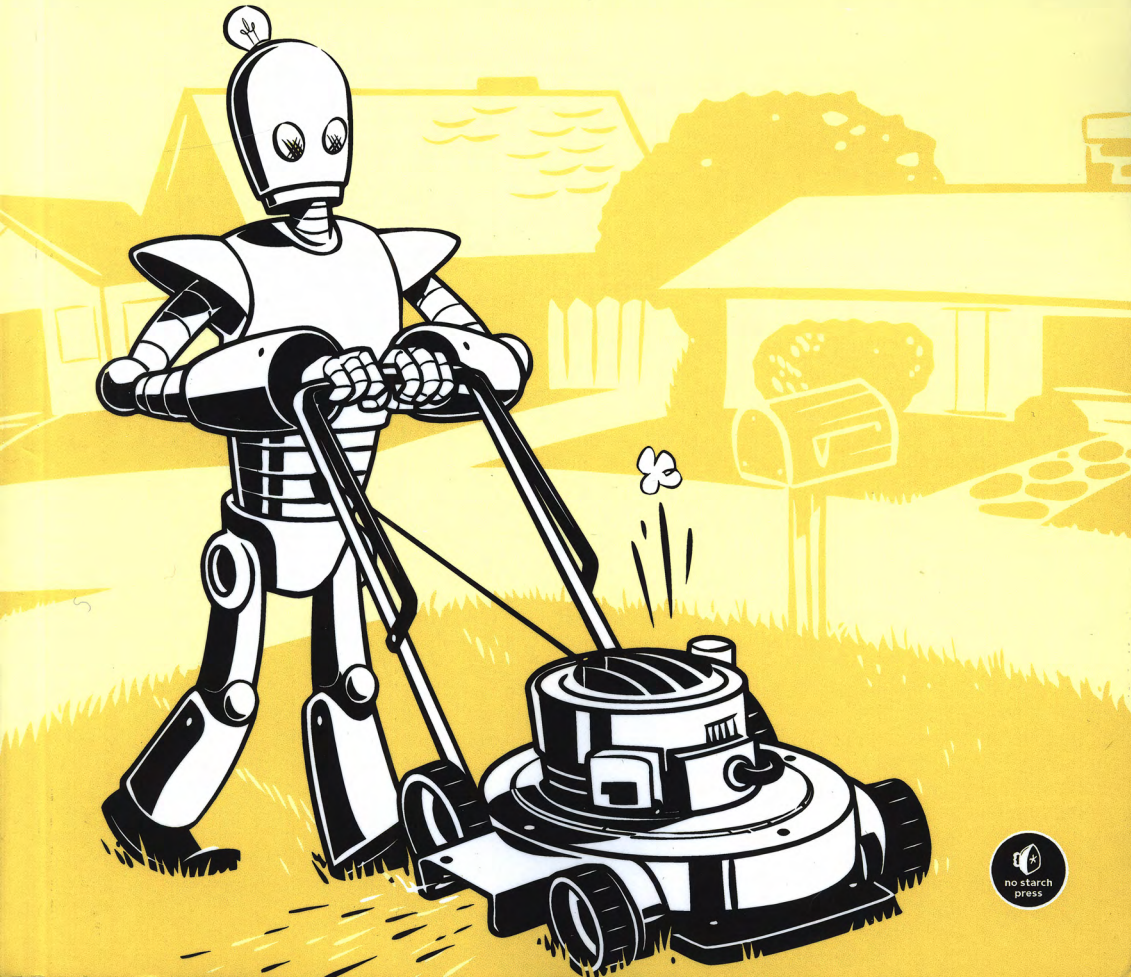
2 Е ИЗДАНИЕ

АВТОМАТИЗАЦИЯ РУТИННЫХ ЗАДАЧ С ПОМОЩЬЮ PYTHON

ПРАКТИЧЕСКОЕ РУКОВОДСТВО ДЛЯ НАЧИНАЮЩИХ

ЭЛ СВЕЙГАРТ



no starch press

AUTOMATE THE BORING STUFF WITH PYTHON

2ND EDITION

Practical Programming for Total Beginners

by Al Sweigart

no starch press

San Francisco

АВТОМАТИЗАЦИЯ РУТИННЫХ ЗАДАЧ С ПОМОЩЬЮ PYTHON

2-Е ИЗДАНИЕ

Практическое руководство для начинающих

Эл Свейгарт

Диалектика

Москва • Санкт-Петербург
2021

ББК 32.973.26-018.2.75
С24
УДК 004.432.2

ООО "Диалектика"

Зав. редакцией *В.Р. Гинзбург*
Перевод с английского канд. хим. наук *А.Г. Гузикевича* и
канд. техн. наук *И.В. Красикова*
Под. редакцией *В.Р. Гинзбурга*

По общим вопросам обращайтесь в издательство "Диалектика" по адресу:
info.dialektika@gmail.com, http://www.dialektika.com

Свейгарт, Эл.

С24 Автоматизация рутинных задач с помощью Python, 2-е изд. : Пер. с англ. – СПб. : ООО "Диалектика", 2021. – 672 с. : ил. – Парал. тит. англ.

ISBN 978-5-907365-55-1 (рус.)

ББК 32.973.26-018.2.75

Научно-популярное издание
Эл Свейгарт
Автоматизация рутинных задач с помощью Python
2-е издание

Подписано в печать 04.06.2021. Формат 70×100/16
Усл. печ. л. 54,2. Уч.-изд. л. 29,9
Тираж 500 экз. Заказ № 5652

Отпечатано в АО "Первая Образцовая типография"
Филиал "Чеховский Печатный Двор"
142300, Московская область, г. Чехов, ул. Полиграфистов, д. 1
Сайт: www.chpd.ru, E-mail: sales@chpd.ru, тел. 8 (499) 270-73-59

ООО "Диалектика", 195027, Санкт-Петербург, Магнитогорская ул., д. 30, лит. А, пом. 848

ISBN 978-5-907365-55-1 (рус.)

ISBN 978-1-59327-992-9 (англ.)

ОГЛАВЛЕНИЕ

СОДЕРЖАНИЕ

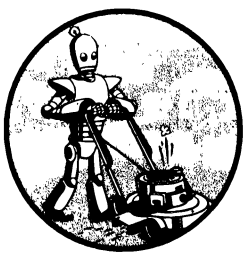
Об авторе

Эл Свейгарт – разработчик программного обеспечения, автор множества компьютерных книг. Python – его любимый язык программирования, для которого он разработал несколько модулей с открытым исходным кодом. Многие книги Эла Свейгарта свободно доступны на условиях лицензии Creative Commons на его сайте `https://inventwithpython.com`.

О техническом рецензенте

Филип Джеймс программирует на Python более десяти лет и часто выступает докладчикам на конференциях, посвященных Python, Unix и сетевым проектам с открытым исходным кодом. Он один из базовых разработчиков проекта BeeWare.

ВВЕДЕНИЕ



“За каких-то пару часов ты сделал то, на что у нас троих ушло бы два дня”. В начале 2000–х годов мой сосед по общежитию в колледже работал в магазине электроники. Время от времени они получали электронные таблицы с прайсами своих конкурентов, включающие тысячи наименований. Распечатка одной таблицы представляла собой толстую стопку бумаги. Обработкой данных занимались три сотрудника магазина. Они сравнивали цены, указанные в таблице, с ценами в своем магазине и отмечали тот товар, который конкуренты продавали по более низкой цене. На эту работу у них уходило примерно два дня.

"Слушайте, если вы дадите мне исходный файл таблицы, то я напишу программу, которая выполнит всю работу за вас", — сказал мой товарищ, увидев, как они копошатся среди груды разбросанных на полу и сложенных в стопки листов.

Через пару часов у него была готова небольшая программа, которая считывала данные о ценах конкурентов из файла, находила для каждого продукта аналог в базе данных магазина и отмечала все товары, цены на которые у конкурентов были ниже. Здесь уместно сказать, что мой товарищ был всего лишь начинающим программистом и бо́льшую часть времени потратил на поиск нужной информации в книге по программированию. Сама программа выполнялась всего несколько секунд, что позволило моему товарищу и его коллегам насладиться в тот день удлиненным обеденным перерывом.

Этот пример наглядно демонстрирует возможности программирования. Компьютер подобен армейскому ножу, который можно использовать в самых разных ситуациях. Многие люди часами сидят за клавиатурой, выполняя одни и те же действия, и даже не догадываются, что компьютер, если снабдить его соответствующими инструкциями, способен сделать все то же самое за считанные секунды.

Для кого предназначена эта книга

В наши дни трудно найти сферу человеческой деятельности, в которой не использовалось бы программное обеспечение. Все мы общаемся в социальных сетях, наши смартфоны — это по сути компьютеры, подключенные к Интернету, а в офисах у всех установлены компьютеры. Как следствие, это привело к стремительному росту спроса на программистов. Бесчисленные книги по программированию, вебинары, семинары для разработчиков — все они обещают превратить амбициозных новичков в компьютерных инженеров, заработная плата которых выражается шестизначными числами.

Это книга не для них. Она предназначена для всех остальных.

Прочтение книги не сделает из вас профессионального разработчика, точно так же, как нескольких уроков игры на гитаре навряд ли будет достаточно для того, чтобы стать рок-звездой. Но если вы офисный работник, администратор, преподаватель или просто используете компьютер для развлечений, то, изучив основы программирования в том объеме, который предлагается в данной книге, вы сможете автоматизировать следующие простые задачи:

- перемещение и переименование тысяч файлов и их сортировка по папкам;
- заполнение веб-форм без ввода данных вручную;

- загрузка файлов или копирование текста с веб-сайта при его обновлении;
- отправка компьютером заранее подготовленных уведомлений;
- обновление и форматирование электронных таблиц Excel;
- проверка электронной почты и рассылка заранее подготовленных писем.

Все это простые задачи, но отнимают у нас массу времени. Кроме того, зачастую они настолько тривиальны или специфичны, что готовых программ для их решения нет. Вооружившись даже минимальными знаниями в области программирования, вы сможете заставить свой компьютер выполнять эти задачи вместо вас.

Исходные предположения

Эта книга — не справочник, а руководство для начинающих. Используемый в ней стиль программирования иногда идет вразрез с общепринятыми практиками (например, в некоторых программах используются глобальные переменные), но это компромиссное решение, позволяющее сделать код более легким для изучения. Книга предназначена для тех, кому будет достаточно научиться писать простой одноразовый код, поэтому стилю оформления программ и приданию им элегантного вида не уделяется особого внимания. В книге не рассматриваются продвинутые концепции программирования, такие как ООП, списковые включения или генераторы, чтобы не усложнять материал. Опытные программисты наверняка найдут в книге те места, где код можно сделать более эффективным, но нас в первую очередь интересует создание работоспособных программ с минимальными усилиями.

Что такое программирование

В сериалах и фильмах часто показывают потоки загадочных нулей и единиц, бегущих по экрану, но реальные компьютерные системы вовсе не такие таинственные, как в "Матрице". *Программирование* — это всего-навсего процесс передачи инструкций компьютеру. Инструкции могут быть связаны с обработкой чисел, редактированием текста, поиском информации в файлах или передачей данных другим компьютерам по сети.

Строительными блоками любых программ служат элементарные инструкции. Вот как выглядят некоторые из них, если перевести их на понятный нам язык:

- "Сделай это, затем сделай то";
- "Если данное условие соблюдается, выполни такое-то действие; в противном случае выполни другое действие";

- "Выполни это действие столько-то раз";
- "Продолжай выполнять эти действия до тех пор, пока данное условие соблюдается".

Эти строительные блоки можно комбинировать для получения более сложных программ. В качестве примера ниже приведены инструкции (*исходный код*) простой программы, написанной на Python. Программа последовательно выполняет каждую строку кода от первой до последней (при этом некоторые инструкции выполняются, только *если* определенное условие выполняется, *иначе* выполняется другая инструкция).

```
❶ passwordFile = open('SecretPasswordFile.txt')
❷ secretPassword = passwordFile.read()
❸ print('Введите пароль.')
  typedPassword = input()
❹ if typedPassword == secretPassword:
❺     print('Доступ разрешен.')
❻     if typedPassword == '12345':
❼         print('Рекомендуем установить более сложный пароль!')
  else:
❽     print('В доступе отказано.')
```

Даже если вы ничего не смыслите в программировании, вы все равно сможете сделать разумные предположения относительно того, что делает этот код, просто читая его. Сначала программа открывает файл *SecretPasswordFile.txt* ❶, из которого считывается пароль ❷, после чего пользователю предлагается ввести свой вариант пароля (с помощью клавиатуры) ❸. Далее оба пароля сравниваются между собой ❹, и если они совпадают, то на экран выводится текст `'Доступ разрешен'` ❺. Затем программа проверяет, не равен ли введенный пароль строке `'12345'` ❻. Если это так, то программа выдает рекомендацию сменить пароль ❼. В случае несовпадения паролей программа выводит на экран сообщение `'В доступе отказано'` ❽.

Что такое Python

Python – это не только язык программирования (со своим синтаксисом, определяющим правила написания корректного кода), но и *интерпретатор*, т.е. программа, предназначенная для чтения исходного кода (написанного на языке Python) и выполнения содержащихся в нем инструкций. Различные версии интерпретатора Python, предназначенные для платформ Linux, macOS и Windows, доступны для бесплатной загрузки на сайте `https://python.org`.

Своим названием Python обязан вовсе не питону, а британской комедийной группе "Монти Пайтон" (Monty Python), работавшей в жанре

абсурдного юмора. Программистов на Python шутливо называют *питонистами*.

Программисту не обязательно в совершенстве знать математику

Многие, кто приступают к изучению программирования, боятся, что им придется интенсивно учить математику. Но в действительности большинству программистов не нужно быть математиками — достаточно знать арифметику. В этом смысле хорошему программисту понадобится не намного больший объем математических знаний по сравнению с тем, который требуется для решения головоломок судоку.

Суть судоку заключается в заполнении цифрами от 1 до 9 каждого из внутренних квадратов размером 3×3, расположенных на игровом поле размером 9×9, причем ни одна строка, ни один столбец и ни один внутренний квадрат игрового поля не должны содержать повторяющихся цифр. Для решения головоломки необходимо использовать дедуктивный метод, исходя из заданной начальной конфигурации цифр. Например, поскольку в головоломке, показанной на рис. 1, цифра 5 находится и в первой сверху, и второй сверху строке, она не может в них повторяться. А раз так, то в правом верхнем квадрате она может быть только в третьей сверху строке. При этом цифра 5 уже стоит в крайнем справа столбце, значит, она может находиться только слева от цифры 6. Последовательное применение подобной логики к строкам, столбцам и внутренним квадратам позволит находить подсказки для заполнения пустых клеток.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Рис. 1. Головоломка судоку (слева) и ее решение (справа). Несмотря на то что это числовая головоломка, никаких математических знаний для ее решения не требуется

Из того факта, что в судоку используются числа, вовсе не следует, что для решения головоломки необходимо быть хорошим математиком. То же самое справедливо и в отношении программирования. Как и в судоку, при

написании программ приходится разбивать задачу более мелкие этапы. Так же и при *отладке* (процесс обнаружения и исправления ошибок в программе) вы кропотливо анализируете действия программы, пытаясь выявить причину ошибки. И чем больше вы программируете, тем лучше у вас это будет получаться.

Учиться программированию никогда не поздно

Второе наиболее распространенное заблуждение относительно изучения программирования заключается в том, что люди думают, будто им уже слишком поздно заниматься этим. Причем так заявляют даже те, кому (вдумайтесь) аж 23 года! Не стоит так рано ставить на себе крест: многие осваивают искусство программирования в гораздо более зрелом возрасте.

Вам вовсе не нужно начинать программировать в школе, чтобы стать программистом. Тем не менее образ юного хакера весьма устойчив. Даже я вношу свою лепту в распространение этого мифа, когда рассказываю, что начал программировать еще в младших классах.

Как бы там ни было, но сегодня учиться программированию намного проще, чем в 1990-е годы. В наши дни доступно множество книг, онлайн-курсов и специализированных сайтов, посвященных программированию. Кроме того, сами языки программирования стали более удобными для изучения. **Все, что я узнал о программировании в школе, сегодня можно выучить за несколько месяцев.** Как видите, мой стартовый багаж знаний оказался не таким уж и большим.

Важно настраивать себя на постоянное обучение, т.е. понимать, что навыки программирования формируются благодаря практике. Люди не рождаются программистами, поэтому отсутствие опыта программирования вовсе не означает, что вы никогда не достигнете уровня эксперта.

Программирование — творческий вид деятельности

Программирование — творческое занятие, как рисование или вязание. Вы начинаете с чистого листа и поначалу ограничены его рамками, но затем перед вами открываются безграничные возможности.

Разница между программированием и другими творческими видами деятельности заключается в том, что все необходимое уже есть в компьютере и вам не нужно ничего докупать. Даже старенького компьютера десятилетней давности вполне достаточно для написания программ. А когда программа готова, ее можно легко скопировать произвольное количество раз. Вязаный свитер будет в конкретный момент времени носить только один человек, тогда как полезной программой можно легко поделиться со всем миром.

Структура книги

В части I рассматриваются основы программирования на Python, тогда как часть II посвящена различным задачам, которые можно автоматизировать. Каждая глава части II включает проекты, которые вам предстоит изучить. Ниже приведено краткое описание глав.

Часть I. Основы программирования на языке Python

- **Глава 1. Основные понятия языка Python.** Здесь рассматриваются выражения — базовые строительные блоки программы, а также описывается, как использовать интерактивную оболочку Python для экспериментов с кодом.
- **Глава 2. Порядок выполнения программы.** Объясняется, как заставить программу выполнять нужные инструкции в зависимости от тех или иных условий.
- **Глава 3. Функции.** Вы узнаете, как создавать собственные функции, разбивая код на логические блоки, с которыми проще работать.
- **Глава 4. Списки.** Вводится понятие *списка* и объясняется, как работать со структурами данных.
- **Глава 5. Словари.** Вводится понятие *словаря* и демонстрируются более мощные структуры данных.
- **Глава 6. Строки.** Описываются способы работы с текстовыми данными (в Python они называются *строками*).

Часть II. Автоматизация задач

- **Глава 7. Регулярные выражения.** Обсуждаются приемы обработки строк и способы поиска образцов текста, соответствующих заданному шаблону, с помощью регулярных выражений.
- **Глава 8. Проверка ввода.** Объясняется, каким образом программа может проверять информацию, которую пользователь предоставляет ей. Тем самым гарантируется, что пользовательские данные поступают в формате, который не вызовет проблем.
- **Глава 9. Чтение и запись файлов.** Будет рассказано, как организовать в программе чтение данных из текстовых файлов и сохранение информации в файлах на диске.
- **Глава 10. Управление файлами.** Рассматриваются автоматизированные способы копирования, перемещения, переименования и удаления файлов, благодаря которым эти операции будут выполняться гораздо быстрее, чем это можно сделать вручную. Также описываются принципы работы со сжатыми файлами.

- **Глава 11. Отладка.** Рассматриваются средства Python, предназначенные для обнаружения ошибок в программах.
- **Глава 12. Веб-скрейпинг.** Будет показано, как писать программы, способные автоматически загружать веб-страницы и извлекать из них данные. Этот процесс называется *веб-скрейпинг.*
- **Глава 13. Работа с таблицами Excel.** Описываются способы автоматизированной работы с электронными таблицами Excel, не требующие открытия самого приложения. Это очень полезно в тех случаях, когда количество обрабатываемых документов исчисляется сотнями или даже тысячами.
- **Глава 14. Работа с приложением Google Таблицы.** Вы узнаете, как загружать и обновлять электронные таблицы веб-приложения Google Таблицы.
- **Глава 15. Работа с документами PDF и Word.** Будут описаны программные методы чтения документов PDF и Word.
- **Глава 16. Работа с CSV-файлами и данными в формате JSON.** Продолжение темы программной обработки документов, только на этот раз в формате CSV и JSON.
- **Глава 17. Работа с датой и временем, планирование заданий и запуск программ.** Объясняется, как обрабатывать в программе значения даты и времени и как запрограммировать компьютер на выполнение задач по расписанию. Также будет показано, как запускать из сценариев Python программы, написанные на других языках.
- **Глава 18. Отправка электронной почты и текстовых сообщений.** Обсуждается написание программ, осуществляющих автоматическую рассылку электронной почты и текстовых сообщений.
- **Глава 19. Работа с изображениями.** Вы узнаете, как обрабатывать изображения, сохраненные в различных форматах, таких как JPEG или PNG.
- **Глава 20. Управление клавиатурой и мышью.** Речь пойдет об управлении клавиатурой и мышью путем программной эмуляции нажатий клавиш и щелчков.
- **Приложение А. Установка сторонних модулей.** Будет показано, каким образом можно расширить возможности Python за счет дополнительных модулей.
- **Приложение Б. Запуск программ.** Вы узнаете, как выполнять программы Python в среде Windows, macOS и Linux, не используя редактор кода.

- **Приложение В. Ответы на контрольные вопросы.** Здесь даны ответы на контрольные вопросы, приведенные в конце каждой главы.

Загрузка и установка Python

Дистрибутивы Python для Windows, macOS и Ubuntu доступны для бесплатной загрузки по адресу `https://python.org/downloads/`. Если вы загрузите текущую версию для своей системы, то все примеры программ, приведенные в книге, должны работать.

Предупреждение

Убедитесь в том, что загружаете версию Python 3 (например, 3.8.0). Все примеры программ в книге написаны с использованием Python 3, и если вы попытаетесь запустить их в версии Python 2, то они могут выполняться неправильно или не выполняться вовсе.

На странице загрузки для каждой операционной системы предлагаются отдельные дистрибутивы, рассчитанные на 64- и 32-разрядные версии, поэтому предварительно определитесь, какой именно вариант вам нужен. Если компьютер был куплен после 2007 года, то, скорее всего, на нем установлена 64-разрядная операционная система. Чтобы убедиться в этом наверняка, выполните следующие действия.

- В Windows выберите Пуск⇨Панель управления⇨Система⇨О программе и проверьте значение поля Тип системы.
- В macOS перейдите в меню Apple, выберите About This Mac⇨More Info⇨System Report⇨Hardware и проверьте значение поля Processor Name. Если там указано "Intel Core Solo" или "Intel Core Duo", то у вас 32-разрядный компьютер. Если же указано что-то другое (включая "Intel Core 2 Duo"), то у вас 64-разрядный компьютер.
- В Ubuntu Linux откройте приложение Terminal и введите команду **`uname -m`**. Ответ `i686` означает, что у вас 32-разрядный компьютер, ответ `x86_64` — 64-разрядный.

В Windows загрузите установщик Python (файл с расширением *.msi*) и дважды щелкните на нем. Далее следуйте инструкциям, отображаемым на экране.

1. Выберите вариант Install for All Users (Установить для всех пользователей) и щелкните на кнопке Next.
2. В следующих окнах примите параметры, заданные по умолчанию, щелкая на кнопке Next.

В macOS загрузите файл с расширением *.dmg*, соответствующий вашей версии macOS, и дважды щелкните на нем. Далее следуйте инструкциям, отображаемым на экране.

1. Когда в новом окне откроется пакет DMG, дважды щелкните на файле *Python.mpkg*. Возможно, вам придется ввести пароль администратора.
2. В следующих окнах щелкайте на кнопках Continue, чтобы принять параметры, заданные по умолчанию, а затем щелкните на кнопке Agree для принятия условий лицензии.
3. В последнем окне щелкните на кнопке Install.

В Ubuntu можно установить Python из окна программы Terminal, выполнив следующие действия.

1. Откройте окно Terminal.
2. Введите команду **`sudo apt-get install python3`**.
3. Введите команду **`sudo apt-get install idle3`**.
4. Введите команду **`sudo apt-get install python3-pip`**.

Загрузка и установка Mu

Интерпретатор Python — это программа, которая выполняет код Python. Редактор Mu — это программа, позволяющая вводить код подобно тому, как вы набираете текст в Word. Редактор Mu доступен для загрузки на сайте `https://codewith.mu/`.

В Windows и macOS загрузите соответствующий инсталлятор и запустите его, дважды щелкнув на файле установки. В macOS при запуске установщика открывается окно, в котором нужно перетащить значок Mu на значок папки *Программы*, чтобы продолжить установку. В Ubuntu необходимо установить Mu как пакет Python. В этом случае щелкните на кнопке Instructions, находящейся в разделе Python Package на странице загрузки.

Запуск Mu

Вот как запустить редактор Mu.

- В Windows щелкните на значке Пуск в левом нижнем углу экрана, введите **`Mu`** в поле поиска и выберите Mu.
- В macOS откройте окно Finder, щелкните на значке Applications, а затем щелкните на значке mu-editor.
- В Ubuntu выберите Applications⇨Accessories⇨Terminal и введите команду **`python3 -m mu`**.

При первом запуске Mu появится окно Select Mode (Выбрать режим), в котором доступны варианты Adafruit CircuitPython, BBC micro:bit, Pygame Zero и Python 3. Выберите Python 3. В дальнейшем вы сможете изменить режим редактора, щелкнув на кнопке Mode в верхней части окна.

Примечание

Чтобы иметь возможность устанавливать сторонние модули, рассматриваемые в книге, загрузите Mu версии 1.1.0 или выше.

Запуск IDLE

В этой книге Mu применяется и как редактор, и как интерактивная оболочка. В то же время для написания кода Python доступно множество других редакторов. IDLE (Integrated Development and Learning Environment) — это интегрированная среда разработки, входящая в состав Python. Она послужит запасным вариантом, если по какой-то причине вам не удастся установить Mu. Вот как запустить IDLE.

- В Windows щелкните на кнопке Пуск в левом нижнем углу экрана, введите **`IDLE`** в поле поиска и выберите IDLE.
- В macOS откройте окно Finder, щелкните последовательно на значках Applications и Python 3.8, а затем щелкните на значке IDLE.
- В Ubuntu выберите Applications⇨Accessories⇨Terminal и введите команду **`idle3`**. (Можете также щелкнуть на кнопке Applications в верхней части экрана, выбрать раздел Programming и щелкнуть на значке IDLE 3.)

Интерактивная оболочка

После запуска Mu появится окно *редактора файла*. Чтобы открыть *интерактивную оболочку*, щелкните на кнопке REPL. Оболочка — это программа, которая позволяет вводить инструкции аналогично тому, как это делается в окне терминала или в командной строке Windows. Команды, вводимые в интерактивной оболочке, тут же выполняются интерпретатором Python.

В Mu интерактивная оболочка представляет собой панель в нижней части окна, где отображается следующий текст.

```
Jupyter QtConsole 4.3.1
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 18:11:49) [MSC v.1900 64
 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information.
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

В случае IDLE интерактивная оболочка – это окно, которое появляется первым. Оно в основном пустое, за исключением текста в верхней части окна.

```
Python 3.8.0b1 (tags/v3.8.0b1:3b5deb0116, Jun 4 2019, 19:52:55)
 [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Обозначения `In [1]:` и `>>>` называются *приглашениями командной строки*. В примерах книги будет использоваться приглашение `>>>`, поскольку оно более распространенное (вы увидите его при запуске Python из окна Terminal или командной строки). Приглашение `In [1]:` появилось в Jupyter Notebook – другом популярном редакторе Python.

Например, введите в интерактивной оболочке следующую команду:

```
>>> print('Hello world!')
```

Как только вы нажмете клавишу <Enter>, оболочка выдаст результат.

```
>>> print('Hello world!')
Hello world!
```

Вы только что передали компьютеру инструкцию, и он выполнил то, о чем вы его попросили!

Установка сторонних модулей

Иногда в программе требуется импортировать дополнительные модули. Некоторые из них поставляются вместе с Python, но есть и сторонние модули, созданные независимыми разработчиками. В приложении А даны инструкции, как использовать утилиту `pip` (в Windows) или `pip3` (в macOS и Linux) для установки сторонних модулей. Обратитесь к нему, когда в книге встретится указание установить тот или иной сторонний модуль.

Как получить справку

Программисты часто получают новые знания, находя в Интернете ответы на свои вопросы. Это сильно отличается от привычного способа обучения через занятия с преподавателем, который читает лекции и может отвечать на вопросы. Преимущество Интернета как обучающей платформы состоит в том, что есть целые сообщества людей, готовых ответить на ваши вопросы. Более того, на многие вопросы наверняка уже были даны ответы, которые остается лишь найти в Интернете. Если вы получили сообщение

об ошибке в программе, то вряд ли вы первый, кто столкнулся с подобной проблемой, и найти решение будет проще, чем вы думаете.

Например, давайте намеренно сгенерируем ошибку следующим образом: введите в интерактивной оболочке выражение `'42' + 3`. Вам необязательно сейчас знать, что оно означает, но результат будет таким, как показано ниже.

```
  >>> '42' + 3
❶ Traceback (most recent call last):
    File "<pyshell#0>", line 1, in <module>
      '42' + 3
❷ TypeError: Can't convert 'int' object to str implicitly
  >>>
```

Появление сообщения об ошибке ❷ обусловлено тем, что смысл введенной вами инструкции оказался непонятным для Python. В той части сообщения, которая касается текущего стека вызовов (`Traceback`) ❶, отображаются конкретная инструкция и номер строки, где интерпретатор столкнулся с проблемой. Если сообщение об ошибке ни о чем вам не говорит, выполните поиск в Интернете по точному тексту сообщения. Введите текст "TypeError: Can't convert 'int' object to str implicitly" (включая внешние кавычки) в поисковой системе, и вы получите тысячи ссылок, по которым можно узнать о том, что означает данное сообщение и что породило ошибку (рис. 2).

Зачастую оказывается, что у кого-то уже возникал аналогичный вопрос и на него уже был дан ответ. Никто не может знать абсолютно все о программировании, поэтому повседневная практика любого разработчика — поиск ответов на различные вопросы технического характера.

Правильно формулируйте вопросы, ответы на которые ищете

Если поиск в Интернете не дал результатов, то попробуйте задать вопрос на таких форумах, как Stack Overflow (`https://ru.stackoverflow.com`) или Reddit (`https://reddit.com/r/learnprogramming`). Но имейте в виду, что при обращении за помощью очень важно правильно формулировать свои вопросы. Обязательно прочитайте разделы "Frequently Asked Questions" (часто задаваемые вопросы) на этих сайтах, где объясняется, как правильно задавать вопросы.

Задавая вопросы, касающиеся программирования, старайтесь придерживаться следующих рекомендаций.

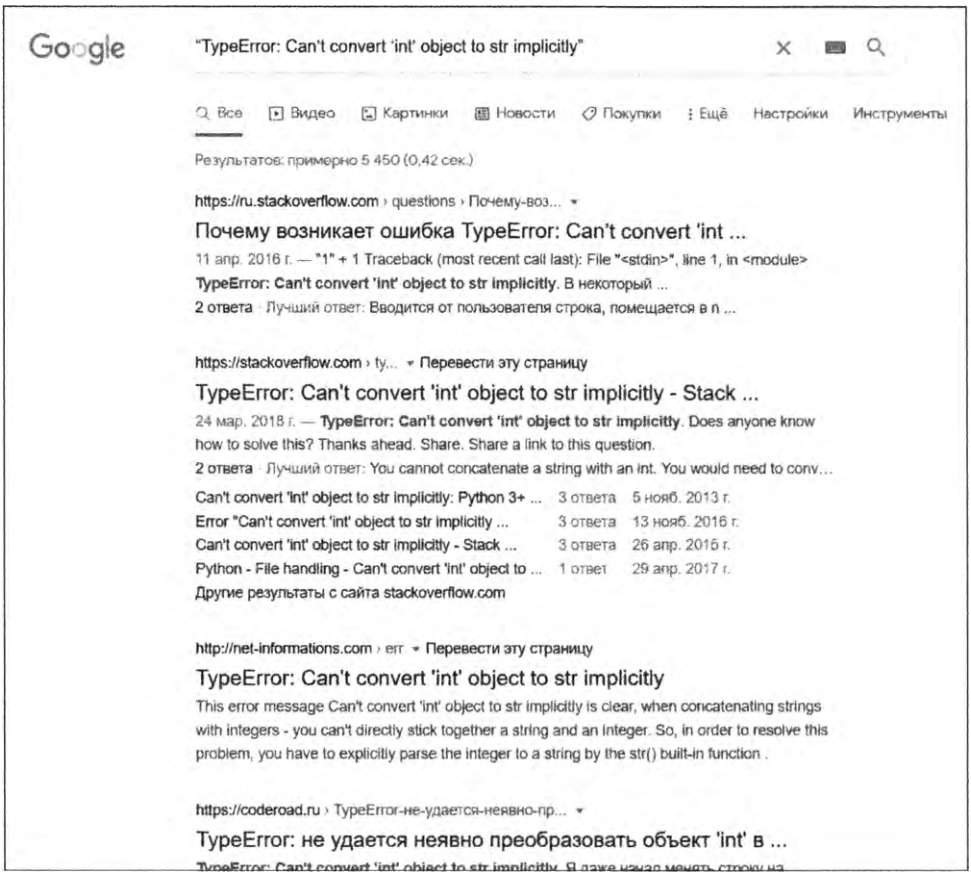


Рис. 2. Получение дополнительной информации о причине ошибки путем поиска в Google с использованием текста сообщения об ошибке в качестве строки поиска

- Объясните, что именно вы *пытаетесь* сделать, а не только то, что вы уже сделали. Это позволит другим понять, где именно вы сбились с пути.
- Укажите, когда именно возникает ошибка: сразу после запуска программы или после того, как выполняются определенные действия.
- Скопируйте и вставьте *полный* текст сообщения об ошибке вместе с программным кодом в хранилище на сайте `https://pastebin.com/` или `https://gist.github.com/`.

  На этих сайтах можно делиться большими объемами кода с другими пользователями, сохраняя форматирование текста. URL-адрес помещенного в хранилище кода можно переслать нужному человеку по электронной почте или опубликовать на форуме. Чтобы увидеть,

как это работает, просмотрите код, который я опубликовал по адресу https://gist.github.com/asweigart/6912168/.

- Объясните, какие меры вы предпринимали для решения возникшей проблемы. Тем самым другие пользователи поймут, что определенная часть работы уже была сделана.
- Укажите используемую версию Python (между интерпретаторами Python версий 2 и 3 имеются существенные различия). Также укажите используемую вами операционную систему и ее версию.
- Если ошибка появилась после того, как вы внесли изменения в код, детально опишите, что именно вы поменяли.
- Укажите, повторяется ли ошибка при каждом запуске программы или же она возникает лишь после того, как вы совершаете определенные действия. Во втором случае опишите, в чем именно заключаются эти действия.

Кроме того, всегда следуйте правилам сетевого этикета. В частности, задавая на форуме вопросы, не набирайте весь текст прописными буквами, пытаясь сделать его более заметным, и не требуйте слишком многого от людей, которые добровольно пытаются вам помочь.

Статья о том, как получить помощь по вопросам программирования, доступна по адресу https://autbor.com/help/. Список часто задаваемых вопросов, связанных с программированием, опубликован по адресу https://www.reddit.com/r/learnprogramming/wiki/faq/. Аналогичный список вопросов, касающихся карьеры программиста, доступен по адресу https://www.reddit.com/r/cscareerquestions/wiki/index/.

Мне нравится помогать людям осваивать Python. Я регулярно публикую соответствующие статьи в своем блоге на сайте https://inventwithpython.com/blog/. Кроме того, мне можно задать вопрос по адресу al@inventwithpython.com. Но если хотите получить более быстрый ответ, опубликуйте вопрос по адресу https://reddit.com/r/inventwithpython/.

Файлы примеров

Архив с файлами Python и дополнительными файлами, которые используются в примерах книги, доступен на сайте издательства No Starch Press:

```
https://nostarch.com/download/
Automate_the_Boring_Stuff_2e_onlinematerials.zip
```

Также архив примеров книги доступен на сайте издательства "Диалектика":

http://go.dialektika.com/automate2

Резюме

Для большинства людей компьютер — всего лишь полезное устройство, а не рабочий инструмент. Но научившись программировать, вы получите доступ к одному из наиболее мощных инструментов в современном мире, работа с которым к тому же доставит вам немалое удовольствие. Программирование — не настолько сложное занятие, и даже любители способны освоить его. Главное — не бояться экспериментировать и совершать ошибки.

Книга подойдет даже для тех, у кого нулевой опыт программирования. Вы многое узнаете из книги, но не рассчитывайте найти в ней ответы на все вопросы. Не забывайте о том, что умение задавать правильные вопросы и находить ответы на них пригодится вам в путешествии в мир программирования.

Итак, приступим!

Ждем ваших отзывов!

Вы, читатель этой книги, и есть главный ее критик. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересны любые ваши замечания в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам электронное письмо либо просто посетить наш веб-сайт и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится ли вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Отправляя письмо или сообщение, не забудьте указать название книги и ее авторов, а также свой обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию новых книг.

Наши электронные адреса:

E-mail: info.dialektika@gmail.com

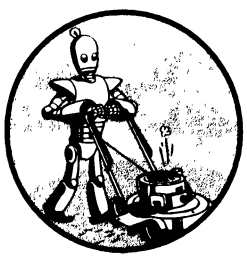
WWW: http://www.dialektika.com

ЧАСТЬ I

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ PYTHON

1

ОСНОВНЫЕ ПОНЯТИЯ ЯЗЫКА PYTHON



Язык программирования Python предлагает огромный набор синтаксических конструкций, функций стандартной библиотеки и средств интерактивной разработки. К счастью, без большинства из них вполне можно обойтись, ведь все, что вам нужно, — это научиться писать короткие полезные программы.

Но прежде чем начинать, необходимо усвоить базовые концепции программирования. Поначалу они могут показаться сложными и запутанными, однако со временем, достаточно попрактиковавшись, вы научитесь управлять компьютером подобно волшебнику, который способен творить чудеса с помощью волшебной палочки.

В этой главе мы разберем несколько примеров работы с *интерактивной оболочкой*, которая позволяет выполнять инструкции Python по одной и сразу же видеть результаты. Интерактивная оболочка станет вашим надежным помощником в изучении основ языка, поэтому мы будем применять ее на протяжении всей книги. Информация усваивается намного лучше, когда вводишь код своими руками, а не просто читаешь книгу.

Ввод выражений в интерактивной оболочке

Чтобы получить доступ к интерактивной оболочке, необходимо запустить редактор Mu, который вы должны были загрузить в процессе выполнения инструкций, содержащихся во введении. В Windows откройте меню Пуск, введите Mu и запустите приложение Mu. В macOS откройте папку Приложения и дважды щелкните на значке Mu. Щелкните на кнопке New и сохраните пустой файл как *blank.py*. Когда вы запустите эту программу, щелкнув на кнопке Run или нажав клавишу <F5>, откроется интерактивная оболочка в виде новой панели, которая появится в нижней части окна редактора Mu. В этой панели вы увидите приглашение >>>.

Введите в командной строке **2 + 2**, чтобы Python выполнил для вас простую математическую операцию.

```
>>> 2 + 2
4
>>>
```

В Python запись 2 + 2 называется *выражением*. Это наиболее фундаментальная разновидность программных инструкций языка. Выражения состоят из *значений* (таких, как 2) и *операторов* (таких, как +), а их результатом всегда будет единственное значение. Это означает, что в коде Python выражения могут использоваться везде, где ожидается значение.

В рассматриваемом примере результатом выражения 2 + 2 будет число 4. Одиночное значение без операторов тоже считается выражением, результат которого равен самому значению.

```
>>> 2
2
```

Не стоит бояться ошибок

Программа может аварийно завершиться, если компьютеру встретится непонятный для него код. В этом случае Python выводит сообщение об ошибке. Такого рода сообщения не могут причинить вред компьютеру, так что не бойтесь совершать ошибки. *Аварийное завершение* всего-навсего означает, что программа неожиданно прекратила работу.

Если хотите получить более подробную информацию об ошибке, введите текст сообщения в поисковой системе Google.

Кроме оператора +, существует множество других операторов. Например, в табл. 1.1 перечислены математические операторы Python.

Таблица 1.1. Математические операторы Python в порядке уменьшения приоритета

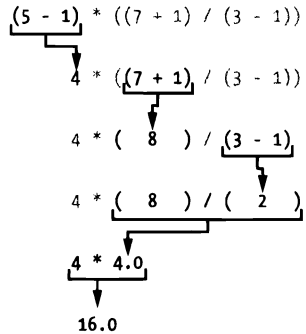
Оператор	Операция	Пример	Результат
**	Возведение в степень	2 ** 3	8
%	Деление по модулю/остаток	22 % 8	6
//	Целочисленное деление с отбрасыванием дробной части	22 // 8	2
/	Деление	22 / 8	2.75
*	Умножение	3 * 5	15
-	Вычитание	5 - 2	3
+	Сложение	2 + 2	4

Приоритет математических операторов Python соответствует порядку выполнения операций, принятому в математике. Сначала применяется оператор **; затем (в порядке слева направо) — операторы *, /, // и %; наконец, последними применяются операторы + и - (тоже в порядке слева направо). В случае необходимости очередность выполнения операций можно изменить с помощью круглых скобок. Количество пробелов между операторами не имеет значения для Python (учитываются только пробелы в начале строки). Общепринятое соглашение — один разделительный пробел. Введите в интерактивной оболочке следующие выражения.

```
>>> 2 + 3 * 6
20
>>> (2 + 3) * 6
30
>>> 48565878 * 578453
28093077826734
>>> 2 ** 8
256
>>> 23 / 7
3.2857142857142856
```

```
>>> 23 // 7
3
>>> 23 % 7
2
>>> 2      +           2
4
>>> (5 - 1) * ((7 + 1) / (3 - 1))
16.0
```

Во всех этих случаях вы как программист лишь вводите выражения, тогда как всю рутинную работу по сведению выражения к единственному значению берет на себя интерпретатор. Python последовательно вычисляет отдельные части выражения до тех пор, пока не получит итоговое значение.



Указанные принципы построения выражений фундаментальны для языка программирования Python точно так же, как правила грамматики позволяют нам общаться на родном языке. Рассмотрим пример.

Это грамматически корректное предложение.
Это корректное грамматически предложение не.

Понять смысл второго высказывания сложно, поскольку оно построено не по правилам языка. Точно так же и Python, встретив неправильно составленную инструкцию, не сможет ее однозначно интерпретировать и выдаст сообщение о синтаксической ошибке (`SyntaxError`).

```
>>> 5 +
  File "<stdin>", line 1
    5 +
      ^
SyntaxError: invalid syntax
>>> 42 + 5 + * 2
  File "<stdin>", line 1
    42 + 5 + * 2
             ^
SyntaxError: invalid syntax
```

Вы всегда можете проверить, работает ли та или иная инструкция, введя ее в интерактивной оболочке. Вам не о чем волноваться — компьютер не выйдет из строя. В худшем случае Python отреагирует на неправильную инструкцию выдачей сообщения об ошибке. Даже профессиональные разработчики постоянно получают подобные сообщения в процессе написания кода.

Типы данных: целые числа, вещественные числа и строки

Помните, что выражения — это просто комбинации значений и операторов, а результат их вычисления всегда сводится к единственному значению. *Тип данных* — это конкретная категория значений, причем каждое значение относится к одному и только к одному типу данных. Самые распространенные типы данных Python приведены в табл. 1.2. Значения наподобие `-2` и `30` называются *целочисленными*. Им соответствует тип `int`. Числа с десятичной точкой, как, например, `3.14`, называются *числами с плавающей точкой* (тип данных `float`) или *вещественными числами*. В частности, значение `42` — целое число, тогда как `42.0` — вещественное.

Таблица 1.2. Базовые типы данных Python

Тип данных	Примеры
Целые числа	`-2, -1, 0, 1, 2, 3, 4, 5`
Числа с плавающей точкой (вещественные)	`-1.25, -1.0, -0.5, 0.0, 0.5, 1.0, 1.25`
Строки	`'a', 'aa', 'aaa', 'Hello!', '11 cats'`

В программах на языке Python часто используются текстовые значения, называемые *строками* (тип данных `str`). Их следует заключать в одинарные кавычки (`'`), чтобы интерпретатор понимал, где начинается и заканчивается строка (например, `'Привет'` или `'Прощай навсегда!'`). Строка может вообще не содержать ни одного символа (`''`); такие строки называются *пустыми* (более подробно строки рассматриваются в главе 6).

Если вы столкнетесь с сообщением об ошибке `SyntaxError: EOL while scanning string literal`, то это означает, что вы пропустили закрывающую кавычку.

```
>>> 'Здравствуй, мир!
SyntaxError: EOL while scanning string literal
```

Конкатенация и репликация строк

Смысл оператора может меняться в зависимости от типа операндов (используемых значений). Например, если оператор + применяется к двум числам (целым или вещественным), то он выполняет операцию сложения. Но если его применить к двум строковым значениям, то он объединит их в одну строку. Такая операция называется *конкатенацией*. Введите в интерактивной оболочке следующее выражение.

```
>>> 'Алиса' + 'Боб'
'АлисаБоб'
```

Результатом будет новая строка, объединяющая текст обеих исходных строк. Если же попытаться применить оператор + к строке и целому числу, то Python не сможет понять, как выполнить такую операцию, и выдаст сообщение об ошибке.

```
>>> 'Алиса' + 42
Traceback (most recent call last):
  File "<pyshell#26>", line 1, in <module>
    'Алиса' + 42
TypeError: can only concatenate str (not "int") to str
```

Сообщение `TypeError: can only concatenate str (not "int") to str` означает, что интерпретатор посчитал, будто вы пытаетесь присоединить число `42` к строке `'Alice'`. В программе необходимо явно преобразовывать целые числа в строки, так как подобная операция не выполняется автоматически. (О преобразовании типов данных мы поговорим в разделе "Анатомия программы".)

Если оператор * применяется к числам (целым или вещественным), то он трактуется как оператор умножения. Но если одно из значений — строка, а второе — целое число, то он становится оператором *репликации строк*. Введите в интерактивной оболочке следующее выражение.

```
>>> 'Алиса' * 5
'АлисаАлисаАлисаАлисаАлиса'
```

Результатом будет строка, представляющая собой многократно повторенную исходную строку, причем количество повторов равно указанному целому числу. Репликация строки — полезный прием, хотя он применяется не так часто, как конкатенация.

Оператор * может применяться только к двум числовым значениям (умножение) или строке и целому числу (репликация строки). В противном случае Python выдаст сообщение об ошибке.

```
>>> 'Алиса' * 'Боб'
Traceback (most recent call last):
  File "<pyshell#32>", line 1, in <module>
    'Алиса' * 'Боб'
TypeError: can't multiply sequence by non-int of type 'str'
>>> 'Alice' * 5.0
Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    'Alice' * 5.0
TypeError: can't multiply sequence by non-int of type 'float'
```

Совершенно очевидно, почему Python не смог определить смысл этих выражений: невозможно умножить одно слово на другое, равно как повторить строку дробное количество раз.

Сохранение значений в переменных

Переменная — это область памяти компьютера, в которой может храниться одиночное значение. Если вы хотите сохранить результат выражения для дальнейшего использования, то необходимо записать его в переменную.

Операции присваивания

Для сохранения значений в переменных используется *операция присваивания*. В ней указывается имя переменной, знак = (*оператор присваивания*) и сохраняемое значение. Например, если ввести **spam = 42**, то значение `42` будет сохранено в переменной `spam`.

Переменную можно сравнить с надписанной коробкой, в которую помещается переменная (рис. 1.1).

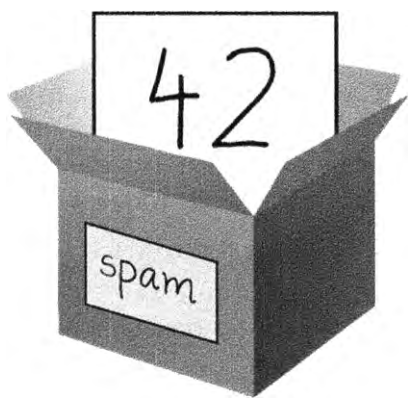


Рис. 1.1. Инструкция `spam = 42` сообщает программе: "Теперь в переменной `spam` хранится целочисленное значение `42`"

Введите в интерактивной оболочке следующие инструкции.

```
❶ >>> spam = 40
  >>> spam
  40
  >>> eggs = 2
❷ >>> spam + eggs
  42
  >>> spam + eggs + spam
  82
❸ >>> spam = spam + 2
  >>> spam
  42
```

При первой записи значения происходит *инициализация* (создание) переменной ❶. После этого переменную можно использовать в выражениях наряду с другими переменными и значениями ❷. Когда в переменную записывается новое значение ❸, прежнее значение теряется; именно поэтому значением переменной `spam` в конце примера становится `42`, а не `40`. В таком случае говорят, что переменная *перезаписана*. Попробуйте перезаписать строку, введя в интерактивной оболочке следующие инструкции.

```
>>> spam = 'Hello'
>>> spam
'Hello'
>>> spam = 'Goodbye'
>>> spam
'Goodbye'
```

В этом примере значение `'Hello'` хранится в переменной `spam` лишь до тех пор, пока вы не замените его значением `'Goodbye'` (рис. 1.2).

Имена переменных

Переменной желательно дать описательное имя, чтобы было понятно, какие именно данные она содержит. В табл. 1.3 приведены примеры допустимых имен переменных. Переменным можно присваивать любые имена, при условии, что они удовлетворяют следующим трем ограничениям:

1) имя переменной должно представлять собой одно слово (без пробелов);
2) в имени переменной могут использоваться только буквы, цифры и символы подчеркивания (_);
3) имя переменной не может начинаться с цифры.

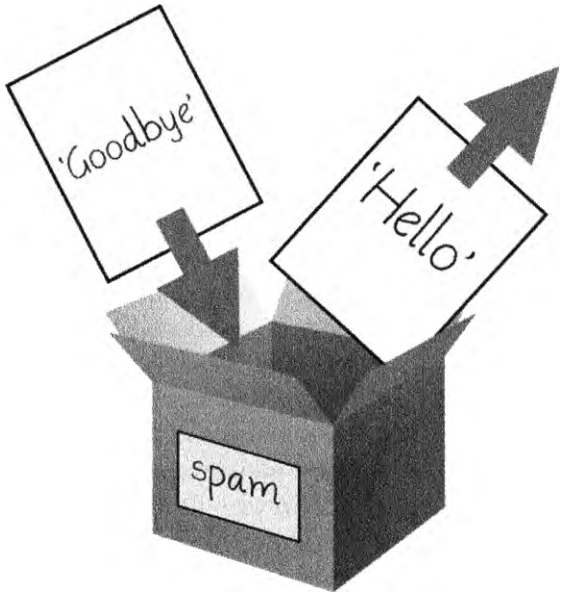


Рис. 1.2. Когда переменной присваивается новое значение, старое значение теряется

Таблица 1.3. Допустимые и недопустимые имена переменных

Допустимые имена переменных	Недопустимые имена переменных
`current_balance`	`current-balance` (дефисы не разрешены)
`currentBalance`	`current balance` (пробелы не разрешены)
`account4`	`4account` (имя не может начинаться с цифры)
`_42`	`42` (имя не может начинаться с цифры)
`TOTAL_SUM`	`total_$um` (специальные символы, такие как `$`, не разрешены)
`hello`	`'hello'` (специальные символы, такие как `'`, не разрешены)

Имена переменных чувствительны к регистру. Это означает, что `spam`, `SPAM`, `Spam` и `sPaM` — четыре разных имени. В соответствии с принятым в Python соглашением имена переменных должны начинаться со строчной буквы, но вы вольны использовать в своих программах все перечисленные имена.

В данной книге для имен переменных применяется так называемый *верблюжий стиль* (CamelCase), в котором имена переменных выглядят как `lookingLikeThis`, а не `looking_like_this`. Некоторые программисты могут возразить, что в соответствии с официальным руководством Python по стилю кодирования — PEP 8 — в именах переменных должны использоваться символы подчеркивания. Но лично для меня "верблюжий стиль" предпочтителен, а своеобразным оправданием может послужить следующая цитата из самого руководства:

> "Согласованность с этим руководством очень важна... Но важно помнить, что иногда это руководство неприменимо, и понимать, когда можно отойти от рекомендаций. Если возникают сомнения, ориентируйтесь на собственную оценку".

Ваша первая программа

Интерактивная оболочка отлично подходит для выполнения инструкций Python одна за другой, но для написания полноценных программ вам нужен *файловый редактор*. Это, по сути, такой же текстовый редактор, как Блокнот или TextMate, но с поддержкой синтаксической разметки кода. Чтобы открыть файловый редактор в Mu, щелкните на кнопке **New**.

В открывшемся окне вы увидите курсор ввода, но это окно отличается от интерактивной оболочки, в которой введенные инструкции выполняются сразу же после нажатия клавиши <Enter>. Файловый редактор позволяет ввести множество инструкций, сохранить файл и запустить программу. Вот как отличить окно интерпретатора от окна редактора:

- в окне оболочки всегда отображается приглашение >>>;
- в окне редактора нет приглашения >>>.

А сейчас пришло время написать первую программу! Введите в окне файлового редактора следующий код.

```
❶ # Эта программа выдает приветствие и запрашивает имя пользователя

❷ print('Здравствуй, мир!')
  print('Как тебя зовут?')     # запрос имени
❸ myName = input()
❹ print('Рад познакомиться с тобой, ' + myName)
❺ print('Длина твоего имени:')
  print(len(myName), ' буквы')
❻ print('Сколько тебе лет?')     # запрос возраста
  myAge = input()
  print('Через год тебе будет ' + str(int(myAge) + 1) + ' лет.')
```

Сохраните исходный код, чтобы не набирать его заново при каждом запуске Mu. Щелкните на кнопке **Save**, введите **`hello.py`** в поле имени файла и сохраните файл.

В процессе ввода кода необходимо периодически сохранять файл, чтобы не потерять сделанную работу, если вдруг произойдет сбой системы или вы случайно выйдете из Mu. Для этого следует запомнить удобную комбинацию клавиш: <Ctrl+S> (Windows и Linux) или <⌘+S> (macOS).

Сохранив файл, попробуем выполнить программу. Нажмите клавишу <F5>. Программа должна запуститься в окне интерактивной оболочки. Не

забывайте о том, что клавишу <F5> следует нажимать в окне файлового редактора, а не в окне оболочки! Введите свое имя в ответ на приглашение программы. Результат работы программы должен выглядеть примерно так.

```
Python 3.7.0b4 (v3.7.0b4:eb96c37699, May 2 2018, 19:02:22)
[MSC v.1913 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()"
for more information.
>>> ========================= RESTART ========================
>>>
Здравствуй, мир!
Как тебя зовут?
Эл
Рад познакомиться с тобой, Эл
Длина твоего имени:
2 буквы
Сколько тебе лет?
4
Через год тебе будет 5 лет.
>>>
```

Выполнив последнюю строку кода, программа *завершается*, т.е. перестает выполняться. (В таком случае говорят о *выходе* из программы.)

Чтобы закрыть окно файлового редактора, щелкните на кнопке × в верхней части окна. Чтобы перезагрузить сохраненную программу, выполните команду File⇨Open..., выберите в открывшемся окне файл *hello.py* и щелкните на кнопке Open. В окне файлового редактора должна открыться программа, которую вы перед этим сохранили в файле *hello.py*.

Для визуализации работы программ имеется удобный инструмент Python Tutor (`http://pythontutor.com/`). Результат выполнения данной конкретной программы (в авторском варианте) можно увидеть по адресу `https://autbor.com/hellopy/`. Щелкайте на кнопке Next >, чтобы последовательно пройти все строки программы. Вы сможете увидеть, как меняются значения переменных и результаты работы программы.

Анализ программы

Сейчас мы кратко рассмотрим все инструкции нашей первой программы, открытой в окне файлового редактора, и проанализируем, что конкретно делает каждая строка кода.

Комментарии

Следующая строка называется *комментарием*.

```
❶ # Эта программа выдает приветствие и запрашивает имя пользователя
```

Python игнорирует комментарии, поэтому их можно использовать для записи примечаний или напоминаний самому себе о том, что делает данный фрагмент программы. Любой текст от символа решетки (#) до конца строки становится частью комментария.

Иногда программисты ставят символ # перед строкой кода для того, чтобы временно (*закомментировать*) отключить ее на этапе тестирования. Этот прием оказывается очень полезным, когда нужно выяснить причины сбоев в работе программы. Впоследствии достаточно удалить символ #, и код строки снова станет выполняемым.

Пустая строка после комментария тоже игнорируется. В программе может быть сколько угодно пустых строк. Они упрощают чтение листинга, подобно абзацам в книге.

Функция `print()`

Функция `print()` выводит содержащуюся в скобках строку на экран.

```
❷ print('Здравствуй, мир!')
  print('Как тебя зовут?')     # запрос имени
```

Инструкция `print('Здравствуй, мир!')` означает: "Отобразить на экране строку `'Здравствуй, мир!'`". Python *вызывает* функцию `print()`, передавая ей указанное строковое значение в качестве *аргумента*. Обратите внимание на то, что кавычки не выводятся на экран. Они лишь задают начало и конец строки, но сами не являются частью строкового значения.

Примечание

Эту же функцию можно использовать для вывода на экран пустой строки. Для этого достаточно вызвать функцию `print()` *без аргументов.*

Наличие пары скобок после имени говорит о том, что перед нами функция. Поэтому в книге пишется `print()`, а не `print` (более подробно функции рассматриваются в главе 3).

Функция `input()`

Функция `input()` ожидает, пока пользователь не введет на клавиатуре какой-нибудь текст и нажмет <Enter>.

```
❸ myName = input()
```

Функция возвращает введенную пользователем строку текста, которая записывается в переменную `myName`.

Вызов функции `input()` можно рассматривать как выражение, значением которого является строка, введенная пользователем. Если пользователь ввел `'Al'`, то операция присваивания выглядит как `myName = 'Al'`.

Примечание: если после вызова функции `input()` появится сообщение об ошибке `NameError: name 'Al' is not defined`, значит, вы пытаетесь выполнить код в среде Python 2, а не Python 3.

Вывод имени пользователя

В следующем коде функция `print()` получает выражение `'Рад познакомиться с тобой, ' + myName` в качестве аргумента.

```
❹ print('Рад познакомиться с тобой, ' + myName)
```

Вспомните, что результатом вычисления выражения всегда будет одиночное значение. Если в строке ❸ в переменную `myName` было записано значение `'Al'`, то функция `print()` получит строку `'Рад познакомиться с тобой, Эл'`. Именно она и будет выведена на экран.

Функция `len()`

Функция `len()` получает в качестве аргумента строку (или строковую переменную) и возвращает целое число, равное количеству символов в данной строке.

```
❺ print('Длина твоего имени:')
  print(len(myName), ' букв')
```

Введите в интерактивной оболочке следующие команды.

```
>>> len('привет')
6
>>> len('А теперь нечто совсем другое.')
29
>>> len('')
0
```

Как и в приведенных выше примерах, результатом вызова `len(myName)` будет целое число. Далее это число передается функции `print()`, которая выводит его на экран. Учтите, что в функцию `print()` можно передавать либо числа, либо строки. Если ввести в интерактивной оболочке следующую инструкцию, то вы получите сообщение об ошибке.

```
>>> print('Мне ' + 29 + ' лет.')
Traceback (most recent call last):
```

```
  File "<pyshell#6>", line 1, in <module>
    print('Мне ' + 29 + ' лет.')
TypeError: can only concatenate str (not "int") to str
```

Ошибка связана не с самой функцией `print()`, а с выражением, которое вы пытаетесь ей передать. То же самое произойдет, если ввести в интерактивной оболочке само выражение.

```
>>> 'Мне ' + 29 + ' лет.'
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    'Мне ' + 29 + ' лет.'
TypeError: can only concatenate str (not "int") to str
```

Python выдает сообщение об ошибке, потому что оператор + может либо суммировать два числа, либо выполнить конкатенацию двух строк. Сложить число со строкой невозможно, потому что такая операция не поддерживается в Python. Проблему можно решить, преобразовав число в строку, о чем пойдет речь ниже.

Функции str(), int() и float()

Если необходимо объединить целое число, например `29`, со строкой, чтобы передать результат в функцию `print()`, то число должно быть представлено в строковом виде как `'29'`. Соответствующее преобразование выполняет функция `str()`.

```
>>> str(29)
'29'
>>> print('Мне ' + str(29) + ' лет.')
Мне 29 лет.
```

Поскольку вызов `str(29)` возвращает строку `'29'`, выражение `'Мне ' + str(29) + ' лет.'` преобразуется в `'Мне ' + '29' + ' лет.'`, что, в свою очередь, трансформируется в строку `'Мне 29 лет.'`. Это значение и передается функции `print()`.

Функции `str()`, `int()` и `float()` возвращают соответственно строковое, целочисленное и вещественное представление аргумента. Попробуем выполнить ряд преобразований в интерактивной оболочке и посмотрим, что произойдет.

```
>>> str(0)
'0'
>>> str(-3.14)
'-3.14'
```

```
>>> int('42')
42
>>> int('-99')
-99
>>> int(1.25)
1
>>> int(1.99)
1
>>> float('3.14')
3.14
>>> float(10)
10.0
```

В этих примерах функции `str()`, `int()` и `float()` вызываются для получения строкового, целочисленного и вещественного представления других типов данных.

Функцию `str()` удобно использовать в тех случаях, когда необходимо конкатенировать целое или вещественное число со строкой. Функция `int()` будет полезной, если необходимо выполнить операции над числом, которое хранится в строковом виде. Например, функция `input()` всегда возвращает строку, даже когда пользователь вводит число. Попробуйте ввести в интерактивной оболочке инструкцию **`spam = input()`**, а затем задать число **`101`**.

```
>>> spam = input()
101
>>> spam
'101'
```

В переменной `spam` сохраняется не число `101`, а строка `'101'`. Если с введенным значением необходимо выполнить определенные вычисления, воспользуйтесь функцией `int()`, чтобы получить целочисленное представление переменной `spam`, и сохраните результат в этой же переменной.

```
>>> spam = int(spam)
>>> spam
101
```

Теперь с переменной `spam` можно работать как с числом, а не строкой.

```
>>> spam * 10 / 5
202.0
```

Если передать в функцию `int()` значение, которое не может быть приведено к целочисленному виду, интерпретатор выдаст сообщение об ошибке.

```
>>> int('99.99')
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    int('99.99')
ValueError: invalid literal for int() with base 10: '99.99'
>>> int('двенадцать')
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    int('двенадцать')
ValueError: invalid literal for int() with base 10: 'двенадцать'
```

Функцию `int()` удобно использовать для округления вещественных чисел "вниз" (с отбрасыванием дробной части).

```
>>> int(7.7)
7
>>> int(7.7) + 1
8
```

В нашей программе функции `int()` и `str()` используются в самом конце для приведения значений к нужным типам данных.

```
❻ print('Сколько тебе лет?')     # запрос возраста
  myAge = input()
  print('Через год тебе будет ' + str(int(myAge) + 1) + ' лет.')
```

В переменной `myAge` содержится результат, возвращаемый функцией `input()`. Поскольку функция `input()` всегда возвращает строку (даже если пользователь ввел число), для преобразования строкового значения, хранящегося в переменной `myAge`, в целочисленное следует использовать вызов `int(myAge)`. Затем это значение увеличивается на `1` в выражении `int(myAge) + 1`.

Результат данного выражения передается функции `str()` в виде вызова `str(int(myAge) + 1)`. Полученная строка конкатенируется со строками `'Через год тебе будет '` и `' лет.'` для формирования результирующей строки, которая передается функции `print()` для вывода на экран.

Предположим, пользователь вводит `4` в качестве значения переменной `myAge`. Строка `'4'` преобразуется в целое число, чтобы к нему можно было прибавить единицу. В результате мы получаем значение `5`. Функция `str()` преобразует этот результат обратно в строку, которую можно конкатенировать со второй строкой, `' лет.'`, для формирования окончательного сообщения. Ниже показана последовательность преобразований.

```
print('Через год тебе будет ' + str(int(myAge) + 1) + ' лет.')
print('Через год тебе будет ' + str(int( '4' ) + 1) + ' лет.')
print('Через год тебе будет ' + str(    4 + 1     ) + ' лет.')
print('Через год тебе будет ' + str(      5       ) + ' лет.')
print('Через год тебе будет ' +          '5'        + ' лет.')
print('Через год тебе будет 5'                      + ' лет.')
print('Через год тебе будет 5 лет.')
```

Сравнение строк и чисел

Строковое представление числа не будет равно самому числу, в то время как целое число может быть равно вещественному.

```
>>> 42 == '42'
False
>>> 42 == 42.0
True
>>> 42.0 == 0042.000
True
```

Резюме

Выражения и их компоненты — операторы, переменные и вызовы функций — это те строительные блоки, на основе которых создаются программы. Освоив выражения, вы сможете с помощью Python оперировать большими объемами данных.

В этой главе мы рассмотрели основные операторы (`+`, `-`, `*`, `/`, `//`, `%` и `**` для математических операций, а также `+` и `*` для строковых операций) и три базовых типа данных (целые и вещественные числа плюс строки).

Вы также познакомились с несколькими функциями. Функции `print()` и `input()` предназначены для вывода текста на экран и ввода текста с клавиатуры. Функция `len()` позволяет определить количество символов в строке. Функции `str()`, `int()` и `float()` возвращают соответственно строковое, целочисленное и вещественное представление переданного им аргумента.

В следующей главе вы узнаете, как в программе принимать решения о том, какой код выполнять, какой — пропускать, а какой — повторять на основе имеющихся значений. Для этого применяются *управляющие инструкции*, с помощью которых можно писать программы, способные принимать гибкие решения.

Контрольные вопросы

1. Что из нижеперечисленного — оператор, а что — значение?

```
*
'привет'
-88.8
-
/
+
5
```

2. Что из этого — переменная, а что — строка?

```
spam
'spam'
```

3. Назовите три основных типа данных.
4. Из чего состоит выражение? К чему сводится любое выражение?
5. В этой главе рассматривались инструкции наподобие `spam = 10`. В чем разница между выражением и инструкцией?
6. Чему будет равна переменная `bacon` после выполнения следующего кода?

```
bacon = 20
bacon + 1
```

7. Каким будет результат вычисления следующих двух выражений?

```
'spam' + 'spamspam'
'spam' * 3
```

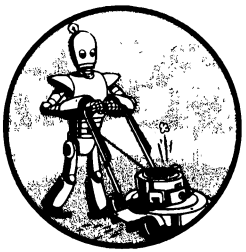
8. Почему `eggs` — допустимое имя переменной, а `100` — нет?
9. Какие три функции можно использовать для получения целочисленного, вещественного и строкового представления числа?
10. Почему следующее выражение вызывает ошибку? Как от нее избавиться?

```
'Я съел ' + 99 + ' котлет.'
```

Дополнительное задание: поищите в Интернете информацию о функции `len()`. Она относится к категории встроенных функций ("Built-in Functions"). Ознакомьтесь со списком других встроенных функций Python, обратив особое внимание на функцию `round()`. Самостоятельно поэкспериментируйте с ней в интерактивной оболочке.

2

ПОРЯДОК ВЫПОЛНЕНИЯ ПРОГРАММЫ



Итак, вы познакомились с простейшими инструкциями и знаете, что последовательность таких инструкций образует программу. Однако программирование заключается не в том, чтобы выполнять инструкции одну за другой, словно вы покупаете в магазине продукты по списку. Программа способна анализировать выражения и самостоятельно принимать решения о том, какие инструкции следует пропустить, а какие — повторить. Навряд ли вам когда-нибудь доведется написать программу, которая будет выполнять инструкции по порядку, от начала до конца. В Python имеются *управляющие инструкции*, или *инструкции ветвления*, которые позволяют выбрать, какой фрагмент программы следует выполнить и при каких условиях.

Управляющие инструкции удобно изображать в виде блок-схем, поэтому в данной главе будут приводиться блок-схемы рассматриваемых программ. На рис. 2.1 показан процесс принятия решений в случае дождя. Попробуйте пройти по стрелкам от начала до конца.

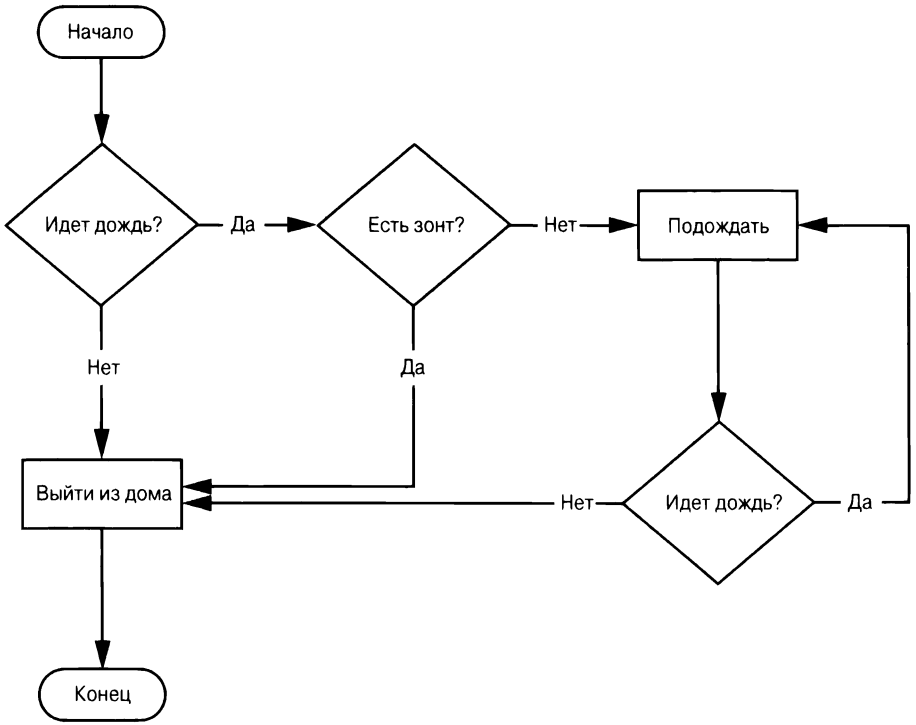


Рис. 2.1. Блок-схема, определяющая порядок действий на случай дождя

Обычно блок-схема предлагает несколько возможных маршрутов, ведущих от начала к концу. То же самое справедливо и для компьютерной программы. Точки принятия решений (условные инструкции) обозначаются на блок-схеме ромбами, тогда как остальные инструкции представлены прямоугольниками. Началу и концу программы соответствуют скругленные прямоугольники.

Но прежде чем приступать к изучению управляющих инструкций, следует узнать, как задавать в программе варианты выбора "да/нет" и как кодировать точки ветвления в коде Python.

Булевы значения

Целочисленные, вещественные и строковые переменные могут иметь неограниченное количество значений, в то время как *булев*[1] тип данных предполагает всего два возможных значения: `True` (истина) и `False` (ложь). В коде Python булевы значения `True` и `False` не заключаются в кавычки и всегда начинаются с прописной буквы `T` или `F`, тогда как остальная часть слова записывается строчными буквами. Введите в интерактивной оболочке следующие инструкции (некоторые из них неправильны, и вы получите сообщение об ошибке).

```
❶ >>> spam = True
  >>> spam
  True
❷ >>> true
  Traceback (most recent call last):
    File "<pyshell#2>", line 1, in <module>
      true
  NameError: name 'true' is not defined
❸ >>> True = 2 + 2
  SyntaxError: can't assign to keyword
```

Как и любые другие значения, булевы значения можно использовать в выражениях и сохранять в переменных ❶. В случае применения неправильного регистра букв ❷, а также при попытке использовать идентификатор `True` или `False` в качестве имени переменной ❸ Python выдаст сообщение об ошибке.

Операторы сравнения

Операторы сравнения сопоставляют два значения и возвращают результат в виде булева значения (табл. 2.1).

Таблица 2.1. Операторы сравнения

Оператор	Операция
==	Равно
!=	Не равно
<	Меньше
>	Больше
<=	Меньше или равно
>=	Больше или равно

[1] Булев тип данных назван так в честь английского математика, основателя математической логики Джорджа Буля.

Результатом операции сравнения будет значение `True` или `False`. Рассмотрим это на примере операторов == и !=.

```
>>> 42 == 42
True
>>> 42 == 99
False
>>> 2 != 3
True
>>> 2 != 2
False
```

Как и следовало ожидать, результатом операции == (равно) будет `True` в случае равенства обоих операндов, тогда как результатом операции != (не равно) будет `True`, если операнды различаются. Операторы == и != поддерживают любые типы операндов.

```
>>> 'привет' == 'привет'
True
>>> 'привет' == 'Привет'
False
>>> 'собака' != 'кот'
True
>>> True == True
True
>>> True != False
True
>>> 42 == 42.0
True
❶ >>> 42 == '42'
False
```

Обратите внимание на то, что целые и вещественные значения никогда не могут быть равны строковым. Результат выражения `42 == '42'` ❶ равен `False`, поскольку для Python целое число `42` и строка `'42'` — разные значения.

В то же время операторы <, >, <= и >= могут работать только с целыми и вещественными значениями.

```
>>> 42 < 100
True
>>> 42 > 100
False
>>> 42 < 42
False
>>> eggCount = 42
❶ >>> eggCount <= 42
True
```

```
>>> myAge = 29
❷ >>> myAge >= 10
True
```

Различие между операторами == и =

Вы наверняка заметили, что оператор == (равно) содержит два знака равенства, тогда как оператор = (присваивание) — один. Их легко перепутать, поэтому запомните следующее:

- оператор == (равно) проверяет равенство двух значений;
- оператор = (присваивание) помещает значение, указанное справа, в переменную, указанную слева.

Возможно, вам будет легче запомнить, что к чему, благодаря оператору != (не равно), который, как и оператор ==, тоже состоит из двух символов.

Операторы сравнения чаще всего используют для того, чтобы сравнить значение переменной с константой, как в выражениях `eggCount <= 42` ❶ и `myAge >= 10` ❷. (В конце концов, нет смысла применять в программе инструкцию `'dog' != 'cat'`, если можно записать просто `True`.) В последующих разделах вы увидите множество подобных примеров.

Булевы операторы

Для сравнения булевых значений используются три булевых оператора: `and`, `or` и `not`. Подобно операторам сравнения, они вычисляют булевы выражения, сводя их к единственному булевому значению. Рассмотрим их более подробно, начав с оператора `and`.

Бинарные булевы операторы

Операторы `and` и `or` всегда работают с двумя булевыми значениями (или выражениями), поэтому их называют *бинарными*. Оператор `and` возвращает `True` только в том случае, когда *оба* булевых операнда равны `True`; в противном случае результат равен `False`. Чтобы увидеть, как это работает, выполните в интерактивной оболочке следующие выражения.

```
>>> True and True
True
>>> True and False
False
```

Все возможные результаты применения булевого оператора можно представить с помощью *таблицы истинности*. Для оператора and такая таблица приведена в табл. 2.2.

Таблица 2.2. Таблица истинности для оператора and

Выражение	Результат
True and True	True
True and False	False
False and True	False
False and False	False

Оператор or возвращает True, когда *любой* из булевых операндов равен True; в противном случае результат равен False

```
>>> False or True
True
>>> False or False
False
```

Таблица истинности для оператора or приведена в табл. 2.3.

Таблица 2.3. Таблица истинности для оператора or

Выражение	Результат
True or True	True
True or False	True
False or True	True
False or False	False

Оператор not

В отличие от операторов and и or, оператор not применяется только к одному булевому значению (выражению), поэтому его называют *унарным*. Он возвращает значение, противоположное значению операнда.

```
  >>> not True
  False
❶ >>> not not not not True
  True
```

Допускается использование вложенных операторов not ❶, хотя в реальных программах в этом вряд ли возникнет необходимость. Таблица истинности для оператора not приведена в табл. 2.4.

Таблица 2.4. Таблица истинности для оператора `not`

Выражение	Результат
`not True`	`False`
`not False`	`True`

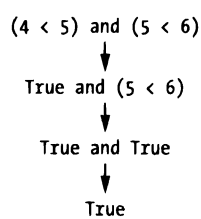
Сочетание операторов сравнения и булевых операторов

Поскольку операторы сравнения возвращают булевы значения, их можно использовать в выражениях совместно с булевыми операторами.

Вспомните, что операторы `and`, `or` и `not` называются булевыми, поскольку их операнды всегда являются булевыми значениями `True` и `False`. Результатом вычисления выражения наподобие `4 < 5` как раз и будет булево значение. Чтобы увидеть, как это работает, выполните в интерактивной оболочке следующие выражения.

```
>>> (4 < 5) and (5 < 6)
True
>>> (4 < 5) and (9 < 6)
False
>>> (1 == 2) or (2 == 2)
True
```

Сначала компьютер вычисляет левое выражение, а затем — правое. Когда оба этих результата становятся известными, вычисляется результат всего выражения в виде единственного булевого значения. Процесс вычисления выражения `(4 < 5) and (5 < 6)` показан ниже.



В выражении может быть несколько булевых операторов и операторов сравнения.

```
>>> 2 + 2 == 4 and not 2 + 2 == 5 and 2 * 2 == 2 + 2
True
```

Подобно математическим операторам, булевы операторы подчиняются приоритету операций. Сначала вычисляются все математические

операторы и операторы сравнения, затем — операторы `not`, далее — операторы `and` и в последнюю очередь — операторы `or` (порядок вычислений можно изменить с помощью скобок).

Элементы структурирования программы

Управляющие инструкции обычно начинаются с *условия*, за которым следует блок кода. Прежде чем изучать конкретные инструкции, рассмотрим их структурные элементы.

Условия

Все булевы выражения, с которыми мы ранее познакомились, могут быть *условиями*. Условие — это такое выражение, которое всегда равно `True` или `False`. Управляющая инструкция выполняет ветвление программы в зависимости от того, какое из этих двух значений принимает условие.

Блоки кода

Строки кода Python могут группироваться в *блоки*. О том, где начинается и заканчивается блок, можно судить по отступам в коде. В отношении блоков действуют следующие три правила.

- Признаком начала блока служит увеличение отступа.
- Блоки могут содержать вложенные блоки.
- Признаком конца блока служит уменьшение отступа до нулевой величины или до величины отступа внешнего блока.

Рассмотрим следующую небольшую программу.

```
name = 'Мэри'
password = 'рыба-меч'
if name == 'Мэри':
❶    print('Привет, Мэри')
    if password == 'рыба-меч':
❷        print('Предоставлен доступ.')
    else:
❸        print('Неверный пароль.')
```

Работу авторского варианта этой программы можно просмотреть на сайте `https://autbor.com/blocks/`. Первый блок кода ❶ начинается строкой `print('Привет, Мэри')` и включает все последующие строки. В этом блоке есть вложенный блок ❷, содержащий всего одну строку: `print('Предоставлен доступ.')`. Третий блок ❸ также состоит только из одной строки: `print('Неверный пароль.')`.

Выполнение программы

В программе *hello.py*, которая рассматривалась в предыдущей главе, инструкции выполнялись строго по очереди, одна за другой, от первой и до последней. *Выполнение программы* — это термин, обозначающий порядок обработки инструкций. Если напечатать исходный код программы на бумаге и перемещать палец по строкам в соответствии с логикой работы программы, то это и будет схематической иллюстрацией ее выполнения.

Однако не все программы выполняются строго сверху вниз. Если попробовать отследить пальцем ход работы программы, в которой есть управляющие инструкции, то пальцу придется прыгать туда-сюда в зависимости от условий и, возможно, пропускать целые блоки.

Управляющие инструкции

Теперь приступим к рассмотрению собственно управляющих инструкций. На блок-схеме, которая была показана на рис. 2.1, они представлены ромбами. Именно в них принимаются решения о том, как должна выполняться программа.

Инструкция `if`

Самая распространенная управляющая инструкция — `if`. Вложенный блок кода будет выполняться только в том случае, если условие равно `True` (т.е. истинно). Если же условие равно `False` (т.е. ложно), то блок выполняться не будет.

Инструкция `if` интерпретируется следующим образом: "Если условие истинно, то выполнить данный блок кода". В Python инструкция `if` содержит такие элементы:

- ключевое слово `if`;
- условие (т.е. выражение, которое равно `True` или `False`);
- двоеточие;
- блок кода с отступом, начинающийся со следующей строки (тело инструкции).

Предположим, программа проверяет, содержит ли переменная `name` имя `'Alice'`.

```
if name == 'Alice':
    print('Hi, Alice.')
```

Все управляющие инструкции заканчиваются двоеточием, за которым следует блок кода. В данном случае блок состоит из одной инструкции:

`print('Hi, Alice.')`. Блок-схема рассматриваемого кода приведена на рис. 2.2.

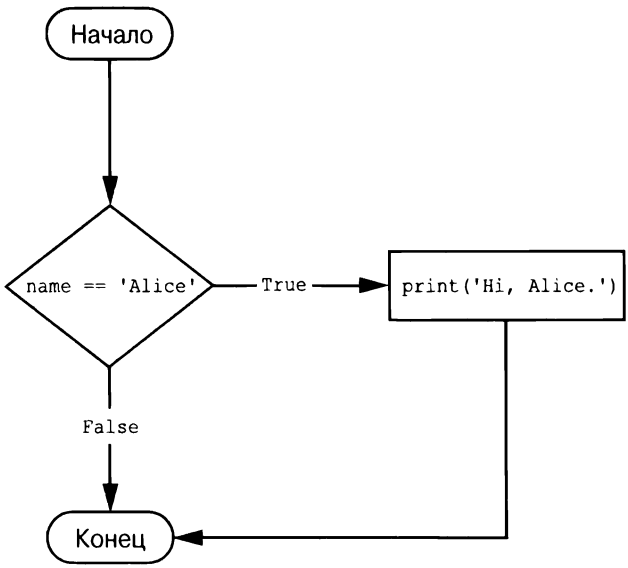


Рис. 2.2. Блок-схема рассматриваемой инструкции `if`

Инструкция `else`

Инструкция `if` может дополняться необязательной инструкцией `else` со своим блоком кода, который выполняется лишь в том случае, если условие `if` ложно. Инструкция `else` интерпретируется следующим образом: "Если условие истинно, выполнить первый блок кода. В противном случае выполнить второй блок кода". В Python инструкция `else` не имеет условия и всегда состоит из следующих элементов:

- ключевое слово `else`;
- двоеточие;
- блок кода с отступом, начинающийся со следующей строки.

Возвращаясь к предыдущему примеру, рассмотрим код, содержащий инструкцию `else`, которая выводит другое приветствие, если имя пользователя — не `'Alice'`.

```
if name == 'Alice':
    print('Hi, Alice.')
else:
    print('Hello, stranger.')
```

Блок-схема этого кода представлена на рис. 2.3.

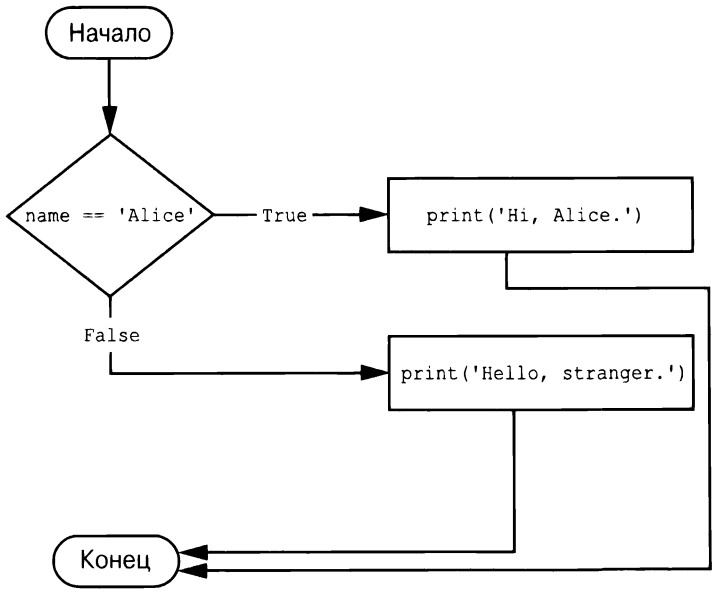


Рис. 2.3. Блок-схема рассматриваемой инструкции `else`

Инструкция `elif`

Из двух блоков кода, связанных с инструкциями `if` и `else`, всегда выполняется только один. Но иногда в программе требуется проверить *несколько* условий, для каждого из которых предусмотрен свой блок кода. Инструкция `elif` (сокращение от "else if") может стоять только после инструкции `if` или другой инструкции `elif`. Она предоставляет еще одно условие, которое проверяется лишь в том случае, если все предыдущие условия оказались ложными. В Python инструкция `elif` всегда состоит из следующих элементов:

- ключевое слово `elif`;
- условие (т.е. выражение, которое равно `True` или `False`);
- двоеточие;
- блок кода с отступом, начинающийся со следующей строки.

Добавим инструкцию `elif` в код проверки имени, чтобы увидеть, как это работает на практике.

```
if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
```

На этот раз дополнительно проверяется возраст пользователя, и если он меньше 12, то текст выводимого сообщения будет другим. Блок-схема этого кода представлена на рис. 2.4.

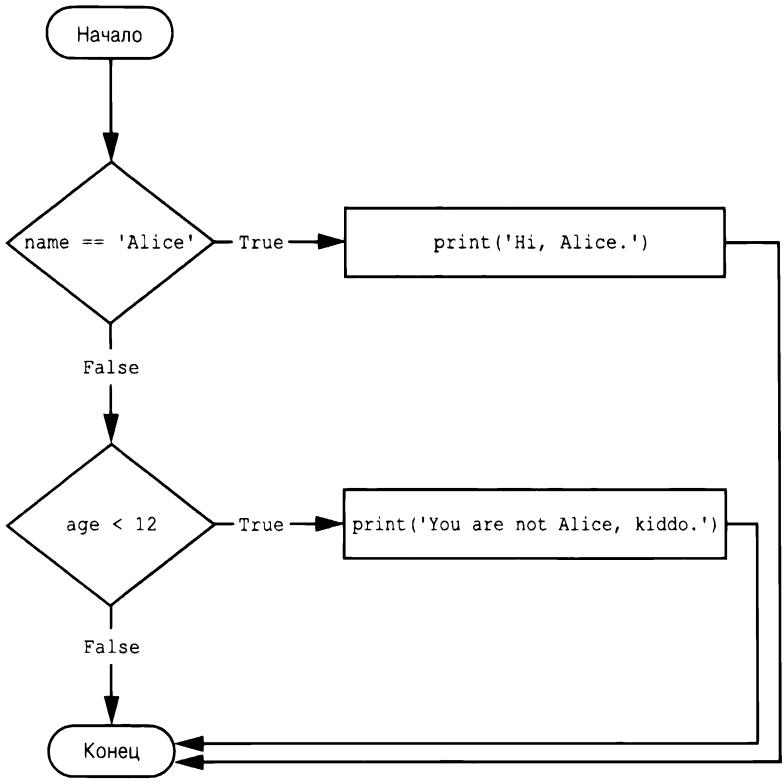


Рис. 2.4. Блок-схема рассматриваемой инструкции `elif`

Блок `elif` выполняется, если выражение `age < 12` истинно, а выражение `name == 'Alice'` ложно. Но если оба условия ложны, то пропускаются оба блока кода. Никаких гарантий того, что выполнится хотя бы один из этих двух блоков, нет. Если имеется цепочка инструкций `elif`, то будет выполнен либо один блок кода, либо ни один из них. Как только обнаруживается, что одно из условий истинно, все остальные блоки `elif` автоматически пропускаются. В качестве примера откройте в файловом редакторе новое окно, введите в нем приведенный ниже код и сохраните его в файле *vampire.py*.

```
name = 'Carol'
age = 3000
if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
```

```
    print('You are not Alice, kiddo.')
elif age > 2000:
    print('Unlike you, Alice is not an undead, immortal vampire.')
elif age > 100:
    print('You are not Alice, grannie.')
```

Выполнение этой программы можно просмотреть на сайте `https://autbor.com/vampire/`. В программу добавлены две дополнительные инструкции `elif`, обеспечивающие вывод разных сообщений в зависимости от возраста (`age`) пользователя. Блок-схема этого кода показана на рис. 2.5.

Учитывайте, что порядок расположения инструкций `elif` важен. Сейчас мы намеренно внесем в код ошибку, переставив инструкции местами. Как вам уже известно, если одно из условий оказывается истинным, все остальные блоки `elif` автоматически пропускаются, поэтому перестановка условий может создавать проблемы. Измените код, как показано ниже, и сохраните его в файле *vampire2.py*.

```
name = 'Carol'
age = 3000
if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
❶ elif age > 100:
    print('You are not Alice, grannie.')
elif age > 2000:
    print('Unlike you, Alice is not an undead, immortal vampire.')
```

Выполнение этой программы можно просмотреть на сайте `https://autbor.com/vampire2/`. Изначально переменная `age` равна `3000`. Вы ожидаете, что программа выведет на экран строку `'Unlike you, Alice is not an undead, immortal vampire.'`. Но поскольку условие `age > 100` равно `True` (ведь `3000` больше, чем `100`) в строке ❶, на экран будет выведена строка `'You are not Alice, grannie.'`, а остальные инструкции `elif` будут автоматически пропущены. Вспомните, что выполняется максимум один блок `elif`, поэтому порядок их следования важен!

Блок-схема предыдущей программы представлена на рис. 2.6. Обратите внимание на то, что ромбы для условий `age > 100` и `age > 2000` переставлены местами.

При необходимости за последней инструкцией `elif` можно поместить инструкцию `else`. В этом случае гарантируется, что будет выполнен хотя бы один (и только один) блок кода. Если условия во всех инструкциях `if` и `elif` окажутся ложными, то выполнится блок кода `else`. Давайте переделаем программу для распознавания имени `'Alice'` таким образом, чтобы в ней использовались инструкции `if`, `elif` и `else`.

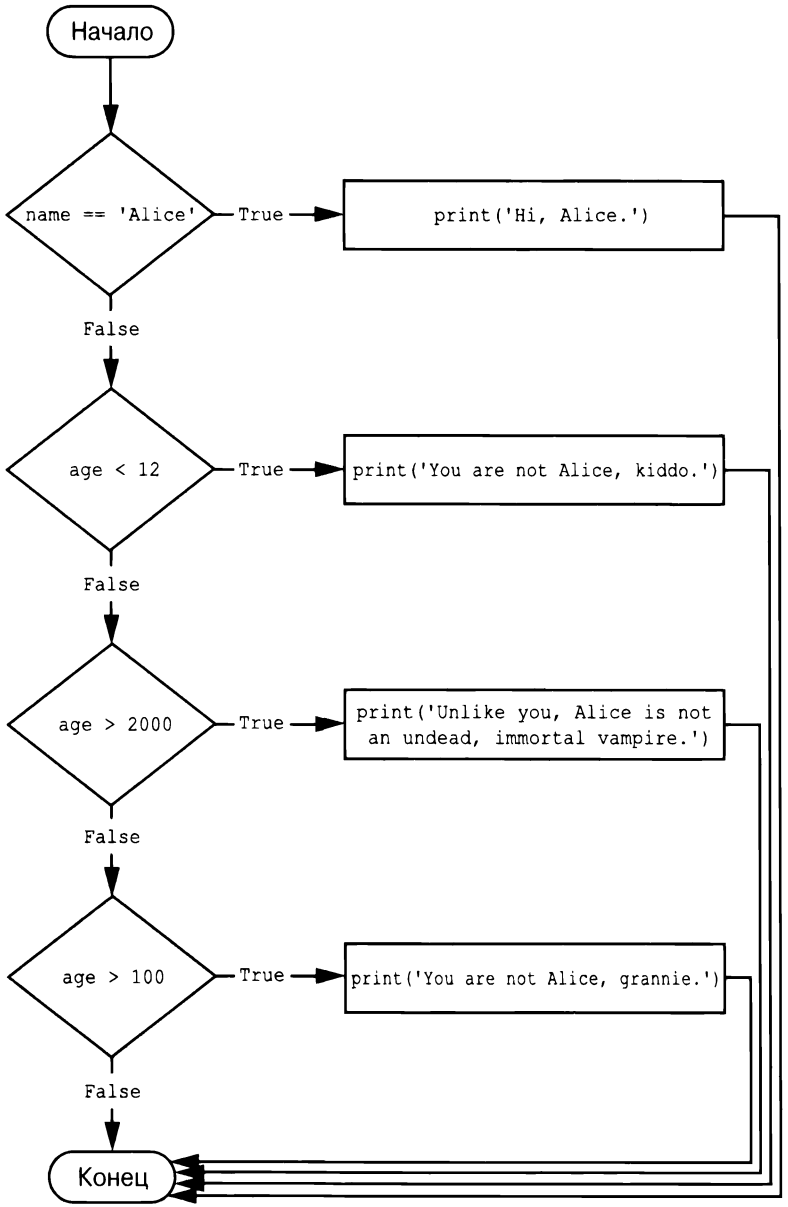


Рис. 2.5. Блок-схема кода с несколькими инструкциями `elif` *в программе vampire.py*

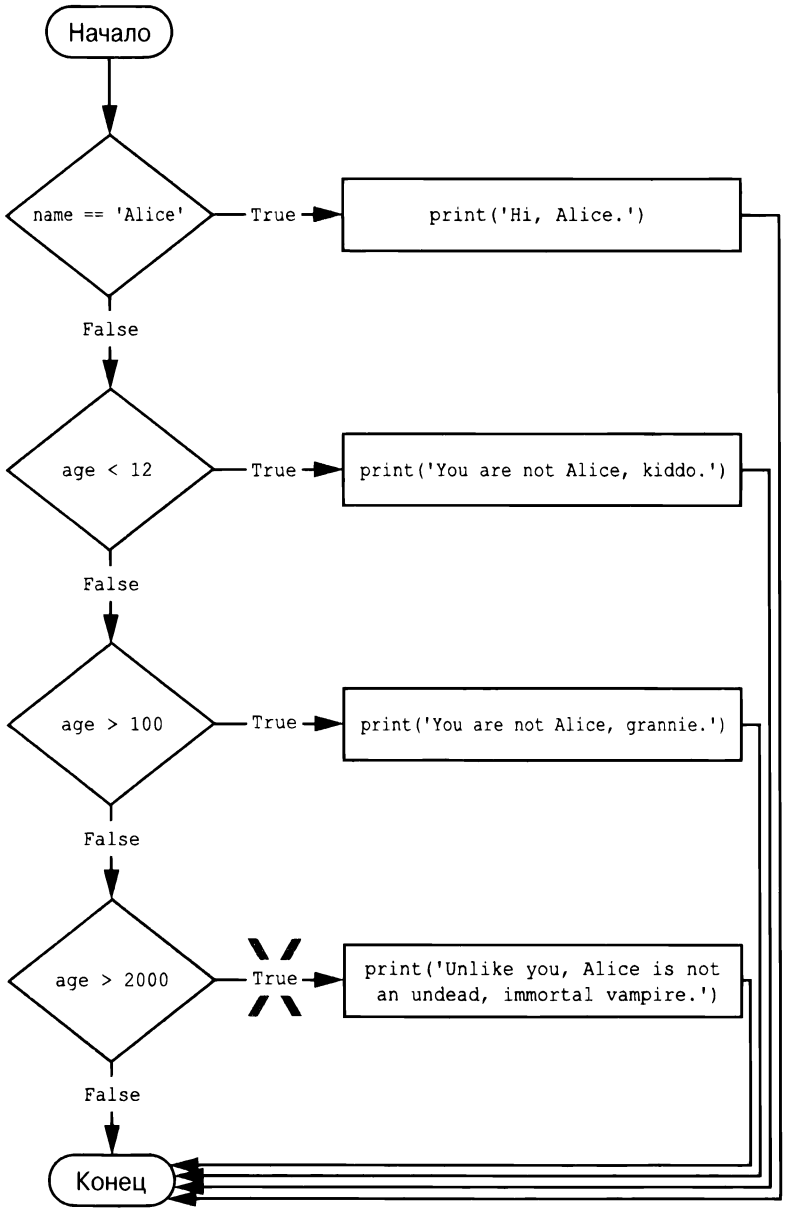


Рис. 2.6. Блок-схема программы vampire2.py. Выполнение программы вдоль перечеркнутого пути логически невозможно, поскольку если значение `age` *больше* `2000`*, то оно заведомо больше* `100`

```
name = 'Carol'
age = 3000
if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
else:
    print('You are neither Alice nor a little kid.')
```

Результат выполнения программы можно увидеть на сайте `https://autbor.com/littlekid/`. На рис. 2.7 показана блок-схема нового кода, который мы сохраним в файле *littleKid.py*.

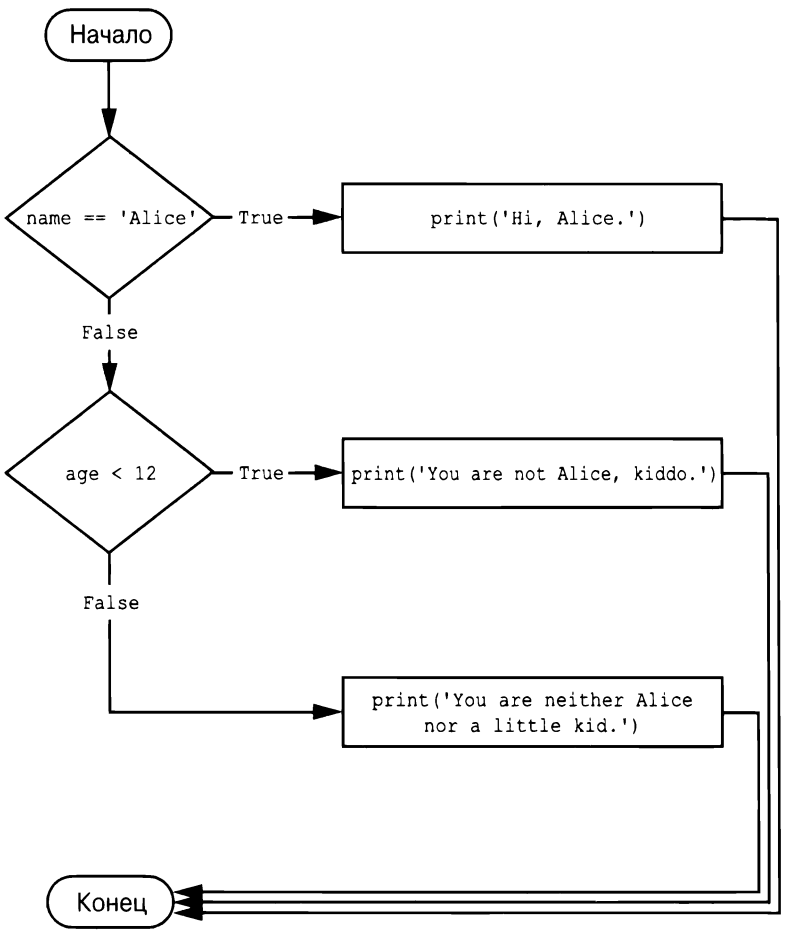


Рис. 2.7. Блок-схема программы littleKid.py

Смысл подобной конструкции можно передать так: "Если первое условие истинно, выполнить первый блок кода. Иначе, если второе условие

истинно, выполнить второй блок кода. В противном случае выполнить последний блок кода". Не забывайте об этих правилах при совместном использовании инструкций `if`, `elif` и `else`, чтобы избежать ошибок, подобных той, которая была проиллюстрирована на рис. 2.6. Во-первых, инструкция `if` должна быть только одна. Любые инструкции `elif`, которые могут вам понадобиться, должны следовать за инструкцией `if`. Во-вторых, если нужно, чтобы выполнился хотя бы один блок кода, используйте завершающую инструкцию `else`.

Цикл `while`

Инструкция `while` позволяет организовать многократное выполнение блока кода. Тело цикла `while` выполняется до тех пор, пока истинно заданное условие. В Python инструкция `while` всегда состоит из следующих элементов:

- ключевое слово `while`;
- условие (т.е. выражение, которое равно `True` или `False`);
- двоеточие;
- блок кода с отступом, начинающийся со следующей строки.

Нетрудно заметить, что инструкция `while` структурно напоминает инструкцию `if`, однако ведет себя иначе. По достижении конца блока `if` управление передается следующей инструкции. В случае же инструкции `while` по достижении конца блока управление возвращается в начало цикла.

Сравним работу инструкции `if` и цикла `while`, имеющих одинаковое условие и тело. Вот как выглядит код с инструкцией `if`.

```
spam = 0
if spam < 5:
    print('Hello, world.')
    spam = spam + 1
```

А вот код с инструкцией `while`.

```
spam = 0
while spam < 5:
    print('Hello, world.')
    spam = spam + 1
```

Как видите, у них много общего. Обе инструкции проверяют значение переменной `spam`, и если оно меньше пяти, то выводится сообщение. Но если выполнить оба фрагмента, то результаты окажутся разными. В случае инструкции `if` выводится одиночное сообщение `"Hello, world."`, тогда

как в случае инструкции while это сообщение выводится пять раз! Чтобы разобраться в том, почему так происходит, обратимся к соответствующим блок-схемам (рис. 2.8 и 2.9).

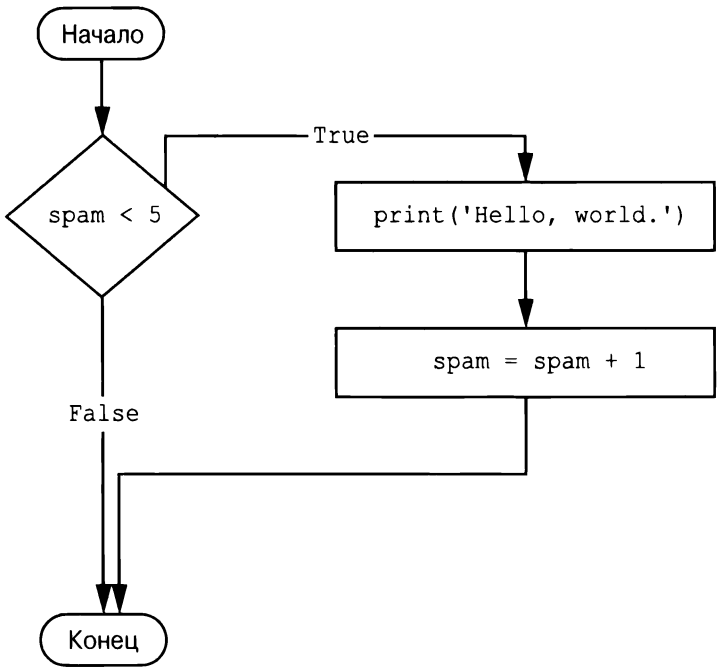


Рис. 2.8. Блок-схема кода с инструкцией if

Инструкция if проверяет условие и выводит текст "Hello, world.", если оно истинно. Инструкция while выводит этот текст пять раз. После пятого раза цикл прекращается, поскольку целочисленное значение, хранящееся в переменной spam и увеличивающееся на единицу на каждой итерации цикла, станет равно 5, а значит, условие spam < 5 перестанет быть истинным.

В цикле while условие всегда проверяется перед началом каждой *итерации* (т.е. перед заходом в тело цикла). Если условие равно True, то блок кода выполняется, после чего условие проверяется вновь. Как только обнаруживается, что условие равно False, цикл while завершается.

Назойливый цикл while

Ниже приведен пример простой программы, которая без устали просит вас ввести свое имя, хотя в действительности она ожидает ввода не вашего имени, а строки 'your name'. Выберите команду File⇨New (Файл⇨Создать), чтобы открыть новое окно файлового редактора, введите приведенный ниже код и сохраните его в файле *yourName.py*.

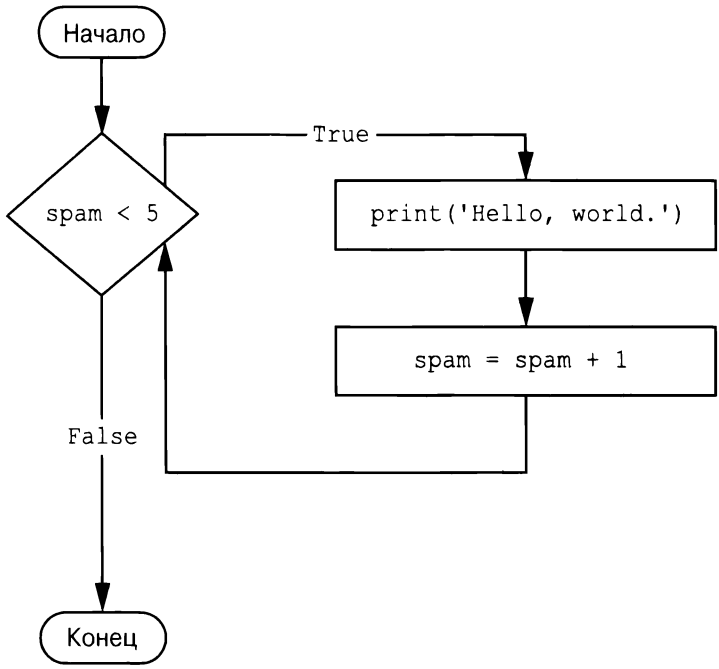


Рис. 2.9. Блок-схема кода с инструкцией `while`

```
❶ name = ''
❷ while name != 'your name':
      print('Please type your name.')
❸     name = input()
❹ print('Thank you!')
```

Сначала программа записывает в переменную `name` ❶ пустую строку. Это делается для того, чтобы первая проверка условия `name != 'your name'` дала результат `True` и программа приступила к выполнению цикла `while` ❷.

В теле цикла программа запрашивает имя пользователя и присваивает введенное значение переменной `name` ❸. Поскольку это последняя строка блока, выполнение возобновляется с начала цикла `while`, где вновь проверяется условие. Если значение `name` *не равно* строке `'your name'`, то условие оказывается истинным, в результате чего вновь начинает выполняться тело цикла.

Но как только пользователь введет текст `'your name'`, условие примет вид `'your name' != 'your name'`, что дает значение `False`. Поскольку теперь условие ложно, выполнение цикла прекращается, и управление передается инструкции, следующей за циклом ❹. Блок-схема программы *yourName.py* приведена на рис. 2.10.

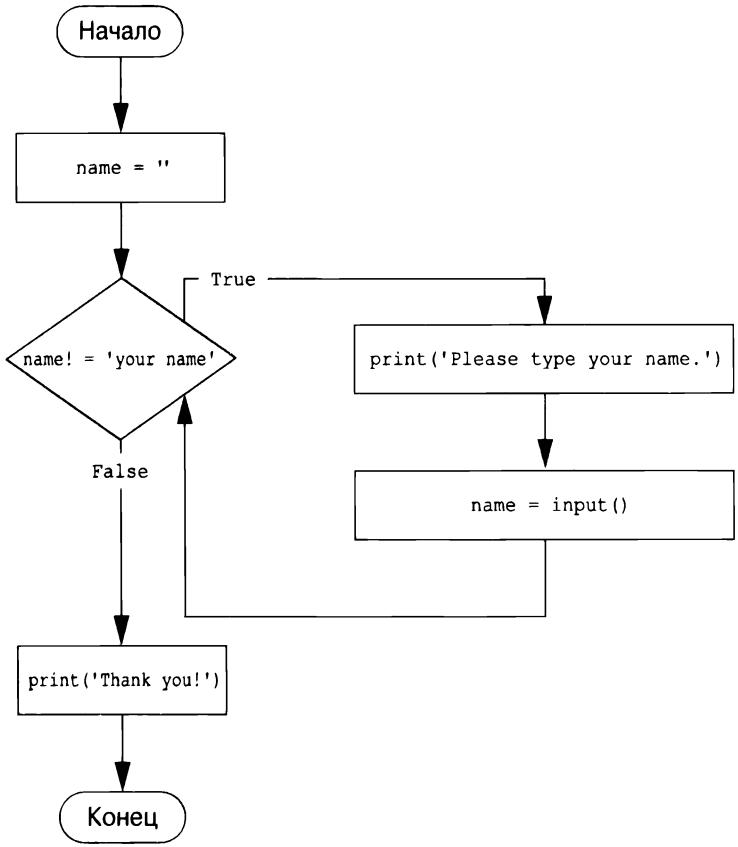


Рис. 2.10. Блок-схема программы yourName.py

Проверим, как работает программа *yourName.py*. Нажмите клавишу <F5>, чтобы запустить программу, и несколько раз введите текст, отличный от `'your name'`, прежде чем предоставить программе то, что ей нужно.

```
Please type your name.
Al
Please type your name.
Albert
Please type your name.
%#@#%*(^&!!!
Please type your name.
your name
Thank you!
```

Если вы так и не введете текст `'your name'`, то условие цикла `while` никогда не окажется ложным, и программа будет бесконечно повторять свой запрос. В данном случае функция `input()` дает пользователю возможность ввести нужную строку, которая позволит программе выйти из цикла. Но

можно написать программу так, что условие никогда не изменится, и это станет проблемой. Рассмотрим, как разорвать цикл `while`.

Инструкция `break`

Существует простой способ заставить программу досрочно выйти из цикла `while`. Если в блоке кода встречается инструкция `break`, то выполнение цикла немедленно прекращается.

Довольно просто, не правда ли? Ниже приведен пример программы, которая делает то же самое, что и предыдущая, но использует инструкцию `break` для выхода из цикла. Введите следующий код и сохраните его в файле *yourName2.py*.

```
❶ while True:
      print('Please type your name.')
❷     name = input()
❸     if name == 'your name':
❹         break
❺ print('Thank you!')
```

Выполнение этой программы можно просмотреть на сайте `https://autbor.com/yourname2/`. В первой строке ❶ создается *бесконечный цикл*, т.е. цикл, условие которого всегда истинно (`True`). Программа выйдет из цикла, только если ей встретится инструкция `break`. (Бесконечный цикл, выйти из которого невозможно, — распространенная программная ошибка.)

Как и прежде, программа запрашивает ввод пользователем текста `'your name'` ❷. Но теперь в цикле имеется инструкция `if` ❸, которая проверяет, равно ли значение переменной `name` строке `'your name'`. В случае истинности этого условия выполняется инструкция `break` ❹, и управление передается инструкции `print('Thank you')` ❺. В противном случае блок `if`, содержащий инструкцию `break`, пропускается, программа переходит в конец цикла `while` и сразу же возвращается в его начало для проверки условия. Поскольку условие — это просто булево значение `True`, программа снова входит в цикл и повторно запрашивает ввод текста `'your name'`. Блок-схема программы приведена на рис. 2.11.

Запустите программу *yourNumber2.py* и введите тот же текст, который вводили для программы *yourNumber.py*. Новая версия программы должна реагировать на ваш ввод так же, как и исходная.

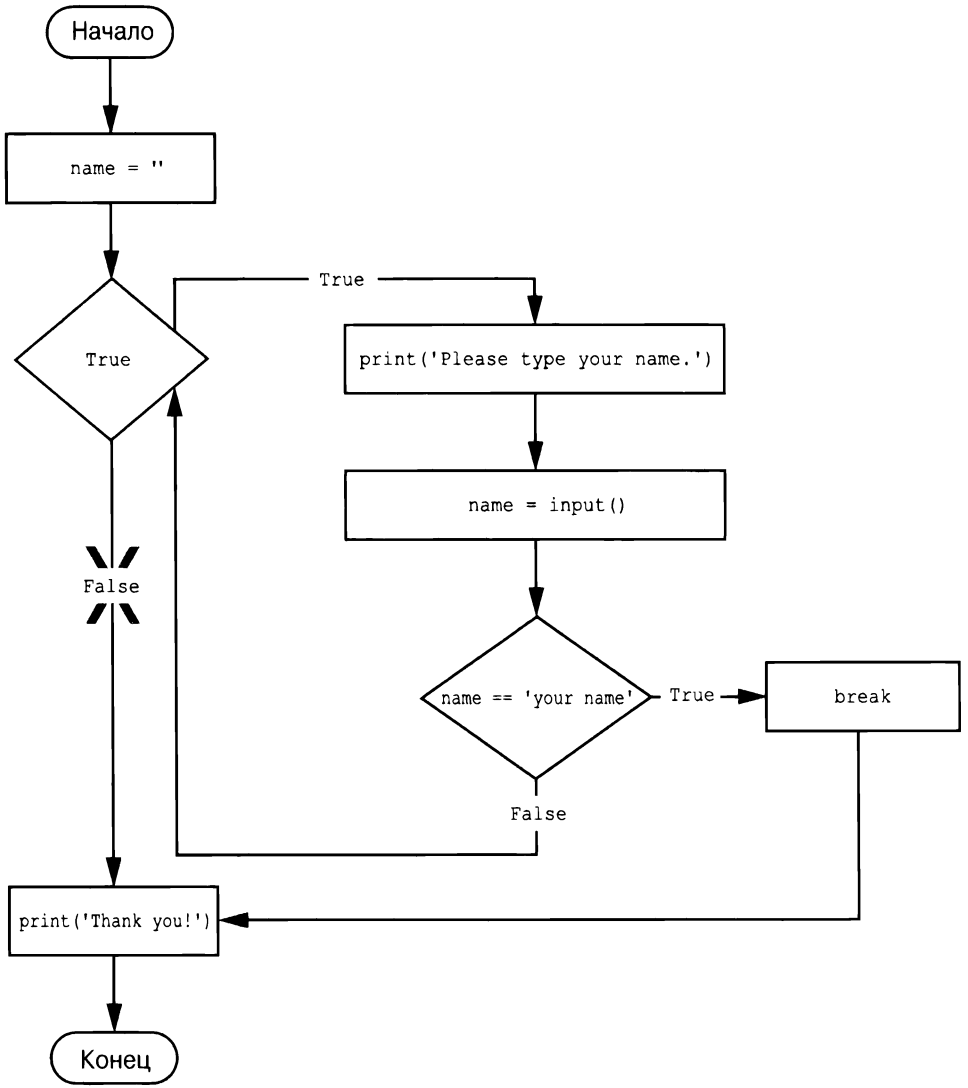


Рис. 2.11. Блок-схема программы yourName2.py с бесконечным циклом. Обратите внимание на то, что перечеркнутый путь никогда не может быть пройден, поскольку условие цикла всегда истинно

Инструкция `continue`

Подобно инструкции `break`, инструкция `continue` тоже применяется внутри цикла. Когда программа встречает такую инструкцию, управление немедленно передается в начало цикла, где заново проверяется условие. (То же самое происходит при достижении программой конца цикла.)

Попали в ловушку бесконечного цикла?

Если вы обнаружили, что программа застряла в бесконечном цикле, нажмите комбинацию клавиш <Ctrl+C> или выберите команду Shell⇨Restart Shell в меню IDLE. В результате будет сгенерирована ошибка `KeyboardInterrupt`, что приведет к немедленному прекращению работы программы. Проверьте, как это работает на практике, создав простой бесконечный цикл в файловом редакторе и сохранив код в файле *infiniteloop.py*.

```
while True:
    print('Здравствуй, мир!')
```

Когда вы запустите эту программу, она начнет безостановочно выводить на экран приветствие `'Здравствуй, мир!'`, поскольку условие инструкции `while` всегда остается истинным. Комбинация клавиш <Ctrl+C> удобна еще и тем, что позволяет немедленно прервать программу независимо от того, находится она в бесконечном цикле или нет.

Рассмотрим программу, которая запрашивает ввод имени пользователя и пароля. Введите следующий код в новом окне файлового редактора и сохраните его в файле *swordfish.py*.

```
  while True:
      print('Who are you?')
      name = input()
❶     if name != 'Joe':
❷         continue
      print('Hello, Joe. What is the password? (It is a fish.)')
❸     password = input()
      if password == 'swordfish':
❹         break
❺ print('Access granted.')
```

Если пользователь вводит любое другое имя, кроме `'Joe'` ❶, инструкция `continue` ❷ заставляет программу вернуться в начало цикла. После проверки условия программа всегда входит в тело цикла, поскольку условие всегда истинно (`True`). В случае прохождения первой инструкции `if` программа запрашивает пароль ❸. Если пользователь вводит пароль `'swordfish'`, то выполняется инструкция `break` ❹, программа выходит из цикла `while` и выводит на экран текст `'Access granted'` ❺. В противном случае управление передается в конец цикла `while` и сразу же возвращается в его начало. Блок-схема программы представлена на рис. 2.12.

Запустите программу и введите какой-нибудь текст. Программа не запросит ввод пароля до тех пор, пока вы не подтвердите, что ваше имя — `'Joe'`. После ввода правильного пароля программа завершится.

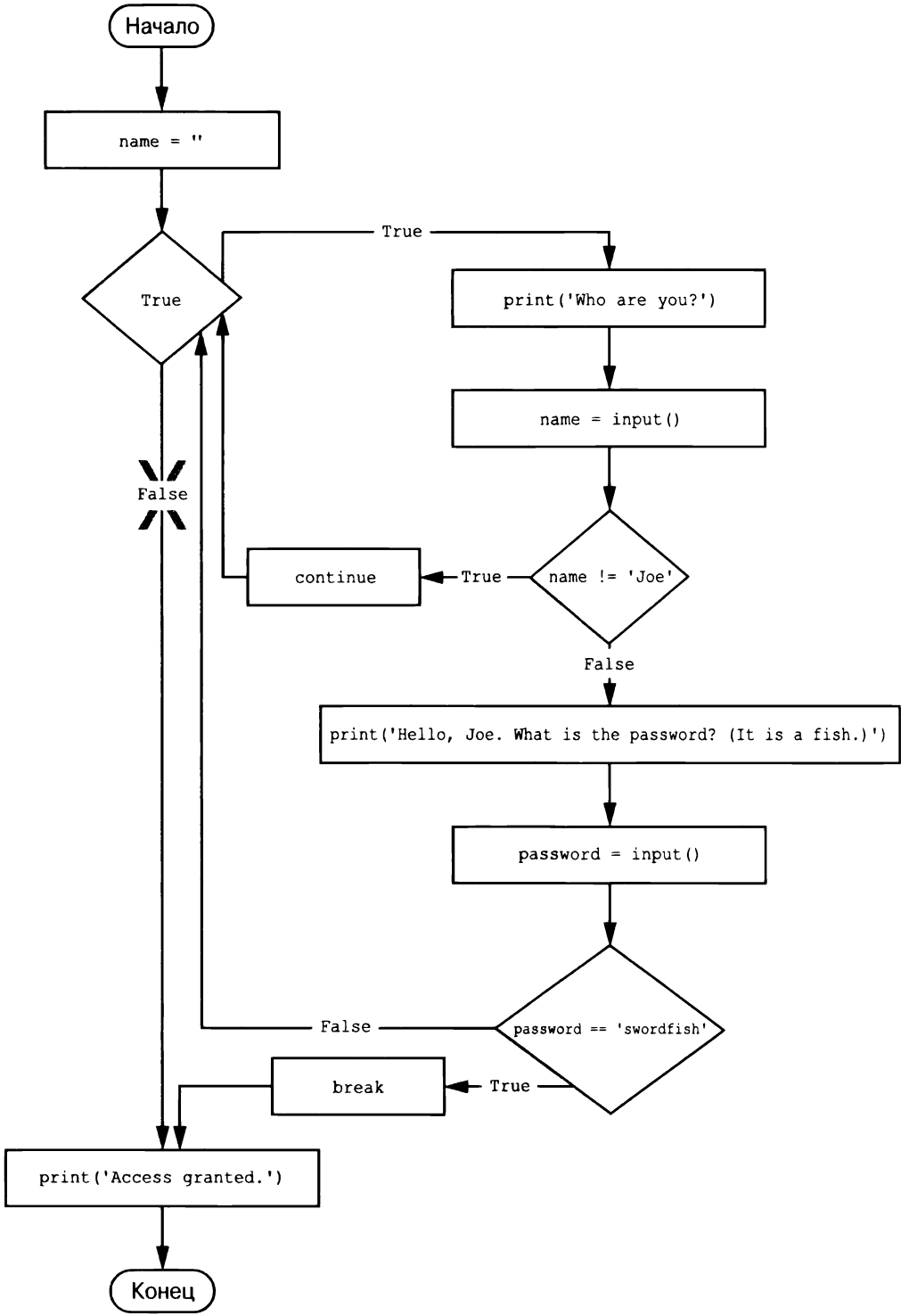


Рис. 2.12. Блок-схема программы swordfish.py. Перечеркнутый путь никогда не может быть пройден, поскольку условие цикла всегда истинно

```
Who are you?
I'm fine, thanks. Who are you?
Who are you?
Joe
Hello, Joe. What is the password? (It is a fish.)
Mary
Who are you?
Joe
Hello, Joe. What is the password? (It is a fish.)
swordfish
Access granted.
```

Истинные и ложные значения

При проверке условий некоторые значения считаются эквивалентами `True` и `False`. В частности, значения `0`, `0.0` и `''` (пустая строка) трактуются как `False`, а все остальные — как `True`. Рассмотрим следующую программу.

```
  name = ''
❶ while not name:
      print('Введите свое имя:')
      name = input()
  print('Сколько гостей вы ждете?')
  numOfGuests = int(input())
❷ if numOfGuests:
❸     print('Убедитесь, что для гостей хватит места.')
  print('Конец работы')
```

Выполнение этой программы можно просмотреть на сайте `https://autbor.com/howmanygests/`. Если пользователь вводит в качестве имени пустую строку, то условие цикла `while` становится равно `True` ❶, и программа продолжает запрашивать имя. Если значение переменной `numOfGuests` не равно `0` ❷, то условие инструкции `if` оказывается истинным, и программа выводит напоминание ❸.

Вместо `not name` можно было бы ввести `not name != ''`, а вместо `numOfGuests` — `numOfGuests != 0`, но это сделало бы код менее компактным и понятным.

Цикл `for` и функция `range()`

Цикл `while` выполняется до тех пор, пока условие остается истинным. Но что если необходимо выполнить блок кода строго определенное количество раз? Это можно сделать с помощью цикла `for` и функции `range()`. Синтаксис выглядит примерно так:

```
for i in range(5):
```

Цикл `for` включает следующие элементы:

- ключевое слово `for`;
- имя переменной (счетчик цикла);
- ключевое слово `in`;
- вызов функции `range()`, которой можно передать до трех целочисленных аргументов;
- двоеточие;
- блок кода с отступом, начинающийся со следующей строки.

Чтобы увидеть на практике, как работает цикл `for`, создадим новую программу и сохраним ее в файле *fiveTimes.py*.

```
print('My name is')
for i in range(5):
    print('Jimmy Five Times (' + str(i) + ')')
```

Выполнение этой программы можно просмотреть на сайте `https://autbor.com/fivetimesfor/`. Тело цикла `for` выполняется пять раз. На первой итерации значение переменной `i` устанавливается равным нулю. Функция `print()` в теле цикла выводит текст `'Jimmy Five Times (0)'`. Когда выполнение блока кода завершается, управление передается в начало цикла, где инструкция `for` увеличивает значение переменной `i` на единицу. Функция `range(5)` обеспечивает пятикратное выполнение тела цикла, последовательно устанавливая счетчик цикла равным `0`, `1`, `2`, `3` и `4`. Значение, указанное в качестве аргумента функции, в этот ряд не входит. Блок-схема программы *fiveTimes.py* показана на рис. 2.13.

Когда вы запустите эту программу, она должна будет пять раз вывести строку `'Jimmy Five Times'` с указанием текущего значения счетчика `i`.

```
My name is
Jimmy Five Times (0)
Jimmy Five Times (1)
Jimmy Five Times (2)
Jimmy Five Times (3)
Jimmy Five Times (4)
```

Примечание

В циклах `for` тоже допускается использование инструкций `break` и `continue`. Инструкция `continue` возвращает управление в начало цикла для выполнения следующей итерации, при этом счетчик цикла увеличивается, как если бы программа достигла конца цикла и вернулась к его началу обычным способом.

Инструкции `continue` и `break` поддерживаются только в циклах `while` и `for`. Если попытаться использовать их где-то еще, Python выдаст сообщение об ошибке.

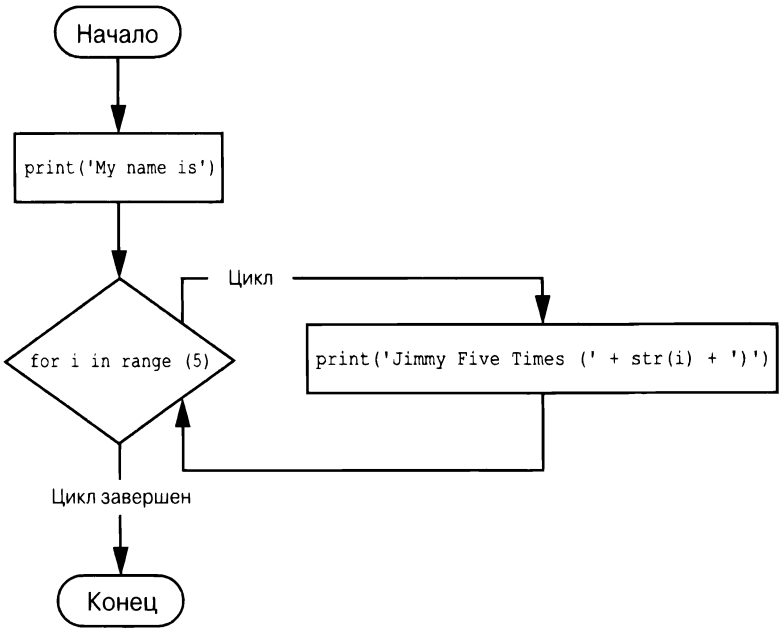


Рис. 2.13. Блок-схема программы fiveTimes.py

Рассмотрим еще один пример, основанный на истории, которая произошла с будущим математиком Карлом Фридрихом Гауссом, когда он еще учился в школе. Однажды учитель дал классу следующее задание: найти сумму всех чисел от 0 до 100. Юный Гаусс сумел быстро сообразить, как найти нужную сумму, и решил задачу буквально за несколько секунд. Мы же напишем программу с циклом `for`, которая проделает соответствующие вычисления за нас.

```
❶ total = 0
❷ for num in range(101):
❸     total = total + num
❹ print(total)
```

Правильный ответ — 5050. Сразу после запуска программы значение переменной `total` устанавливается равным `0` ❶. Затем в цикле `for` ❷ 100 раз выполняется инструкция `total = total + num` ❸. По завершении 100 итераций цикла в переменной `total` будет сохранена сумма всех целых чисел от 0 до 100. После этого значение `total` выводится на экран ❹. Даже на самых медленных компьютерах выполнение этой программы займет менее секунды.

(Юный Гаусс догадался, что всего имеется 50 пар чисел, сумма которых дает 101: 1 + 100, 2 + 99, 3 + 98, ..., 50 + 51. Поскольку 50 · 101 = 5050, то сумма всех чисел от 1 до 100 равна 5050. Сообразительный ребенок!)

Эквивалентный цикл while

Все то, что делает цикл for, можно сделать с помощью цикла while, просто циклы for более компактны. Перепишем код программы *fiveTimes.py*, заменив цикл for эквивалентным циклом while.

```
print('My name is')
i = 0
while i < 5:
    print('Jimmy Five Times (' + str(i) + ')')
    i = i + 1
```

Выполнение этой программы можно просмотреть на сайте https://autbor.com/fivetimeswhile/. Запустив программу, вы увидите, что она выдает те же результаты, что и версия с циклом for.

Аргументы начала, конца и шага функции range()

Некоторым функциям можно передавать список аргументов, разделенных запятыми, и функция range() — одна из них. Это позволяет настраивать диапазон счетчика цикла и начинать отсчет не с нуля.

```
for i in range(12, 16):
    print(i)
```

Первый аргумент определяет, с какого значения начинается отсчет. Второй аргумент задает верхнюю границу счетчика, причем само это значение в диапазон не включается.

```
12
13
14
15
```

Третий аргумент функции range() задает *шаг*, т.е. приращение счетчика в конце каждой итерации цикла.

```
for i in range(0, 10, 2):
    print(i)
```

Вызов range(0, 10, 2) обеспечивает изменение счетчика цикла от 0 до 8 с шагом 2.

```
0
2
4
6
8
```

Функция `range()` очень гибкая в отношении формирования последовательностей целых чисел для циклов `for`. Например, можно задать отрицательный шаг, сделав так, чтобы отсчет шел от *бо́льших* значений к меньшим.

```
for i in range(5, -1, -1):
    print(i)
```

Результат выполнения этого цикла будет таким.

```
5
4
3
2
1
0
```

Импорт модулей

Любой программе на языке Python доступен базовый набор функций, называемых *встроенными*. В их число входят такие функции, как `print()`, `input()` и `len()`, с которыми вы уже успели познакомиться. Кроме того, в состав Python входит набор модулей, называемый *стандартной библиотекой*. Каждый модуль — это программа, содержащая группу родственных функций, которые можно использовать в других программах. Например, модуль `math` включает математические функции, модуль `random` — функции для работы со случайными числами и т.п.

Прежде чем использовать функции, входящие в состав модуля, его необходимо импортировать с помощью инструкции `import`, которая состоит из следующих элементов:

- ключевое слово `import`;
- имя модуля;
- необязательные дополнительные имена модулей, разделенные запятыми.

Как только модуль будет импортирован, станут доступными все входящие в него функции. Проверим это на примере модуля `random`, в котором имеется функция `randint()`.

Введите в файловом редакторе приведенный ниже код и сохраните его в файле *printRandom.py*.

```
import random
for i in range(5):
    print(random.randint(1, 10))
```

Выполнив программу, вы должны получить примерно такие результаты.

```
4
1
8
4
1
```

Выполнение этой программы можно просмотреть на сайте `https://autbor.com/printrandom/`. Функция `random.randint()` возвращает случайное число, лежащее в диапазоне между двумя целочисленными значениями, которые передаются функции в качестве аргументов. Поскольку функция `randint()` находится в модуле `random`, его имя должно указываться в виде префикса (через точку) перед именем функции. Благодаря наличию префикса Python будет знать, что данную функцию следует искать в модуле `random`.

Вот пример инструкции `import`, которая импортирует четыре различных модуля:

```
import random, sys, os, math
```

Теперь мы сможем использовать любую из функций, находящихся в этих четырех модулях.

Инструкция `from import`

Альтернативная форма инструкции `import` состоит из ключевого слова `from`, за которым следуют имя модуля, ключевое слово `import` и символ "звездочка", например `from random import *`.

При использовании такой формы импорта добавлять префикс `random.` к имени функции, вызываемой из модуля `random`, не требуется. В то же время использование полного имени функции делает код более понятным, поэтому лучше все же использовать обычную форму инструкции `import`.

Не переписывайте имена модулей

При сохранении сценариев Python старайтесь не давать им имена, используемые одним из модулей Python, например *random.py*, *sys.py*, *os.py* или *math.py*. Если вы случайно назовете одну из своих программ, допустим, *random.py*, а затем используете инструкцию `import random` в другой программе, то будет импортирован ваш файл *random.py*, а не модуль `random`. Это может привести к появлению сообщений об ошибках вида `AttributeError: module 'random' has no attribute 'randint'`, поскольку файл *random.py* не будет содержать функций, входящих в состав реального модуля `random`. Не используйте также имена любых встроенных функций Python, таких как `print()` или `input()`.

Подобные ошибки встречаются нечасто, но их сложно выявлять. По мере приобретения опыта в программировании вы будете знакомиться с модулями и функциями стандартной библиотеки Python и реже сталкиваться с этими проблемами.

Досрочное завершение программы с помощью функции `sys.exit()`

Нам осталось рассмотреть, как досрочно завершить работу программы. Программа всегда завершается автоматически после выполнения финальной инструкции. Однако существует возможность принудительно прекратить работу программы с помощью функции `sys.exit()`. Поскольку эта функция находится в модуле `sys`, необходимо импортировать его, прежде чем вызывать функцию.

Откройте новое окно файлового редактора, введите в него приведенный ниже код и сохраните его в файле *exitExample.py*.

```
import sys

while True:
    print('Введите "exit" для выхода.')
    response = input()
    if response == 'exit':
        sys.exit()
    print('Вы ввели ' + response + '.')
```

Запустите эту программу в IDLE. В программе выполняется бесконечный цикл, в котором отсутствует инструкция `break`. Единственная возможность завершить работу программы — достичь вызова функции `sys.exit()`. Поскольку значение переменной `response` устанавливается функцией `input()`, для завершения программы пользователь должен ввести слово `'exit'`. Если переменная `response` оказывается равна `'exit'`, выполнение программы прекращается.

Короткая программа: угадай число

Предыдущие примеры были достаточно простыми. Пора создать что-то посложнее. В этом разделе мы рассмотрим игру "угадай число". После запуска программы результат будет выглядеть примерно так.

```
Я загадал число от 1 до 20.
Угадайте число.
10
Я загадал большее число.
Угадайте число.
15
Я загадал большее число.
Угадайте число.
17
Я загадал меньшее число.
Угадайте число.
16
Отлично! Количество попыток: 4.
```

Введите в окне файлового редактора следующий код и сохраните его под именем *guessTheNumber.py*.

```
# Игра в угадывание чисел
import random

secretNumber = random.randint(1, 20)
print('Я загадал число от 1 до 20.')

# Игроку дается 6 попыток
for guessesTaken in range(1, 7):
    print('Угадайте число.')
    guess = int(input())

    if guess < secretNumber:
        print('Я загадал большее число.')
    elif guess > secretNumber:
        print('Я загадал меньшее число.')
    else:
        break    # Число угадано!

if guess == secretNumber:
    print('Отлично! Количество попыток: ' + str(guessesTaken) + '.')
else:
    print('Вам не повезло. Я загадал число ' + str(secretNumber))
```

Работу авторского варианта этой программы можно просмотреть на сайте `https://autbor.com/guessthenumber/`. Рассмотрим код построчно, с самого сначала.

```
# Игра в угадывание чисел
import random

secretNumber = random.randint(1, 20)
```

Комментарий в верхней части кода объясняет назначение программы. Далее программа импортирует модуль `random`, в результате чего появляется возможность использовать функцию `random.randint()` для генерации числа, которое должен угадать пользователь. Возвращаемое значение (случайное целое число в диапазоне от 1 до 20) сохраняется в переменной `secretNumber`.

```
print('Я загадал число от 1 до 20.')

# Игроку дается 6 попыток
for guessesTaken in range(1, 7):
    print('Угадайте число.')
    guess = int(input())
```

Программа сообщает игроку о том, что было загадано некое число, и дает ему шесть попыток для угадывания. Соответствующий код находится в цикле `for`. Этот код будет повторяться не более шести раз. Первое, что происходит в цикле, — ввод игроком предполагаемого числа. Поскольку функция `input()` возвращает строку, возвращаемое значение передается функции `int()`, которая преобразует строку в целочисленное значение. Это значение сохраняется в переменной `guess`.

```
    if guess < secretNumber:
        print('Я загадал большее число.')
    elif guess > secretNumber:
        print('Я загадал меньшее число.')
```

В следующих строках проверяется, является ли введенное число меньшим или большим загаданного числа. В зависимости от результатов проверки на экране отображается соответствующая подсказка.

```
    else:
        break   # Число угадано!
```

Если введенное число не больше и не меньше загаданного числа, значит, оно равно ему! В этом случае программа выходит из цикла `for`.

```
if guess == secretNumber:
    print('Отлично! Количество попыток: ' + str(guessesTaken) + '.')
```

```
else:
    print('Вам не повезло. Я загадал число ' + str(secretNumber))
```

В инструкции `if/else`, которая стоит после цикла `for`, проверяется, угадал ли игрок число, после чего на экран выводится соответствующее сообщение. В обоих случаях программа отображает значение целочисленной переменной (`guessesTaken` или `secretNumber`). Поскольку это значение нужно конкатенировать со строкой, оно передается в функцию `str()`, которая возвращает строковое представление числа.

Короткая программа: камень, ножницы, бумага

В этом разделе мы напишем игру "камень, ножницы, бумага", применив концепции, изученные в данной главе. Результат работы программы будет выглядеть примерно так.

```
КАМЕНЬ, НОЖНИЦЫ, БУМАГА
0 побед, 0 поражений, 0 ничьих
Выберите ход: (к)амень, (н)ожницы, (б)умага или (в)ыход
б
БУМАГА и ...
БУМАГА
Ничья!
0 побед, 0 поражений, 1 ничьих
Выберите ход: (к)амень, (н)ожницы, (б)умага или (в)ыход
н
НОЖНИЦЫ и ...
БУМАГА
Вы победили!
1 побед, 0 поражений, 1 ничьих
Выберите ход: (к)амень, (н)ожницы, (б)умага или (в)ыход
в
```

Введите в окне файлового редактора следующий код и сохраните его под именем *rpsGame.py*.

```
import random, sys

print('КАМЕНЬ, НОЖНИЦЫ, БУМАГА')

# В этих переменных накапливается количество
# побед, поражений и ничьих
wins = 0
losses = 0
ties = 0

while True:    # главный цикл игры
    print('%s побед, %s поражений, %s ничьих' % (wins, losses, ties))
```

```
while True:    # цикл выбора хода
    print('Выберите ход: (к)амень, (н)ожницы, '(б)умага или ' + \
          '(в)ыход')
    playerMove = input()
    if playerMove == 'в':
        sys.exit()    # выход из программы
    if playerMove == 'к' or playerMove == 'н' \
                         or playerMove == 'б':
        break    # выход из цикла выбора хода
    print('Введите "к", "н", "б" или "в".')

# Отображение выбора пользователя
if playerMove == 'к':
    print('КАМЕНЬ и ...')
elif playerMove == 'н':
    print('НОЖНИЦЫ и ...')
elif playerMove == 'б':
    print('БУМАГА и ...')

# Отображение выбора компьютера
randomNumber = random.randint(1, 3)
if randomNumber == 1:
    computerMove = 'к'
    print('КАМЕНЬ')
elif randomNumber == 2:
    computerMove = 'н'
    print('НОЖНИЦЫ')
elif randomNumber == 3:
    computerMove = 'б'
    print('БУМАГА')

# Отображение и учет результата
if playerMove == computerMove:
    print('Ничья!')
    ties = ties + 1
elif playerMove == 'к' and computerMove == 'н':
    print('Вы выиграли!')
    wins = wins + 1
elif playerMove == 'б' and computerMove == 'к':
    print('Вы выиграли!')
    wins = wins + 1
elif playerMove == 'н' and computerMove == 'б':
    print('Вы выиграли!')
    wins = wins + 1
elif playerMove == 'к' and computerMove == 'б':
    print('Вы проиграли!')
    losses = losses + 1
elif playerMove == 'б' and computerMove == 'н':
    print('Вы проиграли!')
    losses = losses + 1
elif playerMove == 'н' and computerMove == 'к':
    print('Вы проиграли!')
    losses = losses + 1
```

Рассмотрим этот код построчно, начиная с первой строки.

```
import random, sys

print('КАМЕНЬ, НОЖНИЦЫ, БУМАГА')

# В этих переменных накапливается количество
# побед, поражений и ничьих
wins = 0
losses = 0
ties = 0
```

Сначала импортируются модули `random` и `sys`, чтобы программа могла вызывать функции `random.randint()` и `sys.exit()`. Также здесь инициализируются три переменные, с помощью которых отслеживается количество побед, поражений и ничьих игрока.

```
while True:    # главный цикл игры
    print('%s побед, %s поражений, %s ничьих' % (wins, losses, ties))
    while True:    # цикл выбора хода
        print('Выберите ход: (к)амень, (н)ожницы, '(б)умага или ' + \
              '(в)ыход')
        playerMove = input()
        if playerMove == 'в':
            sys.exit()    # выход из программы
        if playerMove == 'к' or playerMove == 'н' \
                             or playerMove == 'б':
            break    # выход из цикла выбора хода
        print('Введите "к", "н", "б" или "в".')
```

В программе используется цикл `while`, находящийся внутри другого цикла `while`. Первый из них — это главный цикл игры. На каждой итерации цикла проводится один розыгрыш. Во втором цикле пользователю предлагается сделать ход. Цикл будет продолжаться до тех пор, пока игрок не введет `'к'`, `'н'`, `'б'` или `'в'`. Варианты `'к'`, `'н'` и `'б'` соответствуют камню, ножницам и бумаге, а вариант `'в'` означает выход из игры. В последнем случае вызывается функция `sys.exit()`, которая прекращается работу программы. Если игрок ввел `'к'`, `'н'`, или `'б'`, то цикл завершается. В противном случае программа напоминает игроку о необходимости ввести `'к'`, `'н'`, `'б'` или `'в'` и возвращается к началу цикла.

```
    # Отображение выбора пользователя
    if playerMove == 'к':
        print('КАМЕНЬ и ...')
    elif playerMove == 'н':
        print('НОЖНИЦЫ и ...')
```

```
elif playerMove == 'б':
    print('БУМАГА и ...')
```

Далее на экране отображается выбор игрока.

```
# Отображение выбора компьютера
randomNumber = random.randint(1, 3)
if randomNumber == 1:
    computerMove = 'к'
    print('КАМЕНЬ')
elif randomNumber == 2:
    computerMove = 'н'
    print('НОЖНИЦЫ')
elif randomNumber == 3:
    computerMove = 'б'
    print('БУМАГА')
```

Затем компьютер случайным образом делает ход. Функция `random.randint()` возвращает случайное число в диапазоне 1–3, которое сохраняется в переменной `randomNumber`. Программа присваивает переменной `computerMove` строку `'к'`, `'н'` или `'б'`, исходя из значения переменной `randomNumber`, и отображает ход компьютера.

```
# Отображение и учет результата
if playerMove == computerMove:
    print('Ничья!')
    ties = ties + 1
elif playerMove == 'к' and computerMove == 'н':
    print('Вы выиграли!')
    wins = wins + 1
elif playerMove == 'б' and computerMove == 'к':
    print('Вы выиграли!')
    wins = wins + 1
elif playerMove == 'н' and computerMove == 'б':
    print('Вы выиграли!')
    wins = wins + 1
elif playerMove == 'к' and computerMove == 'б':
    print('Вы проиграли!')
    losses = losses + 1
elif playerMove == 'б' and computerMove == 'н':
    print('Вы проиграли!')
    losses = losses + 1
elif playerMove == 'н' and computerMove == 'к':
    print('Вы проиграли!')
    losses = losses + 1
```

Наконец, программа сравнивает строки, хранящиеся в переменных `playerMove` и `computerMove`, и сообщает результат игры. При этом увеличивается значение переменной `wins`, `losses` или `ties`. Как только программа

достигает конца цикла, она возвращается к началу главного цикла, чтобы провести очередной розыгрыш.

Резюме

Благодаря булевым выражениям, результат вычисления которых равен `True` или `False` (такие выражения называются условиями), можно писать программы, способные принимать решения относительно того, какие фрагменты кода должны выполняться, а какие – пропускаться. Кроме того, можно организовать многократное выполнение кода в цикле до тех пор, пока заданное условие остается истинным. В тех случаях, когда требуется досрочно выйти из цикла или вернуться к его началу, применяются инструкции `break` и `continue`.

Управляющие инструкции позволяют писать более сложные и разветвленные программы. Другой способ структурирования программы – написание собственных функций, о чем мы поговорим в следующей главе.

Контрольные вопросы

1. Каковы два возможных значения булевого типа? Как они записываются?
2. Назовите три булевых оператора.
3. Запишите таблицы истинности для каждого из булевых операторов (т.е. результаты всех возможных комбинаций оператора и двух булевых значений).
4. Каковы результаты вычисления следующих выражений?

```
(5 > 4) and (3 == 5)
not (5 > 4)
(5 > 4) or (3 == 5)
not ((5 > 4) or (3 == 5))
(True and True) and (True == False)
(not False) or (not True)
```

5. Перечислите шесть операторов сравнения.
6. В чем разница между оператором равенства и оператором присваивания?
7. Объясните, что такое условия и где они используются.
8. Укажите три блока в приведенном ниже коде.

```
spam = 0
if spam == 10:
    print('яйца')
    if spam > 5:
        print('бекон')
```

```
    else:
        print('ветчина')
    print('спам')
print('спам')
```

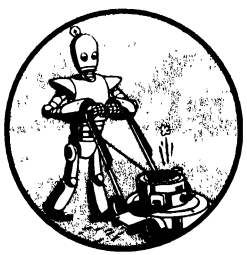
9. Напишите код, который выводит разные сообщения в зависимости от значения, хранящегося в переменной `spam`: `'Hello'` — если переменная равна `1`, `'Howdy'` — если переменная равна `2`, `'Greetings!'` — в остальных случаях.

10. Какую комбинацию клавиш следует нажать, чтобы выйти из бесконечного цикла?
11. Чем разница между инструкциями `break` и `continue`?
12. В чем разница между вызовами `range(10)`, `range(0, 10)` и `range(0, 10, 1)` в цикле `for`?
13. Напишите короткую программу, которая выводит на экран числа от 1 до 10 в цикле `for`. Затем напишите аналогичную программу, в которой используется цикл `while`.
14. Как бы вы вызвали функцию `bacon()`, хранящуюся в модуле `spam`, после того как импортировали этот модуль?

    Дополнительное задание: поищите в Интернете информацию о функциях `round()` и `abs()` и выясните, что они делают. Самостоятельно поэкспериментируйте с ними в интерактивной оболочке.

3

ФУНКЦИИ



В предыдущих главах мы уже познакомились с функциями `print()`, `input()` и `len()`. В Python имеется множество подобных функций, но разрешается писать и собственные функции. *Функция* — это своего рода мини-программа внутри большой программы.

Чтобы лучше понять, как работают функции, рассмотрим конкретный пример. Введите в файловом редакторе приведенный ниже код и сохраните его в файле *helloFunc.py*.

```
❶ def hello():
❷     print('Привет!')
      print('Привет!!!')
      print('Привет всем.')

❸ hello()
  hello()
  hello()
```

Выполнение авторского варианта этой программы можно просмотреть на сайте `https://autbor.com/hellofunc/`. В первой строке находится инструкция `def` ❶, задающая определение функции `hello()`. Блок кода, следующий за инструкцией `def` ❷, образует тело функции. Этот код выполняется при вызове функции, а не при ее первоначальном определении.

Следующие три строки `hello()` ❸ — это вызовы функции. В коде вызов функции обозначается указанием ее имени с последующей парой круглых скобок (в них может содержаться список аргументов, разделенных запятыми). Когда в программе встречается вызов функции, управление передается первой строке тела функции, после чего выполняется весь ее код. По достижении конца функции управление возвращается строке, которая ее вызвала, и программа продолжает выполняться дальше.

Поскольку в данном случае функция `hello()` вызывается три раза, столько же раз выполняется и код этой функции. Запустив программу, вы должны получить следующий результат.

```
Привет!
Привет!!!
Привет всем.
Привет!
Привет!!!
Привет всем.
Привет!
Привет!!!
Привет всем.
```

Основное назначение функции — группирование многократно выполняемого кода. Если бы не пользовательская функция, нам пришлось бы копировать код в буфер обмена и вставлять его в программу везде, где он требуется, в результате чего программа выглядела бы примерно так.

```
print('Привет!')
print('Привет!!!')
```

```
print('Привет всем.')
print('Привет!')
print('Привет!!!')
print('Привет всем.')
print('Привет!')
print('Привет!!!')
print('Привет всем.')
```

В целом рекомендуется всячески избегать дублирования кода, поскольку, если понадобится обновить его (например, при исправлении ошибок), вам придется вносить изменения везде, где этот код был вставлен.

По мере приобретения опыта вы заметите, что все чаще избавляетесь от дублирования кода, сводя к минимуму операции копипастинга. Это позволяет сократить размеры программ, а также упростить их чтение и обновление.

Инструкции `def` с параметрами

Вызывая функции наподобие `print()` или `len()`, вы передаете им значения в скобках, которые называются *аргументами*. Аналогичным образом можно определять собственные функции с аргументами. Введите в файловом редакторе приведенный ниже код и сохраните его в файле *helloFunc2.py*.

```
❶ def hello(name):
❷     print('Привет, ' + name)

❸ hello('Алиса')
  hello('Боб')
```

Запустив программу, вы получите следующий результат.

```
Привет, Алиса
Привет, Боб
```

Выполнение авторского варианта этой программы можно просмотреть на сайте `https://autbor.com/hellofunc2/`. В данном случае определение функции `hello()` включает параметр `name` ❶. *Параметр* — это переменная, в которой сохраняется аргумент функции при ее вызове. Когда функция вызывается в первый раз, ей передается аргумент `'Алиса'` ❸. Внутри функции переменной `name` автоматически присваивается значение `'Алиса'`, которое и выводится на экран функцией `print()` ❷.

Следует учитывать, что после завершения функции сохраненное в параметре значение теряется. Например, если добавить вызов `print(name)` после вызова `hello('Боб')`, то возникнет исключение `NameError`, поскольку вне функции не существует переменной с именем `name`. При выходе

из функции hello() данная переменная уничтожается, поэтому вызов print(name) будет ссылаться на несуществующую переменную.

Это аналогично тому, как при завершении программы теряется информация о ее переменных. Более подробно о том, почему так происходит, мы поговорим при обсуждении локальной области видимости переменных.

Терминология функций

Терминология функций может сбивать с толку новичков. Рассмотрим следующий пример кода.

```
❶ def sayHello(name):
      print('Привет, ' + name)
❷ sayHello('Эл')
```

Определить функцию — значит создать ее, точно так же, как инструкция присваивания наподобие spam = 42 создает переменную spam. Инструкция def определяет функцию sayHello() ❶. Строка sayHello('Эл') ❷ содержит *вызов* только что созданной функции. При вызове управление передается в начало кода функции. Строковое значение 'Эл', передаваемое функции в момент вызова, называется *аргументом*. Этот аргумент присваивается локальной переменной name. Переменные, в которые записываются аргументы, являются *параметрами*.

Инструкция return и возвращаемые значения

Когда вы вызываете функцию len() и передаете ей аргумент, например 'Hello', функция вычисляет длину полученной строки. Целочисленное значение 5, вычисленное функцией, называется *возвращаемым*.

Создавая функцию с помощью инструкции def, можно применить инструкцию return, чтобы указать, какое значение должно возвращаться. Инструкция return состоит из следующих элементов:

- ключевое слово return;
- значение или выражение, которое должна вернуть функция.

Если в инструкции return используется выражение, то возвращается результат вычисления данного выражения. Например, в следующей программе определяется функция, которая каждый раз возвращает другую строку, в зависимости от того, какое число было ей передано в качестве аргумента. Введите в файловом редакторе приведенный ниже код и сохраните его в файле *magic8Ball.py*.

```
❶ import random

❷ def getAnswer(answerNumber):
❸     if answerNumber == 1:
          return 'It is certain'
      elif answerNumber == 2:
          return 'It is decidedly so'
      elif answerNumber == 3:
          return 'Yes'
      elif answerNumber == 4:
          return 'Reply hazy try again'
      elif answerNumber == 5:
          return 'Ask again later'
      elif answerNumber == 6:
          return 'Concentrate and ask again'
      elif answerNumber == 7:
          return 'My reply is no'
      elif answerNumber == 8:
          return 'Outlook not so good'
      elif answerNumber == 9:
          return 'Very doubtful'

❹ r = random.randint(1, 9)
❺ fortune = getAnswer(r)
❻ print(fortune)
```

Выполнение этой программы можно просмотреть на сайте `https://autbor.com/magic8ball/`. При запуске программы Python в первую очередь импортирует модуль `random` ❶. Затем определяется функция `getAnswer()` ❷. Поскольку это всего лишь определение (а не вызов), код функции не выполняется. Далее вызывается функция `random.randint()` с двумя аргументами: `1` и `9` ❹. Эта функция возвращает случайное целое число в диапазоне от `1` до `9` (включительно), которое сохраняется в переменной `r`.

Наконец, вызывается функция `getAnswer()`, которой передается переменная `r` в качестве аргумента ❺. Программа переходит в начало функции `getAnswer()` ❸, и значение переменной `r` сохраняется в параметре `answerNumber`. Затем функция возвращает одно из множества возможных строковых значений, зависящих от значения `answerNumber`. После завершения функции `getAnswer()` управление передается строке, содержащей вызов функции . Возвращаемое значение присваивается переменной `fortune`, которая затем передается функции `print()` ❻ и выводится на экран.

Возвращаемое значение одной функции может служить аргументом для другой функции, поэтому вместо трех инструкций

```
r = random.randint(1, 9)
fortune = getAnswer(r)
print(fortune)
```

можно использовать следующую эквивалентную инструкцию:

```
print(getAnswer(random.randint(1, 9)))
```

Вспомните, что выражения состоят из значений и операторов. Вызов функции можно использовать в выражениях, поскольку это эквивалентно использованию возвращаемого значения функции.

Значение None

В Python определено специальное значение None, которое означает отсутствие значения. None — единственный представитель типа данных NoneType. (Аналогичные значения в других языках программирования могут называться null, nil или undefined.) Подобно булевым значениям True и False, None всегда пишется с прописной буквы.

Это специальное значение может оказаться очень полезным, если в переменной нужно сохранить нечто такое, что невозможно спутать с настоящим значением переменной. В частности, оно используется в качестве возвращаемого значения функции print(), которая всего лишь отображает текст на экране, поэтому ей необязательно возвращать значение, как это делают функции len() и input(). Но поскольку любая функция должна что-то возвращать, функция print() возвращает значение None. Чтобы увидеть, как это работает, введите в интерактивной оболочке следующий код.

```
>>> spam = print('Привет!')
Hello!
>>> None == spam
True
```

Python неявно добавляет инструкцию return None в конец определения любой функции, в которой инструкция return отсутствует. То же самое происходит с циклами while или for, в которые неявно добавляется инструкция continue. Кроме того, если вы используете инструкцию return, но не указываете возвращаемое значение (т.е. вводите только ключевое слово return), то функция автоматически возвращает значение None.

Именованные аргументы и функция print()

Чаще всего аргументы распознаются по их позиции в вызове функции. Например, вызов `random.randint(1, 10)` отличается от вызова `random.randint(10, 1)`. В первом случае возвращается случайное целое число в диапазоне от 1 до 10, поскольку первый аргумент задает нижнюю границу диапазона, а второй — верхнюю, тогда как во втором случае будет выдано сообщение об ошибке.

Именованные аргументы распознаются по имени, которое указывается перед значением аргумента при вызове функции. Такие аргументы часто используются в качестве необязательных параметров. Например, функция `print()` имеет необязательные параметры `end` и `sep`, с помощью которых можно задать соответственно текст, выводимый в конце аргументов, и текст, выводимый между аргументами (разделитель).

Введите в интерактивной оболочке следующий код.

```
print('Привет')
print('Мир')
```

Результат будет таким.

```
Привет
Мир
```

Каждая функция выводит результат на отдельной строке, поскольку в конец текстового аргумента, передаваемого функции `print()`, автоматически добавляется символ новой строки. В случае необходимости это поведение можно изменить, задав другой символ с помощью именованного аргумента `end`. Например, введите следующий код.

```
print('Привет', end='')
print('Мир')
```

Результат будет таким.

```
ПриветМир
```

Теперь все выводится в одну строку, поскольку после текста `'Привет'` не ставится символ новой строки. Вместо него выводится пустая строка. Это полезный прием, когда не нужно, чтобы все сообщения выводились с новой строки.

Если функция `print()` получает несколько строковых аргументов, она автоматически разделяет их пробелом. Введите в интерактивной оболочке следующий код.

```
>>> print('коты', 'собаки', 'мыши')
коты собаки мыши
```

Теперь попробуйте использовать другой разделитель, передав функции именованный аргумент `sep`. Введите следующий код.

```
>>> print('коты', 'собаки', 'мыши', sep=',')
коты,собаки,мыши
```

Именованные аргументы можно добавлять и в пользовательские функции, но сначала необходимо изучить такие типы данных, как списки и словари, о которых будет говориться в следующих двух главах. А пока что достаточно знать лишь то, что у некоторых функций есть именованные аргументы, которые можно задавать при вызове функции.

Стек вызовов

Представьте, что вы беседуете с кем-то о ваших общих знакомых. Вы говорите о своей подруге Алисе, после чего вспоминаете историю о вашем коллеге Бобе, но сначала должны объяснить кое-что о своей кузине Кэрол. Вы завершаете рассказ о Кэрол и возвращаетесь к разговору о Бобе, а когда заканчиваете свой рассказ о Бобе, возвращаетесь к разговору об Алисе. Но тут вам напоминают о вашем брате Дэвиде, поэтому вы рассказываете историю о нем, а затем возвращаетесь к первоначальной истории об Алисе. Структура такого разговора называется стеком, на вершине которого всегда находится текущая тема (рис. 3.1).

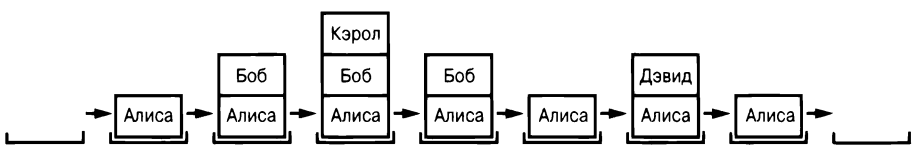


Рис. 3.1. Стек разговора

Нечто подобное происходит при вызовах функций. Python запоминает строку, из которой была вызвана функция, чтобы впоследствии вернуться к ней, когда встретится инструкция `return`. Но если из исходной функции вызывались другие функции, то программа будет сначала возвращаться к ним, прежде чем вернуться из исходной функции.

Откройте окно файлового редактора, введите следующий код и сохраните его в файле *abcdCallStack.py*.

```
  def a():
      print('a() starts')
❶     b()
❷     d()
      print('a() returns')

  def b():
      print('b() starts')
❸     c()
      print('b() returns')

  def c():
❹     print('c() starts')
      print('c() returns')

  def d():
      print('d() starts')
      print('d() returns')

❺ a()
```

Результаты работы программы показаны ниже.

```
a() starts
b() starts
c() starts
c() returns
b() returns
d() starts
d() returns
a() returns
```

Выполнение этой программы можно просмотреть на сайте `https://autbor.com/abcdcallstack/`. Когда вызывается функция `a()` ❺, в ней вызывается функция `b()` ❶, которая, в свою очередь, вызывает функцию `c()` ❸. Функция `c()` просто отображает сообщения `'c() starts'` ❹ и `'c() returns'`, после чего программа возвращается к строке функции `b()`, из которой была вызвана функция `c()` ❸. После завершения функции `b()` программа возвращается к строке функции `a()`, из которой была вызвана функция `b()` ❶. Выполнение продолжается в следующей строке, где содержится вызов функции `d()` ❷. Подобно функции `c()`, функция `d()` просто отображает сообщения `'d() starts'` и `'d() returns'`, после чего управление передается в функцию `a()`, которая ее вызвала ❷. В последней строке функции `a()` отображается сообщение `'a() returns'`, после чего управление передается в основную программу ❺.

Стек вызовов — это структура, с помощью которой интерпретатор Python запоминает, куда следует возвращаться после каждого вызова функции. Стек вызовов не хранится в переменной программы — интерпретатор обрабатывает его самостоятельно. Когда в программе встречается вызов функции, Python создает *объект фрейма* в верхней части стека вызовов. В этом объекте хранится номер строки исходной функции, чтобы Python знал, куда возвращаться по завершении вызова. Если вызывается другая функция, Python помещает другой объект фрейма в стек вызовов над предыдущим объектом.

Когда функция завершается, Python удаляет объект фрейма из верхней части стека и переходит к выполнению строки с номером, хранящимся в нем. Помните, что объекты фреймов всегда добавляются и удаляются только на вершине стека. На рис. 3.2 показано, как меняется стек вызовов в процессе выполнения программы *abcdCallStack.py*.

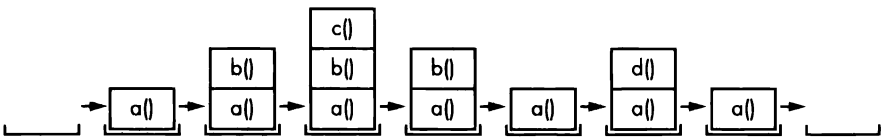


Рис. 3.2. Изменение стека вызовов в программе abcdCallStack.py

На вершине стека вызовов находится функция, которая выполняется в данный момент. Если стек вызовов пуст, значит, точка выполнения находится в основной программе, вне всех остальных функций.

Стек вызовов — это техническая структура, о которой не обязательно знать, чтобы писать программы. Достаточно понимать, что при завершении функции управление передается строке, из которой был осуществлен вызов. В то же время знание стека вызовов помогает разобраться в концепции локальных и глобальных областей видимости.

Локальная и глобальная области видимости

Параметры и переменные, создаваемые в теле вызываемой функции, существуют в *локальной области видимости* этой функции. Переменные, значения которым присваиваются вне функций, существуют в *глобальной области видимости*. Переменные, существующие в локальной области видимости, называются *локальными*, тогда как переменные, существующие в глобальной области видимости, — *глобальными*. Переменная может иметь только одну область видимости. Она не может быть и локальной, и глобальной одновременно.

Область видимости можно представить как контейнер для переменных. При удалении контейнера все значения хранящихся в нем переменных

теряются. Существует только одна глобальная область видимости, которая создается в момент запуска программы. Глобальная область видимости уничтожается, когда программа завершает работу; все глобальные переменные при этом теряются (в противном случае при очередном запуске программы переменные содержали бы те же значения, которые имели в прошлый раз).

Локальная область видимости создается при каждом вызове функции. Любая переменная, которой присваивается значение в этой функции, существует только в локальной области видимости. При завершении функции локальная область видимости уничтожается, и все локальные переменные теряются. Когда вы в следующий раз вызовете эту же функцию, локальные переменные не будут иметь те значения, которые хранились в них в прошлый раз.

Области видимости важны по следующим причинам.

- Код в глобальной области видимости (вне всех функций) не может обращаться к локальным переменным.
- В то же время код в локальной области видимости имеет доступ к глобальным переменным.
- Код, находящийся в локальной области видимости функции, не может использовать переменные из любой другой локальной области видимости.
- Разные переменные могут называться одинаково, если относятся к разным областям видимости. Это означает, что одновременно может существовать как локальная переменная с именем `spam`, так и глобальная переменная с таким же именем.

Причина, по которой в Python существуют различные области видимости вместо одной глобальной, заключается в том, что функции взаимодействуют с остальным кодом только через свои аргументы и возвращаемые значения. Это снижает вероятность возникновения ошибок. Если бы все переменные в программе были глобальными, то ошибочное значение какой-либо переменной было бы слишком сложно отследить, особенно когда программа насчитывает тысячи строк! Зато когда ошибка связана с неверным значением локальной переменной, для поиска причины ошибки достаточно проанализировать лишь код соответствующей функции.

В использовании глобальных переменных в небольших программах нет ничего предосудительного, но придерживаться такого подхода в крупных программах — плохая практика.

Локальные переменные не могут использоваться в глобальной области видимости

Рассмотрим следующую программу, попытка выполнения которой приводит к ошибке.

```
  def spam():
❶     eggs = 31337

  spam()
  print(eggs)
```

Запустив эту программу, вы получите следующее.

```
Traceback (most recent call last):
   File "C:/test1.py", line 4, in <module>
    print(eggs)
NameError: name 'eggs' is not defined
```

Ошибка возникла потому, что переменная `eggs` существует только в локальной области видимости, создаваемой при вызове функции `spam()` ❶. Как только функция `spam()` завершается, эта локальная область видимости уничтожается, и переменная `eggs` прекращает существование. В результате, когда программа пытается выполнить вызов `print(eggs)`, Python выводит сообщение об ошибке, информируя о том, что переменная `eggs` не определена. Если вдуматься, то в этом есть смысл: когда программа выполняется в глобальной области видимости, локальные области видимости не существуют, а значит, нет никаких локальных переменных. Вот почему в глобальной области видимости могут использоваться только глобальные переменные.

В локальных областях видимости не могут использоваться переменные из других локальных областей видимости

Всякий раз, когда вызывается функция (включая случаи, когда она вызывается из другой функции), создается новая локальная область видимости. Рассмотрим следующую программу.

```
  def spam():
❶     eggs = 99
❷     bacon()
❸     print(eggs)

  def bacon():
      ham = 101
```

```
❹     eggs = 0

❺ spam()
```

Выполнение этой программы можно просмотреть на сайте `https://autbor.com/otherlocalscopes/`. В первую очередь вызывается функция `spam()` ❺, после чего создается локальная область видимости, в которой локальной переменной `eggs` ❶ присваивается значение `99`. Затем вызывается функция `bacon()` ❷, и создается вторая локальная область видимости (одновременно могут существовать несколько локальных областей видимости). В этой новой локальной области видимости значение локальной переменной `ham` устанавливается равным `101`, а кроме того, создается новая локальная переменная `eggs` ❹, которая отличается от одноименной переменной, созданной в локальной области видимости функции `spam()`. Эта новая переменная устанавливается равной нулю.

После завершения функции `bacon()` ее локальная область видимости уничтожается. Выполнение программы продолжается в функции `spam()`, которая выводит значение переменной `eggs` ❸, а поскольку локальная область видимости функции `spam()` по-прежнему существует, то значение переменной `eggs` становится равным `99`. Именно это значение и выводит программа.

Таким образом, локальные переменные одной функции полностью изолированы от локальных переменных других функций.

Глобальные переменные доступны из локальной области видимости

Рассмотрим следующую программу.

```
def spam():
    print(eggs)
eggs = 42
spam()
print(eggs)
```

Выполнение этой программы можно просмотреть на сайте `https://autbor.com/readglobal/`. Поскольку имя `eggs` отсутствует в списке параметров функции `spam()` и в теле функции данная переменная не создается, Python считает, что в данном случае имеется в виду ссылка на глобальную переменную `eggs`. Именно поэтому при выполнении программы на экран будет дважды выведено значение `42`.

Локальные и глобальные переменные с одинаковыми именами

Чтобы не усложнять себе жизнь, избегайте использования локальных переменных, имена которых совпадают с именами глобальных или других локальных переменных. Впрочем, с технической точки зрения это вполне допустимо в Python. Чтобы понять, как это происходит, введите в файловом редакторе приведенный ниже код и сохраните его в файле *localGlobalSameName.py*.

```
  def spam():
❶     eggs = 'spam local'
      print(eggs)    # выводится строка 'spam local'

  def bacon():
❷     eggs = 'bacon local'
      print(eggs)    # выводится строка 'bacon local'
      spam()
      print(eggs)    # выводится строка 'bacon local'

❸ eggs = 'global'
  bacon()
  print(eggs)    # выводится строка 'global'
```

Результат работы программы будет таким.

```
bacon local
spam local
bacon local
global
```

Выполнение этой программы можно просмотреть на сайте `https://autbor.com/localglobalsamename/`. Здесь фактически существуют три разные переменные с одним и тем же именем `eggs`:

❶ переменная `eggs`, которая существует в локальной области видимости, когда вызывается функция `spam()`;

❷ переменная `eggs`, которая существует в локальной области видимости, когда вызывается функция `bacon()`;

❸ переменная `eggs`, которая существует в глобальной области видимости.

Тот факт, что все три независимые переменные называются одинаково, может сбивать с толку, если вы хотите отслеживать, какая из них используется в конкретном месте программы. Именно поэтому старайтесь избегать использования одних и тех же имен переменных в разных областях видимости.

Инструкция global

Если в функции возникает потребность изменить глобальную переменную, используйте инструкцию `global`. Например, инструкция `global eggs` в начале функции сообщает интерпретатору следующее: "В этой функции имя `eggs` ссылается на глобальную переменную, поэтому создавать локальную переменную с таким же именем не следует". Введите в файловом редакторе приведенный ниже код и сохраните его в файле *globalStatement.py*.

```
def spam():
❶     global eggs
❷     eggs = 'spam'

eggs = 'global'
spam()
print(eggs)
```

Результат вызова функции `print()` будет таким:

```
spam
```

Выполнение этой программы можно просмотреть на сайте `https://autbor.com/globalstatement/`. Переменная `eggs` объявлена как глобальная в начале функции `spam()` ❶, поэтому, когда ей присваивается значение `'spam'` ❷, эта операция выполняется по отношению к глобальной переменной `eggs`. Никакая локальная переменная `eggs` не создается.

Существуют четыре правила, позволяющие судить о том, в какой области видимости находится переменная: локальной или глобальной.

- Если переменная используется в глобальной области видимости (т.е. вне какой-либо функции), то она всегда является глобальной.
- Если переменная была объявлена в функции с использованием инструкции `global`, то она является глобальной.
- В противном случае, если переменная используется в операции присваивания в функции, она является локальной.
- Но если переменной нигде в функции не присваивается значение, то она является глобальной.

Рассмотрим соответствующий пример. Введите в файловом редакторе приведенный ниже код и сохраните его в файле *sameNameLocalGlobal.py*.

```
def spam():
❶     global eggs
      eggs = 'spam'    # это глобальная переменная
```

```
  def bacon():
❷     eggs = 'bacon'    # это локальная переменная

  def ham():
❸     print(eggs)    # это глобальная переменная

  eggs = 42    # это глобальная переменная
  spam()
  print(eggs)
```

В функции `spam()` переменная `eggs` глобальная, поскольку она объявляется с помощью инструкция `global` ❶. В функции `bacon()` переменная `eggs` локальная, поскольку участвует в операции присваивания ❷. В функции `ham()` ❸ переменная `eggs` глобальная, поскольку для нее нет ни инструкции `global`, ни операции присваивания. Запустив программу *sameNameLocalGlobal.py*, вы должны получить следующий результат:

```
spam
```

Выполнение этой программы можно просмотреть на сайте `https://autbor.com/sameNameLocalGlobal/`. В функции любая переменная будет либо всегда глобальной, либо всегда локальной. Не может быть такого, чтобы в одной и той же функции переменная использовалась сначала как локальная, а затем как глобальная.

Примечание

Если в функции требуется изменить значение, хранящееся в глобальной переменной, то объявите эту переменную с помощью инструкции `global`.

Если попытаться использовать в функции локальную переменную до того, как ей присваивается какое-либо значение, будет выдано сообщение об ошибке. Чтобы убедиться в этом, введите в файловом редакторе следующий код и сохраните его в файле *sameNameError.py*.

```
  def spam():
      print(eggs)    # ОШИБКА!
❶     eggs = 'spam local'

❷ eggs = 'global'
  spam()
```

Запустив программу, вы получите следующее сообщение об ошибке.

```
Traceback (most recent call last):
  File "C:/sameNameError.py", line 6, in <module>
```

```
    spam()
  File "C:/sameNameError.py", line 2, in spam
    print(eggs)    # ОШИБКА!
UnboundLocalError: local variable 'eggs' referenced before assignment
```

Выполнение этой программы можно просмотреть на сайте `https://autbor.com/sameNameError/`. Ошибка обусловлена тем, что интерпретатор, обнаружив присваивание переменной `eggs` в функции `spam()` ❶, начинает считать переменную локальной. Но поскольку функция `print(eggs)` выполняется до того, как переменной `eggs` присваивается какое-либо значение, в момент вызова такой переменной не существует. В этой ситуации Python *не* будет пытаться использовать одноименную глобальную переменную `eggs` ❷.

Функции как "черные ящики"

Обычно все, что вам нужно знать о функции, — это какие входные данные (аргументы) ей следует передавать и какое значение она возвращает. Вам не обязательно обременять себя знанием того, как написан ее код. При подобном высокоуровневом подходе любую функцию можно рассматривать как "черный ящик".

Это фундаментальная концепция современного программирования. В последующих главах вы познакомитесь с модулями, которые содержат функции, написанные другими людьми. Если вы любознательны, то можете заглянуть в их исходный код, однако для того, чтобы использовать эти функции, вам вовсе не обязательно знать их внутреннюю структуру. А поскольку написание функций, в которых глобальные переменные не используются, всячески приветствуется, вам не придется беспокоиться о том, что код этих функций будет нежелательным образом взаимодействовать с остальным кодом вашей программы.

Обработка исключений

На данном этапе возникновение ошибки, или *исключения*, означает крах программы, т.е. ее аварийное завершение. В реальных программах такого происходить не должно. Программа должна выявлять ошибки, обрабатывать их и продолжать работу.

В качестве примера рассмотрим программу, в которой возникает ошибка деления на 0. Введите в файловом редакторе следующий код и сохраните его в файле *zeroDivide.py*.

```
def spam(divideBy):
    return 42 / divideBy

print(spam(2))
```

```
print(spam(12))
print(spam(0))
print(spam(1))
```

Мы определили функцию `spam()` с одним параметром, а затем пытаемся вывести на экран возвращаемое ею значение при различных аргументах, чтобы посмотреть, что при этом произойдет. Запустив программу на выполнение, вы получите следующие результаты.

```
21.0
3.5
Traceback (most recent call last):
  File "C:/zeroDivide.py", line 6, in <module>
    print(spam(0))
  File "C:/zeroDivide.py", line 2, in spam
    return 42 / divideBy
ZeroDivisionError: division by zero
```

Выполнение этой программы можно просмотреть на сайте `https://autbor.com/zerodivide/`. Исключение `ZeroDivisionError` возникает всякий раз, когда предпринимается попытка выполнить деление на нуль. По указанному в сообщении об ошибке номеру строки можно легко определить, что виновницей является инструкция `return` в функции `spam()`.

Ошибки можно обрабатывать с помощью инструкций `try` и `except`. Если в определенном фрагменте программы потенциально может возникнуть ошибка, следует поместить этот код в блок `try`. В случае возникновения ошибки выполнение передается в начало блока `except`.

Попробуем поместить проблемную строку в блок `try` и обработать соответствующее исключение в блоке `except`.

```
def spam(divideBy):
    try:
        return 42 / divideBy
    except ZeroDivisionError:
        print('Error: Invalid argument.')

print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

Когда в коде, заключенном в блок `try`, возникает ошибка, программа немедленно переходит в блок `except`. После завершения этого блока программа продолжает выполняться как обычно. Результат работы программы теперь будет таким.

```
21.0
3.5
Error: Invalid argument.
None
42.0
```

Выполнение этой программы можно просмотреть на сайте `https://autbor.com/tryexceptzerodivide/`. В блоке `try` перехватываются любые ошибки, которые могут возникать при вызовах функций. Рассмотрим следующую программу, в которой вызовы функции `spam()` помещены в блок `try`.

```
def spam(divideBy):
    return 42 / divideBy

try:
    print(spam(2))
    print(spam(12))
    print(spam(0))
    print(spam(1))
except ZeroDivisionError:
    print('Error: Invalid argument.')
```

Результат работы программы будет таким.

```
21.0
3.5
Error: Invalid argument.
```

Выполнение этой программы можно просмотреть на сайте `https://autbor.com/spamintry/`. Инструкция `print(spam(1))` не была выполнена по той причине, что после завершения блока `except` не происходит возврата в блок `try`. Вместо этого выполняются инструкции, следующие за блоком `except`.

Короткая программа: зигзаг

Давайте применим все, что мы уже изучили, для создания небольшой анимационной программы. Эта программа будет создавать зигзагообразный рисунок из звездочек, пока пользователь не остановит ее, щелкнув на кнопке Stop в редакторе Mu или нажав комбинацию клавиш <Ctrl+C>. Результат работы программы будет выглядеть примерно так.

```
    ********
   ********
  ********
```

```
 ********
********
 ********
  ********
   ********
    ********
```

В окне файлового редактора введите следующий код и сохраните файл под именем *zigzag.py*.

```
import time, sys
indent = 0                  # количество пробелов для отступа
indentIncreasing = True     # увеличение или уменьшение отступа

try:
    while True:    # главный цикл программы
        print(' ' * indent, end='')
        print('********')
        time.sleep(0.1)     # пауза длительностью 1/10 секунды

        if indentIncreasing:
            # Увеличение количества пробелов
            indent = indent + 1
            if indent == 20:
                # Изменение направления
                indentIncreasing = False
        else:
            # Уменьшение количества пробелов
            indent = indent - 1
            if indent == 0:
                # Изменение направления
                indentIncreasing = True
except KeyboardInterrupt:
    sys.exit()
```

Рассмотрим этот код построчно.

```
import time, sys
indent = 0                  # количество пробелов для отступа
indentIncreasing = True     # увеличение или уменьшение отступа
```

Сначала импортируются модули `time` и `sys`. В программе используются две переменные: переменная `indent` отслеживает количество пробелов отступа перед полосой из восьми звездочек, а переменная `indentIncreasing` содержит булево значение, позволяющее определить, увеличивается или уменьшается величина отступа.

```
try:
    while True:    # главный цикл программы
        print(' ' * indent, end='')
        print('********')
        time.sleep(0.1)    # пауза длительностью 1/10 секунды
```

Остальная часть программы помещается в блок `try`. Если в процессе выполнения программы пользователь нажмет комбинацию клавиш <Ctrl+C>, возникнет исключение `KeyboardInterrupt`. При отсутствии инструкции `try/except` программа завершит работу, выдав сообщение об ошибке. Но в данном случае мы хотим, чтобы программа обрабатывала исключение `KeyboardInterrupt`, вызывая функцию `sys.exit()`. (Соответствующий код находится в блоке `except` в конце программы.)

Цикл `while True:` будет выполняться бесконечно. Мы используем выражение `' '*indent` для вывода нужного количества пробелов в отступе. Необходимо подавить автоматический переход на новую строку после вывода пробелов, поэтому первой функции `print()` передается аргумент `end=''`. Вторая функция `print()` выводит полосу звездочек. Функцию `time.sleep()` мы еще не рассматривали. О ней достаточно знать, что с ее помощью делается пауза длительностью одна десятая секунды.

```
        if indentIncreasing:
            # Увеличение количества пробелов
            indent = indent + 1
            if indent == 20:
                # Изменение направления
                indentIncreasing = False
```

Далее регулируется размер отступа для очередного вывода группы звездочек. Если значение `indentIncreasing` равно `True`, добавится один отступ. Но как только размер отступа достигнет отметки 20 пробелов, он начнет уменьшаться.

```
        else:
            # Уменьшение количества пробелов
            indent = indent - 1
            if indent == 0:
                # Изменение направления
                indentIncreasing = True
```

Если значение `indentIncreasing` равно `False`, мы уменьшаем размер отступа на единицу. Как только значение отступа достигнет нуля, отступ снова начнет увеличиваться, так как при этом изменяется значение переменной `indentIncreasing`. В любом случае программа возвращается в начало главного цикла, чтобы снова начать вывод звездочек.

```
except KeyboardInterrupt:
    sys.exit()
```

Если пользователь в любой момент времени нажмет комбинацию клавиш <Ctrl+C>, инструкция `except` перехватит возникшее исключение `KeyboardInterrrupt`. Выполнение перейдет в блок `except`, в котором вызывается функция `sys.exit()`, после чего программа завершает работу. Таким образом, даже если главный цикл программы бесконечный, у пользователя есть способ завершить программу.

Резюме

Функции — это основной способ структурирования кода на логические блоки. Поскольку переменные в функциях существуют в собственных локальных областях видимости, код одной функции не может непосредственно воздействовать на переменные другой функции. Эти ограничения, налагаемые на возможность изменения переменных, могут оказаться полезными при отладке кода.

Любую функцию можно представить как "черный ящик". Для вас имеет значение лишь то, какие аргументы ей передаются и какое значение она возвращает, а также то, что код этой функции не может воздействовать на переменные других функций.

В примерах из предыдущих глав единственная ошибка могла привести к краху программы. В данной главе мы изучили инструкции `try` и `except`, которые позволяют продолжить работу программы, даже если в ней возникла ошибка. Это делает программы устойчивыми к распространенным ошибкам.

Контрольные вопросы

1. Зачем нужны функции?
2. Когда именно выполняется код функции: когда она определяется или когда вызывается?
3. С помощью какой инструкции создается функция?
4. В чем разница между определением и вызовом функции?
5. Сколько глобальных областей видимости может иметь программа на языке Python? А сколько локальных?
6. Что происходит с переменными, находящимися в локальной области видимости, при завершении функции?
7. Что такое возвращаемое значение? Может ли возвращаемое значение быть частью выражения?

8. Что возвращает функция, если в ней отсутствует инструкция `return`?
9. Как внутри функции сослаться на глобальную переменную?
10. К какому типу данных относится значение `None`?
11. Что делает инструкция `import areallyourpetsnamederic`?
12. Если имеется функция `bacon()`, содержащаяся в модуле `spam`, то как ее можно вызвать после импорта этого модуля?
13. Как предотвратить аварийное завершение программы при возникновении ошибки?
14. Какой код помещается в блок `try`? Какой код помещается в блок `except`?

Учебные проекты

В качестве практического задания напишите программы для предложенных ниже задач.

Последовательность Коллатца

Напишите функцию `collatz()`, имеющую один параметр `number`. Если `number` — четное число, функция должна вывести на экран и вернуть значение `number // 2`. Если же `number` — нечетное число, то функция должна вывести на экран и вернуть значение `3 * number + 1`.

Далее напишите программу, которая предлагает пользователю ввести целое число, а затем последовательно вызывает функцию `collatz()` для этого числа и значений, возвращаемых очередным вызовом функции, до тех пор пока не будет получено значение 1. (Что любопытно, независимо от выбора начального числа вы все равно рано или поздно получите 1! Даже математики не могут объяснить, почему так происходит. Числовая последовательность, которую вы исследуете с помощью этой программы, называется *последовательностью Коллатца*[1] и иногда характеризуется как "простейшая из неразрешенных проблем математики".)

Не забывайте о том, что функция `input()` возвращает строковое значение, которое должно быть преобразовано в целое число с помощью функции `int()`.

Подсказка: условие четности числа — `number % 2 == 0`, условие нечетности — `number % 2 == 1`.

[1] См. `https://ru.wikipedia.org/wiki/Гипотеза_Коллатца`. — *Примеч. ред.*

Примерные результаты работы этой программы показаны ниже.

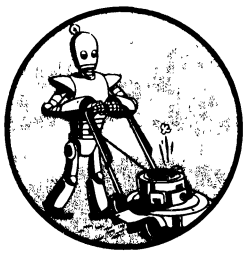
```
Введите число:
3
10
5
16
8
4
2
1
```

Проверка корректности ввода

Добавьте в предыдущий проект инструкции `try` и `except`, чтобы программа могла выявлять ввод пользователем неверных значений. Функция `int()` генерирует исключение `ValueError`, если ей передается строковое представление нецелочисленного значения, например `int('puppy')`. В блоке `except` должно выводиться сообщение о том, что нужно ввести целое число.

4

списки



Еще одна тема, с которой вам обязательно следует познакомиться, прежде чем приступать к написанию собственных программ, — это списки и родственный им тип данных: кортежи. Списки и кортежи могут содержать наборы значений, что упрощает написание программ, обрабатывающих большие объемы данных. А поскольку списки могут включать вложенные списки, появляется возможность создавать иерархические структуры данных.

В этой главе рассматриваются основы работы со списками. Вы также узнаете о методах, которые представляют собой функции, связанные с данными определенного типа. Затем мы вкратце рассмотрим другие списковые типы данных, такие как кортежи и строки, и проанализируем, чем они отличаются друг от друга.

Что такое список

Список — это набор значений, образующих упорядоченную последовательность. Весь набор трактуется как единое значение, которое можно сохранить в переменной или передать в функцию. Например, список может выглядеть так:

```
['кот', 'мышь', 'крыса', 'слон']
```

Подобно тому как строковые значения заключаются в кавычки, показывающие, где начинается и заканчивается строка, список заключается в квадратные скобки ([]). Значения, входящие в список, называются *элементами списка*. Элементы списка разделяются запятыми. Введите в интерактивной оболочке следующие выражения.

```
>>> [1, 2, 3]
[1, 2, 3]
>>> ['кот', 'мышь', 'крыса', 'слон']
['кот', 'мышь', 'крыса', 'слон']
>>> ['привет', 3.1415, True, None, 42]
['привет', 3.1415, True, None, 42]
❶ >>> spam = ['кот', 'мышь', 'крыса', 'слон']
>>> spam
['кот', 'мышь', 'крыса', 'слон']
```

Переменной spam ❶ присваивается одно значение: список. Но само это значение содержит другие значения. Пустой список записывается как []. В таком списке отсутствуют элементы, аналогично тому, как значению '' соответствует пустая строка.

Доступ к элементам списка с помощью индексов

Предположим, имеется список ['кот', 'мышь', 'крыса', 'слон'], хранящийся в переменной spam. Python интерпретирует выражение spam[0] как 'кот', выражение spam[1] — как 'мышь' и т.д. Целое число, указываемое в квадратных скобках после имени списка, называется *индексом*. Первому из значений, входящих в список, соответствует индекс 0, второму — индекс 1, третьему — индекс 2 и т.д. На рис. 4.1 представлен список, сохраняемый в

переменной spam, и показано, какие элементы соответствуют различным индексам. Обратите внимание на то, что индекс последнего элемента на единицу меньше длины списка. Если, к примеру, список содержит четыре элемента, то у последнего из них будет индекс 3.

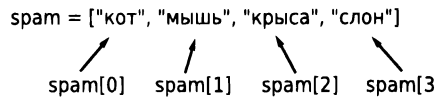


Рис. 4.1. Список, сохраняемый в переменной spam, и индексы, соответствующие его отдельным элементам

Введите в интерактивной оболочке следующие выражения. Сначала мы записываем список в переменную spam.

```
>>> spam = ['кот', 'мышь', 'крыса', 'слон']
>>> spam[0]
'кот'
>>> spam[1]
'мышь'
>>> spam[2]
'крыса'
>>> spam[3]
'слон'
>>> ['кот', 'мышь', 'крыса', 'слон'][3]
'слон'
❶ >>> 'Привет, ' + spam[0]
❷ 'Привет, кот'
>>> spam[0] + ' съедает ' + spam[1] + '.'
'кот съедает мышь.'
```

Обратите внимание на то, что выражение 'Привет, ' + spam[0] ❶ вычисляется как 'Привет, ' + 'кот', поскольку элементу spam[0] соответствует строка 'кот'. Далее выражение преобразуется в строку 'Привет, кот' ❷.

Если попытаться использовать индекс, значение которого превышает количество элементов в списке, будет выдано исключение IndexError.

```
>>> spam = ['кот', 'мышь', 'крыса', 'слон']
>>> spam[10000]
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    spam[10000]
IndexError: list index out of range
```

Индексы могут иметь только целочисленные значения (не вещественные). В следующем примере возникает ошибка TypeError.

```
>>> spam = ['кот', 'мышь', 'крыса', 'слон']
>>> spam[1]
'мышь'
>>> spam[1.0]
Traceback (most recent call last):
    File "<pyshell#13>", line 1, in <module>
    spam[1.0]
TypeError: list indices must be integers or slices, not float
>>> spam[int(1.0)]
'мышь'
```

Элементы списков сами могут быть списками. Доступ к значениям в таких вложенных списках осуществляется с помощью нескольких индексов.

```
>>> spam = [['кот', 'мышь'], [10, 20, 30, 40, 50]]
>>> spam[0]
['кот', 'мышь']
>>> spam[0][1]
'мышь'
>>> spam[1][4]
50
```

Первый индекс указывает, какой из вложенных списков следует использовать, а второй — к какому элементу в этом вложенном списке осуществляется доступ. Например, выражению `spam[0][1]` соответствует значение `'мышь'`, т.е. второе значение в первом списке. Если указан только один индекс, то программа выведет в качестве значения весь вложенный список, соответствующий данному индексу.

Отрицательные индексы

Несмотря на то что отсчет индексов начинается с нуля, в качестве индексов разрешается использовать отрицательные значения. Отрицательному значению `-1` соответствует последний элемент списка, значению `-2` — предпоследний и т.д. Введите в интерактивной оболочке следующие выражения.

```
>>> spam = ['кот', 'мышь', 'крыса', 'слон']
>>> spam[-1]
'слон'
>>> spam[-3]
'мышь'
>>> spam[-1] + ' боится ' + spam[-3] + '.'
'слон боится мышь.'
```

Получение фрагмента списка с помощью среза

С помощью индексов можно извлекать из списка одиночные элементы, тогда как *срезы* позволяют получать сразу несколько значений в виде нового списка. Срез, как и индекс, обозначается квадратными скобками, в которых указываются два индекса, разделенных двоеточием. Рассмотрим разницу между индексом и срезом:

- `spam[2]` — элемент списка с указанным индексом (одно целое число);
- `spam[1:4]` — срез списка (два целых числа).

Первое целое число в квадратных скобках — это индекс, с которого начинается срез. Второе целое число — это индекс, на котором срез заканчивается (сам этот индекс в срез не включается). Значением среза является новый список.

Введите в интерактивной оболочке следующие выражения.

```
>>> spam = ['кот', 'мышь', 'крыса', 'слон']
>>> spam[0:4]
['кот', 'мышь', 'крыса', 'слон']
>>> spam[1:3]
['мышь', 'крыса']
>>> spam[0:-1]
['кот', 'мышь', 'крыса']
```

Допускается сокращенная запись среза с пропуском одного или обоих индексов по обе стороны от двоеточия. Отсутствующий первый индекс равносилен значению `0`, т.е. соответствует началу списка. Отсутствующий второй индекс означает расширение среза до конца списка. Введите в интерактивной оболочке следующие выражения.

```
>>> spam = ['кот', 'мышь', 'крыса', 'слон']
>>> spam[:2]
['кот', 'мышь']
>>> spam[1:]
['мышь', 'крыса', 'слон']
>>> spam[:]
['кот', 'мышь', 'крыса', 'слон']
```

Определение длины списка с помощью функции `len()`

Функция `len()` возвращает количество элементов в переданном ей списке, а в случае строкового значения — количество символов в строке. Введите в интерактивной оболочке следующие инструкции.

```
>>> spam = ['кот', 'собака', 'лось']
>>> len(spam)
3
```

Изменение элементов списка с помощью индексов

Обычно слева от оператора присваивания указывается имя переменной, например `spam = 42`. Но это может быть и элемент списка с заданным индексом. Например, инструкция `spam[1] = 'трубкозуб'` означает следующее: "Записать в элемент списка `spam` с индексом `1` строковое значение `'трубкозуб'`". Введите в интерактивной оболочке следующие выражения.

```
>>> spam = ['кот', 'мышь', 'крыса', 'слон']
>>> spam[1] = 'трубкозуб'
>>> spam
['кот', 'трубкозуб', 'крыса', 'слон']
>>> spam[2] = spam[1]
>>> spam
['кот', 'трубкозуб', 'трубкозуб', 'слон']
>>> spam[-1] = 12345
>>> spam
['кот', 'трубкозуб', 'трубкозуб', 12345]
```

Конкатенация и репликация списков

С помощью оператора + можно объединить два списка в новый список. Оператор *, применяемый к списку и целому числу, позволяет продублировать список заданное количество раз. Введите в интерактивной оболочке следующие выражения.

```
>>> [1, 2, 3] + ['A', 'B', 'C']
[1, 2, 3, 'A', 'B', 'C']
>>> ['X', 'Y', 'Z'] * 3
['X', 'Y', 'Z', 'X', 'Y', 'Z', 'X', 'Y', 'Z']
>>> spam = [1, 2, 3]
>>> spam = spam + ['A', 'B', 'C']
>>> spam
[1, 2, 3, 'A', 'B', 'C']
```

Удаление значений из списка с помощью инструкции `del`

Инструкция `del` удаляет из списка элемент с заданным индексом. Все значения, находящиеся после удаленного, сдвигаются к началу списка на одну позицию. Введите в интерактивной оболочке следующие выражения.

```
>>> spam = ['кот', 'мышь', 'крыса', 'слон']
>>> del spam[2]
>>> spam
['кот', 'мышь', 'слон']
>>> del spam[2]
>>> spam
['кот', 'мышь']
```

Инструкция `del` может также удалять простые переменные. Если попытаться обратиться к удаленной переменной, будет получено сообщение об ошибке `NameError`, поскольку такой переменной больше не существует. На практике такая возможность требуется очень редко. Основное назначение инструкции `del` — удаление элементов из списков.

Работа со списками

У новичков в программировании возникает соблазн создавать множество отдельных переменных для группы родственных значений. Например, если бы я захотел сохранить имена своих котов и кошек, то мог бы сделать это с помощью следующего кода.

```
catName1 = 'Софи'
catName2 = 'Питер'
catName3 = 'Саймон'
catName4 = 'Леди Макбет'
catName5 = 'Толстяк'
catName6 = 'Мисс Клео'
```

Но это далеко не самый удачный способ. Если, например, количество кошек в доме изменится, программа не сможет сохранить имен больше, чем имеется переменных. К тому же в программах такого типа часто происходит дублирование почти идентичных фрагментов кода. Введите следующий код в файловом редакторе и сохраните его в файле *allMyCats1.py*.

```
print('Укажите имя кота или кошки 1:')
catName1 = input()
print('Укажите имя кота или кошки 2:')
catName2 = input()
print('Укажите имя кота или кошки 3:')
catName3 = input()
print('Укажите имя кота или кошки 4:')
catName4 = input()
print('Укажите имя кота или кошки 5:')
catName5 = input()
print('Укажите имя кота или кошки 6:')
catName6 = input()
print('Имена котов и кошек:')
```

```
print(catName1 + ' ' + catName2 + ' ' + catName3 + ' ' +
      catName4 + ' ' + catName5 + ' ' + catName6)
```

Вместо множества однотипных переменных лучше использовать одну переменную-список. Ниже приведена улучшенная версия программы. Здесь используется всего один список, в котором может храниться любое количество имен, введенных пользователем. Откройте новое окно в файловом редакторе, введите в нем приведенный ниже код и сохраните его в файле *allMyCats2.py*.

```
catNames = []
while True:
    print('Укажите имя кота или кошки ' + str(len(catNames) + 1) +
          ' (<Enter> для завершения):')
    name = input()
    if name == '':
        break
    catNames = catNames + [name]    # конкатенация списков
print('Имена котов и кошек:')
for name in catNames:
    print('  ' + name)
```

Запустив эту программу, вы получите примерно следующие результаты.

```
Укажите имя кота или кошки 1 (<Enter> для завершения):
Софи
Укажите имя кота или кошки 2 (<Enter> для завершения):
Питер
Укажите имя кота или кошки 3 (<Enter> для завершения):
Саймон
Укажите имя кота или кошки 4 (<Enter> для завершения):
Леди Макбет
Укажите имя кота или кошки 5 (<Enter> для завершения):
Толстяк
Укажите имя кота или кошки 6 (<Enter> для завершения):
Мисс Клео
Укажите имя кота или кошки 7 (<Enter> для завершения):

Имена котов и кошек:
  Софи
  Питер
  Саймон
  Леди Макбет
  Толстяк
  Мисс Клео
```

Выполнение этих программы можно просмотреть на сайтах `https://autbor.com/allmycats1/` и `https://autbor.com/allmycats2/`. Преимущество списка в том, что теперь все данные хранятся в одной структуре, а

сама программа стала намного более гибкой по сравнению с тем ее вариантом, в котором использовалось множество однотипных переменных.

Использование циклов for со списками

В главе 2 вы узнали о том, как использовать циклы `for` для выполнения одного и того же фрагмента кода заданное количество раз. С технической точки зрения цикл `for` выполняет блок кода по одному разу для каждого элемента указанного списка. Например, выполните следующий код.

```
for i in range(4):
    print(i)
```

Результат будет таким.

```
0
1
2
3
```

Это происходит потому, что возвращаемое функцией `range(4)` значение трактуется как список: `[0, 1, 2, 3]`. Поэтому предыдущий код можно переписать так.

```
for i in [0, 1, 2, 3]:
    print(i)
```

На каждой итерации цикла переменная `i` принимает очередное значение из списка `[0, 1, 2, 3]`.

Популярный прием — использование выражения `range(len(`*список*`))` в цикле `for`, что позволяет пройти по всем индексам в списке. Введите в интерактивной оболочке следующий код.

```
>>> supplies = ['ручки', 'степлеры', 'карандаши', 'скоросшиватели']
>>> for i in range(len(supplies)):
...     print('Индекс ' + str(i) + ': ' + supplies[i])

Индекс 0: ручки
Индекс 1: степлеры
Индекс 2: карандаши
Индекс 3: скоросшиватели
```

Использовать выражение `range(len(supplies))` в цикле `for` очень удобно, так как это дает возможность последовательно получить доступ ко всем индексам списка (переменная `i`) и к соответствующим

элементам списка (выражение `supplies[i]`). Самое главное, что выражение `range(len(supplies))` возвращает правильный результат независимо от количества элементов в списке.

Операторы `in` и `not in`

С помощью операторов `in` и `not in` можно определить, имеется ли заданное значение в списке. Это бинарные операторы: слева от оператора указывается проверяемое значение, справа — список, в котором оно может находиться. Результатом вычисления таких выражений будет булево значение. Введите в интерактивной оболочке следующие выражения.

```
>>> 'здравствуй' in ['привет', 'салют', 'здравствуй', 'эй']
True
>>> spam = ['привет', 'салют', 'здравствуй', 'эй']
>>> 'кот' in spam
False
>>> 'здравствуй' not in spam
False
>>> 'кот' not in spam
True
```

В качестве примера рассмотрим программу, которая предлагает пользователю ввести имя своего домашнего питомца и проверяет, содержится ли оно в списке `pets`. Откройте в файловом редакторе новое окно, введите в него следующий код и сохраните его в файле *myPets.py*.

```
myPets = ['Софи', 'Питер', 'Толстяк']
print('Укажите имя домашнего питомца:')
name = input()
if name not in myPets:
    print('У меня нет домашнего питомца по имени ' + name)
else:
    print(name + ' - мой питомец.')
```

Результат работы программы будет выглядеть примерно так.

```
Укажите имя домашнего питомца:
ФутФут
У меня нет домашнего питомца по имени ФутФут
```

Выполнение этой программы можно просмотреть на сайте `https://autbor.com/mypets/`.

Трюк с групповым присваиванием

Используя трюк с *групповым присваиванием*, можно быстро присвоить значения сразу нескольким переменным в одной строке кода. Рассмотрим следующую последовательность инструкций.

```
>>> cat = ['толстый', 'серый', 'громкий']
>>> size = cat[0]
>>> color = cat[1]
>>> disposition = cat[2]
```

Ее можно переписать более компактно.

```
>>> cat = ['толстый', 'серый', 'громкий']
>>> size, color, disposition = cat
```

Количество переменных должно совпадать с длиной списка, иначе будет выдано исключение `ValueError`.

```
>>> cat = ['толстый', 'серый', 'громкий']
>>> size, color, disposition, name = cat
Traceback (most recent call last):
  File "<pyshell#84>", line 1, in <module>
    size, color, disposition, name = cat
ValueError: not enough values to unpack (expected 4, got 3)
```

Использование функции `enumerate()` со списками

Вместо того чтобы использовать вызов `range(len(`*список*`))` в цикле `for` для получения целочисленных индексов элементов списка, можно вызвать функцию `enumerate()`. На каждой итерации цикла она будет возвращать два значения: индекс элемента в списке и сам элемент. Например, следующий код эквивалентен коду, рассмотренному в разделе "Использование циклов for со списками".

```
>>> supplies = ['ручки', 'степлеры', 'карандаши', 'скоросшиватели']
>>> for index, item in enumerate(supplies):
...     print('Индекс ' + str(index) + ': ' + item)

Индекс 0: ручки
Индекс 1: степлеры
Индекс 2: карандаши
Индекс 3: скоросшиватели
```

Функция `enumerate()` полезна, если в цикле нужен как элемент списка, так и его индекс.

Использование функций random.choice() и random.shuffle() со списками

Модуль random содержит несколько функций, которые в качестве аргументов получают списки. Функция random.choice() возвращает случайно выбранный элемент из списка. Введите в интерактивной оболочке следующий код.

```
>>> import random
>>> pets = ['Собака', 'Кот', 'Лось']
>>> random.choice(pets)
'Собака'
>>> random.choice(pets)
'Кот'
>>> random.choice(pets)
'Кот'
```

Вызов random.choice(*список*) можно рассматривать как более короткую форму записи следующего выражения:

```
список[random.randint(0, len(список) - 1])
```

Функция random.shuffle() переупорядочивает элементы списка. Она изменяет исходный список, а не возвращает новый. Введите в интерактивной оболочке следующий код.

```
>>> import random
>>> people = ['Алиса', 'Боб', 'Кэрол', 'Дэвид']
>>> random.shuffle(people)
>>> people
['Кэрол', 'Дэвид', 'Алиса', 'Боб']
>>> random.shuffle(people)
>>> people
['Алиса', 'Дэвид', 'Боб', 'Кэрол']
```

Комбинированные операторы присваивания

В операции присваивания часто используется текущее значение самой переменной. Например, если переменной spam необходимо присвоить значение 42, а затем увеличить его на 1, то это можно сделать с помощью следующего кода.

```
>>> spam = 42
>>> spam = spam + 1
>>> spam
43
```

Благодаря комбинированному оператору присваивания += можно немного сократить код.

```
>>> spam = 42
>>> spam += 1
>>> spam
43
```

Комбинированные операторы присваивания перечислены в табл. 4.1.

Таблица 4.1. Комбинированные операторы присваивания

Комбинированное присваивание	Эквивалентное обычное присваивание
spam += 1	spam = spam + 1
spam -= 1	spam = spam - 1
spam *= 1	spam = spam * 1
spam /= 1	spam = spam / 1
spam %= 1	spam = spam % 1

Кроме того, оператор += может использоваться для конкатенации, а оператор *= — для репликации строк и списков. Введите в интерактивной оболочке следующие инструкции.

```
>>> spam = 'Здравствуй,'
>>> spam += ' мир!'
>>> spam
'Здравствуй, мир!'
>>> bacon = ['Софи']
>>> bacon *= 3
>>> bacon
['Софи', 'Софи', 'Софи']
```

Методы

Метод — это та же функция, с тем лишь отличием, что она вызывается для конкретного значения. Например, если список хранится в переменной spam, то для него можно вызвать метод index(): spam.index('привет'). Метод указывается после имени переменной (или самого значения) и отделяется от него точкой.

У каждого типа данных есть свой набор методов. В частности, для списков есть полезные методы, позволяющие искать, добавлять и удалять элементы, а также выполнять с ними другие манипуляции.

Поиск значения в списке с помощью метода index()

У списков есть метод index(), который возвращает индекс указанного значения при условии, что оно содержится в списке. Если значение отсутствует в списке, генерируется исключение ValueError. Введите в интерактивной оболочке следующие инструкции.

```
>>> spam = ['привет', 'салют', 'здравствуй', 'эй']
>>> spam.index('привет')
0
>>> spam.index('эй')
3
>>> spam.index('привет привет привет')
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    spam.index('привет привет привет')
ValueError: 'привет привет привет' is not in list
```

При наличии дубликатов возвращается индекс первого из найденных элементов. Введите в интерактивной оболочке следующие инструкции, и вы увидите, что метод index() возвращает 1, а не 3.

```
>>> spam = ['Софи', 'Питер', 'Толстяк', 'Питер']
>>> spam.index('Питер')
1
```

Добавление значений в список с помощью методов append() и insert()

Для добавления новых значений в список используются методы append() и insert(). Введите в интерактивной оболочке следующие инструкции, чтобы применить метод append() к списку, хранящемуся в переменной spam.

```
>>> spam = ['кот', 'собака', 'мышь']
>>> spam.append('лось')
>>> spam
['кот', 'собака', 'мышь', 'лось']
```

Метод append() добавляет элемент в конец списка. Метод insert() позволяет добавить в список элемент с конкретным индексом. Первый аргумент метода insert() — это индекс нового элемента, а второй аргумент — вставляемое значение. Введите в интерактивной оболочке следующие инструкции.

```
>>> spam = ['кот', 'собака', 'мышь']
>>> spam.insert(1, 'курица')
>>> spam
['кот', 'курица', 'собака', 'мышь']
```

Обратите внимание на способ присваивания значений: `spam.append('лось')` и `spam.insert(1, 'курица')`, а не `spam = spam.append('лось')` и `spam = spam.insert(1, 'курица')`. Ни метод `append()`, ни метод `insert()` не возвращают новый список (в действительности оба они возвращают значение `None`, но вряд ли вы захотите присвоить его переменной списка). Вместо этого изменяется исходный список. (Вопросы изменения списков обсуждаются в разделе "Изменяемые и неизменяемые типы данных".)

Методы связаны с конкретными типами данных. В частности, `append()` и `insert()` — это методы списков, которые могут вызываться только в таком контексте. Вызвать их для других типов данных, например для строк или целых чисел, нельзя. Введите в интерактивной оболочке следующие инструкции, и вы получите исключение `AttributeError`.

```
>>> eggs = 'здравствуй'
>>> eggs.append('мир')
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    eggs.append('world')
AttributeError: 'str' object has no attribute 'append'
>>> bacon = 42
>>> bacon.insert(1, 'мир')
Traceback (most recent call last):
  File "<pyshell#22>", line 1, in <module>
    bacon.insert(1, 'world')
AttributeError: 'int' object has no attribute 'insert'
```

Удаление значений из списка с помощью метода `remove()`

Метод `remove()` удаляет из списка указанное значение. Введите в интерактивной оболочке следующие инструкции.

```
>>> spam = ['кот', 'мышь', 'крыса', 'слон']
>>> spam.remove('мышь')
>>> spam
['кот', 'крыса', 'слон']
```

При попытке удалить значение, отсутствующее в списке, будет сгенерировано исключение `ValueError`. Чтобы в этом убедиться, введите в интерактивной оболочке следующие инструкции.

```
>>> spam = ['кот', 'мышь', 'крыса', 'слон']
>>> spam.remove('курица')
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    spam.remove('курица')
ValueError: list.remove(x): x not in list
```

Если в списке имеется несколько одинаковых значений, будет удалено первое из них. Введите в интерактивной оболочке следующие инструкции.

```
>>> spam = ['кот', 'мышь', 'крыса', 'кот', 'шляпа', 'кот']
>>> spam.remove('кот')
>>> spam
['мышь', 'крыса', 'кот', 'шляпа', 'кот']
```

Инструкцию `del` удобно применять в тех случаях, когда известен индекс удаляемого значения, а метод `remove()` — когда известно само удаляемое значение.

Сортировка списка с помощью метода `sort()`

Для сортировки списков, содержащих числа или строки, применяется метод `sort()`. Введите в интерактивной оболочке следующие инструкции.

```
>>> spam = [2, 5, 3.14, 1, -7]
>>> spam.sort()
>>> spam
[-7, 1, 2, 3.14, 5]
>>> spam = ['муравьи', 'коты', 'собаки', 'барсуки', 'слоны']
>>> spam.sort()
>>> spam
['барсуки', 'коты', 'муравьи', 'слоны', 'собаки']
```

Чтобы выполнить сортировку в обратном порядке, следует передать методу `sort()` именованный аргумент `reverse` со значением `True`. Введите в интерактивной оболочке следующие инструкции.

```
>>> spam.sort(reverse=True)
>>> spam
['собаки', 'слоны', 'муравьи', 'коты', 'барсуки']
```

Относительно метода `sort()` следует сделать три замечания. Во-первых, он сортирует исходный список. Не пытайтесь использовать его в выражениях вида `spam = spam.sort()`.

Во-вторых, невозможно отсортировать список, который содержит одновременно и числа, и строки, поскольку Python не знает, как сравнивать

разные типы данных. Введите в интерактивной оболочке следующие инструкции, и вы получите исключение TypeError.

```
>>> spam = [1, 3, 2, 4, 'Алиса', 'Боб']
>>> spam.sort()
Traceback (most recent call last):
  File "<pyshell#70>", line 1, in <module>
    spam.sort()
TypeError: '<' not supported between instances of 'str' and 'int'
```

В-третьих, метод sort() сортирует строки не в алфавитном порядке, а *в соответствии с таблицей ASCII*. Это означает, что буквы в верхнем регистре предшествуют буквам в нижнем регистре. Поэтому, например, буква 'a' будет располагаться в процессе сортировки после буквы 'Я'. Введите в интерактивной оболочке следующие инструкции:

```
>>> spam = ['Алиса', 'муравьи', 'Боб', 'барсуки', 'Кэрол', 'коты']
>>> spam.sort()
>>> spam
['Алиса', 'Боб', 'Кэрол', 'барсуки', 'коты', 'муравьи']
```

Если необходимо отсортировать строку в обычном алфавитном порядке, то передайте методу sort() именованный аргумент key со значением str.lower.

```
>>> spam = ['a', 'я', 'A', 'Я']
>>> spam.sort(key=str.lower)
>>> spam
['a', 'A', 'я', 'Я']
```

Это приведет к тому, что метод sort() будет обрабатывать все элементы списка так, как будто они записаны в нижнем регистре, но сами элементы меняться не будут.

Инверсия списка с помощью метода reverse()

Чтобы быстро инвертировать порядок элементов в списке, воспользуйтесь методом reverse(). Введите в интерактивной оболочке следующие инструкции.

```
>>> spam = ['кот', 'лось', 'собака']
>>> spam.reverse()
>>> spam
['собака', 'лось', 'кот']
```

Как и метод `sort()`, метод `reverse()` не возвращает список, поэтому необходимо писать `spam.reverse()`, а не `spam = spam.reverse()`.

Исключения из правил отступа в Python

В большинстве случаев величина отступа для строки кода сообщает интерпретатору Python о том, к какому блоку относится эта строка. Но из каждого правила есть исключения. В частности, списки могут занимать несколько строк, и отступы в них не играют никакой роли. Список не будет закончен, пока не встретится закрывающая квадратная скобка. Рассмотрим следующий код.

```
spam = ['яблоки',
    'апельсины',
                    'бананы',
'коты']
print(spam)
```

Конечно, на практике никто так не делает. Списки, как и листинги в целом, стараются оформлять аккуратно.

Можно также разделить одну инструкцию на несколько строк, используя *символ продолжения строки* (\). Он как бы сообщает интерпретатору: "Эта инструкция продолжается на следующей строке". Отступ в строке после символа \ не имеет значения. Например, приведенный ниже код совершенно корректен.

```
print('Восемьдесят семь ' + \
      'лет тому назад...')
```

Эти приемы полезны, когда нужно перегруппировать длинные строки кода так, чтобы они стали более читабельными.

Пример программы: Magic 8 Ball со списком

Используя список, можно написать гораздо более элегантную версию программы Magic 8 Ball из предыдущей главы. Вместо того чтобы вводить почти идентичные инструкции `elif`, можно создать единственный список, с которым будет работать программа. Откройте в файловом редакторе новое окно, введите в нем следующий код и сохраните его в файле *magic8Ball2.py*.

```
import random

messages = ['It is certain',
    'It is decidedly so',
    'Yes',
    'Reply hazy try again',
    'Ask again later',
```

```
    'Concentrate and ask again',
    'My reply is no',
    'Outlook not so good',
    'Very doubtful']

print(messages[random.randint(0, len(messages) - 1)])
```

Выполнение этой программы можно просмотреть на сайте `https://autbor.com/magic8ball2/`. Она работает точно так же, как и программа *magic8Ball.py*.

Обратите внимание на выражение, используемое в качестве индекса списка: `random.randint(0, len(messages) - 1)`. Оно позволяет получать случайные числа в нужном диапазоне, независимо от длины списка `messages`. В данном случае генерируется случайное число в диапазоне от `0` до значения `len(messages) - 1`. Преимуществом такого подхода является то, что можно добавлять и удалять элементы списка, не изменяя другие строки кода. Если впоследствии понадобится обновить код, то придется изменять меньшее количество строк кода, а это означает, что уменьшится и вероятность внесения ошибок.

Списковые типы данных

Список — не единственный тип данных, представляющий собой упорядоченную последовательность значений. Строки во многом напоминают списки, если рассматривать строку как список, состоящий из символов. К списковым типам данных относятся списки, строки, объекты диапазона, возвращаемые методом `range()`, и кортежи (рассматриваются далее). Многое из того, что можно делать со списками, можно делать и со строками, а также остальными списковыми типами. В частности, к ним применимы операции индексирования и получения срезов, операторы `in` и `not in` и функции наподобие `len()`. Также они могут использоваться в циклах `for`. Чтобы убедиться в этом на практике, введите в интерактивной оболочке следующие инструкции.

```
>>> name = 'Сократ'
>>> name[0]
'С'
>>> name[-2]
'а'
>>> name[0:4]
'Сокр'
>>> 'Co' in name
True
>>> 'c' in name
False
>>> 'a' not in name
```

```
False
>>> for i in name:
...     print('* * * ' + i + ' * * *')

* * * К * * *
* * * о * * *
* * * к * * *
* * * р * * *
* * * а * * *
* * * т * * *
```

Изменяемые и неизменяемые типы данных

Между списками и строками существует одно важное различие. Список — это *изменяемый* тип данных: значения, хранящиеся в списках, можно добавлять, удалять и изменять. Строки же представляют собой *неизменяемый* тип данных: строку нельзя изменить. Попытка заменить одиночный символ в строке приведет к появлению исключения `TypeError`, в чем вы сможете убедиться, введя в интерактивной оболочке следующий код.

```
>>> name = 'Кошка Софи'
>>> name[7] = '- это'
Traceback (most recent call last):
  File "<pyshell#50>", line 1, in <module>
    name[7] = '- это'
TypeError: 'str' object does not support item assignment
```

Правильный способ "изменения" строки заключается в создании среза и выполнении конкатенации для получения *новой* строки путем копирования фрагментов исходной строки. Введите в интерактивной оболочке следующие инструкции.

```
>>> name = 'Кошка Софи'
>>> newName = name[6:10] + ' - ' + name[0:5]
>>> name
'Кошка Софи'
>>> newName
'Софи - Кошка'
```

Для получения символов, которые мы не собираемся менять, использованы срезы `[0:5]` и `[6:10]`. Заметьте, что исходная строка `'Кошка Софи'` осталась неизменной.

Несмотря на то что список — изменяемый тип данных, в приведенном ниже коде вторая строка не изменяет список `eggs`.

```
>>> eggs = [1, 2, 3]
>>> eggs = [4, 5, 6]
>>> eggs
[4, 5, 6]
```

Список, который хранился в переменной `eggs`, в данном случае не подвергся изменениям. Просто было создано новое значение `[4, 5, 6]`, которое заменило прежнее `[1, 2, 3]` (рис. 4.2).

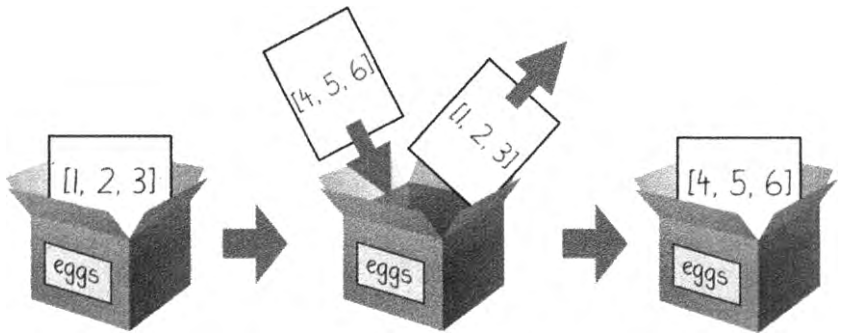


Рис. 4.2. При выполнении инструкции `eggs = [4, 5, 6]` содержимое переменной `eggs` заменяется новым списком

Если бы мы хотели действительно изменить первоначальный список, хранящийся в переменной `eggs`, чтобы он содержал значения `[4, 5, 6]`, то нам пришлось бы сделать примерно следующее.

```
>>> eggs = [1, 2, 3]
>>> del eggs[2]
>>> del eggs[1]
>>> del eggs[0]
>>> eggs.append(4)
>>> eggs.append(5)
>>> eggs.append(6)
>>> eggs
[4, 5, 6]
```

В этом примере переменная `eggs` содержит тот же самый список. Суть в том, что он редактируется, а не перезаписывается. На рис. 4.3 показан процесс изменения списка.

В случае изменяемых типов данных изменению подвергается исходный объект, поскольку значение переменной не заменяется новым списком.

Разделение на изменяемые и неизменяемые типы данных может показаться не имеющим особого смысла, однако, как будет показано в разделе "Передача ссылок", различия проявляются при вызове функций с изменяемыми и неизменяемыми аргументами. Но сначала нам предстоит

рассмотреть кортежи, которые представляют собой неизменяемую разновидность списка.

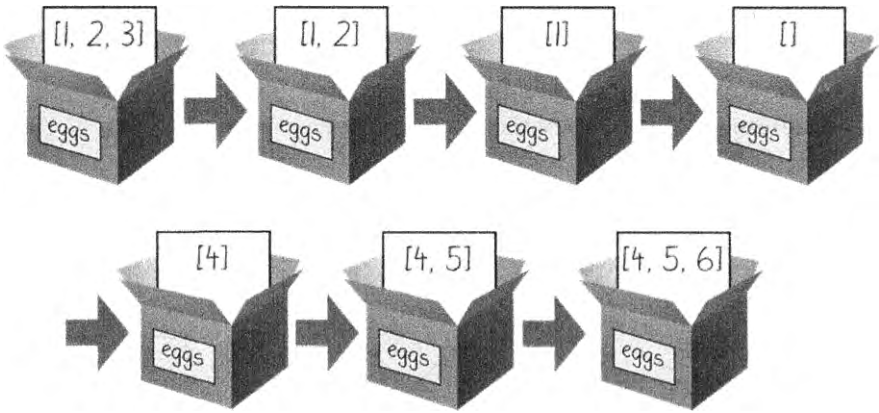


Рис. 4.3. Инструкция `del` и метод `append()` изменяют элементы исходного списка

Кортежи

Кортежи почти идентичны спискам и отличаются от них лишь в двух отношениях. Во-первых, кортежи заключаются в круглые скобки ((и)), а не в квадратные ([и]). Введите в интерактивной оболочке следующие инструкции.

```
>>> eggs = ('привет', 42, 0.5)
>>> eggs[0]
'hello'
>>> eggs[1:3]
(42, 0.5)
>>> len(eggs)
3
```

Но главным отличием кортежей от списков является то, что кортеж, подобно строке, представляет собой неизменяемый тип данных. Его значения нельзя изменять, добавлять или удалять. Введите в интерактивной оболочке следующий код.

```
>>> eggs = ('привет', 42, 0.5)
>>> eggs[1] = 99
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    eggs[1] = 99
TypeError: 'tuple' object does not support item assignment
```

Если кортеж содержит единственное значение, после него внутри скобок необходимо ставить запятую. В противном случае Python считает, что вы просто заключили обычное значение в скобки. Запятая служит признаком кортежа. (В отличие от других языков программирования, в Python допускается ставить запятую в конце кортежа или списка.) Введите в интерактивной оболочке следующие инструкции, чтобы увидеть, к чему приводит отсутствие запятой.

```
>>> type(('привет',))
<class 'tuple'>
>>> type(('привет'))
<class 'str'>
```

Используя кортежи, вы тем самым сообщаете тому, кто будет читать код вашей программы, что не намереваетесь изменять данную последовательность значений. Еще одним преимуществом кортежей по сравнению со списками служит то, что, благодаря неизменяемости их содержимого, Python может реализовать определенные схемы оптимизации, ускоряющие работу программы.

Преобразование типов с помощью функций `list()` и `tuple()`

Подобно тому как функция `str(42)` возвращает значение `'42'`, являющееся строковым представлением целого числа `42`, функции `list()` и `tuple()` возвращают версии переданных им значений в виде списка и кортежа соответственно. Введите в интерактивной оболочке следующие инструкции и обратите внимание на то, что типы передаваемых и возвращаемых значений различаются.

```
>>> tuple(['кот', 'пес', 6])
('кот', 'пес', 6)
>>> list(('кот', 'пес', 6))
['кот', 'пес', 6]
>>> list('привет')
['п', 'р', 'и', 'в', 'е', 'т']
```

Преобразование кортежа в список удобно применять в тех случаях, когда необходимо получить изменяемую версию кортежа.

Ссылки

Как вы уже знаете, переменные хранят строковые и целочисленные значения. Но это упрощенное объяснение. С технической точки зрения

переменные хранят ссылки на области памяти, где находятся значения. Введите в интерактивной оболочке следующие инструкции.

```
>>> spam = 42
>>> cheese = spam
>>> spam = 100
>>> spam
100
>>> cheese
42
```

Когда вы присваиваете переменной spam значение 42, вы в действительности создаете значение 42 в памяти компьютера и сохраняете *ссылку* на него в переменной spam. При копировании переменной spam в переменную cheese копируется ссылка, а не само значение. В результате и переменная spam, и переменная cheese ссылаются на одно и то же значение 42 в памяти компьютера. Когда затем переменной spam присваивается значение 100, это значение создается в памяти компьютера, а в переменной spam сохраняется ссылка на него. Это никак не отражается на содержимом переменной cheese. Целые числа являются *неизменяемыми* значениями. Изменение переменной spam просто приводит к тому, что она начинает ссылаться на другое значение в памяти компьютера.

А вот списки работают совершенно не так, поскольку представляют собой *изменяемый тип данных*. Следующий пример поможет вам разобраться в этом. Введите в интерактивной оболочке приведенные ниже инструкции.

```
❶ >>> spam = [0, 1, 2, 3, 4, 5]
❷ >>> cheese = spam    # копируется ссылка, а не список
❸ >>> cheese[1] = 'Hello!'    # изменение элемента списка
  >>> spam
  [0, 'Hello!', 2, 3, 4, 5]
  >>> cheese    # переменная cheese ссылается на тот же список
  [0, 'Hello!', 2, 3, 4, 5]
```

Возможно, полученные результаты вас несколько озадачат. Несмотря на то что изменялся только список cheese, изменение затронуло также список spam.

Создавая список ❶, вы записываете ссылку на него в переменную spam. В следующей строке ❷ в переменную cheese копируется не сам список, а только ссылка на него, хранящаяся в переменной spam. Это означает, что теперь обе переменные, spam и cheese, ссылаются на один и тот же список. Сам список существует в единственном экземпляре, поскольку он никуда не копировался. Изменяя первый элемент списка с помощью переменной cheese ❸, вы изменяете тот же самый список, на который ссылается и переменная spam.

Вспомните, что переменные можно уподобить коробкам, в которых находятся значения. В этом смысле предыдущие рисунки, на которых были изображены коробки со списками, не совсем точны, поскольку в реальности в списковых переменных хранятся не сами списки, а лишь *ссылки* на них. (Все ссылки снабжаются числовыми идентификаторами, которые используются внутренними механизмами Python; программа о них не знает.) На рис. 4.4 с помощью метафоры коробки показано, что происходит, когда переменной `spam` присваивается список.

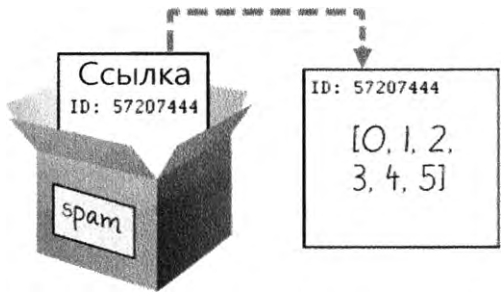


Рис. 4.4. При выполнении инструкции `spam = [0, 1, 2, 3, 4, 5]` в переменной `spam` сохраняется ссылка на список, а не сам список

На рис. 4.5 изображен процесс копирования ссылки из переменной `spam` в переменную `cheese`. В переменной `cheese` сохраняется только ссылка, но не сам список. Заметьте, что обе ссылки указывают на один и тот же список.

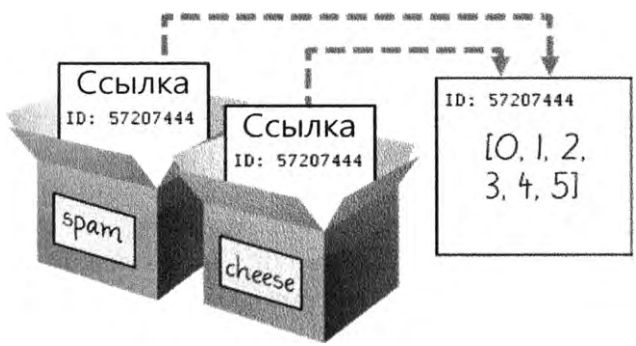


Рис. 4.5. При выполнении инструкции `spam = cheese` копируется ссылка на список, а не сам список

Если вы измените список, на который ссылается переменная `cheese`, то список, на который ссылается переменная `spam`, тоже изменится, поскольку обе эти переменные ссылаются на один и тот же список (рис. 4.6).

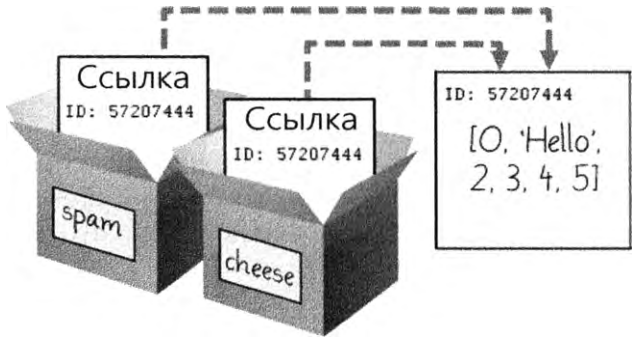


Рис. 4.6. Инструкция `cheese[1] = 'Hello!'` изменяет список, на который ссылаются обе переменные

Несмотря на то что с технической точки зрения в переменных Python хранятся лишь ссылки на значения, программисты привыкли говорить, что "переменная содержит значение".

Тождественность и функция `id()`

Вы наверняка задаетесь вопросом, почему странное поведение изменяемых списков, описанное в предыдущем разделе, не присуще неизменяемым значениям, таким как целые числа или строки. Чтобы ответить на этот вопрос, воспользуемся функцией `id()`. Все значения в Python имеют уникальные идентификаторы, которые можно узнать с помощью функции `id()`. Введите в интерактивной оболочке следующую инструкцию.

```
>>> id('Привет!') # на разных компьютерах возвращаются разные значения
44491136
```

Когда Python выполняет инструкцию `id('Привет!')`, он создает строку `'Привет!'` в памяти компьютера. Функция `id()` возвращает числовое значение адреса памяти, где хранится эта строка. Python выбирает адрес в зависимости от того, какие ячейки памяти свободны в данный момент, поэтому он будет разным при каждом запуске программы.

Как и любые строки, значение `'Привет!'` не подлежит изменению. Если в переменную записывается новая строка, создается новый объект, сохраняемый в другой области памяти, и переменная начинает ссылаться на этот новый объект. Например, введите в интерактивной оболочке следующий код и посмотрите, как меняется идентификатор строки, на которую ссылается переменная `bacon`.

```
>>> bacon = 'Здравствуй,'
>>> id(bacon)
44481156
```

```
>>> bacon += ' мир!' # новая строка, образуемая из строк
                     # 'Здравствуй,' и ' мир!'
>>> id(bacon) # переменная bacon теперь ссылается на другую строку
44609712
```

С другой стороны, списки — изменяемые объекты. Метод `append()` не создает новый объект списка — он изменяет существующий список.

```
>>> eggs = ['кот', 'собака'] # создание нового списка
>>> id(eggs)
35152584
>>> eggs.append('лось') # метод append() изменяет исходный список
>>> id(eggs) # переменная eggs ссылается на тот же самый список
35152584
>>> eggs = ['мышь', 'крыса', корова'] # создание нового списка
                                      # в новой области памяти
>>> id(eggs) # переменная eggs теперь ссылается на другой список
44409800
```

Если две переменные ссылаются на один и тот же список (как, например, `spam` и `cheese` в предыдущем разделе) и его содержимое меняется, то это влияет на обе переменные. Методы списков, в частности `append()`, `extend()`, `remove()`, `sort()`, `reverse()` и др., изменяют исходные списки.

Чтобы освободить память, *автоматический сборщик мусора* в Python удаляет все значения, на которые не ссылаются никакие переменные. Вам не нужно беспокоиться о том, как работает сборщик мусора, и это хорошо: ручное управление памятью в других языках программирования часто приводит к появлению ошибок.

Передача ссылок

Ссылки особенно важны для понимания механизма передачи аргументов в функции. Когда вызывается функция, значения аргументов копируются в переменные параметров. Для списков (и словарей, о которых пойдет речь в следующей главе) это означает, что в параметры копируются ссылки. Чтобы увидеть, к каким последствиям это приводит, введите в новом окне файлового редактора следующий код и сохраните его в файле *passingReference.py*.

```
def eggs(someParameter):
    someParameter.append('Привет')

spam = [1, 2, 3]
eggs(spam)
print(spam)
```

Обратите внимание на то, что при вызове функции `eggs()` в переменную `spam` не записывается возвращаемое значение. Вместо этого список изменяется напрямую. Запустив программу, вы получите следующий результат:

```
[1, 2, 3, 'Привет']
```

Несмотря на то что переменные `spam` и `someParameter` существуют в разных областях видимости, они ссылаются на один и тот же список. Вот почему вызов метода `append('Привет')` внутри функции оказывает влияние на список даже после завершения функции.

Помните о такой особенности списков (и словарей) в Python. Если забыть о том, как обрабатываются изменяемые типы данных, то это чревато возникновением самых неожиданных ошибок.

Функции copy() и deepcopy()

Передача ссылок обычно считается самым удобным способом работы со списками и словарями, но если функция изменяет переданный ей объект, нежелательно, чтобы это изменение отражалось на исходном списке или словаре. Для решения данной проблемы в Python имеется модуль `copy`, в котором содержатся функции `copy()` и `deepcopy()`. Первая из них позволяет создать копию изменяемого значения, такого как список или словарь, а не просто копию ссылки. Введите в интерактивной оболочке следующие инструкции.

```
>>> import copy
>>> spam = ['A', 'B', 'C', 'D']
>>> id(spam)
44684232
>>> cheese = copy.copy(spam)
>>> id(cheese) # cheese - это другой список с другим идентификатором
44685832
>>> cheese[1] = 42
>>> spam
['A', 'B', 'C', 'D']
>>> cheese
['A', 42, 'C', 'D']
```

Как видите, теперь переменные `spam` и `cheese` ссылаются на разные списки. Именно этим объясняется тот факт, что при выполнении инструкции `cheese[1] = 42` изменяется только список `cheese`. Как показано на рис. 4.7, идентификаторы ссылок, хранимых в обеих переменных, больше не совпадают, поскольку теперь эти переменные ссылаются на два независимых списка.

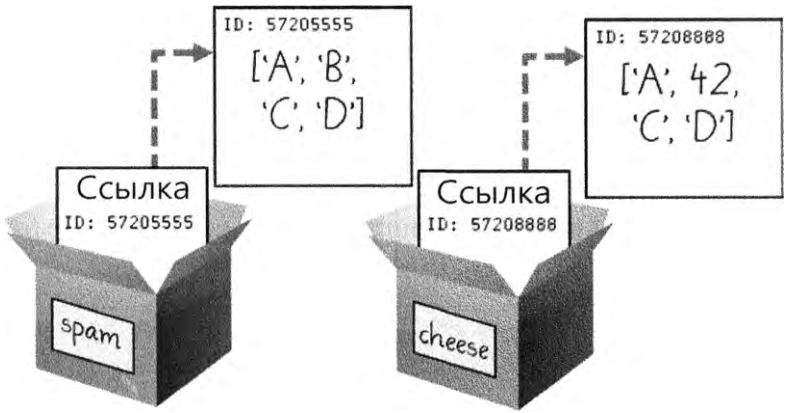


Рис. 4.7. Инструкция `cheese = copy.copy(spam)` создает второй список, который можно изменять независимо от первого

Если копируемый список содержит вложенные списки, то в этом случае вместо функции `copy.copy()` следует использовать функцию `copy.deepcopy()`. Она копирует всю структуру вложенных списков.

Короткая программа: игра "Жизнь"

Игра "Жизнь" – пример *клеточного автомата*: это набор правил, управляющих поведением поля, которое состоит из отдельных клеток. На практике получается симпатичная анимация, иллюстрирующая ход игры. Каждый шаг можно изобразить на разграфленной бумаге, используя квадраты в качестве клеток. Заполненная клетка будет "живой", а пустая – "мертвой". Если у живой клетки есть два или три живых соседа, то она продолжает жить на следующем шаге. Если у мертвой клетки ровно три живых соседа, то на следующем шаге она оживает. Все остальные клетки на следующем шаге умирают (или остаются мертвыми). Пример развития игры представлен на рис. 4.8.

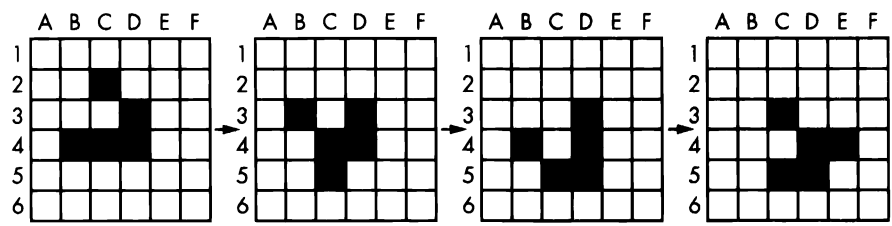


Рис. 4.8. Четыре шага симуляции игры "Жизнь"

Несмотря на простоту правил, в процессе игры возникает много удивительно разнообразных вариантов поведения. Шаблоны фигур в игре "Жизнь" могут перемещаться, самовоспроизводиться и даже порождать

другие фигуры. Но в основе всего этого сложного поведения лежит очень простая концепция.

Для формирования двухмерного игрового поля можно использовать список списков. Внутренний список представляет столбец квадратов; он хранит строку '#' для живых клеток и ' ' (пробел) — для мертвых. Введите в файловом редакторе приведенный ниже код и сохраните его в файле *conway.py*. Даже если вы не вполне понимаете, как работает программа, — не страшно. Я постараюсь объяснить основные моменты.

```
# Игра "Жизнь"
import random, time, copy
WIDTH = 60
HEIGHT = 20

# Создание списка списков для клеток
nextCells = []
for x in range(WIDTH):
    column = []    # создание нового столбца
    for y in range(HEIGHT):
        if random.randint(0, 1) == 0:
            column.append('#')    # добавление живой клетки
        else:
            column.append(' ')    # добавление мертвой клетки
    nextCells.append(column)    # переменная nextCells содержит
                                # список столбцов

while True:    # основной цикл программы
    print('\n\n\n\n\n')    # отделим каждый шаг с помощью
                           # символов новой строки
    currentCells = copy.deepcopy(nextCells)

    # Вывод текущих клеток на экран
    for y in range(HEIGHT):
        for x in range(WIDTH):
            print(currentCells[x][y], end='')    # вывод решетки
                                                 # или пробела
        print()    # вывод символа новой строки в конце

    # Вычисление клеток на следующем шаге
    # на основе клеток текущего шага
    for x in range(WIDTH):
        for y in range(HEIGHT):
            # Получение соседних координат

            # Выражение '% WIDTH' гарантирует, что значение
            # leftCoord всегда находится между 0 и WIDTH - 1
            leftCoord  = (x - 1) % WIDTH
            rightCoord = (x + 1) % WIDTH
            aboveCoord = (y - 1) % HEIGHT
            belowCoord = (y + 1) % HEIGHT
```

```
            # Вычисление количества живых соседних клеток
            numNeighbors = 0
            if currentCells[leftCoord][aboveCoord] == '#':
                numNeighbors += 1    # жива соседняя клетка
                                     # слева сверху
            if currentCells[x][aboveCoord] == '#':
                numNeighbors += 1    # жива соседняя клетка сверху
            if currentCells[rightCoord][aboveCoord] == '#':
                numNeighbors += 1    # жива соседняя клетка
                                     # справа сверху
            if currentCells[leftCoord][y] == '#':
                numNeighbors += 1    # жива соседняя клетка слева
            if currentCells[rightCoord][y] == '#':
                numNeighbors += 1    # жива соседняя клетка справа
            if currentCells[leftCoord][belowCoord] == '#':
                numNeighbors += 1    # жива соседняя клетка
                                     # слева снизу
            if currentCells[x][belowCoord] == '#':
                numNeighbors += 1    # жива соседняя клетка снизу
            if currentCells[rightCoord][belowCoord] == '#':
                numNeighbors += 1    # жива соседняя клетка
                                     # справа снизу

            # Изменение клетки на основе правил игры "Жизнь"
            if currentCells[x][y] == '#' and (numNeighbors == 2 or
                                              numNeighbors == 3):
                # Живые клетки с двумя или тремя живыми
                # соседями остаются живыми
                nextCells[x][y] = '#'
            elif currentCells[x][y] == ' ' and numNeighbors == 3:
                # Мертвые клетки с тремя живыми соседями оживают
                nextCells[x][y] = '#'
            else:
                # Все остальные клетки умирают или остаются мертвыми
                nextCells[x][y] = ' '
    time.sleep(1)    # добавим секундную паузу,
                     # чтобы уменьшить мерцание
```

Рассмотрим этот код построчно.

```
# Игра "Жизнь"
import random, time, copy
WIDTH = 60
HEIGHT = 20
```

Сначала импортируются модули, содержащие нужные нам функции, а именно `random.randint()`, `time.sleep()` **и** `copy.deepcopy()`.

```
# Создание списка списков для клеток
nextCells = []
for x in range(WIDTH):
```

```
    column = []    # создание нового столбца
    for y in range(HEIGHT):
        if random.randint(0, 1) == 0:
            column.append('#')     # добавление живой клетки
        else:
            column.append(' ')     # добавление мертвой клетки
    nextCells.append(column)     # переменная nextCells содержит
                                 # список столбцов
```

Самый первый шаг нашего клеточного автомата будет совершенно случайным. Нам нужно создать список списков для хранения строк `'#'` и `' '`, которые представляют живые и мертвые клетки; их место в списке списков отражает их положение на экране. Каждый из внутренних списков представляет собой столбец клеток. Вызов функции `random.randint(0, 1)` дает равный шанс каждой клетке стать изначально живой или мертвой.

Мы помещаем этот список списков в переменную `nextCells`, поскольку первым шагом в основном цикле программы будет копирование содержимого переменной `nextCells` в новую переменную `currentCells`. Для нашей структуры данных координаты *x* отсчитываются с нуля слева направо, а координаты *y* — с нуля сверху вниз. Поэтому элемент `nextCells[0][0]` соответствует клетке в левом верхнем углу поля, элемент `nextCells[1][0]` — клетке справа от нее, а элемент `nextCells[0][1]` — клетке под ней.

```
while True:    # основной цикл программы
    print('\n\n\n\n\n')    # отделим каждый шаг с помощью
                           # символов новой строки
    currentCells = copy.deepcopy(nextCells)
```

Каждая итерация основного цикла программы представляет собой один шаг нашего клеточного автомата. На каждом шаге мы будем копировать содержимое переменной `nextCells` в переменную `currentCells`, выводить содержимое переменной `currentCells` на экран, а затем использовать клетки списка `currentCells` для вычисления нового поколения клеток в списке `nextCells`.

```
    # Вывод текущих клеток на экран
    for y in range(HEIGHT):
        for x in range(WIDTH):
            print(currentCells[x][y], end='')    # вывод решетки
                                                 # или пробела
        print()    # вывод символа новой строки в конце
```

Вложенные циклы `for` гарантируют, что на экран выводится полный ряд ячеек, за которым следует символ новой строки. Мы повторяем это для каждой строки в списке `currentCells`.

```
# Вычисление клеток на следующем шаге
# на основе клеток текущего шага
for x in range(WIDTH):
    for y in range(HEIGHT):
        # Получение соседних координат

        # Выражение '% WIDTH' гарантирует, что значение
        # leftCoord всегда находится между 0 и WIDTH - 1
        leftCoord  = (x - 1) % WIDTH
        rightCoord = (x + 1) % WIDTH
        aboveCoord = (y - 1) % HEIGHT
        belowCoord = (y + 1) % HEIGHT
```

Затем нужно использовать два других вложенных цикла `for` для вычисления каждой клетки следующего шага. Состояние клетки (живая или мертвая) зависит от состояний соседних клеток, поэтому сначала вычисляются индексы клеток слева, справа, над и под текущими координатами `x` и `y`.

Оператор `%` (деление по модулю) позволяет учитывать границы поля. Левый сосед клетки в крайнем левом столбце с индексом `0` будет иметь индекс `0 - 1`, т.е. `-1`. Вместо этого мы переходим к крайнему правому столбцу, `59`, вычисляя выражение `(0 - 1) % WIDTH`. Поскольку значение `WIDTH` равно `60`, вычисление этого выражения дает `59`. Этот прием работает как по горизонтали, так и по вертикали.

```
        # Вычисление количества живых соседних клеток
        numNeighbors = 0
        if currentCells[leftCoord][aboveCoord] == '#':
            numNeighbors += 1    # жива соседняя клетка
                                 # слева сверху
        if currentCells[x][aboveCoord] == '#':
            numNeighbors += 1    # жива соседняя клетка сверху
        if currentCells[rightCoord][aboveCoord] == '#':
            numNeighbors += 1    # жива соседняя клетка
                                 # справа сверху
        if currentCells[leftCoord][y] == '#':
            numNeighbors += 1    # жива соседняя клетка слева
        if currentCells[rightCoord][y] == '#':
            numNeighbors += 1    # жива соседняя клетка справа
        if currentCells[leftCoord][belowCoord] == '#':
            numNeighbors += 1    # жива соседняя клетка
                                 # слева снизу
        if currentCells[x][belowCoord] == '#':
            numNeighbors += 1    # жива соседняя клетка снизу
        if currentCells[rightCoord][belowCoord] == '#':
            numNeighbors += 1    # жива соседняя клетка
                                 # справа снизу
```

Чтобы выяснить, должна ли клетка `nextCells[x][y]` стать живой или мертвой, нужно подсчитать количество живых соседних клеток у клетки `currentCells[x][y]`. Приведенная выше группа инструкций `if` проверяет каждую из восьми соседних клеток и прибавляет `1` к значению `numNeighbors` для каждой живой клетки.

```
            # Изменение клетки на основе правил игры "Жизнь"
            if currentCells[x][y] == '#' and (numNeighbors == 2 or
                                              numNeighbors == 3):
                # Живые клетки с двумя или тремя живыми
                # соседями остаются живыми
                nextCells[x][y] = '#'
            elif currentCells[x][y] == ' ' and numNeighbors == 3:
                # Мертвые клетки с тремя живыми соседями оживают
                nextCells[x][y] = '#'
            else:
                # Все остальные клетки умирают или остаются мертвыми
                nextCells[x][y] = ' '
    time.sleep(1)    # добавим секундную паузу,
                     # чтобы уменьшить мерцание
```

Теперь, когда мы знаем количество живых соседей клетки `currentCells[x][y]`, можно установить элемент `nextCells[x][y]` равным `'#'` или `' '`. После перебора всех возможных координат *x* и *y* программа делает секундную паузу с помощью вызова `time.sleep(1)`, а затем возвращается к началу основного цикла для выполнения очередной итерации.

В игре было обнаружено несколько шаблонов фигур с такими названиями, как "планер", "пропеллер" или "тяжелый космический корабль". Шаблон планера, изображенный на рис. 4.8, приводит к перемещению фигуры по диагонали на одну клетку через каждые четыре шага. Можно создать планер, заменив строку

```
if random.randint(0, 1) == 0:
```

строкой

```
if (x, y) in ((1, 0), (2, 1), (0, 2), (1, 2), (2, 2)):
```

Дополнительные сведения о необычных фигурах, создаваемых в игре "Жизнь", можно найти в Интернете. Другие короткие текстовые программы доступны по адресу `https://github.com/asweigart/pythonstdiogames`.

Резюме

Списки — очень полезный тип данных, поскольку они позволяют писать код, способный работать с изменяемым количеством значений в одной переменной. В последующих главах вы познакомитесь с примерами программ, позволяющих сделать то, что без использования списков было бы трудно или вообще невозможно осуществить.

Списки — изменяемый тип данных. В то же время кортежи и строки, несмотря на сходство со списками, относятся к неизменяемым типам данных. Переменная, содержащая кортеж или строку, может быть перезаписана путем присваивания ей нового значения в виде кортежа или строки, но это не то же самое, что изменение существующего значения исходного списка, как в случае методов `append()` и `remove()`.

В переменных хранятся не сами списки непосредственно, а *ссылки* на них. Это важное обстоятельство следует учитывать при копировании переменных или передаче списков в виде аргументов функций. Поскольку копируемое значение представляет собой ссылку на список, помните о том, что внесение изменений в список может повлиять на другие переменные в программе. Если вы хотите иметь возможность изменять список в функции таким образом, чтобы это не влияло на исходный список, копируйте список с помощью функции `copy()` или `deepcopy()`.

Контрольные вопросы

1. Что означает выражение `[]`?
2. Как присвоить значение `'hello'` третьему элементу списка, хранящегося в переменной `spam`? (Предполагается, что в переменной `spam` содержится список `[2, 4, 6, 8, 10]`.)

В следующих трех вопросах предполагается, что переменная `spam` содержит список `['a', 'b', 'c', 'd']`.

3. Чему равно выражение `spam[int('3' * 2) // 11]`?
4. Чему равно выражение `spam[-1]`?
5. Чему равно выражение `spam[:2]`?

В следующих трех вопросах предполагается, что переменная `bacon` содержит список `[3.14, 'кот', 11, 'кот', True]`.

6. Чему равно выражение `bacon.index('кот')`?
7. Как будет выглядеть список, хранящийся в переменной `bacon`, после вызова `bacon.append(99)`?
8. Как будет выглядеть список, хранящийся в переменной `bacon`, после вызова `bacon.remove('кот')`?

9. Какие операторы используются для конкатенации и репликации списков?
10. В чем разница между списковыми методами `append()` и `insert()`?
11. Назовите два способа удаления значений из списков.
12. Что общего у списков и строк?
13. Чем кортежи отличаются от списков?
14. Как записать кортеж, содержащий единственное целочисленное значение `42`?
15. Как преобразовать список в кортеж? Как преобразовать кортеж в список?
16. Переменная, "содержащая" список, в действительности не содержит непосредственно сам список. Что же тогда она содержит?
17. В чем разница между функциями `copy.copy()` и `copy.deepcopy()`?

Учебные проекты

В качестве практического задания напишите программы для предложенных ниже задач.

Запятая в качестве разделителя

Предположим, имеется следующий список:

```
spam = ['яблоки', 'бананы', 'тофу', 'коты']
```

Напишите функцию, получающую список в качестве аргумента и возвращающую строку, в которой все элементы списка разделены запятой и пробелом, а перед последним элементом вставлено слово `'и'`. Например, если передать функции показанный выше список `spam`, то вы должны получить строку `'яблоки, бананы, тофу и коты'`. Функция должна работать с любыми списками. Не забудьте проверить случай с пустым списком (`[]`).

Эксперименты с монетой

При выполнении этого упражнения мы попробуем провести эксперимент. Если вы подбросите монету 100 раз и запишите 'Р' для решки и 'О' — для орла, то получите список вида "ООООРРРРОО..." Если попросить человека придумать результаты 100 случайных подбрасываний монеты, то, скорее всего, он предложит примерное чередование орла и решки, например "ОРОРООРОРР". Такие результаты выглядят случайными с точки зрения человека, но они вовсе не являются случайными с математической точки зрения. Человек практически никогда не запишет ряд из шести решек

или шести орлов, хотя при случайных подбрасываниях монеты такое вполне может произойти. Люди плохо имитируют случайные результаты.

Напишите программу, которая позволит узнать, насколько часто серия из шести решек или шести орлов появляется в случайно сгенерированном списке. Программа должна состоять из двух частей: в первой части генерируется список случайно выбранных значений орлов и решек, а во второй части проверяется, есть ли в нем интересующая нас серия. Поместите весь этот код в цикл, который повторяет эксперимент 10 000 раз, чтобы мы могли выяснить частоту появления длинных серий. Подсказка: функция `random.randint(0, 1)` с равной вероятностью (50%) возвращает `0` или `1`.

Можно начать со следующего шаблона.

```
import random

numberOfStreaks = 0

for experimentNumber in range(10000):
    # Код, создающий список из 100 решек или орлов

    # Код, проверяющий наличие серии
    # из 6 орлов или решек подряд

print('Вероятность появления серии: %s%%' % (numberOfStreaks / 100))
```

Конечно, это лишь примерная оценка, но 10 000 — вполне приличный размер выборки. Знание основ статистики позволит получить точный ответ и избавит от необходимости писать программу, но современные программисты, увы, плохо разбираются в математике.

Символьная сетка

Предположим, имеется список списков, в котором каждое значение внутреннего списка представляет собой односимвольную строку, как в показанном ниже примере.

```
grid = [['.', '.', '.', '.', '.', '.'],
        ['.', 'O', 'O', '.', '.', '.'],
        ['O', 'O', 'O', 'O', '.', '.'],
        ['O', 'O', 'O', 'O', 'O', '.'],
        ['.', 'O', 'O', 'O', 'O', 'O'],
        ['O', 'O', 'O', 'O', 'O', '.'],
        ['O', 'O', 'O', 'O', '.', '.'],
        ['.', 'O', 'O', '.', '.', '.'],
        ['.', '.', '.', '.', '.', '.']]
```

Элемент `grid[x][y]` можно интерпретировать как пиксель с координатами `x` и `y` в составе "рисунка", нарисованного текстовыми символами. Точка начала координат `(0, 0)` находится в левом верхнем углу; координата `x` увеличивается слева направо, а координата `y` — сверху вниз.

Скопируйте предыдущее значение `grid` и напишите код, который использует его для вывода следующего изображения.

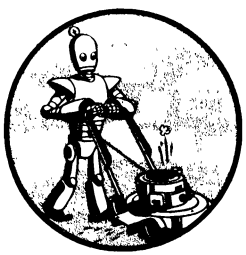
```
..OO.OO..
.OOOOOOO.
.OOOOOOO.
..OOOOO..
...OOO...
....O....
```

Подсказка: используйте цикл в цикле для вывода элементов `grid[0][0]`, `grid[1][0]`, `grid[2][0]` и т.д. вплоть до элемента `grid[8][0]`. Этим вы заполните первую строку, после чего необходимо вывести символ новой строки. Затем программа должна вывести элементы `grid[0][1]`, `grid[1][1]`, `grid[2][1]` и т.д. Последний элемент, который должна вывести программа, — `grid[8][5]`.

Кроме того, не забудьте передать функции `print()` именованный аргумент `end`, если хотите отменить автоматический вывод символа новой строки при каждом вызове функции.

5

СЛОВАРИ



В этой главе речь пойдет о словарях, которые позволяют организовать удобное хранение данных. Объединив словари со списками из предыдущей главы, мы создадим структуру данных для игры в “крестики-нолики”.

Что такое словарь

Подобно списку, *словарь* — это изменяемая коллекция значений. Однако в словарях, в отличие от списков, индексами могут быть не только целые числа, но и другие типы данных. Индексы в словарях называются *ключами*, а ключ вместе с соответствующим ему значением — *парой "ключ – значение"*.

В Python словари обозначаются фигурными скобками ({}). Введите в интерактивной оболочке следующую инструкцию:

```
>>> myCat = {'размер': 'толстый', 'цвет': 'серый', характер': 'шумный'}
```

Здесь переменной myCat присваивается словарь. Ключами в нем служат строки 'размер', 'цвет' и 'характер', а значениями — строки 'толстый', 'серый' и 'шумный' соответственно. Доступ к значениям осуществляется с помощью ключей.

```
>>> myCat['размер']
'толстый'
>>> 'У моего кота ' + myCat['цвет'] + ' мех.'
'У моего кота серый мех.'
```

Индексами в словарях, как и в списках, могут служить целые числа, однако их отсчет не обязательно должен начинаться с нуля. Кроме того, это могут быть любые числа.

```
>>> spam = {12345: 'Код замка', 42: 'Ответ'}
```

Сравнение словарей и списков

В отличие от списков, в словарях элементы не упорядочены. Первым элементом в списке spam был бы spam[0]. Однако к словарям понятие "первый элемент" неприменимо. Порядок элементов важен при проверке идентичности двух списков, но для словарей не имеет значения, в каком порядке в них были включены пары "ключ — значение". Введите в интерактивной оболочке следующие инструкции.

```
>>> spam = ['коты', 'собаки', 'лоси']
>>> bacon = ['собаки', 'лоси', 'коты']
>>> spam == bacon
False
>>> eggs = {'имя': 'Софи', 'вид': 'кот', 'возраст': '8'}
>>> ham = {'вид': 'кот', 'возраст': '8', 'имя': 'Софи'}
>>> eggs == ham
True
```

Поскольку словари не упорядочены, для них нельзя создавать срезы, в отличие от списков.

При попытке обратиться к ключу, отсутствующему в словаре, будет сгенерировано исключение `KeyError`, напоминающее исключение `IndexError`, которое возникает при выходе за пределы допустимого диапазона индексов в списке. Введите в интерактивной оболочке следующие инструкции, и вы получите сообщение об ошибке, поскольку в словаре нет ключа `'цвет'`.

```
>>> spam = {'имя': 'Софи', 'возраст': 7}
>>> spam['цвет']
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    spam['цвет']
KeyError: 'цвет'
```

Несмотря на то что словари не упорядочены, возможность извлечь произвольное значение по его ключу делает их очень удобными структурами. Предположим, в программе необходимо хранить данные о днях рождения ваших друзей. Для этой цели вполне подойдет словарь, в котором ключами будут имена друзей, а значениями — даты их рождения. Откройте в файловом редакторе новое окно, введите в нем следующий код и сохраните его в файле *birthdays.py*.

```
❶ birthdays = {'Алиса': 'Апр 1', 'Боб': 'Дек 12',
               'Кэрол': 'Мар 4'}

  while True:
      print('Введите имя (<Enter> для выхода):')
      name = input()
      if name == '':
          break

❷     if name in birthdays:
❸         print(name + ': день рождения - ' + birthdays[name])
      else:
          print('Я не знаю, когда день рождения у ' + name)
          print('Когда день рождения у этого человека?')
          bday = input()
❹         birthdays[name] = bday
          print('Обновлена информация о днях рождения.')
```

Выполнение авторского варианта этой программы можно просмотреть на сайте `https://autbor.com/bdaydb`. Первоначальный словарь сохраняется в переменной `birthdays` ❶. Проверить, содержится ли введенное имя в качестве ключа в словаре, можно с помощью оператора `in` ❷, точно так же, как и в случае списков. Если имя есть в словаре, то доступ к связанному

с ним значению осуществляется посредством квадратных скобок ❸. Если же имя отсутствует в словаре, то его можно добавить, используя тот же самый синтаксис квадратных скобок в сочетании с оператором присваивания ❹.

Результаты работы программы будут примерно такими.

```
Введите имя (<Enter> для выхода):
Алиса
Алиса: день рождения - Апр 1
Введите имя (<Enter> для выхода):
Ева
Я не знаю, когда день рождения у Ева
Когда день рождения у этого человека?
Дек 5
Обновлена информация о днях рождения.
Введите имя (<Enter> для выхода):
Ева
Ева: день рождения - Дек 5
Введите имя (<Enter> для выхода):
```

Разумеется, по завершении работы программы все введенные вами данные теряются. О том, как сохранить данные на жестком диске, будет рассказано в главе 9.

Методы keys(), values() и items()

Для работы со словарями предусмотрены методы `keys()`, `values()` и `items()`, которые возвращают соответственно ключи, значения и пары "ключ — значение". Возвращаемые этими методами коллекции не являются списками: их нельзя изменять, и у них нет метода `append()`. В то же время эти типы данных (`dict_keys`, `dict_values` и `dict_items` соответственно) *можно* использовать в циклах `for`. Введите в интерактивной оболочке следующий код.

```
>>> spam = {'цвет': 'красный', 'возраст': 42}
>>> for v in spam.values():
...     print(v)

красный
42
```

Здесь цикл `for` проходит по всем значениям, содержащимся в словаре `spam`. То же самое можно сделать для ключей и пар "ключ — значение".

Упорядоченные словари в Python 3.7

В Python 3.7 и более поздних версий словари запоминают порядок вставки пар "ключ — значение", оставаясь при этом неупорядоченными. Например, обратите внимание на то, что порядок элементов в списках, составленных из словарей `eggs` и `ham`, соответствует порядку ввода значений.

```
>>> eggs = {'имя': 'Софи', 'вид': 'кот', 'возраст': '8'}
>>> list(eggs)
['имя', 'вид', 'возраст']
>>> ham = {'вид': 'кот', 'возраст': '8', 'имя': 'Софи'}
>>> list(ham)
['вид', 'возраст', 'имя']
```

Сами словари при этом не упорядочены, и к их элементам нельзя получить доступ, используя целочисленные индексы наподобие `eggs[0]` или `ham[2]`. Но рассчитывать на такое поведение не стоит, поскольку словари в старых версиях Python не помнят порядок вставки пар "ключ — значение". Например, если запустить следующий код в Python 3.5, то порядок элементов в списке не будет соответствовать порядку вставки пар "ключ — значение" в словарь.

```
>>> spam = {}
>>> spam['первый ключ'] = 'значение'
>>> spam['второй ключ'] = 'значение'
>>> spam['третий ключ'] = 'значение'
>>> list(spam)
['первый ключ', 'третий ключ', 'второй ключ']
```

```
>>> for k in spam.keys():
...     print(k)

цвет
возраст
>>> for i in spam.items():
...     print(i)

('цвет', 'красный')
('возраст', 42)
```

Используя методы `keys()`, `values()` и `items()`, можно организовать перебор ключей, значений и пар "ключ — значение" соответственно. Обратите внимание на то, что значения типа `dict_items`, возвращаемые методом `items()`, представляют собой кортежи, образуемые ключами и связанными с ними значениями словаря.

Если необходимо получить результат в виде списка, передайте функции `list()` значение, возвращаемое любым из этих трех методов. Например, введите в интерактивной оболочке следующие инструкции.

```
>>> spam = {'цвет': 'красный', 'возраст': 42}
>>> spam.keys()
dict_keys(['цвет', 'возраст'])
>>> list(spam.keys())
['цвет', 'возраст']
```

В строке `list(spam.keys())` значение типа `dict_keys`, возвращаемое функцией `keys()`, передается функции `list()`, которая формирует список `['цвет', 'возраст']`.

Кроме того, можно воспользоваться групповым присваиванием в цикле `for` для присваивания ключей и связанных с ними значений отдельным переменным. Введите в интерактивной оболочке следующий код.

```
>>> spam = {'цвет': 'красный', 'возраст': 42}
>>> for k, v in spam.items():
...     print('Ключ: ' + k + ', значение: ' + str(v))

Ключ: возраст, значение: 42
Ключ: цвет, значение: красный
```

Проверка наличия ключа или значения в словаре

Как вам уже известно из предыдущей главы, операторы `in` и `not in` позволяют проверить, содержится ли указанное значение в списке. Эти же операторы можно использовать и для того, чтобы проверить, содержится ли в словаре заданный ключ или заданное значение. Введите в интерактивной оболочке следующие инструкции.

```
>>> spam = {'имя': 'Софи', 'возраст': 7}
>>> 'имя' in spam.keys()
True
>>> 'Софи' in spam.values()
True
>>> 'цвет' in spam.keys()
False
>>> 'цвет' not in spam.keys()
True
>>> 'цвет' in spam
False
```

Выражение `'цвет' in spam` представляет собой сокращенную запись выражения `'цвет' in spam.keys()`. Это общее правило: если нужно

проверить, является ли данное значение ключом в словаре, то после ключевого слова in (или not in) достаточно указать только имя словаря.

Метод get()

Было бы слишком утомительно каждый раз проверять наличие ключа в словаре перед обращением к нему. К счастью, для словарей предусмотрен метод get(), имеющий два аргумента: ключ извлекаемого значения и значение по умолчанию, возвращаемое в случае отсутствия данного ключа в словаре. Введите в интерактивной оболочке следующие инструкции.

```
>>> picnicItems = {'яблоки': 5, 'чашки': 2}
>>> 'Я несу ' + str(picnicItems.get('чашки', 0)) + ' чашки.'
'Я несу 2 чашки.'
>>> 'Я несу ' + str(picnicItems.get('яйца', 0)) + ' яйца.'
'Я несу 0 яйца.'
```

Поскольку в словаре picnicItems нет ключа 'яйца', метод get() возвращает заданное по умолчанию значение 0. Если не использовать метод get(), то будет сгенерирована ошибка.

```
>>> picnicItems = {'яблоки': 5, 'чашки': 2}
>>> 'Я несу ' + str(picnicItems['яйца']) + ' яйца.'
Traceback (most recent call last):
  File "<pyshell#34>", line 1, in <module>
    'Я несу ' + str(picnicItems['яйца']) + ' яйца.'
KeyError: 'яйца'
```

Метод setdefault()

Зачастую нужно установить значение для определенного ключа лишь в том случае, если этому ключу еще не присвоено значение. Рассмотрим пример.

```
spam = {'имя': 'Питер', 'возраст': 5}
if 'цвет' not in spam:
    spam['цвет'] = 'черный'
```

С помощью метода setdefault() то же самое можно сделать в одной строке кода. У данного метода два аргумента. Первый из них — это проверяемый ключ, а второй — значение, устанавливаемое для ключа в случае его отсутствия в словаре. Если же ключ существует, метод setdefault() возвращает его значение. Введите в интерактивной оболочке следующие инструкции.

```
>>> spam = {'имя': 'Питер', 'возраст': 5}
>>> spam.setdefault('цвет', 'черный')
'черный'
>>> spam
{'цвет': 'черный', 'возраст': 5, 'имя': 'Питер'}
>>> spam.setdefault('цвет', 'белый')
'черный'
>>> spam
{'цвет': 'черный', 'возраст': 5, 'имя': 'Питер'}
```

При первом вызове метод `setdefault()` изменяет словарь, который теперь выглядит так: `{'цвет': 'черный', 'возраст': 5, 'имя': 'Питер'}`. Метод возвращает значение `'черный'`, которое было назначено ключу `'цвет'`. При втором вызове — `spam.setdefault('цвет', 'белый')` — значение ключа *не* меняется, поскольку в словаре уже есть ключ `'цвет'`.

Метод `setdefault()` удобно использовать в ситуациях, когда требуется гарантировать наличие ключа. Ниже приведена короткая программа, которая подсчитывает, сколько раз в строке встречается каждая из входящих в нее букв. Откройте в файловом редакторе новое окно, введите в нем следующий код и сохраните его в файле *characterCount.py*.

```
message = 'It was a bright cold day in April, and the clocks were
striking thirteen.'
count = {}

for character in message:
❶    count.setdefault(character, 0)
❷    count[character] = count[character] + 1

print(count)
```

Выполнение этой программы можно просмотреть на сайте `https://autbor.com/setdefault`. Программа циклически перебирает все символы строки, содержащейся в переменной `message`, и подсчитывает, как часто встречается каждый из них. Вызов метода `setdefault()` ❶ гарантирует существование ключа в словаре (значение которого по умолчанию равно `0`), поэтому при выполнении инструкции ❷ `count[character] = count[character] + 1` ошибка `KeyError` возникать не будет. Запустив программу, вы получите следующий результат.

```
{' ': 13, ',': 1, '.': 1, 'A': 1, 'I': 1, 'a': 4, 'c': 3, 'b': 1,
 'e': 5, 'd': 3, 'g': 2, 'i': 6, 'h': 3, 'k': 2, 'l': 3, 'o': 2,
 'n': 4, 'p': 1, 's': 3, 'r': 5, 't': 6, 'w': 2, 'y': 1}
```

Как и следовало ожидать, буква 'c' в нижнем регистре встречается 3 раза, пробел – 13 раз, а буква 'A' в верхнем регистре – один раз. Эта программа будет работать со строкой любой длины, даже если в переменной message хранится строка, содержащая миллионы символов!

Красивый вывод

Импортировав модуль pprint, вы получите доступ к функциям pprint() и pformat(), которые обеспечивают красивый вывод значений словаря. Это может понадобиться, если нужно расширить возможности функции print(). Измените предыдущую программу *characterCount.py*, как показано ниже, и сохраните ее в файле *prettyCharacterCount.py*.

```
import pprint
message = 'It was a bright cold day in April, and the clocks were
striking thirteen.'
count = {}

for character in message:
    count.setdefault(character, 0)
    count[character] = count[character] + 1

pprint.pprint(count)
```

Выполнение этой программы можно просмотреть на сайте https://autbor.com/pprint/. На этот раз результат выглядит намного аккуратнее, и к тому же он отсортирован по ключам.

```
{' ': 13,
 ',': 1,
 '.': 1,
 'A': 1,
 'I': 1,
 --Опущено--
 't': 6,
 'w': 2,
 'y': 1}
```

Функция pprint.pprint() особенно полезна в тех случаях, когда словарь содержит вложенные списки или словари.

Если нужно получить аккуратно оформленный текст в виде строки, а не выводить его на экран, то воспользуйтесь функцией pprint.pformat(). Следующие две инструкции эквивалентны.

```
pprint.pprint(someDictionaryValue)
print(pprint.pformat(someDictionaryValue))
```

Использование структур данных для моделирования реальных объектов

Возможность играть в шахматы с партнером, находящимся на другой стороне земного шара, существовала задолго до того, как появился Интернет. Каждый из игроков, сидя у себя дома за шахматной доской, сообщал партнеру о сделанных ходах по почте. Для этого нужен был какой-то способ записи шахматных партий, который позволял бы однозначно описывать положение фигур на доске и их ходы.

В *алгебраической шахматной нотации* клетки шахматной доски обозначаются с использованием букв и цифр (рис. 5.1).

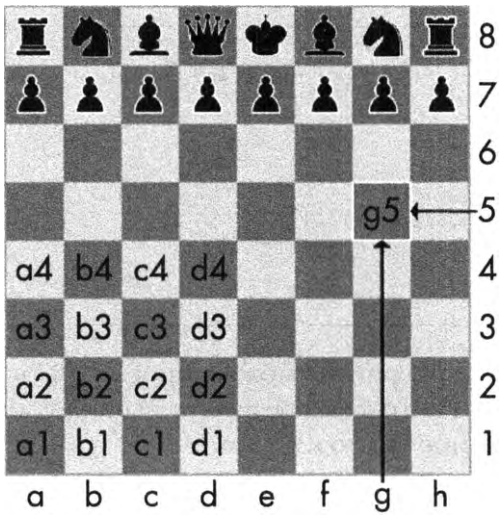


Рис. 5.1. Система координат на шахматной доске, в которой используется алгебраическая нотация

Шахматные фигуры обозначаются такими буквами: 'K' (king) – король, 'Q' (queen) – ферзь, 'R' (rook) – ладья, 'B' (bishop) – слон и 'N' (knight) – конь. Описание хода включает букву, соответствующую фигуре, которая делает ход, и координаты поля, куда ходит данная фигура. Запись пары таких ходов показывает, что происходит, когда каждый игрок делает свой ход (первый ход за белыми). Например, запись `2. Nf3 Nc6` означает, что на втором ходе белые поставили коня на поле f3, а черные – своего коня на поле c6.

Это далеко не полное описание системы записи шахматных партий, но для нас важен тот факт, что существует возможность однозначно описывать игру, даже не находясь за шахматной доской. Ваш оппонент вполне может находиться на другом конце земли! Если у вас хорошая память, вам даже не нужна физическая шахматная доска: вы сможете просто читать

ходы противника, которые он присылает вам по почте, и мысленно обновлять положения фигур на доске.

У компьютеров точно хорошая память. Современные программы позволяют хранить миллиарды строк наподобие `'2. Nf3 Nc6'`. Это позволяет компьютеру играть в шахматы без использования шахматной доски. Он моделирует данные для представления шахматной доски, а вы можете написать код, который работает с компьютерной моделью.

Вот тут-то нам на помощь и приходят списки и словари. Например, словарь `{'h1': 'bking', 'c6': 'wqueen', 'g2': 'bbishop', 'h5': 'bqueen', 'e3': 'wking'}` описывает положение на доске, показанное на рис. 5.2.

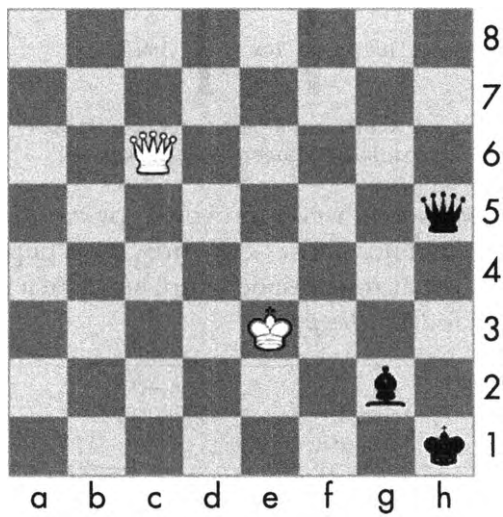


Рис. 5.2. Шахматная доска, смоделированная с помощью словаря `{'h1': 'bking', 'c6': 'wqueen', 'g2': 'bbishop', 'h5': 'bqueen', 'e3': 'wking'}`

Но в качестве примера мы смоделируем более простую, чем шахматы, игру: "крестики-нолики".

Поле для игры в "крестики-нолики"

Поле для игры в "крестики-нолики" напоминает увеличенный символ решетки (`#`) с девятью клетками, каждая из которых может быть пустой либо содержать крестик (`X`) или нолик (`O`). Чтобы представить клетки игрового поля с помощью словаря, можно назначить каждой из них строковый ключ (рис. 5.3).

Содержимое клеток можно описывать с помощью строк `'X'`, `'O'` и `' '` (пробел). Таким образом, всего нам понадобится девять строковых значений. Чтобы связать эти значения с клетками игрового поля, мы используем словарь. Правую верхнюю клетку можно описать с помощью строки,

связанной с ключом 'top-R', левую нижнюю клетку — с помощью строки, связанной с ключом 'low-L', центральную клетку — с помощью строки, связанной с ключом 'mid-M', и т.д.

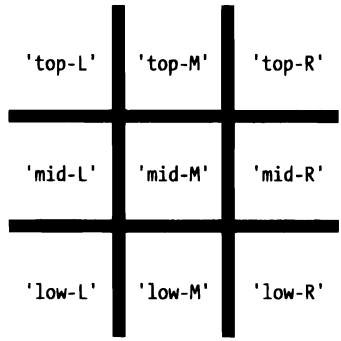


Рис. 5.3. Клетки для игры в "крестики-нолики" с указанием соответствующих ключей

Такой словарь представляет собой структуру данных, моделирующую поле для игры в "крестики-нолики". Сохраним его в переменной theBoard. Откройте в файловом редакторе новое окно, введите в нем следующий код и сохраните его в файле *ticTacToe.py*.

```
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ',
            'mid-L': ' ', 'mid-M': ' ', 'mid-R': ' ',
            'low-L': ' ', 'low-M': ' ', 'low-R': ' '}
```

Структуре данных, сохраненной в переменной theBoard, соответствует состояние поля, представленное на рис. 5.4.

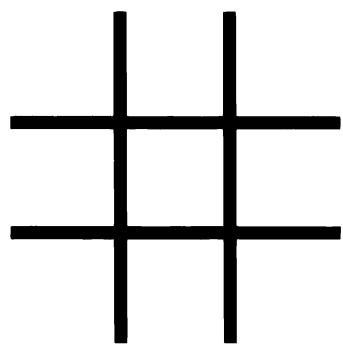


Рис. 5.4. Пустое поле для игры в "крестики-нолики"

Поскольку каждому ключу в словаре theBoard соответствует строка в виде одиночного пробела, такая структура описывает пустое поле. Если

игрок X своим первым ходом выберет центральную клетку, то новое состояние поля будет описываться следующим словарем.

```
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ',
            'mid-L': ' ', 'mid-M': 'X', 'mid-R': ' ',
            'low-L': ' ', 'low-M': ' ', 'low-R': ' '}
```

Теперь структуре данных, сохраненной в переменной `theBoard`, соответствует игровое поле, представленное на рис. 5.5.

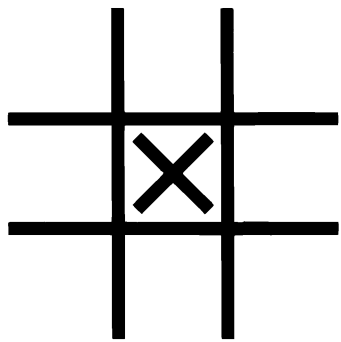


Рис. 5.5. Игровое поле после первого хода

Ниже показана структура данных, которая соответствует победе игрока O, поставившего три нолика в верхние клетки.

```
theBoard = {'top-L': 'O', 'top-M': 'O', 'top-R': 'O',
            'mid-L': 'X', 'mid-M': 'X', 'mid-R': ' ',
            'low-L': ' ', 'low-M': ' ', 'low-R': 'X'}
```

Этой структуре данных соответствует поле, представленное на рис. 5.6.

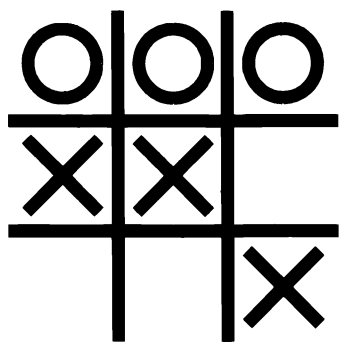


Рис. 5.6. Победил игрок O

Разумеется, игроки могут видеть только то, что выводится на экран, а вовсе не содержимое переменных. Напишем функцию, отображающую содержимое словаря на экране. Внесите в файл *ticTacToe.py* следующие изменения (новый код выделен полужирным шрифтом).

```
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ',
            'mid-L': ' ', 'mid-M': ' ', 'mid-R': ' ',
            'low-L': ' ', 'low-M': ' ', 'low-R': ' '}

def printBoard(board):
    print(board['top-L'] + '|' + board['top-M'] + '|' + \
          board['top-R'])
    print('-+-+-')
    print(board['mid-L'] + '|' + board['mid-M'] + '|' + \
          board['mid-R'])
    print('-+-+-')
    print(board['low-L'] + '|' + board['low-M'] + '|' + \
          board['low-R'])
printBoard(theBoard)
```

Выполнение этой программы можно просмотреть на сайте `https://autbor.com/tictactoe1/`. Функция `printBoard()` выведет на экран пустое игровое поле.

```
 | |
-+-+-
 | |
-+-+-
 | |
```

Функция `printBoard()` способна обрабатывать любую структуру "крестиков-ноликов", которую вы ей передадите. Внесите в код следующие изменения.

```
theBoard = {'top-L': 'O', 'top-M': 'O', 'top-R': 'O',
            'mid-L': 'X', 'mid-M': 'X', 'mid-R': ' ',
            'low-L': ' ', 'low-M': ' ', 'low-R': 'X'}

def printBoard(board):
    print(board['top-L'] + '|' + board['top-M'] + '|' + \
          board['top-R'])
    print('-+-+-')
    print(board['mid-L'] + '|' + board['mid-M'] + '|' + \
          board['mid-R'])
    print('-+-+-')
    print(board['low-L'] + '|' + board['low-M'] + '|' + \
          board['low-R'])
printBoard(theBoard)
```

Выполнение этой программы можно просмотреть на сайте https://autbor.com/tictactoe2/. В данном случае на экран будет выведено следующее состояние игрового поля.

```
O|O|O
-+-+-
X|X|
-+-+-
 | |X
```

Итак, у нас есть структура данных, описывающая игровое поле, и функция printBoard(), способная отображать эту структуру на экране. Таким образом, наша программа корректно моделирует поле для игры в "крестики-нолики". Структуру данных можно было бы организовать и по-другому (например, использовать ключи наподобие 'TOP-LEFT' вместо 'top-L'), но важно, чтобы она корректно распознавалась функцией printBoard().

В частности, функция printBoard() ожидает, что ей будет передан словарь с ключами для всех девяти клеток. Если, допустим, в переданном словаре отсутствует ключ 'mid-L', то программа работать не будет.

```
O|O|O
-+-+-
Traceback (most recent call last):
  File "ticTacToe.py", line 11, in <module>
    printBoard(theBoard)
  File "ticTacToe.py", line 8, in printBoard
    print(board['mid-L'] + '|' + board['mid-M'] + '|' + \
          board['mid-R'])
KeyError: 'mid-L'
```

Теперь добавим код, который позволяет игрокам делать ходы. Внесите показанные ниже изменения в программу *ticTacToe.py*.

```
  theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ',
              'mid-L': ' ', 'mid-M': ' ', 'mid-R': ' ',
              'low-L': ' ', 'low-M': ' ', 'low-R': ' '}

  def printBoard(board):
      print(board['top-L'] + '|' + board['top-M'] + '|' + \
            board['top-R'])
      print('-+-+-')
      print(board['mid-L'] + '|' + board['mid-M'] + '|' + \
            board['mid-R'])
      print('-+-+-')
      print(board['low-L'] + '|' + board['low-M'] + '|' + \
            board['low-R'])
  turn = 'X'
  for i in range(9):
```

```
❶     printBoard(theBoard)
      print('Ход для ' + turn + '. Куда ходить?')
❷     move = input()
❸     theBoard[move] = turn
❹     if turn == 'X':
          turn = 'O'
      else:
          turn = 'X'
printBoard(theBoard)
```

Выполнение авторского варианта этой программы можно просмотреть на сайте `https://autbor.com/tictactoe3/`. Теперь программа выводит состояние игрового поля перед началом каждого хода ❶, запрашивает ход текущего игрока ❷, соответствующим образом обновляет игровое поле ❸, а затем передает право хода другому игроку ❹, прежде чем перейти к следующему ходу.

Результаты работы программы будут выглядеть примерно так.

```
 | |
-+-+-
 | |
-+-+-
 | |
Ход для X. Куда ходить?
mid-M
 | |
-+-+-
 |X|
-+-+-
 | |

-- Опущено --

O|O|X
-+-+-
X|X|O
-+-+-
O| |X
Ход для X. Куда ходить?
low-M
O|O|X
-+-+-
X|X|O
-+-+-
O|X|X
```

Это незавершенный вариант программы, поскольку в ней, например, вообще не определяется, победил ли игрок. Но и такого кода вполне достаточно для того, чтобы понять, как применять структуры данных.

Примечание

Любознательные читатели смогут ознакомиться с полной версией программы для игры в "крестики-нолики" по адресу `http://inventwithpython.com/chapter10.html`.

Вложенные словари и списки

Моделировать игру в "крестики-нолики" относительно легко: для описания игрового поля требуется всего один словарь с девятью парами "ключ — значение". В более сложных играх могут потребоваться словари и списки, содержащие другие списки и словари. Списки удобны для хранения упорядоченных последовательностей, а словари — для сопоставления значений с ключами. Ниже приведена программа, в которой используются вложенные словари для описания того, что приносит с собой каждый из гостей, приглашенных на пикник. Функция `totalBrought()` считывает эту структуру данных и вычисляет общее количество предметов, принесенных гостями.

```
allGuests = {'Алиса': {'яблоки': 5, 'конфеты': 12},
             'Боб':   {'бутерброды': 3, 'яблоки': 2},
             'Кэрол': {'чашки': 3, 'пироги': 1}}

def totalBrought(guests, item):
    numBrought = 0
❶   for k, v in guests.items():
❷       numBrought = numBrought + v.get(item, 0)
    return numBrought

print('Количество принесенных предметов:')
print(' - Яблоки      ' + str(totalBrought(allGuests, 'яблоки')))
print(' - Чашки       ' + str(totalBrought(allGuests, 'чашки')))
print(' - Булочки     ' + str(totalBrought(allGuests, 'булочки')))
print(' - Бутерброды  ' + str(totalBrought(allGuests, 'бутерброды')))
print(' - Пироги      ' + str(totalBrought(allGuests, 'пироги')))
```

Выполнение авторского варианта этой программы можно просмотреть на сайте `https://autbor.com/guestpicnic/`. В функции `totalBrought()` цикл `for` проходит по всем парам "ключ — значение", хранящимся в переменной `guests` ❶. В цикле переменной `k` присваивается строка с именем гостя, а переменной `v` — словарь с информацией о том, что принес этот гость. Если в словаре имеется ключ, соответствующий параметру `item`, то его значение (количество принесенных предметов) прибавляется к переменной `numBrought` ❷. В случае отсутствия ключа метод `get()` возвращает `0`.

Результаты работы программы будут выглядеть так.

```
Количество принесенных предметов:
- Яблоки       7
- Чашки        3
- Булочки      0
- Бутерброды   3
- Пироги       1
```

Может показаться, что эта модель слишком простая, чтобы обременять себя написанием программы для нее. Однако задумайтесь над тем, что функция `totalBrought()` способна с легкостью обрабатывать словари, включающие *тысячи* гостей с тысячами различных предметов. Наличие функции для такой огромной структуры данных сохранит вам массу времени.

С помощью структур данных можно создавать модели любой сложности, в зависимости от назначения программы. Если вы только начинаете заниматься программированием, не стоит слишком беспокоиться о выборе "наилучшей" модели. Со временем вы научитесь оптимизировать модели хранения данных, а пока главное, чтобы выбранная модель соответствовала тем задачам, которые решает программа.

Резюме

Списки и словари — это изменяемые типы данных, которые могут содержать множество значений, включая другие списки и словари. Словари удобны тем, что позволяют сопоставлять одни элементы (ключи) с другими (значения), в отличие от списков, которые просто содержат упорядоченные последовательности значений. Доступ к элементам словаря осуществляется посредством квадратных скобок, как и в случае списков. Но вместо целочисленных индексов в словарях допускается использование ключей самых разных типов: целых и вещественных чисел, строк и даже кортежей. С помощью таких структур данных можно моделировать реальные объекты, как было показано на примере игры в "крестики-нолики".

Контрольные вопросы

1. Как выглядит пустой словарь?
2. Как выглядит словарь, содержащий ключ `'foo'` со значением `42`?
3. В чем основная разница между словарем и списком.
4. Что произойдет при попытке получить доступ к элементу `spam['foo']`, если `spam` — это словарь вида `{'bar': 100}`?
5. Если в переменной `spam` хранится словарь, то в чем разница между выражениями `'кот' in spam` и `'кот' in spam.keys()`?

6. Если в переменной `spam` хранится словарь, то в чем разница между выражениями `'кот' in spam` и `'кот' in spam.values()`?
7. Как можно короче записать приведенный ниже код?

```
if 'цвет' not in spam:
    spam['цвет'] = 'черный'
```

8. Какую функцию можно использовать для "красивого вывода" значений словаря?

Учебные проекты

В качестве практического задания напишите программы для предложенных ниже задач.

Валидатор словаря для игры в шахматы

В главе рассматривался словарь `{'1h': 'bking', '6c': 'wqueen', '2g': 'bbishop', '5h': 'bqueen', '3e': 'wking'}` для представления шахматной доски. Напишите функцию `isValidChessBoard()`, которая получает словарь в качестве аргумента и возвращает `True` или `False` в зависимости от того, корректна ли позиция на доске.

В корректной позиции на доске имеется ровно один черный и один белый король. Каждый игрок может иметь не более 16 фигур и не более 8 пешек, а все фигуры должны находиться в допустимом пространстве координат от `'1a'` до `'8h'`. Другими словами, фигура не может находиться, к примеру, в клетке `'9z'`. Названия фигур начинаются с `'w'` или `'b'`, что соответствует белому или черному цвету. Далее указывается обозначение самой фигуры: `'pawn'` (пешка), `'knight'` (конь), `'bishop'` (слон), `'rook'` (ладья), `'queen'` (ферзь) или `'king'` (король).

Инвентарь приключенческой игры

Предположим, вы разрабатываете приключенческую видеоигру. Структурой данных для инвентаря игрока должен быть словарь, в котором ключи — это строки, описывающие инвентарь, а значения — количество имеющихся у игрока единиц данного инвентаря. Например, словарь может выглядеть так.

```
{'веревка': 1, 'факел': 6, 'золотая монета': 42, 'кинжал': 1,
 'стрела': 12}
```

Это означает, что у игрока есть одна веревка, 6 факелов, 42 золотые монеты, один кинжал и 12 стрел.

Напишите функцию `displayInventory()`, которая получает в качестве аргумента инвентарный словарь и отображает его в следующем виде.

```
Инвентарь:
  веревка - 1
  золотая монета - 42
  кинжал - 1
  стрела - 12
  факел - 6
Всего элементов: 62
```

Подсказка: для просмотра всех ключей словаря можно использовать цикл `for`.

```
# inventory.py
stuff = {'веревка': 1, 'факел': 6, 'золотая монета': 42,
         'кинжал': 1, 'стрела': 12}

def displayInventory(inventory):
    print("Инвентарь:")
    item_total = 0
    for k, v in inventory.items():
        # Вставьте сюда свой код
    print("Всего элементов: " + str(item_total))

displayInventory(stuff)
```

Функция добавления списка в словарь для приключенческой игры

Предположим, что трофеи за победу над драконом представлены в виде следующего списка.

```
dragonLoot = ['золотая монета', 'кинжал', 'золотая монета',
              'золотая монета', 'рубин']
```

Напишите функцию `addToInventory(inventory, addedItems)`, в которой параметр `inventory` — это словарь, представляющий инвентарь игрока (как в предыдущем проекте), а параметр `addedItems` — это список наподобие `dragonLoot`. Функция должна возвращать словарь, представляющий обновленный инвентарь. Обратите внимание на то, что в списке `addedItems` один и тот же элемент может встречаться несколько раз. Код программы может выглядеть примерно так.

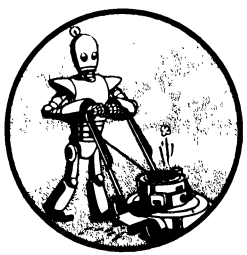
```
def addToInventory(inventory, addedItems):
    # Вставьте сюда свой код

inv = {'золотая монета': 42, 'веревка': 1}
dragonLoot = ['золотая монета', 'кинжал', 'золотая монета',
              'золотая монета', 'рубин']
inv = addToInventory(inv, dragonLoot)
displayInventory(inv)
```

Функция `displayInventory()`, рассмотренная в предыдущем разделе, должна вывести следующее.

```
Инвентарь:
  веревка - 1
  золотая монета - 45
  кинжал - 1
  рубин - 1
Всего элементов: 48
```

6

СТРОКИ



Чаще всего в программах приходится работать именно с текстом. Вы уже знаете, как конкатенировать две строки с помощью оператора +, но возможности Python гораздо шире. Можно извлекать фрагменты строк, добавлять и удалять пробелы, преобразовывать буквы из нижнего регистра в верхний и обратно, а также проверять форматирование строк. Можно даже обратиться к буферу обмена для копирования и вставки текста.

В этой главе будет показано, как работать со строками, после чего мы реализуем два программных проекта: простой буфер обмена, способный хранить множество текстовых строк, и программа для автоматического форматирования текста.

Работа со строками

Рассмотрим основные способы записи строк, а также их вывода на экран и получения доступа к ним в коде Python.

Строковые литералы

Строки в Python начинаются и заканчиваются одинарными кавычками. Но как быть, если кавычка стоит в самой строке? Вводить строки вида `'That is Alice's cat.'` нельзя, поскольку Python интерпретирует апостроф как закрывающую кавычку, и оставшаяся часть текста (`s cat.'`) будет воспринята как недопустимый код. К счастью, есть разные способы ввода строк.

Двойные кавычки

Начало и конец строки можно обозначать не только одинарными, но и двойными кавычками. Преимущество двойных кавычек в том, что они позволяют трактовать одинарную кавычку как апостроф. Введите в интерактивной оболочке следующую инструкцию:

```
>>> spam = "That is Alice's cat."
```

Поскольку строка начинается с двойной кавычки, Python считает одинарную кавычку апострофом в составе строки и не помечает оставшийся текст как ошибочный. Если же в строке нужны как одинарные, так и двойные кавычки, то необходимо прибегнуть к экранированию символов.

Экранирование символов

Благодаря экранированию в строке можно использовать символы, вставить которые по-другому невозможно. *Экранированный символ* предваряется обратной косой чертой (\), за которой следует сам символ, добавляемый в строку. (Несмотря на то что экранированный символ состоит из двух частей, его рассматривают как одиночный символ.) Например, экранированная кавычка записывается так: `\'`. Ее можно использовать даже в строке, которая начинается и заканчивается одинарной кавычкой. Чтобы увидеть, как работают экранированные символы, введите в интерактивной оболочке следующую инструкцию:

```
>>> spam = 'Say hi to Bob\'s mother.'
```

Поскольку в слове `Bob\'s` апострофу предшествует обратная косая черта, Python знает, что в данном случае это не маркер конца строки. Экранированные символы `\'` и `\"` позволяют включать в строку соответственно одинарные и двойные кавычки.

Доступные экранированные символы перечислены в табл. 6.1.

Таблица 6.1. Экранированные символы

Экранированный символ	Отображаемый символ
`\'`	Одинарная кавычка (апостроф)
`\"`	Двойная кавычка
`\t`	Табуляция
`\n`	Новая строка (разрыв строки)
`\\`	Обратная косая черта

В качестве примера введите в интерактивной оболочке следующую инструкцию.

```
>>> print("Hello there!\nHow are you?\nI\'m doing fine.")
Hello there!
How are you?
I'm doing fine.
```

Необработанные строки

Поместив символ `r` перед открывающей кавычкой, вы помечаете строку как *необработанную*. В такой строке экранирование полностью игнорируется, поэтому на экран выводятся все символы обратной косой черты, которые встречаются в строке. Введите в интерактивной оболочке следующую инструкцию.

```
>>> print(r'That is Carol\'s cat.')
That is Carol\'s cat.
```

Поскольку это необработанная строка, Python считает все символы обратной косой черты ее частью, а не началом экранированного символа. Такая возможность удобна в тех случаях, когда вводятся строки, содержащие множество символов обратной косой черты, как, например, в файловых путях вида `r'C:\Users\Al\Desktop'` или в регулярных выражениях, описанных в следующей главе.

Многострочные текстовые блоки с тройными кавычками

Несмотря на то что в строку всегда можно добавить экранированный символ новой строки (\n), во многих случаях удобнее использовать многострочные блоки. В Python многострочный текст представляет собой группу строк, заключенных в тройные кавычки (три одинарные или три двойные). Любые кавычки, табуляции или символы новой строки в блоке, ограниченном тройными кавычками, считаются частью строки. Правила Python, регламентирующие форматирование блоков кода с помощью отступов, в отношении многострочных блоков не действуют.

Введите в файловом редакторе следующий код и сохраните его в файле *catnapping.py*.

```
print('''Dear Alice,

Eve's cat has been arrested for catnapping, cat burglary,
and extortion.

Sincerely,
Bob''')
```

После запуска программы вы должны получить следующее.

```
Dear Alice,

Eve's cat has been arrested for catnapping, cat burglary,
and extortion.

Sincerely,
Bob
```

Обратите внимание на то, что для апострофа в слове `Eve's` не понадобилось экранирование. В многострочных блоках экранировать одинарные и двойные кавычки необязательно. Аналогичный результат можно получить с помощью следующего вызова функции `print()`.

```
print('Dear Alice,\n\nEve\'s cat has been arrested for catnapping,
      cat burglary,\nand extortion.\n\nSincerely,\nBob')
```

Многострочные комментарии

Однострочный комментарий начинается с символа решетки (#) и длится до конца строки. Если требуется ввести многострочный комментарий, то для этого можно использовать текстовый блок. Приведенный ниже код Python абсолютно корректен.

```
"""Это тестовая программа.
Написана Элом Свейгартом (al@inventwithpython.com).

Программа предназначена для Python 3, а не для Python 2.
"""

def spam():
    """Это многострочный комментарий, объясняющий
    назначение функции spam()."""
    print('Привет!')
```

Индексирование строк и извлечение срезов

В случае строк операции индексирования и извлечения срезов выполняются точно так же, как и в случае списков. Например, строку `'Hello, world!'` можно рассматривать как список, в котором каждый символ имеет соответствующий индекс.

```
' H  e  l  l  o  ,     w  o  r  l  d  !  '
  0  1  2  3  4  5  6  7  8  9  10 11 12
```

Пробел и восклицательный знак тоже являются частью строки, поэтому фраза `'Hello, world!'` содержит 13 символов, от символа `H` с индексом `0` до символа `!` с индексом `12`.

Введите в интерактивной оболочке следующие выражения.

```
>>> spam = 'Hello, world!'
>>> spam[0]
'H'
>>> spam[4]
'o'
>>> spam[-1]
'!'
>>> spam[0:5]
'Hello'
>>> spam[:5]
'Hello'
>>> spam[7:]
'world!'
```

Указав индекс, вы получаете символ, находящийся в соответствующей позиции строки. В случае диапазона индексов, т.е. среза, элемент с начальным индексом включается в срез, а элемент с конечным индексом — нет. Поэтому срез `spam[0:5]` содержит строку `'Hello'`. Подстрока, получаемая с помощью среза `spam[0:5]`, будет включать в себя все символы от `spam[0]` до `spam[4]`, тогда как символ запятой, имеющий индекс `5`, в нее не войдет.

Имейте в виду, что операция создания среза не сопровождается изменением исходной строки. Срез, извлеченный из одной переменной, можно сохранить в другой переменной. Введите в интерактивной оболочке следующие инструкции.

```
>>> spam = 'Hello world!'
>>> fizz = spam[0:5]
>>> fizz
'Hello'
```

Сохранив срез в другой переменной, вы получаете доступ как ко всей строке, так и к ее подстроке.

Использование операторов `in` и `not in` со строками

Операторы `in` и `not in` применяются к строкам точно так же, как и к спискам. Результатом операции будет булево значение `True` или `False`. Введите в интерактивной оболочке следующие выражения.

```
>>> 'Hello' in 'Hello, World'
True
>>> 'Hello' in 'Hello'
True
>>> 'HELLO' in 'Hello, World'
False
>>> '' in 'spam'
True
>>> 'cats' not in 'cats and dogs'
False
```

В этих выражениях проверяется, содержится ли первая строка во второй строке (с учетом регистра).

Вставка строк в другие строки

В программах часто приходится вставлять строки в другие строки. До сих пор мы применяли для этого оператор +, выполняющий конкатенацию строк.

```
>>> name = 'Эл'
>>> age = 4000
>>> 'Меня зовут ' + name + '. Мне ' + str(age) + ' лет.'
'Меня зовут Эл. Мне 4000 лет.'
```

Такой подход немного утомителен. Более простой способ — *строковая интерполяция* (подстановка), при которой оператор `%s` внутри строки

действует как маркер, который следует заменить значениями, указанными после строки. Одно из преимуществ интерполяции заключается в том, что нет необходимости вызывать функцию `str()` для преобразования чисел в строки. Введите в интерактивной оболочке следующие инструкции.

```
>>> name = 'Эл'
>>> age = 4000
>>> 'Меня зовут %s. Мне %s лет.' % (name, age)
'Меня зовут Эл. Мне 4000 лет.'
```

В Python 3.6 появились *f-строки*, имеющие аналогичное назначение, за исключением того, что вместо оператора `%s` в строку включаются выражения в фигурных скобках. Подобно необработанным строкам, f-строки предваряются префиксом `f` перед открывающей кавычкой. Введите в интерактивной оболочке следующие инструкции.

```
>>> name = 'Эл'
>>> age = 4000
>>> f'Меня зовут {name}. В следующем году мне будет {age + 1}.'
'Меня зовут Эл. В следующем году мне будет 4001.'
```

Не забывайте добавлять префикс `f`, в противном случае фигурные скобки и их содержимое станут частью строки.

```
>>> 'Меня зовут {name}. В следующем году мне будет {age + 1}.'
'Меня зовут {name}. В следующем году мне будет {age + 1}.'
```

Полезные методы для работы со строками

Существует целый ряд методов, позволяющих анализировать строки и выполнять над ними различные преобразования. В этом разделе описаны наиболее популярные методы.

Методы `upper()`, `lower()`, `isupper()` и `islower()`

Методы `upper()` и `lower()` возвращают новую строку, в которой все буквы исходной строки преобразованы соответственно в верхний или нижний регистр. Небуквенные символы не затрагиваются. Введите в интерактивной оболочке следующие инструкции.

```
>>> spam = 'Здравствуй, мир!'
>>> spam = spam.upper()
>>> spam
'ЗДРАВСТВУЙ, МИР!'
```

```
>>> spam = spam.lower()
>>> spam
'здравствуй, мир!'
```

Имейте в виду, что эти методы возвращают новые строковые значения, не изменяя исходную строку. Чтобы изменить саму строку, необходимо вызвать для нее метод `upper()` или `lower()` и присвоить результат той же переменной, в которой хранилась исходная строка. Именно поэтому для изменения строки, хранящейся в переменной `spam`, потребовалось выполнить операцию присваивания `spam = spam.upper()`, а не просто вызвать функцию `spam.upper()`. (В качестве аналогии рассмотрим числовую переменную `eggs`, содержащую значение `10`. Выражение `eggs + 3` не приведет к изменению переменной, но это можно сделать с помощью инструкции `eggs = eggs + 3`.)

Методы `upper()` и `lower()` удобно применять в тех случаях, когда при сравнении строк не должен учитываться регистр букв. Строки `'отлично'` и `'ОТлично'` считаются разными. Но в приведенной ниже небольшой программе не имеет значения, как именно будет введено это слово — `'Отлично'`, `'ОТЛИЧНО'` или `'отЛИЧНО'`, — поскольку строка предварительно переводится в нижний регистр.

```
print('Как дела?')
feeling = input()
if feeling.lower() == 'отлично':
    print('Я тоже чувствую себя отлично.')
else:
    print('Я надеюсь, что остаток дня будет лучше.')
```

Даже если в ответ на запрос программы вы введете `'ОТЛично'`, то все равно будет выведена строка `'Я тоже чувствую себя отлично'`. Добавляя в программу код, нивелирующий различия в написании слов и игнорирующий ошибки, связанные с неправильным использованием регистра букв, вы упростите работу с ней и уменьшите вероятность ее аварийного завершения из-за ошибок, допущенных пользователем при вводе текста.

```
Как дела?
ОТЛично
Я тоже чувствую себя отлично.
```

Выполнение этой программы можно просмотреть на сайте `https://autbor.com/convertlowercase/`. Методы `isupper()` и `islower()` возвращают булево значение `True`, если в строке имеется хотя бы одна буква и все буквы записаны соответственно в верхнем или нижнем регистре. В противном случае возвращается значение `False`. Введите в интерактивной

оболочке следующие инструкции и обратите внимание на возвращаемые методами значения.

```
>>> spam = 'Здравствуй, мир!'
>>> spam.islower()
False
>>> spam.isupper()
False
>>> 'ПРИВЕТ'.isupper()
True
>>> 'abc12345'.islower()
True
>>> '12345'.islower()
False
>>> '12345'.isupper()
False
```

Поскольку методы `upper()` и `lower()` сами возвращают строки, для этих строк тоже можно вызывать строковые методы. Соответствующие выражения выглядят как цепочки вызовов. Введите в интерактивной оболочке следующие инструкции.

```
>>> 'Привет'.upper()
'HELLO'
>>> 'Привет'.upper().lower()
'hello'
>>> 'Привет'.upper().lower().upper()
'HELLO'
>>> 'ПРИВЕТ'.lower()
'hello'
>>> 'ПРИВЕТ'.lower().islower()
True
```

Строковые методы `isX()`

Наряду с методами `islower()` и `isupper()` существует ряд других строковых методов, имена которых начинаются со слова `'is'`. Методы этой группы возвращают булево значение, соответствующее определенной характеристике строки. Ниже приведен перечень наиболее популярных методов группы `isX()`.

- **`isalpha()`** — возвращает `True`, если строка непустая и состоит только из букв.
- **`isalnum()`** — возвращает `True`, если строка непустая и состоит только из буквенно-цифровых символов.
- **`isdecimal()`** — возвращает `True`, если строка непустая и состоит только из цифр.

- **isspace()** — возвращает `True`, если строка непустая и состоит только из символов пробела, табуляции и новой строки.
- **istitle()** — возвращает `True`, если строка состоит только из слов, в которых все буквы строчные, кроме первой.

Введите в интерактивной оболочке следующие инструкции.

```
>>> 'Привет'.isalpha()
True
>>> 'hello123'.isalpha()
False
>>> 'hello123'.isalnum()
True
>>> 'hello'.isalnum()
True
>>> '123'.isdecimal()
True
>>> '    '.isspace()
True
>>> 'Это Строка Заголовка'.istitle()
True
>>> 'Это Строка Заголовка 123'.istitle()
True
>>> 'Это не Строка Заголовка'.istitle()
False
>>> 'И Это НЕ Строка Заголовка'.istitle()
False
```

Такие методы удобно применять для проверки допустимости введенных пользователем значений. Например, приведенная ниже программа повторяет запрос до тех пор, пока пользователь не введет корректный возраст и пароль. Откройте в файловом редакторе новое окно, введите в нем следующий код и сохраните его в файле *validateInput.py*.

```
while True:
    print('Укажите ваш возраст:')
    age = input()
    if age.isdecimal():
        break
    print('Пожалуйста, введите число.')

while True:
    print('Выберите новый пароль (только буквы и цифры):')
    password = input()
    if password.isalnum():
        break
    print('Пароли могут состоять только из букв и цифр.')
```

В первом цикле while программа просит пользователя указать свой возраст и сохраняет введенное значение. Если пользователь ввел возраст в виде допустимого значения (десятичного числа), первый цикл прерывается, и управление передается второму циклу while, в котором запрашивается пароль. В противном случае программа информирует пользователя о том, что должно быть введено число, и предлагает повторно указать возраст. Во втором цикле while запрашивается пароль. Если пользователь ввел буквенно-цифровое значение, цикл завершается. В противном случае программа информирует пользователя о том, что допускаются только пароли, состоящие из букв и цифр, и предлагает повторно ввести пароль.

Запустив программу, вы должны получить примерно такие результаты.

```
Укажите ваш возраст:
сорок два
Пожалуйста, введите число.
Укажите ваш возраст:
42
Выберите новый пароль (только буквы и цифры):
secr3t!
Пароли могут состоять только из букв и цифр.
Выберите новый пароль (только буквы и цифры):
secr3t
```

Выполнение авторского варианта этой программы можно просмотреть на сайте https://autbor.com/validateinput/. Вызывая методы isdecimal() и isalnum() для переменных, мы можем выяснить, являются ли введенные пользователем значения цифровыми или буквенно-цифровыми. В данном случае эти проверки позволили отвергнуть ввод пользователем строки 'сорок два' при указании возраста, но принять ввод числа 42, а также отвергнуть ввод значения 'secr3t!' в качестве пароля, но принять ввод значения 'secr3t'.

Методы startswith() и endswith()

Методы startswith() и endswith() возвращают True, если строка, для которой они вызываются, соответственно начинается или заканчивается строкой, переданной методу. В противном случае возвращается False. Введите в интерактивной оболочке следующие инструкции.

```
>>> 'Здравствуй, мир!'.startswith('Здравствуй')
True
>>> 'Здравствуй, мир!'.endswith('мир!')
True
>>> 'abc123'.startswith('abcdef')
False
>>> 'abc123'.endswith('12')
```

```
False
>>> 'Здравствуй, мир!'.startswith('Здравствуй, мир!')
True
>>> 'Здравствуй, мир!'.endswith('Здравствуй, мир!')
True
```

Эти методы — полезная альтернатива оператору сравнения ==, если сравнение с другой строкой требуется выполнить не для всей исходной строки, а только для первой или последней ее части.

Методы join() и split()

Метод `join()` удобно использовать в тех случаях, когда несколько строк, представленных в виде списка, необходимо объединить в одну строку. Этот метод вызывается для строки, которая используется в качестве разделителя. Он получает список строк в качестве аргумента и возвращает объединенную строку. Введите в интерактивной оболочке следующие инструкции.

```
>>> ', '.join(['коты', 'крысы', 'мыши'])
'коты, крысы, мыши'
>>> ' '.join(['Меня', 'зовут', 'Саймон'])
'Меня зовут Саймон'
>>> 'ABC'.join(['Меня', 'зовут', 'Саймон'])
'МеняABCзовутABCСаймон'
```

Обратите внимание на то, что строка, для которой вызывается метод `join()`, вставляется между элементами списка. Например, если вызвать метод `join(['коты', 'крысы', 'мыши'])` для строки `', '`, то будет получена строка `'коты, крысы, мыши'`.

Метод `split()` имеет противоположное назначение: он вызывается для строки и разбивает ее на список слов. Введите в интерактивной оболочке следующую инструкцию.

```
>>> 'Меня зовут Саймон'.split()
['Меня', 'зовут', 'Саймон']
```

По умолчанию строка `'Меня зовут Саймон'` разбивается на слова в тех местах, где встречаются пробельные символы: пробел, табуляция или символ новой строки. Сами эти символы не включаются в строки, возвращаемые в виде списка. Можно задать другую строку-разделитель, передав ее методу `split()`. Например, введите в интерактивной оболочке следующие инструкции.

```
>>> 'MyABCnameABCisABCSimon'.split('ABC')
['Меня', 'зовут', 'Саймон']
```

```
>>> 'Меня зовут Саймон'.split('н')
['Ме', 'я зовут Саймо']
```

Типичный способ применения метода `split()` — разбиение многострочного блока по символам новой строки. Введите в интерактивной оболочке следующие инструкции.

```
>>> spam = '''Дорогая Алиса!
Как твои дела? У меня все хорошо.
В холодильнике хранится контейнер
с этикеткой "Молочный эксперимент".

Не выпей его.
Искренне твой,
Боб'''
>>> spam.split('\n')
['Дорогая Алиса!', 'Как твои дела? У меня все хорошо.',
'В холодильнике хранится контейнер',
' с этикеткой "Молочный эксперимент".', '', 'Не выпей его.',
'Искренне твой,', 'Боб']
```

Передача методу `split()` строки `'\n'` в качестве аргумента позволяет выполнить разбивку многострочного блока, сохраненного в переменной `spam`, в позициях символов новой строки, и вернуть список, каждый элемент которого соответствует одной строке текста.

Разбиение строк с помощью метода `partition()`

Строковый метод `partition()` разбивает строку на текст, стоящий до и после разделителя. Этот метод находит в строке, для которой вызывается, строку-разделитель, которая передается в качестве аргумента, и возвращает кортеж из трех подстрок: до разделителя, сам разделитель и после разделителя. Введите в интерактивной оболочке следующие инструкции.

```
>>> 'Здравствуй, мир!'.partition('м')
('Здравствуй, ', 'м', 'ир!')
>>> 'Здравствуй, мир!'.partition('мир')
('Здравствуй, ', 'мир', '!')
```

Если строка-разделитель, переданная методу `partition()`, встречается в исходной строке несколько раз, то метод разбивает строку только на первом разделителе.

```
>>> 'Здравствуй, мир!'.partition('р')
('Зд', 'р', 'авствуй, мир!')
```

Если строка-разделитель не найдена, то первым элементом возвращаемого кортежа будет исходная строка, а два других элемента будут пустыми.

```
>>> 'Здравствуй, мир!'.partition('XYZ')
('Здравствуй, мир!', '', '')
```

С помощью операции множественного присваивания можно записать три возвращаемые строки в три переменные.

```
>>> before, sep, after = 'Здравствуй, мир!'.partition(' ')
>>> before
'Здравствуй,'
>>> after
'мир!'
```

Метод `partition()` полезен для разбиения строки на части по конкретному разделителю.

Выравнивание текста с помощью методов `rjust()`, `ljust()` и `center()`

Строковые методы `rjust()` и `ljust()` возвращают версию строки, для которой они вызываются, выровненную за счет вставки пробелов. В обоих методах первый аргумент – целое число, определяющее длину выровненной строки. Введите в интерактивной оболочке следующие инструкции.

```
>>> 'Здравствуй'.rjust(15)
'     Здравствуй'
>>> 'Здравствуй'.rjust(20)
'           Здравствуй'
>>> 'Здравствуй, мир'.rjust(20)
'     Здравствуй, мир'
>>> 'Здравствуй'.ljust(15)
'Здравствуй     '
```

Выражение `'Здравствуй'.rjust(15)` означает, что необходимо выровнять строку `'Здравствуй'` вправо в пределах 15 символов. В слове `'Здравствуй'` насчитывается 10 символов, поэтому слева от него будут добавлены 5 пробелов, в результате чего общая длина строки составит 15 символов.

Необязательный второй аргумент в обоих методах задает символ-заполнитель, отличающийся от пробела. Введите в интерактивной оболочке следующие инструкции.

```
>>> 'Здравствуй'.rjust(20, '*')
'**********Здравствуй'
>>> 'Здравствуй'.ljust(20, '-')
'Здравствуй----------'
```

Метод `center()` работает аналогично методам `ljust()` и `rjust()`, но центрирует текст, а не выравнивает его по правому или левому краю. Введите в интерактивной оболочке следующие инструкции.

```
>>> 'Здравствуй'.center(20)
'     Здравствуй     '
>>> 'Здравствуй'.center(20, '=')
'=====Здравствуй====='
```

Эти методы особенно полезны в ситуациях, когда необходимо вывести табулированные значения. Откройте в файловом редакторе новое окно, введите в нем следующий код и сохраните его в файле *picnicTable.py*.

```
def printPicnic(itemsDict, leftWidth, rightWidth):
    print('БЕРЕМ НА ПИКНИК'.center(leftWidth + rightWidth, '-'))
    for k, v in itemsDict.items():
        print(k.ljust(leftWidth, '.') + str(v).rjust(rightWidth))

picnicItems = {'сэндвичи': 4, 'яблоки': 12, 'чашки': 4,
               'печенье': 8000}
printPicnic(picnicItems, 16, 5)
printPicnic(picnicItems, 24, 7)
```

Выполнение этой программы можно просмотреть на сайте `https://autbor.com/picnictable/`. Здесь мы определяем функцию `printPicnic()`, которая получает данные в виде словаря и использует методы `center()`, `ljust()` и `rjust()` для отображения этих данных в виде аккуратно выровненной страницы.

В функцию `printPicnic()` передается словарь `picnicItems`, согласно которому мы берем на пикник 4 сэндвича, 12 яблок, 4 чашки и 8000 печений. Содержимое словаря должно быть выведено в две колонки: слева — название, справа — количество. Для этого нужно решить, какой ширины должны быть левая и правая колонки. Соответствующие значения передаются функции `printPicnic()` вместе со словарем.

Функция `printPicnic()` получает словарь, а также значения ширины для левой (`leftWidth`) и правой (`rightWidth`) колонок. Над таблицей выводится центрированный заголовок `'БЕРЕМ НА ПИКНИК'`. Затем элементы словаря обрабатываются в цикле, и каждая пара "ключ — значение" выводится в отдельной строке, причем ключ выравнивается влево, заполняясь точками, а значение — вправо, заполняясь пробелами.

После создания функции `printPicnic()` мы определяем словарь `picnicItems` и дважды вызываем функцию `printPicnic()`, передавая ей различные значения ширины левого и правого столбцов.

Программа выводит две таблицы. В первом случае ширина левой колонки составляет 16 символов, а правой — 5. Во втором случае эти значения составляют 24 и 7 символов соответственно.

```
---БЕРЕМ НА ПИКНИК---
сэндвичи.....    4
яблоки.....   12
чашки.....    4
печенье..... 8000
--------БЕРЕМ НА ПИКНИК--------
сэндвичи.....................      4
яблоки.....................     12
чашки.....................      4
печенье.....................   8000
```

Используя методы `rjust()`, `ljust()` и `center()`, мы можем быть уверены в том, что колонки таблицы аккуратно выравниваются, даже если точное количество символов, содержащихся в каждой строке, неизвестно.

Удаление пробелов с помощью методов strip(), rstrip() и lstrip()

Иногда возникает необходимость удалить из строки ведущие и/или замыкающие пробельные символы (пробелы, табуляции, символы новой строки). Метод `strip()` возвращает новую строку без начальных и конечных пробельных символов. Методы `lstrip()` и `rstrip()` удаляют пробельные символы соответственно в начале и в конце строки. Введите в интерактивной оболочке следующие инструкции.

```
>>> spam = '    Здравствуй, мир    '
>>> spam.strip()
'Здравствуй, мир'
>>> spam.lstrip()
'Здравствуй, мир    '
>>> spam.rstrip()
'    Здравствуй, мир'
```

С помощью необязательного строкового аргумента можно указать, какие именно символы должны удаляться с обоих концов строки. Введите в интерактивной оболочке следующие инструкции.

```
>>> spam = 'SpamSpamBaconSpamEggsSpamSpam'
>>> spam.strip('ampS')
'BaconSpamEggs'
```

Передавая методу `strip()` аргумент `'ampS'`, мы сообщаем ему, что в начале и в конце строки должны быть удалены все вхождения символов `'a'`, `'m'`, `'p'` и `'S'`. Порядок символов в строке, передаваемой методу `strip()`, не важен: вызов `strip('mapS')` или `strip('Spam')` даст тот же самый результат, что и вызов `strip('ampS')`.

Получение числовых значений символов с помощью функций `ord()` и `chr()`

Компьютеры хранят информацию в виде байтов, т.е. двоичных чисел, поэтому важно иметь возможность преобразовывать текст в числа. Каждому текстовому символу соответствует числовое значение, называемое *кодом Unicode*. Например, символу `'A'` соответствует код 65, символу `'4'` — код 52, а символу `'!'` — код 33. С помощью функции `ord()` можно узнать код символа, а с помощью функции `chr()` — символ, соответствующий целочисленному коду. Введите в интерактивной оболочке следующие инструкции.

```
>>> ord('A')
65
>>> ord('4')
52
>>> ord('!')
33
>>> chr(65)
'A'
```

Эти функции полезны, когда нужно выполнить сравнение символов или математическую операцию над ними.

```
>>> ord('B')
66
>>> ord('A') < ord('B')
True
>>> chr(ord('A'))
'A'
>>> chr(ord('A') + 1)
'B'
```

Рассмотрение стандарта Unicode выходит за рамки книги. Для тех, кому эта тема интересна, рекомендую просмотреть выступление Неда

Батчелдера на конференции PyCon 2012, доступное по адресу `https://youtu.be/sgHbC6udIqc`.

> **Выполнение сценариев Python вне Mu**
>
> До сих пор мы выполняли сценарии Python с помощью интерактивной оболочки или файлового редактора Mu. Но вряд ли вам понравится запускать Mu всякий раз, когда потребуется выполнить сценарий. К счастью, существуют более удобные способы запуска сценариев Python. Соответствующие процедуры для Windows, macOS и Linux немного различаются, но все они описаны в приложении Б. Загляните в него, если хотите узнать о том, как передавать сценариям аргументы командной строки (в Mu такая возможность отсутствует).

Копирование и вставка строк с помощью модуля `pyperclip`

В модуле `pyperclip` имеются функции `copy()` и `paste()`, которые позволяют выполнять операции копирования и вставки текста через буфер обмена. Например, результаты работы программы можно скопировать в буфер обмена, а затем вставить в сообщение электронной почты или в текстовый документ.

Модуль `pyperclip` не входит в состав Python. Чтобы его инсталлировать, следуйте указаниям по установке сторонних модулей, приведенным в приложении А. После этого введите в интерактивной оболочке следующие инструкции.

```
>>> import pyperclip
>>> pyperclip.copy('Здравствуй, мир!')
>>> pyperclip.paste()
'Hello world!'
```

Разумеется, если содержимое буфера будет изменено внешней программой, то метод `paste()` вернет другой результат. Например, если я скопирую данное предложение в буфер обмена, а затем вызову метод `paste()`, то получу следующее.

```
>>> pyperclip.paste()
'Например, если я скопирую данное предложение в буфер обмена,
а затем вызову метод paste(), то получу следующее.'
```

Проект: автоматическая рассылка сообщений с помощью нескольких буферов обмена

Если вам приходится отвечать на большое количество похожих писем, то вам, вероятно, приходится вводить много одинакового текста. Возможно, у вас есть заранее составленный текстовый документ с готовыми фразами, чтобы быстро копировать и вставлять их с помощью буфера обмена. Но проблема в том, что системный буфер обмена одновременно хранит только одно сообщение. Мы напишем программу, которая хранит в буфере несколько фраз.

> **Проекты в главах**
>
> Это первый из проектов, предлагаемых в книге. Во всех последующих главах будут рассматриваться проекты, предназначенные для закрепления изученного материала. Особенность всех проектов заключается в том, что каждый из них начинается "с чистого листа" (пустого окна файлового редактора) и заканчивается полностью функциональным вариантом программы. Желательно, чтобы вы не просто читали эти разделы, но и выполняли сами проекты.

Шаг 1. Проектирование программы и структур данных

Мы хотим иметь возможность запускать программу с аргументом командной строки, который представляет собой короткую ключевую фразу, например `'согласен'` или `'занят'`. Сообщение, связанное с этой ключевой фразой, будет скопировано в буфер обмена, чтобы пользователь мог вставить его в электронное письмо. Это позволяет не вводить длинные сообщения каждый раз заново.

Откройте новое окно файлового редактора и сохраните программу в файле *mclip.py*. Программа должна начинаться со строки `#!` (об этом рассказано в приложении Б). Также нужно добавить строку комментария с кратким описанием программы. Поскольку с каждой ключевой фразой связано текстовое сообщение, соответствующие строки хранятся в словаре. Ниже показан начальный фрагмент программы.

```
#! python3
# mclip.py - программа с несколькими буферами обмена.

TEXT = {'Согласен': """Да, я согласен. Мне это подходит.""",
        'Занят': """Извините, можно перенести это на конец недели
                    или на следующую неделю?""",
        'Продать': """Хотите сделать такой платеж ежемесячным?"""}
```

Шаг 2. Обработка аргументов командной строки

Аргументы командной строки хранятся в переменной `sys.argv` (дополнительная информация об использовании аргументов командной строки приведена в приложении Б). Первым элементом в списке `sys.argv` всегда будет строка, содержащая имя файла программы (`'mclip.py'`), а вторым элементом будет первый аргумент командной строки. В нашей программе это ключевое слово сообщения, помещаемого в буфер обмена. Поскольку аргумент командной строки обязателен, пользователь получит сообщение о синтаксисе вызова, если забудет добавить ключевое слово (т.е. если в списке `sys.argv` меньше двух элементов). Теперь программа должна выглядеть так.

```
#! python3
# mclip.py - программа с несколькими буферами обмена.

TEXT = {'Согласен': """Да, я согласен. Мне это подходит.""",
        'Занят': """Извините, можно перенести это на конец недели
                    или на следующую неделю?""",
        'Продать': """Хотите сделать такой платеж ежемесячным?"""}

import sys
if len(sys.argv) < 2:
    print('Использование: py mclip.py [ключевое слово] - ' \
          'копирование соответствующего текста в буфер обмена)
    sys.exit()

keyphrase = sys.argv[1]     # первый аргумент командной строки -
                            # ключевое слово
```

Шаг 3. Копирование фразы в буфер

Теперь, когда ключевое слово хранится в виде строки в переменной `keyphrase`, нужно проверить, есть ли соответствующий ключ в словаре. Если да, то значение ключа копируется в буфер обмена с помощью вызова `pyperclip.copy()` (модуль `pyperclip` необходимо предварительно импортировать). Отметим, что переменная `keyphrase` не особо нужна – вместо нее можно просто использовать выражение `sys.argv[1]`. Но переменная с именем `keyphrase` выглядит гораздо понятнее, чем загадочное `sys.argv[1]`.

Теперь программа выглядит следующим образом.

```
#! python3
# mclip.py - программа с несколькими буферами обмена.

TEXT = {'Согласен': """Да, я согласен. Мне это подходит.""",
        'Занят': """Извините, можно перенести это на конец недели
```

```
                или на следующую неделю?""",
        'Продать': """Хотите сделать такой платеж ежемесячным?"""}

import sys, pyperclip
if len(sys.argv) < 2:
    print('Использование: py mclip.py [ключевое слово] - ' \
          'копирование соответствующего текста в буфер обмена)
    sys.exit()

keyphrase = sys.argv[1]     # первый аргумент командной строки -
                            # ключевое слово

if keyphrase in TEXT:
    pyperclip.copy(TEXT[keyphrase])
    print('Текст для \'' + keyphrase + '\' скопирован в буфер обмена.')
else:
    print('Текст для \'' + keyphrase + '\' отсутствует')
```

Новый код ищет в словаре `TEXT` ключевое слово. Если данное слово является ключом в словаре, мы получаем значение, соответствующее этому ключу, копируем его в буфер обмена и выводим сообщение о том, что значение скопировано. В противном случае выводится сообщение о том, что данного ключевого слова нет в словаре.

Это полный текст программы. Обратитесь к приложению Б, чтобы узнать, как запускать такие программы из командной строки. Теперь у вас есть возможность быстро копировать сообщения в буфер обмена. Нужно будет только изменять содержимое словаря `TEXT` всякий раз, когда вам понадобится дополнить программу новым сообщением.

В Windows можно создать пакетный файл, чтобы иметь возможность запускать программу в окне **Выполнить** (оно вызывается нажатием комбинации клавиш <Win+R>). Введите в файловом редакторе следующий код и сохраните его в файле *C:\Windows\mclip.bat*.

```
@py.exe C:\путь_к_файлу\mclip.py %*
@pause
```

Когда пакетный файл будет создан, для запуска программы в среде Windows достаточно будет нажать комбинацию клавиш <Win+R> и ввести **mclip *ключевое_слово***.

Проект: добавление маркеров в разметку Wiki-документов

Редактируя статьи в Википедии, можно создавать маркированные списки, вводя каждый элемент списка в отдельной строке, которая предваряется маркером в виде звездочки. Но что, если имеется очень большой список,

в который нужно добавить маркеры? Можно сделать это вручную, вводя символы звездочки в начале каждой строки, строка за строкой. Но лучше автоматизировать такую легкую задачу с помощью короткого сценария Python.

Сценарий *bulletPointAdder.py* получает текст из буфера обмена, добавляет звездочку и пробел в начало каждой строки, а затем обратно копирует новый текст в буфер обмена. Предположим, в буфер обмена скопирован следующий текст.

```
Список животных
Список аквариумных рыб
Список биологов
Список сортов растений
```

Если выполнить программу *bulletPointAdder.py*, то в буфере обмена будет содержаться следующий текст.

```
* Список животных
* Список аквариумных рыб
* Список биологов
* Список сортов растений
```

Этот дополненный звездочками текст можно вставить в статью Википедии в качестве маркированного списка.

Шаг 1. Копирование и вставка посредством буфера обмена

Итак, программа *bulletPointAdder.py* должна делать следующее:

1) получать текст из буфера обмена;
2) выполнять над ним требуемые действия;
3) копировать измененный текст обратно в буфер обмена.

Со вторым пунктом придется немного повозиться, а вот пункты 1 и 3 достаточно просты. Они требуют использования всего лишь двух вызовов: `pyperclip.paste()` и `pyperclip.copy()`. Поэтому на данном этапе мы напишем только ту часть программы, которая реализует пп. 1 и 3. Введите в файловом редакторе приведенный ниже код и сохраните его в файле *bulletPointAdder.py*.

```
#! python3
# bulletPointAdder.py - добавляет маркеры Википедии в начало
# каждой строки текста, сохраненного в буфере обмена.

import pyperclip
text = pyperclip.paste()
```

```
# СДЕЛАТЬ: разделить строки и добавить звездочки.

pyperclip.copy(text)
```

Комментарий 'СДЕЛАТЬ' — напоминание о том, что эту часть программы еще предстоит написать. Следующий шаг — реализация п. 2.

Шаг 2. Разбивка текста на строки и добавление звездочек

Метод pyperclip.paste() возвращает весь текст из буфера в виде одной большой строки. В рассмотренном выше примере строка, сохраненная в переменной text, выглядела бы так.

```
'Список животных\nСписок аквариумных рыб\n
Список биологов\nСписок сортов растений'
```

Наличие символов новой строки \n приведет к тому, что строка будет отображаться в виде многострочного текста при выводе на экран или вставке из буфера обмена. В этой общей строке содержатся элементы списка, и нам нужно добавить звездочку в начало каждого из них.

Можно было бы написать код, который ищет все символы \n и вставляет после каждого из них звездочку. Однако гораздо проще использовать метод split() для получения списка строк, а затем добавить звездочку в начало каждого элемента списка.

Допишем программу следующим образом.

```
#! python3
# bulletPointAdder.py - добавляет маркеры Википедии в начало
# каждой строки текста, сохраненного в буфере обмена.

import pyperclip
text = pyperclip.paste()

# Разбивка текста на строки и добавление звездочек
lines = text.split('\n')
for i in range(len(lines)):     # цикл по списку "lines"
    lines[i] = '* ' + lines[i]     # добавляем звездочку в каждую
                                   # строку в списке "lines"

pyperclip.copy(text)
```

Выполняя разбивку текста по символам новой строки, мы получаем список, каждый элемент которого представляет собой отдельную строку текста. Мы сохраняем этот список в переменной lines, а затем проходим в цикле по всем элементам списка, добавляя в начало каждой строки звездочку и пробел. Теперь каждая строка списка начинается со звездочки.

Шаг 3. Объединение измененных строк

Итак, список `lines` содержит измененные строки, начинающиеся со звездочек. Но метод `pyperclip.copy()` ожидает аргумент в виде одиночной строки, а не списка строк. Чтобы сформировать такой аргумент, передайте список строк методу `join()`, который объединит их в одну строку. Дополните программу, как показано ниже.

```
#! python3
# bulletPointAdder.py - добавляет маркеры Википедии в начало
# каждой строки текста, сохраненного в буфере обмена.

import pyperclip
text = pyperclip.paste()

# Разбивка текста на строки и добавление звездочек
lines = text.split('\n')
for i in range(len(lines)):    # цикл по списку "lines"
    lines[i] = '* ' + lines[i]    # добавляем звездочку в каждую
                                  # строку в списке "lines"
text = '\n'.join(lines)
pyperclip.copy(text)
```

Теперь программа заменяет текст в буфере обмена текстом, в начале каждой строки которого стоит звездочка. Это готовый вариант программы. Попробуйте применить ее к тексту, скопированному в буфер обмена.

Не факт, что вам понадобится автоматизировать именно эту задачу, но при работе с текстом часто приходится автоматизировать различные задачи, такие как удаление замыкающих пробелов в конце строк или преобразование текста в верхний или нижний регистр. Подобные действия удобно выполнять через буфер обмена.

Короткая программа: поросячья латынь

Поросячья латынь (Pig Latin)[1] — это шутливый выдуманный язык, в котором английские слова изменяются следующим образом:

- если слово начинается с гласной, то в конце добавляется *yay*;
- если слово начинается с одной или нескольких согласных (например, *ch* или *gr*), то они перемещаются в конец слова, и к ним добавляется *ay*.

Мы напишем программу, которая будет выводить нечто наподобие следующего.

[1] См. https://ru.wikipedia.org/wiki/Поросячья_латынь. — *Примеч. ред.*

```
Введите английский текст для перевода на поросячью латынь:
My name is AL SWEIGART.
Ymay amenay isyay ALYAY EIGARTSWAY.
```

В программе применяются строковые методы, рассмотренные в этой главе. Введите в файловом редакторе следующий код и сохраните его в файле *pigLat.py*.

```
# Перевод английского текста на поросячью латынь
print('Введите английский текст для перевода на поросячью латынь:')
message = input()

VOWELS = ('a', 'e', 'i', 'o', 'u', 'y')

pigLatin = []    # список слов на поросячьей латыни
for word in message.split():
    # Отделяем небуквенные символы в начале слова
    prefixNonLetters = ''
    while len(word) > 0 and not word[0].isalpha():
        prefixNonLetters += word[0]
        word = word[1:]
    if len(word) == 0:
        pigLatin.append(prefixNonLetters)
        continue

    # Отделяем небуквенные символы в конце слова
    suffixNonLetters = ''
    while not word[-1].isalpha():
        suffixNonLetters += word[-1]
        word = word[:-1]

    # Запоминаем регистр слова
    wasUpper = word.isupper()
    wasTitle = word.istitle()

    word = word.lower()    # перевод в нижний регистр

    # Отделяем согласные в начале слова
    prefixConsonants = ''
    while len(word) > 0 and not word[0] in VOWELS:
        prefixConsonants += word[0]
        word = word[1:]

    # Добавляем финальный слог к слову
    if prefixConsonants != '':
        word += prefixConsonants + 'ay'
    else:
        word += 'yay'

    # Возвращаем исходный регистр
    if wasUpper:
```

```
        word = word.upper()
    if wasTitle:
        word = word.title()

    # Возвращаем небуквенные символы в начало или конец слова
    pigLatin.append(prefixNonLetters + word + suffixNonLetters)

# Соединяем слова обратно в строку
print(' '.join(pigLatin))
```

Рассмотрим этот код построчно.

```
# Перевод английского текста на поросячью латынь
print('Введите английский текст для перевода на поросячью латынь:')
message = input()

VOWELS = ('a', 'e', 'i', 'o', 'u', 'y')
```

Сначала пользователю предлагается ввести текст на английском языке для перевода на поросячью латынь. Кроме того, мы создаем константу, которая содержит все строчные гласные (включая `'y'`) в виде кортежа строк. Она понадобится позже.

Далее мы создаем переменную `pigLatin`, в которой будет храниться список слов, переводимых на поросячью латынь.

```
pigLatin = []    # список слов на поросячьей латыни
for word in message.split():
    # Отделяем небуквенные символы в начале слова
    prefixNonLetters = ''
    while len(word) > 0 and not word[0].isalpha():
        prefixNonLetters += word[0]
        word = word[1:]
    if len(word) == 0:
        pigLatin.append(prefixNonLetters)
        continue
```

Нам нужно, чтобы каждое слово представляло собой отдельную строку, поэтому мы вызываем метод `message.split()` для получения списка слов. Например, строка `'My name is AL SWEIGART.'` будет разбита следующим образом: `['My', 'name', 'is', 'AL', 'SWEIGART.']`.

Далее необходимо удалить все небуквенные символы, стоящие в начале и конце каждого слова, чтобы строки наподобие `'SWEIGART.'` переводились как `'EIGARTSWAY.'`, а не `'EIGART.SWAY'`. Небуквенные символы будут храниться в переменных `prefixNonLetters` и `suffixNonLetters`.

В цикле `while`, в котором функция `isalpha()` вызывается для первого символа в слове, определяется, нужно ли удалить символ из слова и добавить его в конец строки `prefixNonLetters`. Если все слово состоит из

небуквенных символов, например из цифр, мы просто добавляем его в список pigLatin и переходим к следующему слову.

```
# Отделяем небуквенные символы в конце слова
suffixNonLetters = ''
while not word[-1].isalpha():
    suffixNonLetters += word[-1]
    word = word[:-1]
```

Нам также нужно сохранить все небуквенные символы, стоящие в конце строки word. Этот цикл напоминает предыдущий.

Далее программа запоминает регистр слова, чтобы его можно было восстановить после перевода слова на поросячью латынь.

```
# Запоминаем регистр слова
wasUpper = word.isupper()
wasTitle = word.istitle()

word = word.lower()     # перевод в нижний регистр
```

В оставшейся части цикла for мы будем работать с версией слова в нижнем регистре.

Чтобы преобразовать слово (например, 'name' в 'amenay'), необходимо удалить все согласные, стоящие в начале.

```
# Отделяем согласные в начале слова
prefixConsonants = ''
while len(word) > 0 and not word[0] in VOWELS:
    prefixConsonants += word[0]
    word = word[1:]
```

Мы используем цикл, аналогичный тому, в котором удалялись небуквенные символы в начале слова, только теперь мы извлекаем согласные и сохраняем их в переменной prefixConsonants.

Если в начале слова были согласные, то теперь они находятся в переменной prefixConsonants, и нам нужно присоединить эту переменную вместе со строкой 'ay' к концу слова. В противном случае предполагается, что слово начинается с гласной, и тогда нам нужно лишь добавить в конце 'yay'.

```
# Добавляем финальный слог к слову
if prefixConsonants != '':
    word += prefixConsonants + 'ay'
else:
    word += 'yay'
```

Напомним, что мы перевели слово в нижний регистр с помощью инструкции `word = word.lower()`. Если слово изначально было записано в другом регистре, то следующий код возвращает прежний регистр.

```
    # Возвращаем исходный регистр
    if wasUpper:
        word = word.upper()
    if wasTitle:
        word = word.title()
```

В конце цикла `for` мы добавляем слово вместе с небуквенным префиксом или суффиксом в список `pigLatin`.

```
    # Возвращаем небуквенные символы в начало или конец слова
    pigLatin.append(prefixNonLetters + word + suffixNonLetters)

# Соединяем слова обратно в строку
print(' '.join(pigLatin))
```

После завершения цикла мы объединяем все слова в одну общую строку, вызывая метод `join()`. Эта строка передается методу `print()` для вывода на экран.

Другие короткие текстовые программы доступны по адресу `https://github.com/asweigart/pythonstdiogames/`.

Резюме

В Python имеется множество полезных методов, предназначенных для работы с текстом и строковыми переменными. Эти методы будут применяться практически во всех программах, которые вам предстоит написать.

Пока что наши программы не слишком сложные: в них нет графического интерфейса с красочными изображениями и цветным текстом. Мы просто выводим текст с помощью метода `print()` и даем пользователю возможность вводить текст, используя метод `input()`. Работать с текстом можно также через буфер обмена. Это позволяет писать программы, обрабатывающие большие объемы текстовых данных. И пусть такие программы не содержат окон или графики, зато они могут быстро выполнять массу полезных действий.

Другой способ работы с большими объемами текста — это чтение и запись текстовых файлов, хранящихся на жестком диске. Об этом мы поговорим в главе 9.

Итак, мы рассмотрели практически все основные концепции программирования на Python. В последующих главах вы узнаете еще много интересного, но имеющихся знаний уже достаточно, чтобы начать писать полезные

сценарии автоматизации различных задач. Если хотите увидеть коллекцию небольших программ Python, основанных на изученных концепциях, обратитесь по адресу `https://github.com/asweigart/pythonstdiogames/`. Попробуйте скопировать исходный код каждой программы вручную и внести в него различные изменения, чтобы увидеть, как они влияют на поведение программы. Когда вы поймете, как программа работает, попытайтесь написать ее с нуля. Не нужно в точности воспроизводить ее исходный код: просто сосредоточьтесь на том, что делает программа, а не на том, как она это делает.

Возможно, вы посчитаете, что у вас недостаточно знаний Python, чтобы выполнять такие операции, как загрузка веб-страниц, обновление электронных таблиц или отправка текстовых сообщений. Но здесь вам на помощь придут модули Python. Эти модули, написанные другими программистами, содержат функции, которые облегчают решение самых разных задач. В следующей части мы займемся написанием полезных программ, решающих те или иные задачи автоматизации.

Контрольные вопросы

1. Что такое экранированные символы?
2. Что означают экранированные символы `\n` и `\t`?
3. Как вставить обратную косую черту (`\`) в строку?
4. Строка `"Howl's Moving Castle"` вполне допустима. Почему она не вызовет ошибку, несмотря на наличие неэкранированного апострофа в слове `"Howl's"`?
5. Если вам не хочется вставлять в строку символы `\n`, то как записать многострочный текст?
6. Чему равны следующие выражения?
   - `'Здравствуй, мир!'[1]`
   - `'Здравствуй, мир!'[0:5]`
   - `'Здравствуй, мир!'[:5]`
   - `'Здравствуй, мир'[3:]`
7. Чему равны следующие выражения?
   - `'Здравствуй'.upper()`
   - `'Здравствуй'.upper().isupper()`
   - `'Здравствуй'.upper().lower()`
8. Чему равны следующие выражения?
   - `'Помни о том, что было, не забывай.'.split()`
   - `'-'.join('Должен остаться только один.'.split())`

9. Какие методы применяются для выравнивания строки по правому краю, по левому краю и по центру?
10. Как удалить пробельные символы в начале или в конце строки?

Учебные проекты

Чтобы закрепить полученные знания на практике, напишите программы для предложенных ниже задач.

Табличный вывод данных

Напишите функцию `printTable()`, которая получает список списков строк и отображает его в виде аккуратной таблицы с выравниванием текста по правому краю в каждом столбце. Предполагается, что все внутренние списки содержат одинаковое количество строк. Например, список может выглядеть так.

```
tableData = [['яблоки', 'апельсины', 'вишни', 'бананы'],
             ['Алиса', 'Боб', 'Кэрол', 'Дэвид'],
             ['собаки', 'кошки', 'лось', 'гусь']]
```

Функция `printTable()` должна отобразить этот список в следующем виде.

```
    яблоки Алиса собаки
 апельсины   Боб  кошки
    вишни Кэрол   лось
   бананы Дэвид   гусь
```

Подсказка: в первую очередь программа должна найти самую длинную строку в каждом из внутренних списков. Ширина столбца должна быть достаточно большой для того, чтобы в нем поместилась любая строка. Значения максимальной ширины столбцов могут храниться в виде списка целых чисел. Код функции `printTable()` может начинаться с инструкции `colWidths = [0] * len(tableData)`, которая создает список, содержащий значения `0` в количестве, равном количеству внутренних списков в переменной `tableData`. Таким образом, элемент `colWidths[0]` будет содержать длину самой большой строки в списке `tableData[0]`, элемент `colWidths[1]` — длину самой большой строки в списке `tableData[1]` и т.д. Далее можно найти самую большую длину строки в каждом вложенном списке, чтобы определить, какое целочисленное значение следует передать строковому методу `rjust()`.

Боты Zombie Dice

Игра для программистов – это компьютерная игра, в которой человек не участвует напрямую, а вместо этого пишет бот, который будет играть автономно. Автор книги разработал симулятор Zombie Dice, который позволяет программистам попрактиковаться в написании интеллектуальных игровых ботов. Боты Zombie Dice могут быть как простыми, так и невероятно сложными и отлично подходят для занятий в классе или для решения индивидуальных задач.

Zombie Dice – это быстрая, веселая игра в кости от Steve Jackson Games. Здесь игроками являются зомби, пытающиеся съесть как можно больше человеческих мозгов, прежде чем их подстрелят три раза. Имеется стакан с 13 кубиками, на которые нанесены пиктограммы мозгов, следов и дробовиков. Пиктограммы окрашены, и каждый цвет обозначает различную вероятность наступления события. У каждого кубика есть две грани с пиктограммами следов, но у зеленых кубиков больше граней с пиктограммами мозгов, у красных кубиков больше граней с пиктограммами дробовиков, а у желтых кубиков одинаковое количество граней с пиктограммами мозгов и дробовиков. На каждом ходе выполняются следующие действия.

1. Все 13 кубиков помещаются в стакан. Игрок случайным образом вытаскивает три кубика и бросает их (игроки всегда бросают ровно три кубика).
2. Подсчитайте выпавшие пиктограммы мозгов (люди, чьи мозги были съедены) и дробовиков (люди, которые отбились от нападения). Накопление трех дробовиков автоматически оставляет игрока с нулем очков (независимо от того, сколько он съел мозгов). Если у игрока выпало не более двух дробовиков, то при желании он может продолжить бросать кости, а может выбрать завершение хода и получить по одному очку за каждый съеденный мозг.
3. Если игрок решает продолжать бросать кости, то он должен повторно бросить все кости с пиктограммами следов. Помните, что игрок всегда должен бросать три кости, а значит, он должен взять из стакана дополнительные кости, если у него их меньше трех. Игрок может продолжать бросать кости до тех пор, пока либо не соберет три дробовика, потеряв все, либо все 13 костей не будут брошены. Игрок не может повторно бросить только одну или две кости.
4. Как только кто-то собирает 13 мозгов, остальные игроки завершают раунд. Игрок, собравший наибольшее количество мозгов, побеждает. В случае ничьей игроки с одинаковым количеством очков играют финальный раунд.

Zombie Dice — это азартная игра с механикой проверки удачи (проверка на жадность): чем дольше вы бросаете кости, тем больше можете собрать мозгов, но тем выше вероятность, что в конечном итоге вы получите три дробовика и все потеряете. Как только игрок набирает 13 очков, остальные игроки делают свой ход (чтобы получить шанс наверстать упущенное), и игра заканчивается. Игрок с наибольшим количеством очков побеждает. Полные правила игры можно прочитать по адресу `https://github.com/asweigart/zombiedice/`.

Установите модуль `zombiedice` с помощью утилиты `pip` (см. приложение А). Можете запустить демоверсию симулятора с несколькими готовыми ботами, выполнив в интерактивной оболочке следующие инструкции.

```
>>> import zombiedice
>>> zombiedice.demo()
Zombie Dice Visualization is running. Open your browser
to http://localhost:51810 to view it.
Press Ctrl-C to quit.
```

Программа выполняется в окне браузера (рис. 6.1).

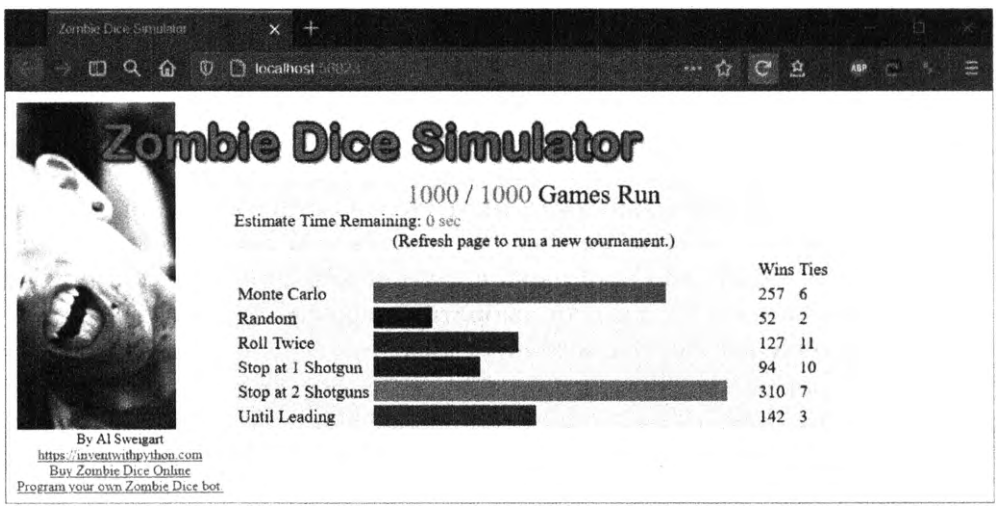


Рис. 6.1. Веб-интерфейс симулятора Zombie Dice

Для создания ботов нужно написать класс, включающий метод `turn()`, который вызывается симулятором, когда наступает очередь вашего бота бросать кости. Подробное рассмотрение классов выходит за рамки книги, поэтому код класса уже содержится в готовом виде в программе *myzombie.py*, которая, в свою очередь, содержится в архиве материалов к книге (адреса, по которым можно скачать архив, были указаны во введении). Метод класса — это, по сути, та же функция, так что можете использовать код метода

`turn()` в программе *myZombie.py* в качестве шаблона. В методе `turn()` нужное число раз вызывается функция `zombiedice.roll()`, имитирующая бросание костей ботом.

```
import zombiedice

class MyZombie:
    def __init__(self, name):
        # А каждого зомби должно быть имя
        self.name = name

    def turn(self, gameState):
        # gameState - это словарь с информацией о текущем
        # состоянии игры (его можно игнорировать)

        diceRollResults = zombiedice.roll()    # первый бросок
        # Метод roll() возвращает словарь с ключами 'brains',
        # 'shotgun' и 'footsteps', показывающими, сколько раз
        # выпали кубики каждого типа. Ключ 'rolls' - это список
        # кортежей (цвет, пиктограмма) с информацией о броске.
        # Пример словаря, возвращаемого функцией roll():
        # {'brains': 1, 'footsteps': 1, 'shotgun': 1,
        #  'rolls': [('yellow', 'brains'), ('red', 'footsteps'),
        #            ('green', 'shotgun')]}

        # ЗАМЕНИТЕ ЭТОТ КОД СОБСТВЕННЫМ
        brains = 0
        while diceRollResults is not None:
            brains += diceRollResults['brains']

            if brains < 2:
                diceRollResults = zombiedice.roll()    # еще бросок
            else:
                break

zombies = (
    zombiedice.examples.RandomCoinFlipZombie(name='Random'),
    zombiedice.examples.RollsUntilInTheLeadZombie(name= \
        'Until Leading'),
    zombiedice.examples.MinNumShotgunsThenStopsZombie(name= \
        'Stop at 2 Shotguns', minShotguns=2),
    zombiedice.examples.MinNumShotgunsThenStopsZombie(name= \
        'Stop at 1 Shotgun', minShotguns=1),
    MyZombie(name='My Zombie Bot'),
    # Добавьте здесь любых других зомби
)

# Раскомментируйте одну из следующих строк,
# чтобы запустить игру в режиме командной строки
# или в режиме браузера
#zombiedice.runTournament(zombies=zombies, numGames=1000)
zombiedice.runWebGui(zombies=zombies, numGames=1000)
```

У метода `turn()` два параметра: `self` и `gameState`. Можете проигнорировать их при создании первых зомби-ботов или узнать подробности в онлайн-документации, если потребуется дополнительная информация. В методе `turn()` должен хотя бы один раз вызываться метод `zombiedice.roll()`, чтобы сделать начальный бросок. Затем, в зависимости от стратегии, которую применяет бот, он может повторно вызвать метод `zombiedice.roll()` столько раз, сколько захочет. В файле *myZombie.py* метод `zombiedice.roll()` вызывается в двух местах. Зомби-бот будет бросать кости, пока не съест более одного мозга.

Значение, возвращаемое методом `zombiedice.roll()`, описывает результаты броска. Это словарь с четырьмя ключами. Три ключа, `'shotgun'`, `'brains'` и `'footsteps'`, имеют целочисленные значения, соответствующие количеству выпавших игральных костей с данными пиктограммами. Значение четвертого ключа, `'rolls'`, представляет собой список кортежей для каждого кубика. Кортежи содержат две строки: цвет кубика (индекс 0) и выпавшая пиктограмма (индекс 1). Пример был приведен в комментариях к коду. Если бот уже собрал три дробовика, то метод `zombiedice.roll()` вернет `None`.

Напишите несколько своих ботов Zombie Dice и сравните их с другими ботами. В частности, попробуйте создать следующих ботов:

- бот, который после первого броска случайным образом решает, продолжать или остановиться;
- бот, который перестает бросать кости после того, как выпало два мозга;
- бот, который перестает бросать кости после того, как выпало два дробовика;
- бот, который решает бросать кости от одного до четырех раз, но досрочно останавливается, если выпало два дробовика;
- бот, который перестает бросать кости после того, как выпало больше дробовиков, чем мозгов.

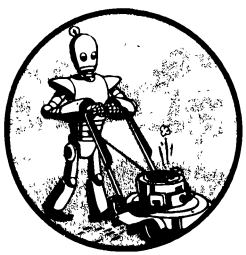
Запустите эти боты с помощью симулятора и посмотрите, как они сражаются друг с другом. Можете также ознакомиться с кодом нескольких готовых ботов по адресу `https://github.com/asweigart/zombiedice/`. На практике результаты тысяч смоделированных игр говорят о том, что лучшая стратегия — остановиться, когда выпало два дробовика. Но никто не мешает вам проверить свою удачу...

ЧАСТЬ II

АВТОМАТИЗАЦИЯ ЗАДАЧ

7

РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ



Наверняка вам не раз приходилось выполнять текстовый поиск, нажимая комбинацию клавиш <Ctrl+F> и вводя в появившемся окне слова, которые нужно найти. *Регулярные выражения* — более продвинутый поисковый механизм: они позволяют задавать *шаблон* искомого текста. Вы можете не знать номер телефона компании, но если вы живете в США или Канаде, то знаете, что он будет включать три цифры, дефис и еще четыре цифры (иногда вначале указывается необязательный трехзначный код региона). Поэтому вы сразу понимаете, что 415-555-1234 —телефонный номер, а 4 155 551 234 — нет.

Мы ежедневно наблюдаем различные текстовые шаблоны: адреса электронной почты содержат символ @, номера социального страхования в США содержат девять цифр и два дефиса, URL-адреса сайтов содержат двоеточие и две косые черты, хештеги социальных сетей начинаются с # и не содержат пробелов и т.п.

Несмотря на безусловную пользу регулярных выражений, лишь немногие из тех, кто не являются программистами, знают, что это такое, притом что в большинстве современных текстовых процессоров, таких как Microsoft Word или OpenOffice, имеются средства поиска и замены на основе регулярных выражений. Регулярные выражения сэкономят массу времени не только пользователям, но и программистам. Более того, по мнению Кори Доктороу, изучение регулярных выражений должно предшествовать изучению самого программирования:

> "Знание [регулярных выражений] может означать разницу между решением задачи за 3 шага или за 3 тысячи шагов. Когда ты профессиональный программист, ты легко забываешь о том, что на задачу, которую ты решаешь нажатием нескольких клавиш, другие потратят несколько дней напряженной работы, чреватой внесением дополнительных ошибок"[1].

В этой главе мы сначала напишем программу для поиска образцов текста *без* использования регулярных выражений, а затем узнаем, как сделать то же самое с помощью регулярных выражений, получив гораздо более компактный код. Мы рассмотрим, как работают простые шаблоны на основе регулярных выражений, после чего мы займемся более сложными операциями, такими как замена строк и создание собственных символьных классов. В конце главы вам предстоит написать программу для автоматического извлечения телефонных номеров и адресов электронной почты из текста.

Поиск образцов текста без использования регулярных выражений

Предположим, требуется найти американский телефонный номер, содержащийся в строке. Исходный текстовый шаблон известен: три цифры, дефис, три цифры, дефис, четыре цифры (например, `415-555-4242`).

Чтобы проверить соответствие строки данному образцу, мы напишем функцию `isPhoneNumber()`, возвращающую `True` либо `False`. Откройте в файловом редакторе новое окно, введите в нем приведенный ниже код и сохраните его в файле *isPhoneNumber.py*.

[1] Cory Doctorow, "Here's what ICT should really teach kids: how to do regular expressions," Guardian, December 4, 2012, `http://www.theguardian.com/technology/2012/dec/04/ict-teach-kids-regular-expressions/`.

```
  def isPhoneNumber(text):
❶     if len(text) != 12:
          return False
      for i in range(0, 3):
❷         if not text[i].isdecimal():
              return False
❸     if text[3] != '-':
          return False
      for i in range(4, 7):
❹         if not text[i].isdecimal():
              return False
❺     if text[7] != '-':
          return False
      for i in range(8, 12):
❻         if not text[i].isdecimal():
              return False
❼     return True

  print('415-555-4242 - это телефонный номер?')
  print(isPhoneNumber('415-555-4242'))
  print('Moshi moshi - это телефонный номер?')
  print(isPhoneNumber('Moshi moshi'))
```

Запустив эту программу, вы получите следующие результаты.

```
415-555-4242 - это телефонный номер?
True
Moshi moshi - это телефонный номер?
False
```

В функции `isPhoneNumber()` выполняются несколько проверок, цель которых — выяснить, содержит ли переменная `text` строку телефонного номера в корректном формате. При отрицательном исходе любой из проверок функция возвращает значение `False`. Сначала проверяется, содержит ли строка в точности 12 символов ❶. Затем проверяется, состоит ли код региона (т.е. первые три символа) из одних цифр ❷. Остальные проверки таковы: за кодом региона должен идти первый дефис ❸, потом еще три цифры ❹, еще один дефис ❺ и еще четыре цифры ❻. Если все проверки пройдены успешно, функция возвращает значение `True` ❼.

При вызове функции `isPhoneNumber()` с аргументом `'415-555-4242'` возвращается `True`. Для аргумента `'Moshi moshi'` будет получено значение `False`: в данном случае не проходит самый первый тест, поскольку количество символов в строке `'Moshi moshi'` не равно 12.

Если нужно найти телефонный номер в более длинной строке, придется написать дополнительный код. Замените последние четыре вызова функции `print()` в файле *isPhoneNumber.py* следующим кодом.

```
  message = 'Позвони мне завтра по номеру 415-555-1011.
            415-555-9999 - это номер телефона моего офиса.'
  for i in range(len(message)):
❶     chunk = message[i:i+12]
❷     if isPhoneNumber(chunk):
          print('Найденный номер телефона: ' + chunk)
  print('Готово')
```

Запустив программу, вы получите следующее.

```
Найденный номер телефона: 415-555-1011
Найденный номер телефона: 415-555-9999
Готово
```

На каждой итерации цикла `for` в переменную `chunk` записываются очередные 12 последовательных символов строки `message` ❶. Например, на первой итерации `i` равно `0`, а значит, переменной `chunk` присваивается срез `message[0:12]` (т.е. строка `'Позвони мне '`). На следующей итерации `i` равно `1`, и переменной `chunk` присваивается срез `message[1:13]` (т.е. строка `'озвони мне з'`) и т.д.

Каждый такой фрагмент передается функции `isPhoneNumber()`, чтобы проверить, соответствует ли он образцу телефонного номера ❷, и если результат проверки оказывается положительным, то найденный номер выводится на экран.

В цикле просматривается вся строка, и выводятся те 12-символьные фрагменты, которые удовлетворяют условиям, проверяемым в функции `isPhoneNumber()`. В конце выводится слово `'Готово'`.

В данном примере строка, содержащаяся в переменной `message`, очень короткая, но в других случаях она может содержать миллионы символов. Тем не менее программа выполнится менее чем за секунду. Аналогичная программа, выполняющая поиск телефонных номеров с помощью регулярных выражений, также будет выполняться менее секунды, зато регулярные выражения упрощают написание подобных программ.

Поиск образцов текста с помощью регулярных выражений

Предыдущая программа нахождения телефонных номеров вполне функциональна, но при этом обладает достаточно ограниченными возможностями: функция `isPhoneNumber()`, содержащая 17 строк кода, способна находить телефонные номера, соответствующие лишь одному шаблону. Что, если телефонный номер будет записан в формате 415.555.4242 или (415) 555-4242? Что, если телефонный номер включает добавочный номер,

например 415-555-4242 x99? Функция `isPhoneNumber()` не в состоянии распознать подобные номера. Для учета других возможных шаблонов придется писать дополнительный код, однако существует более простой способ.

Компактное описание текстовых шаблонов можно создавать с помощью *регулярных выражений*. Например, регулярному выражению `\d` соответствует любой цифровой символ, т.е. любая одиночная цифра от `0` до `9`. Регулярное выражение `\d\d\d-\d\d\d-\d\d\d\d` позволяет находить текстовые строки того же формата, что и в функции `isPhoneNumber()`: строка из трех цифр, дефис, еще три цифры, другой дефис и еще четыре цифры. Никакая другая строка не будет соответствовать регулярному выражению `\d\d\d-\d\d\d-\d\d\d\d`.

В то же время регулярные выражения могут быть гораздо более сложными. Например, указав `3` в фигурных скобках после шаблона (`{3}`), мы сообщаем следующее: "Искать троекратное соответствие данному шаблону". Поэтому корректному телефонному номеру будет соответствовать следующий короткий шаблон: `\d{3}-\d{3}-\d{4}`.

Создание объектов Regex

В Python все функции, предназначенные для работы с регулярными выражениями, содержатся в модуле `re`. Введите в интерактивной оболочке следующую инструкцию, чтобы импортировать этот модуль:

```
>>> import re
```

Примечание

В большинстве примеров главы требуется модуль `re`*, поэтому не забывайте импортировать его в начале любого сценария или при каждом перезапуске IDLE. В противном случае вы получите сообщение об ошибке* `NameError: name 're' is not defined`*.*

Функция `re.compile()` возвращает объект `Regex`, соответствующий переданной строке регулярного выражения.

Чтобы создать объект `Regex`, соответствующий шаблону телефонного номера, введите в интерактивной оболочке следующую инструкцию (вспомните, что выражение `\d` означает "цифровой символ"):

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
```

Теперь в переменной `phoneNumRegex` содержится объект `Regex`.

Поиск соответствий объектам Regex

Метод `search()` объекта `Regex` ищет в переданной ему строке любые совпадения с регулярным выражением. В случае отсутствия совпадений возвращается значение `None`. Если совпадения обнаружены, то возвращается объект `Match`. У такого объекта есть метод `group()`, который возвращает найденные соответствия шаблону (о том, что такое группы, будет рассказано далее). В качестве примера введите в интерактивной оболочке следующие инструкции.

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
>>> mo = phoneNumRegex.search('Мой номер: 415-555-4242.')
>>> print('Найденный номер телефона: ' + mo.group())
Найденный номер телефона: 415-555-4242
```

В данном случае переменная `mo` содержит объект `Match`. Поначалу пример может показаться сложноватым, но он значительно короче приведенной ранее программы *isPhoneNumber.py*, хотя делает то же самое.

В этом примере мы передаем требуемый шаблон методу `re.compile()` и сохраняем полученный объект `Regex` в переменной `phoneNumRegex`. Затем для этой переменной вызывается метод `search()`, получающий строку, в которой необходимо найти соответствия шаблону. Результат поиска сохраняется в переменной `mo`. Нам заранее известно, что искомый образец содержится в строке, поэтому метод `search()` вернет объект `Match`. Зная, что переменная `mo` содержит объект `Match`, а не значение `None`, мы можем вызвать для нее метод `group()`, возвращающий найденное соответствие. В результате функция `print()` отображает искомый телефонный номер `415-555-4242`.

Пошаговая процедура

Процедура использования регулярных выражений в Python достаточно проста.

1. Импортируйте модуль регулярных выражений с помощью инструкции `import re`.
2. Создайте объект `Regex` с помощью функции `re.compile()`. (Ему должна быть передана необработанная строка поискового шаблона регулярного выражения.)
3. Передайте строку, в которой выполняется поиск, методу `search()` объекта `Regex`. Этот метод возвращает объект `Match`.
4. Вызовите метод `group()` объекта `Match`, чтобы получить строку, которая содержит найденный текст, соответствующий заданному регулярному выражению.

Примечание

Помимо выполнения кода примеров в интерактивной оболочке я рекомендую использовать доступные в Интернете тестировщики регулярных выражений, которые позволяют проверить соответствие введенного текста заданному шаблону. В частности, удобный тестировщик доступен на сайте `https://pythex.org/`.

Другие шаблоны регулярных выражений

Теперь, когда вы знаете, как создавать объекты регулярных выражений и находить совпадения с шаблонами, рассмотрим расширенные возможности поиска.

Создание групп с помощью круглых скобок

Предположим, вы хотите отделить код региона от остальной части телефонного номера. Добавление круглых скобок приводит к созданию *групп* в регулярном выражении: `(\d\d\d)-(\d\d\d-\d\d\d\d)`. Теперь можно использовать метод `group()` объекта `Match` для получения текста, соответствующего только одной группе.

Первый набор круглых скобок в строке регулярного выражения будет группой 1, второй набор — группой 2 и т.д. Передавая *целые* числа `1` или `2` методу `group()`, вы сможете отбирать различные фрагменты совпавшего текста. Если метод `group()` вызывается с аргументом `0` или вообще без аргументов, то он возвращает весь найденный текст, соответствующий шаблону. Введите в интерактивной оболочке следующие инструкции.

```
>>> phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')
>>> mo = phoneNumRegex.search('Мой номер: 415-555-4242.')
>>> mo.group(1)
'415'
>>> mo.group(2)
'555-4242'
>>> mo.group(0)
'415-555-4242'
>>> mo.group()
'415-555-4242'
```

Если нужно извлечь сразу все группы, используйте метод `groups()`.

```
>>> mo.groups()
('415', '555-4242')
>>> areaCode, mainNumber = mo.groups()
>>> print(areaCode)
415
>>> print(mainNumber)
555-4242
```

Поскольку метод `mo.groups()` возвращает кортеж, состоящий из нескольких значений, можно использовать операцию группового присваивания, как в инструкции `areaCode, mainNumber = mo.groups()`.

Круглые скобки трактуются как спецсимволы в регулярных выражениях, но что если нужно найти в тексте сами скобки? Например, в телефонных номерах круглые скобки часто используются для выделения кода региона. В таких случаях символы `(` и `)` должны экранироваться с помощью обратной косой черты. Введите в интерактивной оболочке следующие инструкции.

```
>>> phoneNumRegex = re.compile(r'(\(\d\d\d\)) (\d\d\d-\d\d\d\d)')
>>> mo = phoneNumRegex.search('Мой номер: (415) 555-4242.')
>>> mo.group(1)
'(415)'
>>> mo.group(2)
'555-4242'
```

Экранированные символы `\(` и `\)` в необработанной строке, передаваемой методу `re.compile()`, означают соответствие фактическим символам круглых скобок. В регулярных выражениях следующие символы имеют специальное значение:

```
. ^ $ * + ? { } [ ] \ | ( )
```

Если требуется найти их в текстовом шаблоне, нужно экранировать их с омощью обратной косой черты:

```
\. \^ \$ \* \+ \? \{ \} \[ \] \\ \| \( \)
```

Следите за тем, чтобы по ошибке не принять скобки `\(` и `\)` за спецсимволы `(` и `)`. Если будет получено сообщение об ошибке `"missing )"` или `"unbalanced parenthesis"`, то, скорее всего, вы просто забыли включить закрывающую неэкранированную скобку для группы, как в следующем примере.

```
>>> re.compile(r'(\(Parentheses\)')
Traceback (most recent call last):
  --Опущено--
re.error: missing ), unterminated subpattern at position 0
```

Сообщение об ошибке говорит о том, что в позиции `0` строки `r'(\(Parentheses\)'` стоит открывающая скобка, для которой отсутствует закрывающая скобка.

Выбор альтернативных групп с помощью канала

Символ | в регулярном выражении называется *каналом* (pipe). Его можно использовать, когда требуется найти соответствие одному из нескольких альтернативных выражений. Например, регулярному выражению `r'Бэтмен|Тина Фей'` будут соответствовать как строка `'Бэтмен'`, так и строка `'Тина Фей'`.

Если в тексте найдены *обе* эти строки, то в объект `Match` будет записана та из них, которая встретится первой. Введите в интерактивной оболочке следующие инструкции.

```
>>> heroRegex = re.compile (r'Бэтмен|Тина Фей')
>>> mo1 = heroRegex.search('Бэтмен и Тина Фей.')
>>> mo1.group()
'Бэтмен'

>>> mo2 = heroRegex.search('Тина Фей и Бэтмен.')
>>> mo2.group()
'Тина Фей'
```

Примечание

Для поиска всех совпадений с шаблоном можно использовать метод `findall()`, который обсуждается далее.

С помощью канала удобно выбирать альтернативные варианты при поиске совпадений. Предположим, к примеру, что ищется совпадение с любой из следующих строк: `'Бэтмен'`, `'Бэтмобиль'`, `'Бэткоптер'` и `'Бэтбэт'`. Поскольку все они начинаются с `'Бэт'`, желательно задать префикс лишь один раз. Это можно сделать с помощью круглых скобок. Введите в интерактивной оболочке следующие инструкции.

```
>>> batRegex = re.compile(r'Бэт(мен|мобиль|коптер|бэт)')
>>> mo = batRegex.search('Бэтмобиль потерял колесо')
>>> mo.group()
'Бэтмобиль'
>>> mo.group(1)
'мобиль'
```

Метод `mo.group()` возвращает весь совпавший с шаблоном текст, т.е. строку `'Бэтмобиль'`, тогда как метод `mo.group(1)` возвращает лишь фрагмент совпавшего текста, соответствующий первой группе в круглых скобках, т.е. `'мобиль'`. Используя символ канала и группирующие скобки, можно задать несколько альтернативных шаблонов для поиска соответствий с помощью единственного регулярного выражения.

Если требуется найти в строке сам символ канала, то его необходимо экранировать с помощью обратной косой черты: \|.

Указание необязательной группы с помощью вопросительного знака

Иногда встречаются шаблоны, содержащие необязательные символы. Другими словами, регулярное выражение должно найти совпадение независимо от того, содержится ли в строке определенный фрагмент текста. Символ ? означает, что предшествующая ему группа представляет собой необязательную часть поискового шаблона. Введите в интерактивной оболочке следующие инструкции.

```
>>> batRegex = re.compile(r'Бэт(ву)?мен')
>>> mo1 = batRegex.search('Мой герой - Бэтмен')
>>> mo1.group()
'Бэтмен'

>>> mo2 = batRegex.search('Моя героиня - Бэтвумен')
>>> mo2.group()
'Бэтвумен'
```

Часть (ву)? регулярного выражения означает, что шаблон 'ву' — это необязательная группа. Регулярному выражению будет соответствовать текст, в котором подстрока 'ву' либо вообще не встречается, либо встречается один раз. Именно поэтому при поиске регулярного выражения находится как слово 'Бэтвумен', так и слово 'Бэтмен'.

В примере с телефонным номером можно составить регулярное выражение, позволяющее находить номера, которые могут как содержать, так и не содержать код региона. Введите в интерактивной оболочке следующие инструкции.

```
>>> phoneRegex = re.compile(r'(\d\d\d-)?\d\d\d-\d\d\d\d')
>>> mo1 = phoneRegex.search('Мой номер: 415-555-4242')
>>> mo1.group()
'415-555-4242'

>>> mo2 = phoneRegex.search('Мой номер: 555-4242')
>>> mo2.group()
'555-4242'
```

Символ ? имеет следующий смысл: "Искать соответствие тексту, в котором группа, предшествующая вопросительному знаку, встречается нуль или один раз".

Если требуется найти в строке сам вопросительный знак, то его необходимо экранировать с помощью обратной косой черты: \?.

Указание группы, повторяющейся нуль или несколько раз, с помощью звездочки

Символ * (звездочка) означает "найти нулевое или большее количество экземпляров", т.е. группа, предшествующая звездочке, может встречаться в тексте любое количество раз. Она может либо вообще отсутствовать, либо повторяться снова и снова. Вернемся к примеру с Бэтменом.

```
>>> batRegex = re.compile(r'Бэт(ву)*мен')
>>> mo1 = batRegex.search('Мой герой - Бэтмен')
>>> mo1.group()
'Бэтмен'

>>> mo2 = batRegex.search('Моя героиня - Бэтвумен')
>>> mo2.group()
'Бэтвумен'

>>> mo3 = batRegex.search('Моя героиня - Бэтвувувувумен')
>>> mo3.group()
'Бэтвувувувумен'
```

В случае слова `'Бэтмен'` часть `(ву)*` регулярного выражения соответствует нулевому количеству (т.е. отсутствию) экземпляров группы `'ву'` в строке. В случае слова `'Бэтвумен'` часть `(ву)*` совпадает с одним экземпляром `'ву'`, а в случае слова `'Бэтвувувувумен'` часть `(ву)*` совпадает с четырьмя экземплярами `'ву'`.

Если требуется найти в строке сам символ звездочки, то его необходимо экранировать с помощью обратной косой черты: `\*`.

Указание группы, повторяющейся один или несколько раз, с помощью знака "плюс"

Если символ * в регулярном выражении означает совпадение с нулевым или большим количеством экземпляров, то символ + означает совпадение с единичным или большим количеством экземпляров. В отличие от символа *, который не требует появления предшествующей ему группы в исследуемой строке, группа, предшествующая знаку +, должна появиться в строке *хотя бы один раз*. Такая группа не является необязательной. Введите в интерактивной оболочке следующие инструкции и сравните полученные результаты с результатами из предыдущего раздела.

```
>>> batRegex = re.compile(r'Бэт(ву)+мен')
>>> mo1 = batRegex.search('Моя героиня - Бэтвумен')
>>> mo1.group()
'Бэтвумен'
```

```
>>> mo2 = batRegex.search('Моя героиня - Бэтвувувувумен')
>>> mo2.group()
'Бэтвувувувумен'

>>> mo3 = batRegex.search('Мой герой - Бэтмен')
>>> mo3 == None
True
```

Регулярное выражение `Бэт(ву)+мен` не находит соответствия в строке `'Мой герой - Бэтмен'`, поскольку символ + требует наличия по крайней мере одного экземпляра подстроки `'ву'`.

Если требуется найти в строке сам символ "плюс", то его необходимо экранировать с помощью обратной косой черты: `\+`.

Указание количества повторений с помощью фигурных скобок

Если имеется группа, которая должна повторяться определенное количество раз, укажите за ней число повторений в фигурных скобках. Например, регулярному выражению `(На){3}` будет соответствовать строка `'НаНаНа'` (но не строка `'НаНа'`, поскольку в этом случае группа `(На)` повторяется всего два раза).

Вместо одного числа можно указать диапазон, записав в фигурных скобках минимальное и максимальное число допустимых повторений. Например, регулярному выражению `(На){3, 5}` будут соответствовать строки `'НаНаНа'`, `'НаНаНаНа'` и `'НаНаНаНаНа'`.

Как первое, так и второе из чисел в фигурных скобках можно опустить, оставив минимальное или максимальное количество повторений неограниченным. Например, регулярному выражению `(На){3, }` будут соответствовать три и более экземпляров группы `(На)`, тогда как регулярному выражению `(На){, 5}` будут соответствовать от нуля до пяти экземпляров. Фигурные скобки позволяют записывать регулярные выражения в более компактном виде. Следующие два регулярных выражения идентичны.

```
(На){3}
(На)(На)(На)
```

Приведенные ниже регулярные выражения тоже идентичны.

```
(На){3, 5}
((На)(На)(На))|((На)(На)(На)(На))|((На)(На)(На)(На)(На))
```

Введите в интерактивной оболочке следующие инструкции.

```
>>> haRegex = re.compile(r'(Ha){3}')
>>> mo1 = haRegex.search('HaHaHa')
>>> mo1.group()
'HaHaHa'

>>> mo2 = haRegex.search('Ha')
>>> mo2 == None
True
```

Здесь регулярное выражение `(Ha){3}` совпадает со строкой `'HaHaHa'`, но не со строкой `'Ha'`, поэтому во втором случае метод `search()` возвращает значение `None`.

Жадный и нежадный виды поиска

Поскольку регулярному выражению `(Ha){3, 5}` могут соответствовать три, четыре или пять вхождений подстроки `'Ha'` в строке `'HaHaHaHaHa'`, вас может удивить, почему вызов метода `group()` объекта `Match` в таком случае возвращает строку `'HaHaHaHaHa'`, а не ее более короткие варианты. В конце концов, строки `'HaHaHa'` и `'HaHaHaHa'` тоже соответствуют поисковому шаблону `(Ha){3, 5}`.

Регулярные выражения Python по умолчанию *жадные* в том смысле, что в неоднозначных ситуациях они будут пытаться соответствовать как можно более длинной строке. *Нежадная* версия выражения с фигурными скобками, которая пытается соответствовать самой короткой из возможных строк, помечается вопросительным знаком после закрывающей фигурной скобки.

Введите в интерактивной оболочке следующие инструкции и обратите внимание на различия между жадным и нежадным видами поиска при работе с одной и той же тестируемой строкой.

```
>>> greedyHaRegex = re.compile(r'(Ha){3, 5}')
>>> mo1 = greedyHaRegex.search('HaHaHaHaHa')
>>> mo1.group()
'HaHaHaHaHa'

>>> nongreedyHaRegex = re.compile(r'(Ha){3, 5}?')
>>> mo2 = nongreedyHaRegex.search('HaHaHaHaHa')
>>> mo2.group()
'HaHaHa'
```

Следует учитывать, что в регулярных выражениях вопросительный знак имеет двойное назначение: он может обозначать как нежадный поиск, так и необязательную группу. Эти две его функции никак не связаны между собой.

Метод findall()

Помимо метода search(), у объектов Regex есть метод findall(). Если метод search() возвращает объект Match *первого* совпадения, найденного в тестируемой строке, то метод findall() возвращает строки *каждого* из найденных совпадений. Чтобы увидеть, как работает метод search(), введите в интерактивной оболочке следующие инструкции.

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
>>> mo = phoneNumRegex.search('Мобильный: 415-555-9999 \
...                           Рабочий: 212-555-0000')
>>> mo.group()
'415-555-9999'
```

В то же время метод findall() возвращает не объект Match, а список строк, *если в регулярном выражении отсутствуют группы.* Введите в интерактивной оболочке следующие инструкции.

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d') # нет групп
>>> phoneNumRegex.findall('Мобильный: 415-555-9999 \
...                       Рабочий: 212-555-0000')
['415-555-9999', '212-555-0000']
```

При наличии групп метод findall() вернет список кортежей. Каждый кортеж представляет найденное совпадение, а его элементы — совпавшие строки для каждой группы в регулярном выражении. Чтобы увидеть, как работает метод findall(), введите в интерактивной оболочке следующие инструкции (обратите внимание на то, что теперь в компилируемом регулярном выражении имеются группы, заключенные в круглые скобки).

```
>>> phoneNumRegex =
    re.compile(r'(\d\d\d)-(\d\d\d)-(\d\d\d\d)') # есть группы
>>> phoneNumRegex.findall('Мобильный: 415-555-9999 \
...                       Рабочий: 212-555-0000')
[('415', '555', '1122'), ('212', '555', '0000')]
```

Вкратце подытожим работу метода findall().

- Будучи вызванным для регулярного выражения, не содержащего групп, например \d\d\d-\d\d\d-\d\d\d\d, метод findall() возвращает список совпавших строк: ['415-555-9999', '212-555-0000'].
- Будучи вызванным для регулярного выражения, содержащего группы, например (\d\d\d)-(\d\d\d)-(\d\d\d\d), метод findall() возвращает список строковых кортежей (по одной строке для каждой группы): [('415', '555', '1122'), ('212','555','0000')].

Символьные классы

Из предыдущих примеров вам уже известно, что символ \d означает любую цифру. Другими словами, \d — это сокращенное обозначение регулярного выражения (0|1|2|3|4|5|6|7|8|9). Аналогичные сокращения существуют для многих других *символьных классов* (табл. 7.1).

Таблица 7.1. Сокращенные обозначения распространенных символьных классов

Сокращение	Представляемые символы
\d	Любая цифра в диапазоне от 0 до 9
\D	Любой символ, *не* являющийся цифрой в диапазоне от 0 до 9
\w	Любая буква, цифра или символ подчеркивания
\W	Любой символ, *не* являющийся буквой, цифрой или символом подчеркивания
\s	Пробел, табуляция или символ новой строки (так называемые *пробельные* символы)
\S	Любой символ, *не* являющийся пробелом, табуляцией или символом новой строки

Символьные классы удобно использовать для компактной записи регулярных выражений. Например, классу [0-5] будет соответствовать любая одиночная цифра в диапазоне от 0 до 5. Это намного короче, чем вводить (0|1|2|3|4|5). Отметим, что не существует символьного класса только для букв. Можно разве что использовать запись [a-zA-Z], как будет объясняться далее.

Введите в интерактивной оболочке следующие инструкции.

```
>>> xmasRegex = re.compile(r'\d+\s\w+')
>>> xmasRegex.findall('12 барабанщиков, 11 волынщиков, 10 лордов,
 9 леди, 8 горничных, 7 лебедей, 6 гусей, 5 колец, 4 птицы,
 3 курицы, 2 голубя, 1 куропатка')

['12 барабанщиков', '11 волынщиков', '10 лордов', '9 леди',
 '8 горничных', 7 лебедей', '6 гусей', '5 колец', '4 птицы',
 '3 курицы', '2 голубя', '1 куропатка']
```

Регулярному выражению \d+\s\w+ будет соответствовать текст, содержащий одну или несколько цифр (\d+), за которыми следует пробельный символ (\s), а за ним — один или несколько алфавитно-цифровых символов: буква, цифра или символ подчеркивания (\w+). Метод findall() возвращает все совпавшие строки шаблона регулярного выражения в виде списка.

Создание собственных символьных классов

Иногда возникает необходимость сопоставить регулярное выражение символам из определенного набора, для которого сокращенные

символьные классы (\d, \w, \s и т.п.) оказываются слишком широкими. В таком случае можно определить собственный символьный класс, используя квадратные скобки. Например, символьному классу [aeiouAEIOU] будет соответствовать любая гласная буква в нижнем или верхнем регистре. Введите в интерактивной оболочке следующие инструкции.

```
>>> vowelRegex = re.compile(r'[aeiouAEIOU]')
>>> vowelRegex.findall('RoboCop eats baby food. BABY FOOD.')
['o', 'o', 'o', 'e', 'a', 'a', 'o', 'o', 'A', 'O', 'O']
```

В классы можно также включать диапазоны букв и цифр, используя дефис. Например, классу [a-zA-Z0-9] будут соответствовать все буквы в нижнем и верхнем регистрах, а также цифры.

Учтите, что внутри квадратных скобок обычные символы регулярных выражений как таковые не интерпретируются. Это означает, что перед символами ., *, ?, (и) не следует ставить обратную косую черту. Например, классу [0-5.] будут соответствовать цифры от 0 до 5 и точка. Не следует записывать этот класс как [0-5\.].

Поместив сразу за открывающей квадратной скобкой символ ^, можно создать *инвертированный символьный класс*. Такому классу будет соответствовать любой символ, *не* входящий в исходный класс. Например, введите в интерактивной оболочке следующие инструкции.

```
>>> consonantRegex = re.compile(r'[^aeiouAEIOU]')
>>> consonantRegex.findall('RoboCop eats baby food. BABY FOOD.')
['R', 'b', 'c', 'p', ' ', 't', 's', ' ', 'b', 'b', 'y', ' ', 'f',
'd', '.', ' ', 'B', 'B', 'Y', ' ', 'F', 'D', '.']
```

Теперь регулярному выражению будут соответствовать все символы, не являющиеся гласными.

Символ ^ и знак доллара

Поставив в начале регулярного выражения символ ^ (карет), вы тем самым указываете, что соответствие регулярному выражению следует искать в *начале* текста. Аналогичным образом знак доллара ($) в конце регулярного выражения указывает на то, что строка должна *заканчиваться* данным шаблоном регулярного выражения. Можно также использовать символы ^ и $ совместно. Это означает, что данному регулярному выражению должна соответствовать вся строка, т.е. совпадения подстроки будет недостаточно.

Например, регулярному выражению r'^Здравствуй' будут соответствовать строки, начинающиеся со слова 'Здравствуй'. Введите в интерактивной оболочке следующие инструкции.

```
>>> beginsWithHello = re.compile(r'^Здравствуй')
>>> beginsWithHello.search('Здравствуй, мир!')
<re.Match object; span=(0, 10), match='Здравствуй'>
>>> beginsWithHello.search('Он сказал здравствуй.') == None
True
```

Регулярному выражению `r'\d$'` будут соответствовать строки, оканчивающиеся любой цифрой в диапазоне от `0` до `9`. Введите в интерактивной оболочке следующие инструкции.

```
>>> endsWithNumber = re.compile(r'\d$')
>>> endsWithNumber.search('Ваше число - 42')
<re.Match object; span=(14, 15), match='2'>
>>> endsWithNumber.search('Ваше число - сорок два.') == None
True
```

Регулярному выражению `r'^\d+$'` будут соответствовать строки, которые состоят из одной или нескольких цифр. Введите в интерактивной оболочке следующие инструкции.

```
>>> wholeStringIsNum = re.compile(r'^\d+$')
>>> wholeStringIsNum.search('1234567890')
<re.Match object; span=(0, 10), match='1234567890'>
>>> wholeStringIsNum.search('12345xyz67890') == None
True
>>> wholeStringIsNum.search('12 34567890') == None
True
```

Последние два вызова метода `search()` показывают, что при одновременном использовании символов `^` и `$` вся строка должна соответствовать регулярному выражению.

Символ подстановки

Символ `.` (точка) в регулярных выражениях называется *символом подстановки* и представляет собой сокращенную форму записи символьного класса, совпадающего с любым одиночным символом, за исключением символа новой строки. Например, введите в интерактивной оболочке следующие инструкции.

```
>>> atRegex = re.compile(r'.at')
>>> atRegex.findall('The cat in the hat sat on the flat mat.')
['cat', 'hat', 'sat', 'lat', 'mat']
```

Символу подстановки соответствует только один символ, поэтому в слове 'flat' была выбрана подстрока 'lat'. Если требуется выполнить поиск самой точки, то ее необходимо экранировать с помощью обратной косой черты: \..

Поиск любого текста с помощью комбинации "точка – звездочка"

Иногда нужно найти соответствие произвольному тексту. Допустим, вы ищете строку 'Имя:', за которой следует некий текст, за ним – строка 'Фамилия:', а за ней – снова некий текст. Шаблону "некий текст" соответствует комбинация "точка – звездочка" (.*). Вспомните о том, что символ . означает "любой одиночный символ, за исключением символа новой строки", а символ * означает "нуль или более вхождений предыдущего символа".

Введите в интерактивной оболочке следующие инструкции.

```
>>> nameRegex = re.compile(r'Имя: (.*) Фамилия: (.*)')
>>> mo = nameRegex.search('Имя: Эл Фамилия: Свейгарт')
>>> mo.group(1)
'Эл'
>>> mo.group(2)
'Свейгарт'
```

Комбинация .* работает в режиме *жадного поиска*: она всегда будет стремиться захватить как можно больше текста. Чтобы заставить ее работать в нежадном режиме, дополните ее вопросительным знаком: .*?. Как и в случае фигурных скобок, вопросительный знак означает использование *нежадного* поиска.

Чтобы увидеть, чем различаются жадная и нежадная версии регулярного выражения, введите в интерактивной оболочке следующие инструкции.

```
>>> nongreedyRegex = re.compile(r'<.*?>')
>>> mo = nongreedyRegex.search('<Приготовить мужу> ужин.>')
>>> mo.group()
'<Приготовить мужу>'

>>> greedyRegex = re.compile(r'<.*>')
>>> mo = greedyRegex.search('<Приготовить мужу> ужин.>')
>>> mo.group()
'<Приготовить мужу> ужин.>'
```

Смысл обоих регулярных выражений можно описать так: "Найти открывающую угловую скобку, за которой следует произвольный текст, завершающийся закрывающей угловой скобкой". Однако в строке '<Приготовить мужу> ужин.>' имеются две закрывающие угловые скобки. В нежадной версии регулярного выражения Python ограничивается самой короткой из

возможных строк: '<Приготовить мужу>'. В жадной версии Python стремится к тому, чтобы найти совпадение в виде как можно более длинной строки из всех возможных: '<Приготовить мужу> ужин.>'.

Поиск символов новой строки с помощью точки

Комбинации "точка – звездочка" будет соответствовать все, за исключением символа новой строки. Передав методу `re.compile()` дополнительный аргумент `re.DOTALL`, можно установить режим, при котором точке соответствуют *все* символы, включая символ новой строки.

Введите в интерактивной оболочке следующие инструкции.

```
>>> noNewlineRegex = re.compile('.*')
>>> noNewlineRegex.search('Служить обществу.\nЗащищать невиновных.\n
Соблюдать закон.').group()
'Служить обществу.'

>>> newlineRegex = re.compile('.*', re.DOTALL)
>>> newlineRegex.search('Служить обществу.\nЗащищать невиновных.\n
Соблюдать закон.').group()
'Служить обществу.\nЗащищать невиновных.\nСоблюдать закон.'
```

Регулярному выражению `noNewlineRegex`, которое компилировалось без аргумента `re.DOTALL`, будет соответствовать весь текст, но только до первого символа новой строки, тогда как регулярному выражению `newlineRegex`, которое компилировалось с аргументом `re.DOTALL`, будет соответствовать весь текст. Именно поэтому метод `newlineRegex.search()` возвращает всю строку целиком, включая символы новой строки.

Сводка синтаксиса регулярных выражений

Мы уже рассмотрели множество элементов регулярных выражений, поэтому имеет смысл привести их полный перечень.

- `?` – нулевое или единичное вхождение предшествующей группы.
- `*` – нулевое или произвольное количество вхождений предшествующей группы.
- `+` – одно или несколько вхождений предшествующей группы.
- `{n}` – ровно *n* вхождений предшествующей группы.
- `{n, }` – *n* или более вхождений предшествующей группы.
- `{, m}` – отсутствие или вплоть до *m* вхождений предшествующей группы.
- `{n, m}` – не менее чем *n* и не более чем *m* вхождений предшествующей группы.

- `{n, m}?`, или `*?`, или `+?` — нежадный поиск вхождений предшествующей группы.
- `^spam` — строка должна начинаться символами `'spam'`.
- `spam$` — строка должна заканчиваться символами `'spam'`.
- `.` — любой символ, за исключением символа новой строки.
- `\d`, `\w` и `\s` — одиночный цифровой, алфавитно-цифровой и пробельный символ соответственно.
- `\D`, `\W` и `\S` — одиночный символ, не являющийся цифровым, алфавитно-цифровым и пробельным соответственно.
- `[abc]` — любой одиночный символ из числа тех, которые указаны в квадратных скобках (например, `'a'`, `'b'` или `'c'`).
- `[^abc]` — любой одиночный символ, кроме тех, которые указаны в квадратных скобках.

Поиск без учета регистра

Обычно поиск регулярного выражения проводится с учетом регистра. Например, следующим поисковым шаблонам будут соответствовать разные строки.

```
>>> regex1 = re.compile('РобоКоп')
>>> regex2 = re.compile('РОБОКОП')
>>> regex3 = re.compile('робОкоп')
>>> regex4 = re.compile('РобокОп')
```

Но иногда нас интересует лишь сам факт совпадения букв, независимо от их регистра. Можно включить режим, в котором регистр букв игнорируется, передав методу `re.compile()` дополнительный аргумент `re.IGNORECASE` или `re.I`. Введите в интерактивной оболочке следующие инструкции.

```
>>> robocop = re.compile(r'робокоп', re.I)
>>> robocop.search('РобоКоп - это полицейский, частично человек,
частично машина.').group()
'РобоКоп'

>>> robocop.search('РОБОКОП защищает невиновных.').group()
'РОБОКОП'

>>> robocop.search('Эл, почему в твоей книге по
программированию так часто упоминается робокоп?').group()
'робокоп'
```

Замена строк с помощью метода sub()

С помощью регулярных выражений можно не только находить заданные образцы текста, но и заменять их новым текстом. У объектов Regex есть метод sub(), имеющий два аргумента. Первый аргумент – это строка, которая должна подставляться вместо найденных совпадений. Второй аргумент – это строка, в которой выполняется поиск регулярного выражения. Метод sub() возвращает строку, в которой выполнены замены.

Введите в интерактивной оболочке следующие инструкции.

```
>>> namesRegex = re.compile(r'агент \w+', re.I)
>>> namesRegex.sub('ЦЕНЗУРА', 'Агент Алиса передала секретные
документы. Агент Боб.')
'ЦЕНЗУРА передала секретные документы. ЦЕНЗУРА.'
```

Иногда сам найденный текст должен включаться в строку замены. Для этого в первом аргументе метода sub() можно использовать ссылки \1, \2, \3 и т.д., которые означают следующее: "Вставить в подстановочный текст группы 1, 2, 3 и т.д."

Предположим, вы хотите скрыть в тексте имена секретных агентов, показав лишь первые буквы. Для этого можно воспользоваться регулярным выражением агент (\w)\w* и передать методу sub() в качестве первого аргумента строку r'\1****'. Ссылка \1 в этой строке будет заменяться тем текстом, который будет захвачен группой 1, т.е. группой (\w) регулярного выражения.

```
>>> agentNamesRegex = re.compile(r'агент (\w)\w*', re.I)
>>> agentNamesRegex.sub(r'\1****', 'Агент Алиса передала, что агент
Ева знает: агент Боб - двойной агент.')
А**** передала, что Е**** знает: Б**** - двойной агент.'
```

Работа со сложными регулярными выражениями

С регулярными выражениями легко работать в случае простых текстовых шаблонов. Но для поиска более сложных шаблонов могут требоваться длинные и запутанные регулярные выражения. Один из способов внесения ясности в подобных ситуациях заключается в том, чтобы заставить метод re.compile() игнорировать пробелы и комментарии в строке регулярного выражения. Для этого следует передать методу re.compile() дополнительный аргумент re.VERBOSE.

Предположим, имеется следующее головоломное регулярное выражение:

```
phoneRegex = re.compile(r'((\d{3}|\(\d{3}\))?(\s|-|\.)?\d{3}
(\s|-|\.)\d{4}(\s*(ext|x|ext.)\s*\d{2,5})?)')
```

Его можно разбить на несколько строк и снабдить комментариями, благодаря которым понять смысл регулярного выражения станет значительно легче.

```
phoneRegex = re.compile(r'''(
    (\d{3}|\(\d{3}\))?              # код региона
    (\s|-|\.)?                      # разделитель
    \d{3}                           # первые 3 цифры
    (\s|-|\.)                       # разделитель
    \d{4}                           # последние 4 цифры
    (\s*(ext|x|ext.)\s*\d{2, 5})?   # добавочный номер
    )''', re.VERBOSE)
```

Обратите внимание на то, что строка регулярного выражения взята в тройные кавычки (`'''`). Это позволило разбить текст на несколько строк, благодаря чему его стало намного легче читать.

Для комментариев в пределах строки регулярного выражения действуют те же правила, что и в коде Python: символ # и весь текст до конца строки игнорируются. Кроме того, дополнительные пробелы в многострочном регулярном выражении не считаются частью поискового шаблона. Это позволяет записывать регулярное выражение таким образом, чтобы его было легче читать.

Комбинация констант `re.IGNORECASE`, `re.DOTALL` и `re.VERBOSE`

Что, если вы хотите включить режим `re.VERBOSE` для добавления комментариев в регулярное выражение, и при этом необходимо игнорировать регистр символов с помощью константы `re.IGNORECASE`? К сожалению, у метода `re.compile()` только два аргумента. Это ограничение можно обойти, объединив константы `re.IGNORECASE`, `re.DOTALL` и `re.VERBOSE` с помощью символа `|`, который в данном контексте является оператором *побитового ИЛИ*.

Таким образом, если требуется, чтобы регулярное выражение игнорировало регистр, а символам точки в нем соответствовали бы также символы новой строки, вызовите метод `re.compile()` следующим образом.

```
>>> someRegexValue = re.compile('foo', re.IGNORECASE | re.DOTALL)
```

А вот пример объединения всех трех констант во втором аргументе.

```
>>> someRegexValue = re.compile('foo', re.IGNORECASE | re.DOTALL |
                                re.VERBOSE)
```

Этот синтаксис немного старомоден и унаследован от ранних версий Python. Детальное рассмотрение побитовых операторов выходит за рамки книги. Вы сможете найти более подробную информацию о них на сайте `https://wiki.python.org/moin/BitwiseOperators/`. Кроме того, во втором аргументе могут использоваться и другие константы компиляции (см. `https://docs.python.org/3/howto/regex.html`).

Проект: извлечение телефонных номеров и адресов электронной почты

Предположим, вам предстоит заняться рутинной работой — найти все телефонные номера и адреса электронной почты, которые содержатся на длинной веб-странице или в большом документе. Если прокручивать страницу вручную, то на такой поиск уйдет много времени. Но если бы у вас была программа, способная выполнять поиск телефонных номеров и электронных адресов в буфере обмена, то можно было бы просто нажать комбинацию клавиш <Ctrl+A>, чтобы выделить весь текст, и комбинацию клавиши <Ctrl+C>, чтобы скопировать выделенный текст в буфер, а затем запустить программу, чтобы она заменила находящийся в буфере текст найденными телефонными номерами и адресами электронной почты.

Всякий раз, когда вы беретесь за новый проект, возникает соблазн сразу же приступить к написанию кода. Но лучше не спешить и сначала обдумать проект в целом. Я рекомендую всегда начинать с составления высокоуровневого плана того, что должна делать программа. На данном этапе не стоит думать о коде — им вы займетесь позже. Начинайте с общей картины.

Например, вот что должна делать программа, предназначенная для извлечения телефонных номеров и адресов электронной почты:

1) получать текст из буфера обмена;
2) находить в тексте все телефонные номера и адреса электронной почты;
3) переносить найденный текст в буфер обмена.

Вот теперь уже можно подумать над тем, как реализовать все это в коде. Программа должна выполнить следующее:

1) импортировать модуль `pyperclip`, содержащий функции копирования и вставки строк;
2) создать два регулярных выражения, первое из которых соответствует телефонным номерам, а второе — адресам электронной почты;

3) найти все совпадения (не только первое) для обоих регулярных выражений;
4) аккуратно отформатировать найденные строки, объединив их в одну строку для вставки в буфер обмена;
5) вывести сообщение, если искомые шаблоны в тексте не были обнаружены.

Это своего рода дорожная карта проекта. В процессе написания программы вы сможете сфокусироваться на каждом этапе по отдельности. Все необходимое вы уже знаете.

Шаг 1. Создание регулярного выражения для поиска телефонных номеров

Прежде всего, необходимо составить регулярное выражение для поиска телефонных номеров. Создайте новый файл, введите в него следующий код и сохраните его в файле *phoneAndEmail.py*.

```
#! python3
# phoneAndEmail.py - находит телефонные номера
# и адреса электронной почты в буфере обмена

import pyperclip, re

phoneRegex = re.compile(r'''(
    (\d{3}|\(\d{3}\))?              # код региона
    (\s|-|\.)?                      # разделитель
    (\d{3})                         # первые 3 цифры
    (\s|-|\.)                       # разделитель
    (\d{4})                         # последние 4 цифры
    (\s*(ext|x|ext.)\s*(\d{2, 5}))?  # добавочный номер
    )''', re.VERBOSE)

# СДЕЛАТЬ: создать регулярное выражение для адресов
#          электронной почты

# СДЕЛАТЬ: найти совпадения в тексте, содержащемся
#          в буфере обмена

# СДЕЛАТЬ: скопировать результаты в буфер обмена
```

Комментарии `'СДЕЛАТЬ'` обозначают "скелет" программы. Впоследствии они будут заменены фактическим кодом.

Телефонный номер начинается с *необязательного* кода региона, поэтому соответствующая группа дополняется вопросительным знаком. Поскольку код региона содержит ровно три цифры (`\d{3}`) *или* ровно три цифры в круглых скобках (`\(\d{3}\)`), эти две альтернативы следует соединить

символом канала. Также в регулярное выражение добавлен комментарий `# код региона`, уточняющий назначение данного фрагмента.

В качестве разделителя групп цифр в телефонном номере может использоваться пробел (`\s`), дефис (`-`) или точка (`.`), поэтому данные компоненты регулярного выражения тоже должны быть соединены символами канала. В следующих трех компонентах нет ничего сложного: три цифры, за которыми идет еще один разделитель, а затем еще четыре цифры. Последняя часть — это необязательный добавочный номер, состоящий из произвольного количества пробелов, за которыми следует буквенное обозначение `'ext'`, `'x'` или `'ext.'`, а затем — сам добавочный номер, содержащий от двух до пяти цифр.

Примечание

Можно легко запутаться с регулярными выражениями, которые содержат группы с круглыми скобками () и экранированными круглыми скобками \(\). Если появится сообщение об ошибке "missing), unterminated subpattern", проверьте корректность использования скобок.

Шаг 2. Создание регулярного выражения для поиска адресов электронной почты

Нам также требуется регулярное выражение для поиска адресов электронной почты. Добавьте в программу новый код, выделенный полужирным шрифтом.

```
#! python3
# phoneAndEmail.py - находит телефонные номера
# и адреса электронной почты в буфере обмена

import pyperclip, re

phoneRegex = re.compile(r'''(
-- Опущено --

# Создание регулярного выражения для адресов электронной почты
emailRegex = re.compile(r'''(
❶    [a-zA-Z0-9._%+-]+     # имя пользователя
❷    @                     # символ @
❸    [a-zA-Z0-9.-]+        # домен
     (\.[a-zA-Z]{2, 4})    # остальная часть адреса
     )''', re.VERBOSE)

# СДЕЛАТЬ: найти совпадения в тексте, содержащемся
#          в буфере обмена

# СДЕЛАТЬ: скопировать результаты в буфер обмена
```

Часть адреса, содержащая имя пользователя ❶, включает один или несколько символов из следующего перечня: буквы в верхнем или нижнем регистре, цифры, точка, символ подчеркивания, знак процента, знак "плюс" и дефис. Все эти символы указываются в виде символьного класса: `[a-zA-Z0-9._%+-]`.

Домен отделяется от имени пользователя символом @ ❷. Доменное имя ❸ содержит более узкий класс символов, включающий только буквы, цифры, точку и дефис: `[a-zA-Z0-9.-]`. Последняя часть (домен верхнего уровня) может содержать только точку и буквы (от двух до четырех символов).

Формат адресов электронной почты порой бывает достаточно причудливым. Данному регулярному выражению будут соответствовать *не все* корректные адреса электронной почты, но его будет достаточно для всех типичных адресов.

Шаг 3. Поиск всех совпадений в тексте, скопированном в буфер обмена

Теперь, когда у нас есть регулярные выражения для поиска телефонных номеров и адресов электронной почты, можно поручить модулю `re` выполнить всю рутинную работу по поиску в буфере обмена всех строк, соответствующих составленным регулярным выражениям. Метод `pyperclip.paste()` получает строку, хранящуюся в буфере обмена, а метод `findall()` объекта `Regex` возвращает список кортежей.

Добавьте в программу новый код, выделенный полужирным шрифтом.

```
#! python3
# phoneAndEmail.py - находит телефонные номера
# и адреса электронной почты в буфере обмена

import pyperclip, re

phoneRegex = re.compile(r'''(
-- Опущено --

# Поиск совпадений в тексте, содержащемся
# в буфере обмена
text = str(pyperclip.paste())

❶ matches = []
❷ for groups in phoneRegex.findall(text):
      phoneNum = '-'.join([groups[1], groups[3], groups[5]])
      if groups[8] != '':
          phoneNum += ' x' + groups[8]
      matches.append(phoneNum)
```

```
❸ for groups in emailRegex.findall(text):
      matches.append(groups[0])

  # СДЕЛАТЬ: скопировать результаты в буфер обмена
```

Для каждого совпадения создается один кортеж, и каждый кортеж содержит строки для каждой группы в регулярном выражении. Не забывайте о том, что группе 0 соответствует все регулярное выражение, поэтому то, что нам нужно, — это группа с индексом 0 в кортеже.

Найденные совпадения сохраняются в списке matches. Поначалу этот список пуст ❶. Далее следуют два цикла for. В случае адресов электронной почты достаточно присоединять к списку matches группу 0 каждого найденного совпадения ❸. В случае же телефонных номеров алгоритм должен быть другим. Поскольку программа ищет телефонные номера, формат которых может быть разным, прежде чем присоединять их к списку, их нужно привести к единому формату. В переменной phoneNum содержится строка, скомпонованная из групп 1, 3, 5 и 8 найденного текста ❷. (Этими группами будут код региона, первые три цифры, последние четыре цифры и добавочный номер.)

Шаг 4. Объединение совпадений в одну строку для копирования в буфер обмена

Теперь, когда адреса электронной почты и телефонные номера сохранены в списке matches, их необходимо скопировать в буфер обмена. Функция pyperclip.copy() получает одиночную строку, а не список строк, поэтому для переменной matches необходимо вызвать метод join().

Чтобы было легче проверить работу программы, выведем все найденные совпадения на экран. Если ни телефонных номеров, ни адресов электронной почты в тексте не найдено, будет выдано соответствующее сообщение.

Внесите в программу следующие изменения.

```
#! python3
# phoneAndEmail.py - находит телефонные номера
# и адреса электронной почты в буфере обмена

-- Опущено --
for groups in emailRegex.findall(text):
  matches.append(groups[0])

# Копирование результатов в буфер обмена
if len(matches) > 0:
    pyperclip.copy('\n'.join(matches))
    print('Скопировано в буфер обмена:')
```

```
    print('\n'.join(matches))
else:
    print('Телефонные номера и адреса электронной \
          почты не обнаружены.')
```

Запуск программы

Для примера откройте в браузере страницу контактов сайта No Starch Press по адресу `https://www.nostarch.com/contactus/`, нажмите комбинацию клавиш <Ctrl+A>, чтобы выделить весь текст на странице, а затем — комбинацию клавиш <Ctrl+C>, чтобы скопировать этот текст в буфер обмена. Запустив программу, вы должны получить следующие результаты.

```
Скопировано в буфер обмена:
800-420-7240
415-863-9900
415-863-9950
info@nostarch.com
media@nostarch.com
academic@nostarch.com
conferences@nostarch.com
help@nostarch.com
```

Идеи для создания похожих программ

Распознавание образцов текста (и их замена с помощью метода `sub()`) — задача, которая находит множество возможных областей применения, например:

- нахождение URL-адресов веб-сайтов, начинающихся с префикса `http://` или `https://`;
- унификация дат, записанных в различных форматах (например, `14/03/2019`, `14-03-2019` и `2019/3/14`), путем их замены датами, представленными в едином формате;
- удаление конфиденциальной информации, такой как номера социального страхования или кредитных карт;
- исправление типичных опечаток, таких как наличие нескольких пробелов между словами, случайное повторение слов или наличие нескольких восклицательных знаков в конце предложения. Это так раздражает!!!

Резюме

Несмотря на то что компьютер способен очень быстро выполнять текстовый поиск, он нуждается в четких указаниях относительно того, что именно необходимо найти. Регулярные выражения позволяют задать символьные шаблоны для поиска вместо того, чтобы указывать конкретный текст. Средства поиска и замены на основе регулярных выражений есть во многих текстовых процессорах и приложениях электронных таблиц.

Модуль `re`, входящий в стандартную библиотеку Python, позволяет компилировать объекты регулярных выражений (`Regex`). У таких объектов есть методы `search()` для поиска одиночных совпадений с регулярным выражением, `findall()` для поиска всех экземпляров совпадений и `sub()` для поиска с заменой текста.

Дополнительную информацию о модуле `re` можно найти в официальной документации Python, доступной по адресу `https://docs.python.org/3/library/re.html`. Другой полезный ресурс — обучающий сайт `https://www.regular-expressions.info/`.

Контрольные вопросы

1. С помощью какой функции создаются объекты регулярных выражений (`Regex`)?
2. Почему при создании объектов `Regex` часто используются необработанные строки?
3. Что возвращает метод `search()`?
4. Как с помощью объекта `Match` получить фактические строки, соответствующие шаблону регулярного выражения?
5. Имеется объект регулярного выражения, созданный на основе строки `r'(\d\d\d)-(\d\d\d-\d\d\d\d)'`. Чему в нем соответствует группа 0? Группа 1? Группа 2?
6. В синтаксисе регулярных выражений круглые скобки и точки имеют специальное назначение. Как указать в регулярном выражении, что символы круглых скобок и точки сами являются объектом поиска?
7. В каком случае метод `findall()` возвращает список строк, а в каком — список кортежей строк.
8. Что означает символ `|` в регулярных выражениях?
9. Какие две функции выполняет символ `?` в регулярных выражениях?
10. В чем разница между символами `+` и `*` в регулярных выражениях?
11. В чем разница между шаблонами `{3}` и `{3, 5}` в регулярных выражениях?

12. Что означают сокращенные символьные классы `\d`, `\w` и `\s` в регулярных выражениях?
13. Что означают сокращенные символьные классы `\D`, `\W` и `\S` в регулярных выражениях?
14. В чем разница между шаблонами `.*` и `.*?`?
15. Как записать символьный класс, которому соответствуют все цифры и буквы в нижнем регистре?
16. Как сделать регулярное выражение нечувствительным к регистру?
17. Чему обычно соответствует символ `.`? Чему он соответствует, если методу `re.compile()` в качестве второго аргумента передана константа `re.DOTALL`?
18. Если `numRegex = re.compile(r'\d+')`, то что вернет вызов `numRegex.sub('X', '12 барабанщиков, 11 волынщиков, пять колец, 3 курицы')`?
19. К чему приводит передача константы `re.VERBOSE` в качестве второго аргумента метода `re.compile()`?
20. Как записать регулярное выражение, которому соответствуют числа, содержащие запятую в качестве разделителя после каждых трех цифр? Этому выражению должно соответствовать следующее:
    - `'42'`
    - `'1,234'`
    - `'6,368,745'`

    и не должно соответствовать следующее:
    - `'12,34,567'` (только две цифры между запятыми);
    - `'1234'` (отсутствуют запятые).
21. Как записать регулярное выражение, которому соответствуют все полные имена, включающие фамилию `'Watanabe'`? Предполагается, что имя, предшествующее фамилии, всегда состоит из одного слова, начинающегося с прописной буквы. Этому регулярному выражению должно соответствовать следующее:
    - `'Haruto Watanabe'`
    - `'Alice Watanabe'`
    - `'RoboCop Watanabe'`

    и не должно соответствовать следующее:
    - `'haruto Watanabe'` (имя не начинается с прописной буквы);
    - `'Mr. Watanabe'` (в предшествующем слове имеется небуквенный символ);

- `'Watanabe'` (отсутствует имя);
- `'Haruto watanabe'` (фамилия `'Watanabe'` начинается со строчной буквы).

22. Как записать регулярное выражение, ищущее совпадения с предложениями, в которых первым идет имя `'Алиса'`, `'Боб'` или `'Кэрол'`, вторым идет слово `'ест'`, `'заводит'` или `'кидает'`, третьим — слово `'яблоки'`, `'кошек'` или `'мячи'`, а в конце стоит точка? Это регулярное выражение должно быть нечувствительным к регистру. Ему должны соответствовать следующие предложения:

    - `'Алиса ест яблоки.'`
    - `'Боб заводит кошек.'`
    - `'Кэрол кидает мячи.'`
    - `'Алиса кидает яблоки.'`
    - `'БОБ ЕСТ КОШЕК.'`

    и не должны соответствовать следующие:

    - `'РобоКоп ест яблоки.'`
    - `'АЛИСА КИДАЕТ ФУТБОЛЬНЫЕ МЯЧИ.'`
    - `'Кэрол ест 7 кошек.'`

Учебные проекты

Чтобы закрепить полученные знания на практике, напишите программы для предложенных ниже задач.

Обнаружение даты

Составьте регулярное выражение, которое позволяет находить даты в формате *ДД/ММ/ГГГГ*. Предполагается, что дни изменяются от 01 до 31, месяцы — от 01 до 12, а годы — от 1000 до 2999. Если день или месяц задан одиночной цифрой, то добавляется начальный нуль.

Регулярное выражение не обязано определять корректное количество дней для каждого месяца или високосного года. Оно может находить несуществующие даты, например 31/02/2020 или 31/04/2021. Сохраните возвращаемые строки в переменных `month`, `day` и `year` и напишите дополнительный код для проверки корректности даты. В апреле, июне, сентябре и ноябре по 30 дней, в феврале — 28 дней, в остальных месяцах — 31 день. В високосные годы в феврале 29 дней. Високосные годы кратны 4, за исключением тех, которые кратны 100, если только год при этом не кратен 400. Такие вычисления слишком громоздкие, чтобы можно было составить для них регулярное выражение разумного размера.

Выявление сильных паролей

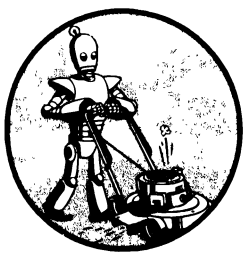
Напишите функцию, которая использует регулярные выражения для проверки того, что переданная ей строка содержит сильный пароль. Сильными считаются пароли, которые состоят по крайней мере из восьми символов, содержат символы в верхнем и нижнем регистрах и включают хотя бы одну цифру. Строку пароля можно сравнивать с несколькими шаблонами регулярных выражений.

Версия метода `strip()`, использующая регулярные выражения

Напишите функцию, которая получает строку и делает то же, что и строковый метод `strip()`. В отсутствие других аргументов, кроме строки, функция должна удалить из нее начальные и конечные пробельные символы. В противном случае из строки должны быть удалены символы, переданные в качестве второго аргумента.

8

ПРОВЕРКА ВВОДА



В программе необходимо проверять корректность значений, вводимых пользователем с помощью той же функции `input()`. В частности, если пользователь должен указать свой возраст, то программа не должна принимать бессмысленные ответы, например отрицательные числа (недопустимый диапазон целых чисел) или слова (неправильный тип данных). Проверка ввода также позволяет предотвратить ошибки. Если вы реализуете функцию `withdrawFromAccount()`, которая содержит аргумент для снятия суммы со счета, то убедитесь, что эта сумма положительна. При попытке снять со счета отрицательную сумму функция в конечном итоге добавит деньги на счет!

Как правило, мы выполняем проверку ввода, неоднократно запрашивая данные у пользователя, пока он не введет корректный текст, как в следующем примере.

```
while True:
    print('Укажите ваш возраст:')
    age = input()
    try:
        age = int(age)
    except:
        print('Пожалуйста, введите цифры.')
        continue
    if age < 1:
        print('Пожалуйста, введите положительное число.')
        continue
    break

print(f'Вам {age} лет.')
```

После запуска программы результат будет выглядеть примерно так.

```
Укажите ваш возраст:
тридцать
Пожалуйста, введите цифры.
Укажите ваш возраст:
-2
Пожалуйста, введите положительное число.
Укажите ваш возраст:
30
Вам 30 лет.
```

Программа предлагает ввести возраст до тех пор, пока не будет введено правильное значение. Тем самым гарантируется, что при выходе из цикла `while` переменная `age` будет содержать допустимое значение, которое впоследствии не приведет к сбою программы.

Впрочем, писать код проверки для каждого вызова функции `input()` быстро надоедает. Кроме того, возникает риск пропустить те или иные ошибочные варианты ввода, и в результате в программе окажутся некорректные данные. В этой главе мы рассмотрим использование модуля PyInputPlus для проверки ввода.

Модуль PyInputPlus

Модуль PyInputPlus содержит функции, аналогичные `input()`, которые предназначены для ввода различных типов данных: чисел, дат, адресов электронной почты и т.п. Если пользователь введет недопустимое значение, например неверно отформатированную дату или число за пределами

допустимого диапазона, модуль PyInputPlus предложит ввести данные повторно, как в приведенном выше примере. В модуле PyInputPlus реализованы и другие полезные возможности, такие как ограничение количества повторных запросов или тайм-аут, если пользователь должен уложиться в определенные временны́е рамки.

Модуль PyInputPlus не является частью стандартной библиотеки Python, поэтому его нужно установить отдельно с помощью утилиты `pip` (сведения по установке сторонних модулей приведены в приложении А). Для этого в командной строке выполните следующую инструкцию:

```
pip install --user pyinputplus
```

Чтобы проверить, корректно ли установлен модуль PyInputPlus, импортируйте его в интерактивной оболочке:

```
>>> import pyinputplus
```

Если при импорте модуля не появилось никаких сообщений об ошибках, значит, он успешно установлен.

Модуль PyInputPlus содержит несколько функций, предназначенных для обработки различных типов вводимых данных.

- **`inputStr()`**. Аналогична встроенной функции `input()`, но поддерживает расширенные возможности модуля PyInputPlus. Ей можно передать пользовательскую функцию для проверки введенных данных.
- **`inputNum()`**. Гарантирует ввод числа и возвращает значение типа `int` или `float`, в зависимости от того, содержит ли введенное число десятичную точку.
- **`inputChoice()`**. Гарантирует выбор одного из предложенных вариантов.
- **`inputMenu()`**. Аналогична функции `inputChoice()`, но отображает меню с числовыми или буквенными вариантами.
- **`inputDatetime()`**. Гарантирует ввод значений даты и времени.
- **`inputYesNo()`**. Гарантирует, что пользователь введет ответ "да/нет".
- **`inputBool()`**. Аналогична функции `inputYesNo()`, но распознает ответ `'True'` или `'False'` и возвращает соответствующее булево значение.
- **`inputEmail()`**. Гарантирует ввод корректного адреса электронной почты.

- **inputFilepath()**. Гарантирует ввод правильного имени файла (включая путь) и может дополнительно проверять, существует ли файл с таким именем.
- **inputPassword()**. Аналогична встроенной функции `input()`, но отображает символы * вместо вводимых символов, что позволяет безопасно вводить пароли и другую конфиденциальную информацию.

Эти функции автоматически выводят новый запрос до тех пор, пока не будут введены корректные данные.

```
>>> import pyinputplus as pyip
>>> response = pyip.inputNum()
пять
'пять' is not a number.
42
>>> response
42
```

Выражение `as pyip` в инструкции `import` избавляет от необходимости указывать `pyinputplus` каждый раз, когда нужно вызвать функцию из этого модуля. Вместо длинного имени используется более короткий псевдоним `pyip`. В отличие от функции `input()`, функция `inputNum()` возвращает значение типа `int` или `float`: в данном случае это `42`, а не `'42'`.

Как и в случае функции `input()`, функциям модуля PyInputPlus можно передать строку приглашения с помощью именованного аргумента `prompt`.

```
>>> response = input('Введите число: ')
Введите число: 42
>>> response
'42'
>>> import pyinputplus as pyip
>>> response = pyip.inputInt(prompt='Введите число: ')
Введите число: кот
'кот' is not an integer.
Введите число: 42
>>> response
42
```

Чтобы больше узнать о каждой из этих функций, воспользуйтесь функцией `help()`. Например, если ввести в интерактивной оболочке **help(pyip.inputChoice)**, отобразится справочная информация по функции `inputChoice()`. Полная документация к модулю доступна по адресу `https://pyinputplus.readthedocs.io/`.

В отличие от встроенной функции `input()`, функции модуля PyInputPlus имеют ряд дополнительных возможностей проверки, о которых будет рассказано далее.

Именованные аргументы `min`, `max`, `greaterThan` и `lessThan`

Функции `inputNum()`, `inputInt()` и `inputFloat()`, которые работают с целыми числами и числами с плавающей точкой, поддерживают именованные аргументы `min`, `max`, `greaterThan` и `lessThan`, с помощью которых можно задать диапазон допустимых значений. Введите в интерактивной оболочке следующие инструкции.

```
>>> import pyinputplus as pyip
>>> response = pyip.inputNum('Введите число: ', min=4)
Введите число: 3
Input must be at minimum 4.
Введите число: 4
>>> response
4
>>> response = pyip.inputNum('Введите число: ', greaterThan=4)
Введите число: 4
Input must be greater than 4.
Введите число: 5
>>> response
5
>>> response = pyip.inputNum('>', min=4, lessThan=6)
> 6
Input must be less than 6.
> 3
Input must be at minimum 4.
> 4
>>> response
4
```

Это необязательные аргументы, но если они указаны, то вводимые значения не могут быть меньше аргумента `min` или больше аргумента `max` (но могут быть равны им). Кроме того, вводимые значения должны быть больше, чем аргумент `moreThan`, и меньше, чем аргумент `lessThan` (т.е. они не могут быть равны им).

Именованный аргумент `blank`

По умолчанию ввод пустой строки не допускается, если только для аргумента `blank` не задано значение `True`.

```
>>> import pyinputplus as pyip
>>> response = pyip.inputNum('Введите число: ')
Введите число: (введено пустое значение)
```

```
Blank values are not allowed.
Введите число: 42
>>> response
42
>>> response = pyip.inputNum(blank=True)
(введено пустое значение)
>>> response
''
```

Используйте аргумент `blank = True` в том случае, когда пользователю разрешается ничего не вводить.

Именованные аргументы `limit`, `timeout` и `default`

По умолчанию функции модуля PyInputPlus будут продолжать запрашивать у пользователя ввод корректных данных бесконечно долго (ну или до тех пор, пока выполняется программа). Если хотите, чтобы функция перестала запрашивать данные после определенного количества попыток или по истечении определенного времени, используйте именованные аргументы `limit` и `timeout`. Целочисленный аргумент `limit` определяет, сколько попыток будет предпринято функцией для получения корректных данных, прежде чем она завершит работу, а целочисленный аргумент `timeout` определяет, сколько секунд отведено пользователю для ввода корректных данных, прежде чем функция завершится.

Если пользователь так и не введет корректных данных за указанное количество попыток или отведенное время, функции сгенерируют исключение `RetryLimitException` или `TimeoutException` соответственно. Например, введите в интерактивной оболочке следующий код.

```
>>> import pyinputplus as pyip
>>> response = pyip.inputNum(limit=2)
бла-бла-бла
'бла-бла-бла' is not a number.
Enter num: число
'число' is not a number.
Traceback (most recent call last):
    -- Опущено --
pyinputplus.RetryLimitException
>>> response = pyip.inputNum(timeout=10)
42 (введено после 10 секунд ожидания)
Traceback (most recent call last):
    -- Опущено --
pyinputplus.TimeoutException
```

Если помимо этих аргументов указан также аргумент `default`, функция не сгенерирует исключение, а вернет значение, переданное ей с помощью этого аргумента. Введите в интерактивной оболочке следующий код.

```
>>> response = pyip.inputNum(limit=2, default='N/A')
Здравствуй
'Здравствуй' is not a number.
мир
'мир' is not a number.
>>> response
'N/A'
```

Вместо того чтобы генерировать исключение `RetryLimitException`, функция `inputNum()` возвращает строку `'N/A'`.

Именованные аргументы `allowRegexes` и `blockRegexes`

Указывать допустимые значения можно также с помощью регулярных выражений. Именованные аргументы `allowRegexes` и `blockRegexes` позволяют задать списки регулярных выражений, определяющих, какие значения функция принимает в качестве допустимых, а какие — отклоняет. Например, введите в интерактивной оболочке следующий код, чтобы функция `inputNum()` помимо обычных чисел поддерживала римские цифры.

```
>>> import pyinputplus as pyip
>>> response=pyip.inputNum(allowRegexes=[r'(I|V|X|L|C|D|M)+',
                           r'zero'])
XLII
>>> response
'XLII'
>>> response=pyip.inputNum(allowRegexes=[r'(i|v|x|l|c|d|m)+',
                           r'zero'])
xlii
>>> response
'xlii'
```

Конечно, эти регулярные выражения задают только буквы, которые разрешается вводить пользователю. Порядок цифр в числе может оказаться неправильным, например `'XVX'` или `'MILLI'`, потому что регулярное выражение `r'(I|V|X|L|C|D|M)+'` допускает такое число.

С помощью именованного аргумента `blockRegexes` можно также указать список регулярных выражений, которые функция не будет принимать. Введите в интерактивной оболочке следующий код, чтобы функция `inputNum()` не принимала четные числа.

```
>>> import pyinputplus as pyip
>>> response = pyip.inputNum(blockRegexes=[r'[02468]$'])
42
This response is invalid.
44
This response is invalid.
```

```
43
>>> response
43
```

Если указать оба аргумента — и `allowRegexes`, и `blockRegexes`, — то список разрешений перекрывает список блокировок. Например, введите в интерактивной оболочке следующий код, который разрешает ввод слов `'caterpillar'` и `'category'`, но блокирует любые другие строки, содержащие слово `'cat'`.

```
>>> import pyinputplus as pyip
>>> response = pyip.inputStr(allowRegexes=[r'caterpillar',
...                                        'category'],
...                          blockRegexes=[r'cat'])
cat
This response is invalid.
catastrophe
This response is invalid.
category
>>> response
'category'
```

Функции модуля PyInputPlus помогут избавить вас от утомительного написания кода проверки вводимых данных. Полная документация к этому модулю доступна по адресу `https://pyinputplus.readthedocs.io/`.

Передача пользовательской функции проверки в функцию `inputCustom()`

Можно написать функцию, реализующую требуемую логику проверки, и передать ее функции `inputCustom()`. Допустим, необходимо, чтобы пользователь ввел последовательность цифр, в сумме равных 10. Функции `pyinputplus.inputAddsUpToTen()` не существует, но можно создать свою собственную функцию, которая:

- имеет один строковый аргумент (значение, введенное пользователем);
- генерирует исключение, если строка не проходит проверку;
- возвращает `None` (инструкция `return` может быть опущена), если функция `inputCustom()` должна вернуть строку без изменений;
- возвращает значение, отличное от `None`, если функция `inputCustom()` должна вернуть строку, отличную от той, которую ввел пользователь;
- передается в качестве первого аргумента функции `inputCustom()`.

Например, можно создать пользовательскую функцию `addUpToTen()` и передать ее функции `inputCustom()`. При этом вызов должен выглядеть как `inputCustom(addUpToTen)`, а не `inputCustom(addUpToTen())`, потому что мы передаем ссылку на функцию `addUpToTen()`, а не возвращаемое ею значение.

```
>>> import pyinputplus as pyip
>>> def addsUpToTen(numbers):
...     numbersList = list(numbers)
...     for i, digit in enumerate(numbersList):
...         numbersList[i] = int(digit)
...     if sum(numbersList) != 10:
...         raise Exception('Сумма должна быть 10, а не %s.' %
(sum(numbersList)))
...     return int(numbers)    # вернуть целое число
...
>>> response = pyip.inputCustom(addsUpToTen) # без скобок после имени
123
Сумма должна быть 10, а не 6.
1235
Сумма должна быть 10, а не 11.
1234
>>> response # функция inputCustom() возвращает целое число, а не строку
1234
>>> response = pyip.inputCustom(addsUpToTen)
Здравствуй
invalid literal for int() with base 10: 'З'
55
>>> response
55
```

Функция `inputCustom()` тоже поддерживает именованные аргументы `blank`, `limit`, `timeout`, `default`, `allowRegexes` и `blockRegexes`. Написание собственной функции проверки целесообразно в том случае, когда трудно или невозможно написать регулярное выражение для проверки допустимых данных, как в приведенном выше примере.

Проект: как занять дурака на несколько часов

Воспользуемся модулем PyInputPlus для создания простой программы, которая выполняет следующие действия:

1) спрашивает пользователя, не хотел бы он узнать, как занять дурака на протяжении нескольких часов;
2) завершает работу, если пользователь отвечает "нет";
3) возвращается к п. 1, если пользователь отвечает "да".

Поскольку мы не знаем, будет ли пользователь вводить что-то кроме 'да' или 'нет', необходимо проверить правильность введенных данных. Было бы также удобно, чтобы пользователь мог вводить 'д' или 'н' вместо полных слов. Эти проверки выполняет функция inputYesNo() из модуля PyInputPlus, которая независимо от регистра введенных символов возвращает строку 'да' или 'нет' в нижнем регистре.

Результаты работы программы будут выглядеть примерно так.

```
Хотите узнать, как занять дурака на несколько часов?
конечно
'конечно' is not a valid yes/no response.
Хотите узнать, как занять дурака на несколько часов?
да
Хотите узнать, как занять дурака на несколько часов?
д
Хотите узнать, как занять дурака на несколько часов?
Да
Хотите узнать, как занять дурака на несколько часов?
ДА
Хотите узнать, как занять дурака на несколько часов?
ДА!!!!!!
'ДА!!!!!!' is not a valid yes/no response.
Хотите узнать, как занять дурака на несколько часов?
СКАЖИ МНЕ, КАК.
'СКАЖИ МНЕ, КАК.' is not a valid yes/no response.
Хотите узнать, как занять дурака на несколько часов?
нет
Спасибо! Желаю хорошего дня.
```

Откройте новую вкладку в редакторе файлов и сохраните файл под именем *idiot.py*. Затем введите следующий код:

```
import pyinputplus as pyip
```

В результате будет импортирован модуль PyInputPlus. Поскольку имя pyinputplus слишком длинное, мы назначаем ему псевдоним pyip.

```
while True:
    prompt = 'Хотите узнать, как занять дурака на несколько часов?'
    response = pyip.inputYesNo(prompt)
```

Выражение while True: создает бесконечный цикл, который выполняется до тех пор, пока не встретится инструкция break. В цикле вызывается функция pyip.inputYesNo(), которая не завершится до тех пор, пока пользователь не введет корректный ответ.

```
    if response == 'no':
        break
```

Функция `pyip.inputYesNo()` гарантированно возвращает `'yes'` или `'no'`. Если возвращается `'no'`, программа выходит из бесконечного цикла и продолжает выполняться до последней строки, после чего прощается с пользователем:

```
print('Спасибо! Желаю хорошего дня.')
```

В противном случае цикл повторяется снова.

Функция `inputYesNo()` поддерживает языки, отличные от английского, благодаря именованным аргументам `yesVal` и `noVal`. Например, русскоязычная версия программы должна содержать следующий код.

```
    response = pyip.inputYesNo(prompt, yesVal= 'да', noVal='нет')
    if response == 'нет':
```

Теперь пользователь может ввести `'да'` или `'д'` (в нижнем или верхнем регистре) вместо `'yes'` или `'y'` в качестве утвердительного ответа.

Проект: тест на умножение

Модуль PyInputPlus может оказаться полезным для создания теста на умножение с лимитом времени. Благодаря именованным аргументам `allowRegexes`, `blockRegexes`, `timeout` и `limit` функции `pyip.inputStr()` бо́льшая часть операций реализуется в самом модуле PyInputPlus. Чем меньше кода предстоит написать, тем быстрее можно получить готовую программу. Мы напишем программу, которая выводит пользователю 10 заданий на умножение. Откройте в редакторе файлов новую вкладку и сохраните файл под именем *multiplicationQuiz.py*.

Сначала необходимо импортировать модули `pyinputplus`, `random` и `time`. Мы будем отслеживать, сколько вопросов задала программа и сколько правильных ответов дал пользователь, с помощью переменных `numberOfQuestions` и `correctAnswers`. В цикле `for` будет случайным образом 10 раз выдаваться задание на умножение.

```
import pyinputplus as pyip
import random, time

numberOfQuestions = 10
correctAnswers = 0
for questionNumber in range(numberOfQuestions):
```

В цикле `for` программа выбирает две перемножаемые цифры. Эти цифры отображаются в приглашении `#Q: N × N=`, где *Q* — номер вопроса (от 1 до 10), а *N* — числа, которые нужно умножить.

```
# Выбираем два случайных числа
num1 = random.randint(0, 9)
num2 = random.randint(0, 9)
prompt = '#%s: %s x %s = ' % (questionNumber, num1, num2)
```

Основная логика программы реализована в функции `pyip.inputStr()`. Именованный аргумент `allowRegexes` представляет собой список, содержащий регулярное выражение `'^%s$'`, где `%s` заменяется правильным ответом. Символы `^` и `$` гарантируют, что ответ начинается и заканчивается правильным числом, хотя функции PyInputPlus сначала удаляют любые ведущие и замыкающие пробелы на тот случай, если пользователь случайно нажмет пробела до или после ответа. Именованный аргумент `blocklistRegexes` содержит список с одним кортежем `('.*', 'Неправильно!')`. Первая строка в кортеже — это регулярное выражение, которое соответствует любой возможной строке. Следовательно, если пользователь вводит неправильный ответ, программа отклоняет его. В таком случае отображается строка `'Неправильно!'`, после чего пользователю предлагается ответить снова. Кроме того, аргументы `timeout=8` и `limit=3` гарантируют, что у пользователя есть только 8 секунд и 3 попытки дать правильный ответ.

```
try:
    # Правильные ответы задаются аргументом allowRegexes;
    # неправильные ответы задаются аргументом blockRegexes
    # (в случае неправильного ответа отображается
    # пользовательское сообщение)
    pyip.inputStr(prompt, allowRegexes=['^%s$' % (num1 * num2)], \
                  blockRegexes=[('.*', 'Неправильно!')], \
                  timeout=8, limit=3)
```

Если пользователь отвечает по истечении 8-секундного тайм-аута, то даже в случае правильного ответа функция `pyip.inputStr()` генерирует исключение `TimeoutException`. Если пользователь отвечает неправильно более трех раз, генерируется исключение `RetryLimitException`. Оба этих типа исключений определены в модуле PyInputPlus, поэтому их нужно предварять префиксом `pyip`.

```
except pyip.TimeoutException:
    print('Время истекло!')
except pyip.RetryLimitException:
    print('Закончилось количество попыток!')
```

Помните, что блок `else` может следовать не только за блоком `if` или `elif`, но и за блоком `except`. Следующий код будет выполняться, если в блоке `try` не было сгенерировано исключение. В нашем случае это означает, что пользователь ввел правильный ответ.

```
    else:
        # Этот блок выполняется, если в блоке try
        # не возникло исключений
        print('Правильно!')
        correctAnswers += 1
```

Независимо от того, какое из трех сообщений (`'Время истекло!'`, `'Закончилось количество попыток!'` или `'Правильно'`) отображается, в конце цикла делается секундная пауза, которая дает пользователю время на прочтение сообщения. После того как программа задала 10 вопросов, пользователь увидит количество правильных ответов.

```
    time.sleep(1)    # короткая пауза, позволяющая пользователю
                     # увидеть результат
print('Счет: %s/%s'%(correctAnswers, numberOfQuestions))
```

Модуль PyInputPlus достаточно гибкий, благодаря чему его можно использовать в самых разных программах, которые принимают ввод пользователя с клавиатуры.

Резюме

Многие программисты забывают писать код для проверки вводимых данных, но без него в программах практически наверняка будут возникать ошибки. Значения, которые вы ожидаете от пользователей, и значения, которые они фактически вводят, могут оказаться совершенно разными, и программы должны быть достаточно надежными, чтобы справляться с такими ситуациями. Для создания кода проверки вводимых данных можно использовать регулярные выражения, но обычно проще использовать готовый модуль, такой как PyInputPlus. Импортируйте этот модуль с помощью инструкции `import pyinputplus as pyip`, чтобы при вызове функций модуля указывать более короткий псевдоним `pyip`.

Модуль PyInputPlus содержит функции для ввода различных типов данных, включая строки, числа, даты, булевы значения, варианты "да/нет", адреса электронной почты и файлы. В то время как функция `input()` всегда возвращает строку, функции модуля PyInputPlus возвращают значения соответствующего типа данных. Функция `inputChoice()` позволяет выбрать один из нескольких предварительно заданных вариантов, а функция `inputMenu()` добавляет цифры или буквы к вариантам выбора.

Все эти функции поддерживают следующие стандартные возможности: удаление ведущих и замыкающих пробелов, установка тайм-аута и количества допустимых повторов с помощью именованных аргументов `timeout` и `limit`, а также передача списков регулярных выражений с помощью аргументов `allowRegexes` и `blockRegexes`, позволяющих принимать или отвергать определенные ответы. Вам больше не нужно писать утомительные циклы `while`, в которых проверяется правильность введенных данных и выводятся повторные запросы.

Если ни одна из функций модуля PyInputPlus не соответствует вашим задачам, можно написать собственную функцию проверки и передать ее функции `inputCustom()`. Полный список функций модуля доступен по адресу `https://pyinputplus.readthedocs.io/en/latest/`. В онлайн-документации содержится гораздо больше информации, чем было приведено в этой главе. Не стоит изобретать велосипед – лучше научиться использовать этот модуль, чем писать (и отлаживать!) собственный код.

Теперь, когда вы умеете работать с текстом и проверять его правильность, пора научиться считывать и записывать файлы, хранящиеся на жестком диске. Этой теме посвящена следующая глава.

Контрольные вопросы

1. Входит ли модуль PyInputPlus в стандартную библиотеку Python?
2. Почему модуль PyInputPlus обычно импортируют с помощью инструкции `import pyinputplus as pyip`?
3. Чем отличается функция `inputInt()` от функции `inputFloat()`?
4. Как с помощью модуля PyInputPlus гарантировать, что пользователь введет целое число в диапазоне от `0` до `99`?
5. Что передается с помощью именованных аргументов `allowRegexes` и `blockRegexes`?
6. Что сделает функция `inputStr(limit=3)`, если три раза ввести пустую строку?
7. Что сделает функция `inputStr(limit=3, default='привет')`, если три раза ввести пустую строку?

Учебные проекты

Чтобы закрепить полученные знания на практике, напишите программы для предложенных ниже задач.

Изготовитель бутербродов

Напишите программу, которая спрашивает пользователя о его бутербродных предпочтениях. Программа должна использовать модуль PyInputPlus, чтобы гарантировать ввод корректных данных.

- Используйте функцию `inputMenu()` для определения типа хлеба: цельнозерновой, белый или ржаной.
- Используйте функцию `inputMenu()` для определения типа белкового продукта: курица, индейка, ветчина или тофу.
- Используйте функцию `inputYesNo()`, чтобы спросить, хочет ли пользователь добавить сыр.
- Если пользователь ответил утвердительно, используйте функцию `inputMenu()`, чтобы узнать тип сыра: чеддер, швейцарский или моцарелла.
- Используйте функцию `inputYesNo()`, чтобы узнать у пользователя, хочет ли он добавить майонез, горчицу, салат или помидор.
- Используйте функцию `inputInt()`, чтобы узнать, сколько бутербродов хочет пользователь. Убедитесь в том, что это число не меньше 1.

Придумайте цены для каждого из параметров бутерброда, и пусть программа отобразит общую стоимость после того, как пользователь сделает свой выбор.

Собственный тест на умножение

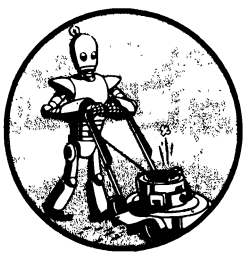
Чтобы оценить реальные возможности модуля PyInputPlus, попробуйте воссоздать тест на умножение без использования этого модуля. Программа должна предложить пользователю 10 заданий на умножение в диапазоне от 0-0 до 9-9. Необходимо реализовать следующий функционал.

- Если пользователь вводит правильный ответ, программа в течение одной секунды отображает сообщение "Правильно!" и переходит к следующему вопросу.
- Пользователь получает три попытки для ввода правильного ответа, прежде чем программа перейдет к следующему вопросу.
- Через восемь секунд после первого отображения вопроса он помечается как неправильный, даже если пользователь вводит правильный ответ после 8-секундной паузы.

Сравните свой код с кодом на основе модуля PyInputPlus, который был приведен в главе.

9

ЧТЕНИЕ И ЗАПИСЬ ФАЙЛОВ



Переменные — отличное средство хранения данных на этапе выполнения программы, но если требуется, чтобы данные существовали и после ее завершения, их необходимо сохранить в файле. Содержимое файла можно рассматривать как огромную строку, размер которой способен исчисляться гигабайтами. В этой главе мы поговорим о том, как использовать Python для создания, чтения и сохранения файлов на жестком диске.

Файлы и папки

Две ключевые характеристики файла — *имя* (обычно записывается в виде одного слова) и *путь*. Путь определяет, где именно в структуре каталогов располагается файл. Например, в ноутбуке автора есть файл с именем *project.docx*, который находится в каталоге *C:\Users\Al\Documents*. Часть имени файла, стоящая после точки, называется *расширением*. Оно определяет тип файла. Файл *project.docx* — это документ Word, а *Users*, *Al* и *Documents* — названия папок. Папки могут содержать файлы и другие папки. Например, файл *project.docx* находится в папке *Documents*, которая сама находится в папке *Al*, а та, в свою очередь, содержится в папке *Users* (рис. 9.1).

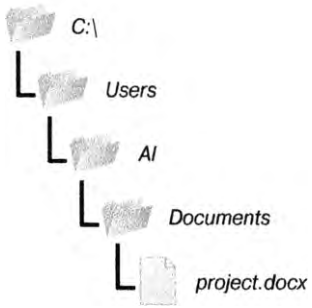


Рис. 9.1. Расположение файла в иерархии папок

Компонент *C:* — это *корневая папка*, которая содержит все остальные папки. В Windows корневая папка — *C:*, в macOS и Linux — */*. В данной книге корневая папка обозначается в стиле Windows (*C:*). Если вы будете выполнять примеры в интерактивной оболочке macOS или Linux, то используйте вместо этого обозначение */*.

Дополнительные *тома*, соответствующие DVD-приводу или USB-носителям, будут отображаться по-разному в разных операционных системах. В Windows они представлены буквами дисков: *D:*, *E:* и т.д. В macOS они отображаются в виде новых папок в папке */Volumes*, а в Linux — в виде новых папок в папке */mnt* (точки монтирования). Кроме того, не забывайте, что в Linux имена файлов и папок чувствительны к регистру символов, а в Windows и macOS — нет.

Примечание

В вашей системе структура файлов и папок наверняка отличается от моей, поэтому вы не сможете в точности следовать примерам данной главы. Пытайтесь выполнять примеры, ориентируясь на свою систему.

Использование обратной косой черты в Windows и косой черты в macOS и Linux

В Windows имена папок разделяются обратной косой чертой (\). В macOS и Linux разделителем служит косая черта (/). Если хотите, чтобы ваши программы работали во всех операционных системах, пишите их так, чтобы обрабатывались оба случая.

К счастью, это делается очень просто — с помощью функции `Path()` из модуля `pathlib`. Если передать ей строки с именами папок, то она вернет строку пути с использованием корректной версии разделителя. Введите в интерактивной оболочке следующие инструкции.

```
>>> from pathlib import Path
>>> Path('spam', 'bacon', 'eggs')

WindowsPath('spam/bacon/eggs')
>>> str(Path('spam', 'bacon', 'eggs'))
'spam\\bacon\\eggs'
```

Обратите внимание на инструкцию импорта: в ней из модуля `pathlib` импортируется только функция `Path()`. При импорте всего модуля нам пришлось бы писать полное имя функции `pathlib.Path` везде, где сейчас пишется просто `Path`.

Примеры этой главы выполняются в Windows, поэтому команда `Path('spam', 'bacon', 'eggs')` возвращает объект `WindowsPath` для строки пути: `WindowsPath('spam/bacon/eggs')`. Несмотря на то что в Windows разделителем служит обратная косая черта, представление объекта `WindowsPath` в интерактивной оболочке отображается с использованием косой черты, поскольку разработчики открытого программного обеспечения исторически предпочитают операционную систему Linux.

Если хотите получить простую текстовую строку пути, передайте ее функции `str()`, которая в нашем примере возвращает строку `'spam\\bacon\\eggs'` (обратите внимание на удвоение обратной косой черты, потому что она должен экранироваться другим символом обратной косой черты). В Linux функция `Path()` вернула бы объект `PosixPath`, а функция `str()` вернула бы строку `'spam/bacon/eggs'`. (*POSIX* — это набор стандартов для Unix-подобных операционных систем, таких как Linux.)

Эти объекты `Path` (`WindowsPath` или `PosixPath`, в зависимости от операционной системы) можно передавать различным функциям, как будет показано далее. Например, в следующем коде имена из списка имен файлов добавляются к концу имени папки.

```
>>> from pathlib import Path
>>> myFiles = ['accounts.txt', 'details.csv', 'invite.docx']
```

```
>>> for filename in myFiles:
        print(Path(r'C:\Users\Al', filename))
C:\Users\Al\accounts.txt
C:\Users\Al\details.csv
C:\Users\Al\invite.docx
```

В Windows имена папок разделяются обратной косой чертой, поэтому использовать ее в именах файлов нельзя. Но ее можно использовать в именах файлов в macOS и Linux. Таким образом, в Windows вызов `Path(r'spam\eggs')` относится к двум разным папкам (или к файлу `eggs` в папке `spam`), но в macOS или Linux этот же вызов будет относиться к одной папке (или файлу) с именем `spam\eggs`. Вот почему в коде Python рекомендуется всегда использовать косую черту (именно так мы и будем поступать в последующих примерах). Модуль `pathlib` гарантирует работоспособность такого кода во всех операционных системах.

Модуль `pathlib` появился в Python 3.4 для замены устаревших функций `os.path`. Стандартная библиотека Python поддерживает его начиная с Python 3.6, но если вы работаете с устаревшими версиями Python 2, то рекомендую использовать модуль `pathlib2`, который реализует аналогичную функциональность для Python 2.7. Обратитесь к приложению A, в котором содержатся инструкции по установке сторонних модулей с помощью утилиты `pip`. Документация по устаревшим функциям `os.path` доступна по адресу `https://docs.python.org/3/library/os.path.html`.

Использование оператора / для объединения путей

Обычно для сложения двух целых чисел или чисел с плавающей точкой применяется оператор +. Например, результатом выражения `2 + 2` будет целочисленное значение `4`. Оператор + также служит для конкатенации двух строк, как, например, в выражении `'Здравствуй, ' + 'мир'`, которое в результате дает строку `'Здравствуй, мир'`. Точно так же оператор /, который обычно обозначает деление, позволяет объединять объекты `Path` и строки. Это удобно, если объект `Path` требуется изменить уже после того, как он был создан с помощью функции `Path()`.

Например, введите в интерактивной оболочке следующие инструкции.

```
>>> from pathlib import Path
>>> Path('spam') / 'bacon' / 'eggs'
WindowsPath('spam/bacon/eggs')

>>> Path('spam') / Path('bacon/eggs')
WindowsPath('spam/bacon/eggs')

>>> Path('spam') / Path('bacon', 'eggs')
WindowsPath('spam/bacon/eggs')
```

Благодаря оператору / объединять имена папок можно так же легко, как и выполнять конкатенацию строк. Это более быстрый и безопасный способ, чем конкатенация строк или использование метода `join()`, как показано ниже.

```
>>> homeFolder = r'C:\Users\Al'
>>> subFolder = 'spam'
>>> homeFolder + '\\' + subFolder
'C:\\Users\\Al\\spam'
>>> '\\'.join([homeFolder, subFolder])
'C:\\Users\\Al\\spam'
```

Такой код небезопасен, поскольку применяемая в данном случае обратная косая черта будет работать только в Windows. Можно, конечно, повсюду добавлять блок `if`, в котором будет проверяться значение `sys.platform` (содержащее строку с описыванием операционной системы компьютера) и приниматься решение о том, какой тип косой черты использовать, но это чревато ошибками.

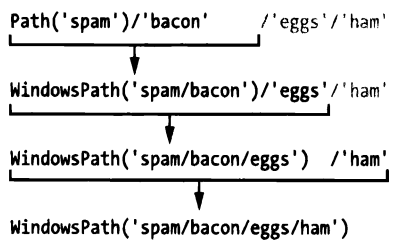
Модуль `pathlib` устраняет описанные проблемы за счет применения оператора / для корректного объединения путей, независимо от того, в какой операционной системе выполняется программа. В следующем примере показано объединение тех же путей, что и в предыдущем примере.

```
>>> homeFolder = Path('C:/Users/Al')
>>> subFolder = Path('spam')
>>> homeFolder / subFolder
WindowsPath('C:/Users/Al/spam')
>>> str(homeFolder / subFolder)
'C:\\Users\\Al\\spam'
```

Единственное, о чем следует помнить при использовании оператора / для объединения путей, — это то, что одно из первых двух значений должно быть объектом `Path`. Python выдаст сообщение об ошибке, если попытаться ввести в интерактивной оболочке следующее выражение.

```
>>> 'spam' / 'bacon' / 'eggs'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

Python выполняет операцию / слева направо и возвращает объект `Path`, поэтому либо первый, либо второй операнд должен быть объектом `Path`. Только тогда все выражение будет интерпретировано как объект `Path`. Вот как это происходит.



Если появится приведенное выше сообщение об ошибке `TypeError: unsupported operand type(s) for /: 'str' and 'str'`, переместите в левую часть выражения объект `Path`.

Оператор `/` заменяет устаревшую функцию `os.path.join()`, информация о которой доступна по адресу `https://docs.python.org/3/library/os.path.html#os.path.join`.

Текущий каталог

Каждой программе, выполняемой на компьютере, назначается *текущий каталог* (current working directory – cwd). Предполагается, что любые имена файлов и папок, которые не начинаются с указания корневой папки, заданы относительно текущего каталога.

Примечание

Несмотря на то что более современный эквивалент термина каталог – папка, обычно говорят текущий каталог (или рабочий каталог), а не текущая папка.

Функция `Path.cwd()` возвращает объект текущего каталога. Сменить текущий каталог можно с помощью функции `os.chdir()`. Введите в интерактивной оболочке следующие инструкции.

```
>>> from pathlib import Path
>>> import os
>>> Path.cwd()
WindowsPath('C:/Users/Al/AppData/Local/Programs/Python/Python37')
>>> os.chdir('C:\\Windows\\System32')
>>> Path.cwd()
WindowsPath('C:/Windows/System32')
```

В данном случае текущий каталог – *C:\Users\Al\AppData\Local\Programs\Python\Python37*, поэтому имя файла *project.docx* будет трактоваться как *C:\Users\Al\AppData\Local\Programs\Python\Python37\project.docx*. После смены текущего каталога на *C:\Windows\System32* имя файла *project.docx* будет трактоваться как *C:\Windows\System32\project.docx*.

Если попытаться перейти в несуществующий каталог, Python выдаст сообщение об ошибке.

```
>>> os.chdir('C:/ThisFolderDoesNotExist')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [WinError 2] The system cannot find the file
specified: 'C:/ThisFolderDoesNotExist'
```

В модуле `pathlib` не существует функции для смены текущего каталога. Это связано с тем, что изменение текущего каталога во время работы программы часто приводит к трудно обнаруживаемым ошибкам.

Функция `os.getcwd()` — это устаревший способ получения имени текущего каталога в виде строки.

Домашний каталог

У каждого пользователя системы имеется папка для собственных файлов, называемая *домашним каталогом*. Функция `Path.home()` возвращает объект `Path` домашнего каталога.

```
>>> Path.home()
WindowsPath('C:/Users/Al')
```

Расположение домашних каталогов пользователей зависит от операционной системы:

- Windows — папка *C:\Users*;
- macOS — папка */Users*;
- Linux — папка */home*.

Ваши сценарии почти наверняка будут иметь право чтения/записи файлов в домашнем каталоге, так что это идеальное место для размещения файлов, с которыми будут работать программы Python.

Абсолютные и относительные пути

Есть два способа задать путь к файлу:

- *абсолютный путь* — всегда начинается с имени корневой папки;
- *относительный путь* — указывается относительно текущего каталога программы.

Существуют также каталоги, обозначаемые одной (`.`) или двумя (`..`) точками. Это не реальные папки, а специальные имена, которые можно

использовать при задании путей. Одиночная точка соответствует текущей папке, а двойная точка – родительской папке.

На рис. 9.2 приведен пример расположения папок и файлов. Текущий каталог в данном случае – *C:\bacon*.

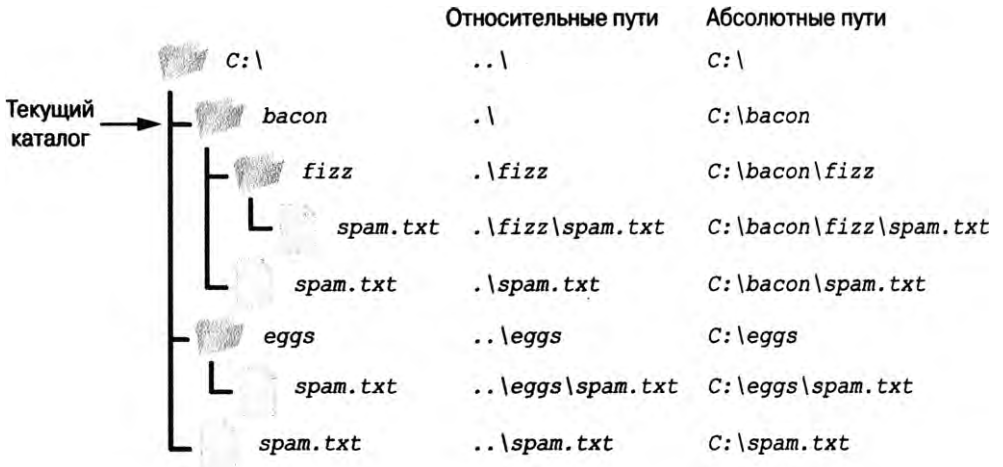


Рис. 9.2. Относительные пути доступа к папкам и файлам в текущем каталоге C:\bacon

Имя .\ в начале относительного пути необязательное. Например, пути *.\spam.txt* и *spam.txt* ведут к одному и тому же файлу.

Создание новых папок с помощью функции os.makedirs()

В программе можно создавать новые папки с помощью функции `os.makedirs()`. Введите в интерактивной оболочке следующие инструкции.

```
>>> import os
>>> os.makedirs('C:\\delicious\\walnut\\waffles')
```

В результате будет создана не только папка *C:\delicious*, но и расположенная в ней папка *walnut*, а также расположенная в папке *C:\delicious\walnut* папка *waffles*. Таким образом, функция `os.makedirs()` создает все необходимые промежуточные папки, гарантируя существование полного пути. Соответствующая иерархия папок показана на рис. 9.3.

Чтобы создать каталог из объекта `Path`, вызовите для него метод `mkdir()`. Например, ниже создается папка *spam*, находящаяся в домашнем каталоге автора книги.

```
>>> from pathlib import Path
>>> Path(r'C:\Users\Al\spam').mkdir()
```

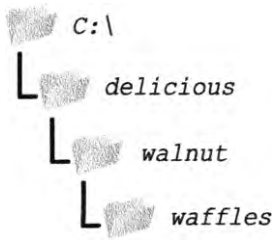


Рис. 9.3. Результат выполнения функции `os.makedirs('C:\\delicious\\walnut\\waffles')`

Учтите, что метод `mkdir()` способен за один раз создать всего один каталог. Он не может создавать несколько подкаталогов одновременно, в отличие от функции `os.makedirs()`.

Обработка абсолютных и относительных путей

Модуль `pathlib` содержит методы, позволяющие проверить, является ли данный путь абсолютным, и получить абсолютный путь из относительного пути.

Метод `is_absolute()` объекта `Path` вернет `True`, если объект представляет абсолютный путь, или `False`, если объект представляет относительный путь. Введите в интерактивной оболочке следующие инструкции, только в последнем случае укажите собственную папку.

```
>>> Path.cwd()
WindowsPath('C:/Users/Al/AppData/Local/Programs/Python/Python37')
>>> Path.cwd().is_absolute()
True
>>> Path('spam/bacon/eggs').is_absolute()
False
```

Чтобы получить абсолютный путь на основе относительного, можно поместить `Path.cwd() /` перед объектом `Path` относительного пути. В конце концов, когда говорят "относительный путь", почти всегда подразумевают путь относительно текущего каталога. Введите в интерактивной оболочке следующие инструкции.

```
>>> Path('my/relative/path')
WindowsPath('my/relative/path')
>>> Path.cwd() / Path('my/relative/path')
WindowsPath('C:/Users/Al/AppData/Local/Programs/Python/Python37/
my/relative/path')
```

Если путь задан относительно какого-то другого пути, подставьте этот путь вместо `Path.cwd()`. В следующем примере абсолютный путь формируется на основе домашнего каталога, а не текущего.

```
>>> Path('my/relative/path')
WindowsPath('my/relative/path')
>>> Path.home() / Path('my/relative/path')
WindowsPath('C:/Users/Al/my/relative/path')
```

Модуль `os.path` тоже содержит несколько полезных функций для работы с абсолютными и относительными путями.

- Функция `os.path.abspath(`*путь*`)` возвращает строку абсолютного пути для заданного аргумента. Это удобный способ преобразовать относительный путь в абсолютный.
- Функция `os.path.isabs(`*путь*`)` возвращает `True`, если аргумент представляет абсолютный путь, и `False`, если аргумент — относительный путь.
- Функция `os.path.relpath(`*путь*`, `*начало*`)` возвращает строку относительного пути, который формируется от каталога *начало* до каталога *путь*. Если аргумент *начало* не задан, в качестве начальной точки используется текущий каталог.

Попробуйте протестировать эти функции в интерактивной оболочке.

```
>>> os.path.abspath('.')
'C:\\Users\\Al\\AppData\\Local\\Programs\\Python\\Python37'
>>> os.path.abspath('.\Scripts')
'C:\\Users\\Al\\AppData\\Local\\Programs\\Python\\Python37\\Scripts'
>>> os.path.isabs('.')
False
>>> os.path.isabs(os.path.abspath('.'))
True
```

В данном случае каталогу `.` соответствует абсолютный путь *C:\Users\Al\AppData\Local\Programs\Python\Python37*.

Введите в интерактивной оболочке следующие инструкции.

```
>>> os.path.relpath('C:\Windows', 'C:\')
'Windows'
>>> os.path.relpath('C:\Windows', 'C:\spam\eggs')
'..\..\Windows'
```

Если в первом аргументе задана папка, не являющаяся подкаталогом второго аргумента, то при формировании относительного пути будут использованы каталоги `..`, позволяющие подняться вверх по дереву каталогов.

Получение отдельных частей пути

С помощью атрибутов объекта `Path` можно извлекать различные фрагменты пути в строковом виде. Это может быть полезно для создания новых каталогов на основе существующих. Соответствующая терминология представлена на рис. 9.4.

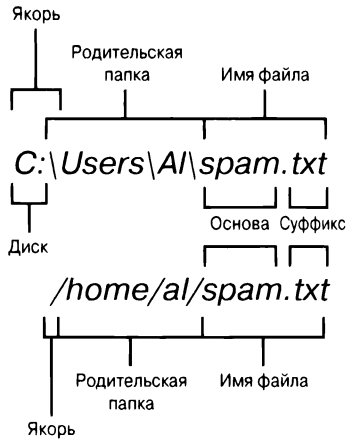


Рис. 9.4. Фрагменты пути к файлу в Windows (сверху) и macOS/Linux (снизу)

Полное имя файла состоит из следующих частей.

- *Якорь* (`anchor`), представляющий собой корневой каталог файловой системы.
- В Windows *диск* (`drive`) — это одиночная буква, которая обычно обозначает физический жесткий диск или другое устройство хранения.
- *Родительская папка* (`parent`), которая представляет собой папку, содержащую файл.
- *Имя файла* (`name`), состоящее из *основы* (`stem`) и *суффикса* (или *расширения*, `suffix`).

Обратите внимание на то, что в Windows объекты `Path` включают атрибут `drive`, которого нет у объектов `Path` в macOS и Linux. Атрибут `drive` не содержит начальную обратную косую черту.

Введите в интерактивной оболочке следующие инструкции.

```
>>> p = Path('C:/Users/Al/spam.txt')
>>> p.anchor
'C:\'
>>> p.parent    # объект Path, а не строка
WindowsPath('C:/Users/Al')
```

```
>>> p.name
'spam.txt'
>>> p.stem
'spam'
>>> p.suffix
'.txt'
>>> p.drive
'C:'
```

Все атрибуты содержат строки, кроме атрибута `parent`, который содержит объект `Path`.

С помощью атрибута `parents` можно узнать родительские папки объекта `Path` (целочисленный индекс определяет количество переходов вверх по дереву каталогов).

```
>>> Path.cwd()
WindowsPath('C:/Users/Al/AppData/Local/Programs/Python/Python37')
>>> Path.cwd().parents[0]
WindowsPath('C:/Users/Al/AppData/Local/Programs/Python')
>>> Path.cwd().parents[1]
WindowsPath('C:/Users/Al/AppData/Local/Programs')
>>> Path.cwd().parents[2]
WindowsPath('C:/Users/Al/AppData/Local')
>>> Path.cwd().parents[3]
WindowsPath('C:/Users/Al/AppData')
>>> Path.cwd().parents[4]
WindowsPath('C:/Users/Al')
>>> Path.cwd().parents[5]
WindowsPath('C:/Users')
>>> Path.cwd().parents[6]
WindowsPath('C:/')
```

В более старом модуле `os.path` имеются похожие функции, позволяющие получить различные части пути. В частности, функция `os.path.dirname(`*`путь`*`)` возвращает имя папки (все, что находится перед завершающей косой чертой). Функция `os.path.basename(`*`путь`*`)` возвращает имя файла (все, что идет после завершающей косой черты). Разница проиллюстрирована на рис. 9.5.

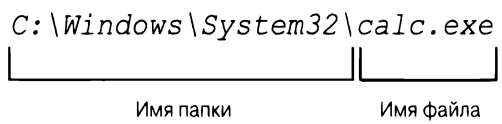


Рис. 9.5. Имя файла идет после завершающей косой черты, имя папки — все, что находится перед завершающей косой чертой

Введите в интерактивной оболочке следующие инструкции.

```
>>> calcFilePath = 'C:\\Windows\\System32\\calc.exe'
>>> os.path.basename(calcFilePath)
'calc.exe'
>>> os.path.dirname(calcFilePath)
'C:\\Windows\\System32'
```

Если нужно получить имя папки вместе с именем файла, вызовите функцию `os.path.split()`, которая возвращает кортеж из двух строк.

```
>>> calcFilePath = 'C:\\Windows\\System32\\calc.exe'
>>> os.path.split(calcFilePath)
('C:\\Windows\\System32', 'calc.exe')
```

Аналогичного результата можно добиться, поместив результаты вызовов функций `os.path.dirname()` и `os.path.basename()` в кортеж.

```
>>> (os.path.dirname(calcFilePath), os.path.basename(calcFilePath))
('C:\\Windows\\System32', 'calc.exe')
```

Конечно же, проще вызвать функцию `os.path.split()`.

Следует также отметить, что функция `os.path.split()` не возвращает список папок в строке пути. Чтобы получить такой список, используйте строковый метод `split()` и разбейте строку по переменной `os.sep`. Эта переменная содержит корректную версию разделителя для той операционной системы, в которой выполняется программа (`'\\'` в Windows и `'/'` в macOS и Linux).

Например, введите в интерактивной оболочке следующую инструкцию.

```
>>> calcFilePath.split(os.sep)
['C:', 'Windows', 'System32', 'calc.exe']
```

Она возвращает имена всех папок, образующих данный путь.

В macOS и Linux возвращаемый список папок начинается с пустой строки.

```
>>> '/usr/bin'.split(os.sep)
['', 'usr', 'bin']
```

Определение размеров файлов и содержимого папок

Научившись работать с путями доступа к файлам, можно приступить к сбору информации о конкретных файлах и папках. Модуль `os.path`

содержит функции, позволяющие узнавать размеры файлов (в байтах) и определять, какие файлы и папки содержатся в заданной папке.

- Функция `os.path.getsize(`*путь*`)` возвращает размер заданного файла в байтах.
- Функция `os.listdir(`*путь*`)` возвращает список всех файлов в каталоге *путь*. (Эта функция содержится в модуле `os`, а не `os.path`.)

Введите в интерактивной оболочке следующие инструкции.

```
>>> os.path.getsize('C:\\Windows\\System32\\calc.exe')
27648
>>> os.listdir('C:\\Windows\\System32')
['0409', '12520437.cpx', '12520850.cpx', '5U877.ax', 'aaclient.dll',
-- Опущено --
'xwtpdui.dll', xwtpw32.dll', 'zh-CN', 'zh-HK', 'zh-TW', 'zipfldr.dll']
```

Как видите, программа *calc.exe* имеет размер 27 648 байт, а в папке *C:\Windows\system32* содержится множество файлов. Если требуется узнать суммарный объем всех файлов, находящихся в заданной папке, то это можно сделать так, как показано ниже.

```
>>> totalSize = 0
>>> for filename in os.listdir('C:\\Windows\\System32'):
      totalSize = totalSize +
os.path.getsize(os.path.join('C:\\Windows\\System32', filename))
>>> print(totalSize)
2559970473
```

В цикле последовательно перебираются все файлы, содержащиеся в папке *C:\Windows\System32*, и значение переменной `totalSize` каждый раз увеличивается на размер очередного файла. Функция `os.path.join()` используется для присоединения имени папки к текущему имени файла при вызове функции `os.path.getsize()`. Целочисленное значение, возвращаемое функцией `os.path.getsize()`, суммируется с текущим значением переменной `totalSize`. По завершении цикла выводится значение `totalSize`, содержащее суммарный объем содержимого папки *C:\Windows\System32*.

Изменение списка файлов с помощью шаблонов

Метод `glob()` объекта `Path` — более удобный аналог метода `listdir()`. С его помощью можно получить список содержимого папки в соответствии с шаблоном. *Шаблоны glob* — это упрощенная разновидность регулярных выражений для применения в командной строке. Метод `glob()` возвращает объект-генератор (его описание выходит за рамки книги), который нужно передать методу `list()` для удобного просмотра в интерактивной оболочке.

```
>>> p = Path('C:/Users/Al/Desktop')
>>> p.glob('*')
<generator object Path.glob at 0x000002A6E389DED0>
>>> list(p.glob('*'))      # создание списка на основе генератора
[WindowsPath('C:/Users/Al/Desktop/1.png'),
 WindowsPath('C:/Users/Al/Desktop/22-ap.pdf'),
 WindowsPath('C:/Users/Al/Desktop/cat.jpg'),
  -- Опущено --
 WindowsPath('C:/Users/Al/Desktop/zzz.txt')]
```

Звездочка (*) означает "произвольное количество любых символов", поэтому вызов `p.glob('*')` возвращает генератор для всех файлов в папке `p`.

Как и в случае регулярных выражений, шаблоны бывают самыми разными.

```
>>> list(p.glob('*.txt')    # вывод всех текстовых файлов
[WindowsPath('C:/Users/Al/Desktop/foo.txt'),
  -- Опущено --
 WindowsPath('C:/Users/Al/Desktop/zzz.txt')]
```

Шаблон `'*.txt'` позволяет отобрать файлы, имена которых начинаются с любой комбинации символов и заканчиваются строкой `'.txt'` (расширение текстового файла).

В отличие от звездочки, вопросительный знак (?) означает "любой одиночный символ".

```
>>> list(p.glob('project?.docx')
[WindowsPath('C:/Users/Al/Desktop/project1.docx'),
 WindowsPath('C:/Users/Al/Desktop/project2.docx'),
  -- Опущено --
 WindowsPath('C:/Users/Al/Desktop/project9.docx')]
```

Выражение `'project?.docx'` соответствует имени *project1.docx* или *project5.docx*, но не *project10.docx*, поскольку знак ? соответствует только *одному* символу, но не, например, строке `'10'`.

Наконец, можно объединить звездочку и знак вопроса для создания еще более сложного шаблона, как показано ниже.

```
>>> list(p.glob('*.?x?')
[WindowsPath('C:/Users/Al/Desktop/calc.exe'),
 WindowsPath('C:/Users/Al/Desktop/foo.txt'),
  -- Опущено --
 WindowsPath('C:/Users/Al/Desktop/zzz.txt')]
```

Шаблон `'*.?x?'` соответствует файлам с любым именем и любым трехсимвольным расширением, в котором средний символ — `'x'`.

С помощью метода `glob()` можно отобрать файлы в каталоге, с которыми требуется выполнить определенную операцию. Используйте цикл `for` для обхода списка, возвращаемого методом `glob()`.

```
>>> p = Path('C:/Users/Al/Desktop')
>>> for textFilePathObj in p.glob('*.txt'):
...     print(textFilePathObj)    # вывод объекта Path в виде строки
...     # Выполнение операций с текстовым файлом
...
C:\Users\Al\Desktop\foo.txt
C:\Users\Al\Desktop\spam.txt
C:\Users\Al\Desktop\zzz.txt
```

Если необходимо выполнить какую-либо операцию с каждым файлом в каталоге, можно использовать метод `os.listdir(p)` или `p.glob('*')`.

Проверка существования пути

Многие функции Python аварийно завершаются с выдачей сообщения об ошибке, если предоставленный им путь не существует. К счастью, у объектов `Path` есть методы для проверки того, существует ли заданный путь и соответствует ли он файлу или папке. Если переменная `p` содержит объект `Path`, то можно ожидать следующее:

- метод `p.exists()` возвращает `True`, если путь существует; в противном случае возвращается `False`;
- метод `p.is_file()` возвращает `True`, если путь существует и соответствует файлу; в противном случае возвращается `False`;
- метод `p.is_dir()` возвращает `True`, если путь существует и соответствует каталогу; в противном случае возвращается `False`.

Вот, что получится, если протестировать эти методы в интерактивной оболочке.

```
>>> winDir = Path('C:/Windows')
>>> notExistsDir = Path('C:/This/Folder/Does/Not/Exist')
>>> calcFile = Path('C:/Windows/System32/calc.exe')
>>> winDir.exists()
True
>>> winDir.is_dir()
True
>>> notExistsDir.exists()
False
>>> calcFile.is_file()
True
>>> calcFile.is_dir()
False
```

Можно определить, подключен ли в данный момент к компьютеру привод DVD или флеш-накопитель USB, проверив существование диска с помощью метода `exists()`. Например, если нужно проверить наличие диска *D:* в Windows, то это можно сделать с помощью следующих инструкций.

```
>>> dDrive = Path('D:/')
>>> dDrive.exists()
False
```

В устаревшем модуле `os.path` содержатся аналогичные функции `os.path.exists(`*путь*`)`, `os.path.isfile(`*путь*`)` и `os.path.isdir(`*путь*`)`. Начиная с Python 3.6 эти функции поддерживают аргументы типа `Path`.

Процесс чтения и записи файлов

Теперь вы знаете, как задать путь к файлу для операций чтения и записи. Функции, рассматриваемые в следующих разделах, будут применяться к *простым текстовым файлам*. Такие файлы содержат только текстовые символы, не сопровождающиеся информацией о шрифте, кегле и цвете текста. В качестве примера можно привести файлы с расширением *.txt* или файлы сценариев Python с расширением *.py*. Подобные файлы можно открыть с помощью приложения Блокнот в Windows или TextEdit в macOS. Содержимое простых текстовых файлов можно обрабатывать как обычную строку.

Другой тип файлов — *бинарные файлы*, к которым относятся, в частности, документы, создаваемые текстовыми процессорами, PDF-файлы, графические файлы, файлы электронных таблиц, а также исполняемые программы. Открыв бинарный файл в приложении Блокнот или TextEdit, вы увидите бессмысленный набор странных символов (рис. 9.6).

Поскольку различные типы бинарных файлов должны обрабатываться по-разному, мы не будем пытаться непосредственно читать и записывать такие файлы. Существуют специальные модули, которые упрощают работу с бинарными файлами. С одним из них — `shelve` — мы познакомимся далее. Метод `read_text()` модуля `pathlib` возвращает строку с полным содержимым текстового файла. Метод `write_text()` создает новый текстовый файл (или перезаписывает существующий) на основе переданной ему строки содержимого. Введите в интерактивной оболочке следующие инструкции.

```
>>> from pathlib import Path
>>> p = Path('spam.txt')
>>> p.write_text('Здравствуй, мир!')
16
>>> p.read_text()
'Здравствуй, мир!'
```

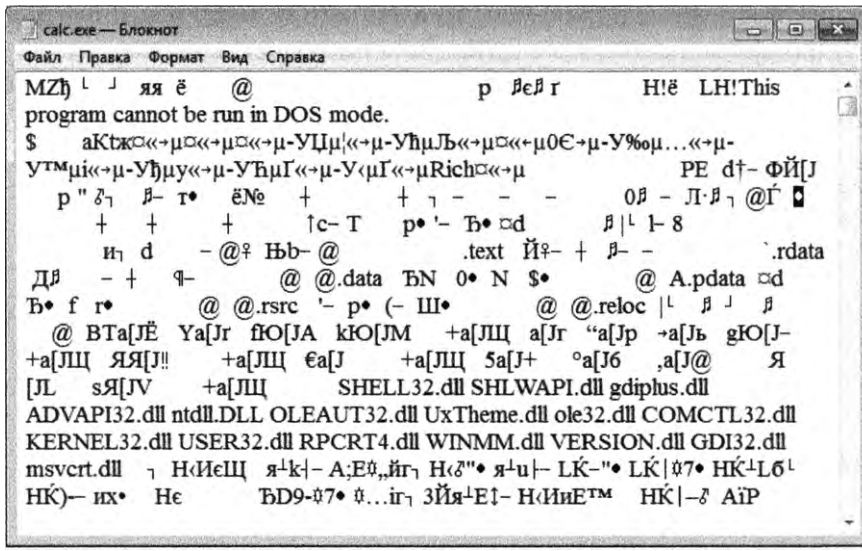


Рис. 9.6. Программа calc.exe, открытая в приложении Блокнот

Здесь создается файл *spam.txt*, содержащий строку `'Здравствуй, мир!'`. Значение `16`, возвращаемое методом `write_text()`, сообщает о том, что в файл было записано 16 символов (обычно эта информация игнорируется). Метод `read_text()` считывает содержимое нового файла и возвращает его в виде строки:

```
'Здравствуй, мир!'
```

Методы объекта `Path` реализуют только базовые операции с файлами. Более распространенный способ записи в файл предполагает использование функции `open()` и файловых объектов. В Python операции чтения/записи файлов выполняются в три этапа:

1) вызов функции `open()`, которая возвращает объект `File`;
2) вызов метода `read()` или `write()` объекта `File`;
3) закрытие файла путем вызова метода `close()` объекта `File`.

Открытие файла с помощью функции open()

Чтобы открыть файл с помощью функции `open()`, передайте ей строку пути к файлу. Это может быть как абсолютный, так и относительный путь. Функция `open()` возвращает объект `File`.

Создайте текстовый файл *hello.txt* с помощью приложения Блокнот или TextEdit. Введите в него строку **`'Здравствуй, мир!'`** и сохраните файл

в своей домашней папке. Затем введите в интерактивной оболочке следующую инструкцию:

```
>>> helloFile = open(Path.home() / 'hello.txt')
```

В функцию `open()` можно также передать строку пути. Если вы работаете в Windows, введите в интерактивной оболочке следующую инструкцию:

```
>>> helloFile = open('C:\\Users\\ваша_папка\\hello.txt')
```

В macOS введите следующее:

```
>>> helloFile = open('/Users/ваша_папка/hello.txt')
```

Подставьте вместо *ваша_папка* имя своей домашней папки, например `'C:\\Users\\Al\\hello.txt'`. Следует отметить, что в Python 3.6 функция `open()` поддерживает только объекты `Path`. В предыдущих версиях нужно было всегда передавать строку пути.

Обе приведенные выше команды открывают файл в режиме чтения простого текста или, для краткости, *в режиме чтения*. В этом режиме Python позволяет только считывать данные из файла; вы не сможете записать данные в файл или каким-то образом изменить его содержимое. Такой режим устанавливается по умолчанию для файлов, открываемых в Python. Но если вы не хотите полагаться на установки по умолчанию, то можете явно задать этот режим, передав функции `open()` строку `'r'` в качестве второго аргумента. Таким образом, вызовы `open('/Users/Al/hello.txt', 'r')` и `open('/Users/Al/hello.txt')` означают одно и то же.

Функция `open()` возвращает объект `File`, который представляет файл. Это еще один тип данных в Python, как списки или словари, с которыми вы уже знакомы. В предыдущем примере объект `File` был сохранен в переменной `helloFile`. Теперь всякий раз, когда понадобится прочитать или записать данный файл, достаточно будет вызвать соответствующий метод объекта `File` для переменной `helloFile`.

Чтение содержимого файла

Если требуется прочитать все содержимое файла в виде одной большой строки, используйте метод `read()` объекта `File`. Продолжим работу с файлом *hello.txt*, который хранится в переменной `helloFile`. Введите в интерактивной оболочке следующие инструкции.

```
>>> helloContent = helloFile.read()
>>> helloContent
'Здравствуй, мир!'
```

Альтернативный вариант – использование метода `readlines()` для чтения списка строк из файла. Например, создайте файл *sonnet29.txt* в той же папке, в которой находится файл *hello.txt*, и введите в него следующий текст.

```
When, in disgrace with fortune and men's eyes,
I all alone beweep my outcast state,
And trouble deaf heaven with my bootless cries,
And look upon myself and curse my fate,
```

В процессе ввода текста не забывайте нажимать клавишу <Enter> в конце каждой строки. Затем введите в интерактивной оболочке следующие инструкции.

```
>>> sonnetFile = open(Path.home() / 'sonnet29.txt')
>>> sonnetFile.readlines()
["When, in disgrace with fortune and men's eyes,\n",
 'I all alone beweep my outcast state,\n',
 'And trouble deaf heaven with my bootless cries,\n',
 'And look upon myself and curse my fate,']
```

Обратите внимание на то, что каждое из строковых значений, за исключением последнего, заканчивается символом новой строки (`\n`). Зачастую работать со списком строк проще, чем с одной длинной строкой.

Запись в файл

Python позволяет записывать содержимое в файл аналогично тому, как функция `print()` выводит строки на экран. Но записать что-либо в файл, открытый в режиме чтения, невозможно. Вместо этого файл должен быть открыт в *режиме записи*.

В режиме записи содержимое существующего файла удаляется, и новые данные записываются "с чистого листа", аналогично тому, как в операции присваивания старое значение переменной заменяется новым. Чтобы открыть файл в режиме записи, следует передать методу `open()` строку `'w'` в качестве второго аргумента. Поддерживается также *режим добавления*, в котором новый текст добавляется в конец существующего файла. Эту операцию можно рассматривать как присоединение нового значения к списку, хранящемуся в переменной, а не полную перезапись содержимого переменной. Чтобы открыть файл в режиме добавления, следует передать методу `open()` строку `'a'` в качестве второго аргумента.

Если файла с именем, переданным методу `open()`, не существует, то как в режиме записи, так и в режиме добавления будет создан новый, пустой файл. Прежде чем повторно открывать файл после завершения операции чтения или записи, его предварительно нужно закрыть с помощью метода `close()`.

Введите в интерактивной оболочке следующие инструкции.

```
>>> baconFile = open('bacon.txt', 'w')
>>> baconFile.write('Здравствуй, мир!\n')
17
>>> baconFile.close()
>>> baconFile = open('bacon.txt', 'a')
>>> baconFile.write('Бекон - не овощ.')
25
>>> baconFile.close()
>>> baconFile = open('bacon.txt')
>>> content = baconFile.read()
>>> baconFile.close()
>>> print(content)
Здравствуй, мир!
Бекон - не овощ.
```

Сначала мы открываем файл *bacon.txt* в режиме записи. Поскольку такого файла пока что не существует, Python создает его. В результате вызова метода `write()` в открытый файл записывается строка `'Здравствуй, мир!\n'` и возвращается количество записанных символов, включая символ новой строки. После этого мы закрываем файл.

Чтобы дополнить содержимое существующего файла новым текстом, не заменяя только что записанную строку, мы открываем файл в режиме добавления текста, записываем в файл строку `'Бекон - не овощ.'` и закрываем его. Наконец, чтобы вывести содержимое файла на экран, мы открываем файл в режиме чтения, вызываем метод `read()`, сохраняем полученный объект `File` в переменной `content`, закрываем файл и выводим на экран его содержимое.

Учтите, что метод `write()` не добавляет автоматически символ новой строки в конец записываемой строки, как это делает функция `print()`. Данный символ нужно добавлять самостоятельно.

Начиная с Python 3.6 вместо строки имени файла можно передавать в функцию `open()` объект `Path`.

Сохранение переменных с помощью модуля `shelve`

С помощью модуля `shelve` можно сохранять переменные в бинарных файлах-хранилищах. Благодаря этому программа сможет впоследствии восстановить значения переменных, считывая данные с жесткого диска, что

позволяет реализовать в программе функции сохранения и открытия файлов. Например, можно задать конфигурационные настройки, сохранить их в файле хранилища и загрузить при последующем запуске программы.

Введите в интерактивной оболочке следующие инструкции.

```
>>> import shelve
>>> shelfFile = shelve.open('mydata')
>>> cats = ['Софи', 'Питер', 'Саймон']
>>> shelfFile['кошки'] = cats
>>> shelfFile.close()
```

Сначала мы импортируем модуль `shelve`. Далее вызывается метод `shelve.open()`, которому передается имя файла. Полученное содержимое хранилища записывается в переменную `shelfFile`. Доступ к хранилищу осуществляется по ключу, как при работе со словарями. Мы создаем список `cats` и записываем его в хранилище с помощью инструкции `shelfFile['кошки'] = cats`, которая ассоциирует список с ключом `'кошки'`. Отметим, что в Python 3.7 методу `open()` модуля `shelve` нужно передать имя файла как строку: объекты `Path` в данном случае не поддерживаются. По завершении работы с хранилищем следует вызвать метод `close()`.

Выполнив этот код в Windows, вы увидите в текущем каталоге три новых файла: *mydata.bak*, *mydata.dat* и *mydata.dir*. В macOS будет создан только один файл: *mydata.db*. Это бинарные файлы, в которых содержатся данные, помещенные в хранилище. Точный формат хранения данных не имеет значения: достаточно знать лишь то, *что* делает модуль `shelve`, а не *как* он это делает. Данный модуль освобождает вас от всех забот, связанных с организацией хранения данных в файлах.

Программа может использовать модуль `shelve` для последующего открытия файлов хранилища и извлечения из них данных. Хранилища не нужно открывать в режиме чтения или записи — как только хранилище будет открыто, вы сможете выполнять оба типа операций. Введите в интерактивной оболочке следующие инструкции.

```
>>> shelfFile = shelve.open('mydata')
>>> type(shelfFile)
<class 'shelve.DbfilenameShelf'>
>>> shelfFile['кошки']
['Софи', 'Питер', 'Саймон']
>>> shelfFile.close()
```

Здесь мы открываем файл хранилища для проверки того, что данные были корректно сохранены. Команда `shelfFile['кошки']` возвращает тот же список, который был сохранен ранее, что подтверждает корректность данных. Метод `close()` закрывает хранилище.

Как и словари, хранилища поддерживают методы `keys()` и `values()`, извлекающие из хранилища коллекции ключей и значений. Эти коллекции не являются истинными списками. Если нужно получить список, следует передать коллекцию функции `list()`. Введите в интерактивной оболочке следующие инструкции.

```
>>> shelfFile = shelve.open('mydata')
>>> list(shelfFile.keys())
['кошки']
>>> list(shelfFile.values())
[['Софи', 'Питер', 'Саймон']]
>>> shelfFile.close()
```

Формат простого текста удобно использовать для создания файлов, которые будут просматриваться в текстовом редакторе наподобие Блокнот или TextEdit. Если же нужно сохранять данные из программ Python, используйте модуль `shelve`.

Сохранение переменных с помощью функции `pprint.pformat()`

В главе 5 говорилось о том, что функция `pprint.pprint()` обеспечивает красивый вывод содержимого списка или словаря на экран, тогда как функция `pprint.pformat()` возвращает тот же самый текст в виде строки. Эта строка не только отформатирована так, что ее удобно читать, но и представляет собой синтаксически правильный код Python. Предположим, в программе имеется словарь и вы хотите сохранить переменную-словарь для будущего использования. Применив функцию `pprint.pformat()`, вы получите строку, которую можно записать в *.py*-файл. Этот файл будет вашим собственным модулем, который вы сможете импортировать всякий раз, когда понадобится использовать хранящуюся в нем переменную.

Введите в интерактивной оболочке следующие инструкции.

```
>>> import pprint
>>> cats = [{'имя': 'Софи', 'описание': 'упитанная'},
            {'имя': 'Питер', 'описание': 'пушистый'}]
>>> pprint.pformat(cats)
"[{'описание': 'упитанная', 'имя': 'Софи'}, {'описание': 'пушистый',
'имя': 'Питер'}]"
>>> fileObj = open('myCats.py', 'w')
>>> fileObj.write('cats = ' + pprint.pformat(cats) + '\n')
92
>>> fileObj.close()
```

Сначала мы импортируем модуль `pprint`, в котором содержится функция `pprint.pformat()`. У нас есть список словарей, сохраненный в переменной `cats`. Чтобы иметь возможность обращаться к списку даже после того, как будет закрыта оболочка, мы используем функцию `pprint.pformat()`, возвращающую список в виде строки, и записываем эту строку в файл *myCats.py*.

Модули, импортируемые с помощью инструкции `import`, представляют собой обычные сценарии Python. После того как строка, возвращаемая `pprint.pformat()`, будет сохранена в *.py*-файле, этот файл станет модулем, который можно импортировать подобно любому другому модулю.

Поскольку сценарии Python — это простые текстовые файлы с расширением *.py*, программы Python способны генерировать другие программы. Впоследствии эти файлы можно импортировать в сценарии.

```
>>> import myCats
>>> myCats.cats
[{'имя': 'Софи', 'описание': 'упитанная'}, {'имя': 'Питер',
'описание': 'пушистый'}]
>>> myCats.cats[0]
{'имя': 'Софи', 'описание': 'упитанная'}
>>> myCats.cats[0]['имя']
'Софи'
```

Преимуществом создания *.py*-файлов (в отличие от сохранения переменных с помощью модуля `shelve`) является то, что они представляют собой простые текстовые файлы, а значит, их содержимое можно легко читать и изменять с помощью обычного текстового редактора. Но для большинства приложений применение модуля `shelve` — более предпочтительный способ сохранения переменных в файле. В простой текстовый файл можно записать только данные элементарных типов, таких как целые числа, числа с плавающей точкой, строки, списки и словари. А вот объекты `File`, например, не могут быть закодированы в виде текста.

Проект: генерирование случайных билетов

Предположим, вы преподаете географию группе из 35 студентов и хотите провести контрольную работу на знание столиц штатов США. К сожалению, вы не можете быть уверены в том, что студенты не будут списывать друг у друга. Вы хотите составить билеты таким образом, чтобы вопросы в них располагались в случайном порядке, благодаря чему все билеты будут разными, и это затруднит списывание ответов. Разумеется, составлять такие билеты вручную — долгая и утомительная задача. К счастью, в вашем распоряжении есть Python.

Вот примерный план того, что должна делать программа:

1) создать 35 разных билетов;
2) создать для каждого билета по 50 вопросов с несколькими вариантами ответа, расположив их в случайном порядке;
3) предоставить на каждый вопрос правильный ответ и три случайным образом выбранных неправильных ответа, располагая их в случайном порядке;
4) записать билеты в 35 текстовых файлов;
5) записать ключи ответов в 35 текстовых файлов.

Это означает, что программа должна будет выполнять следующие операции.

1) сохранять названия штатов и их столиц в словаре;
2) вызывать методы `open()`, `write()` и `close()` для текстовых файлов, в которых хранятся билеты и ключи ответов;
3) использовать функцию `random.shuffle()` для рандомизации (перемешивания) вопросов и вариантов ответов.

Шаг 1. Сохранение данных в словаре

Первый шаг заключается в том, чтобы составить "каркас" сценария и наполнить его данными. Создайте файл *randomQuizGenerator.py* и введите в него следующий код.

```
#! python3
# randomQuizGenerator.py - создает билеты с вопросами и ответами,
# расположенными в случайном порядке, вместе с ключами ответов

❶ import random
# Данные билетов: ключи - названия штатов, а значения - столицы
❷ capitals = {'Айдахо': 'Бойсе', 'Айова': 'Де-Мойн', 'Алабама':
'Монтгомери', 'Аляска': 'Джуно', 'Аризона': 'Финикс', 'Арканзас':
'Литл-Рок', 'Вайоминг': 'Шайенн', 'Вашингтон': 'Олимпия',
'Вермонт': 'Монтпилиер', 'Виргиния': 'Ричмонд', 'Висконсин':
'Мадисон', 'Гавайи': 'Гонолулу', 'Делавэр': 'Довер', 'Джорджия':
'Атланта', 'Западная Виргиния': 'Чарлстон', 'Иллинойс':
'Спрингфилд', 'Индиана': 'Индианаполис', 'Калифорния':
'Сакраменто', 'Канзас': 'Топика', 'Кентукки': 'Франкфорт',
'Колорадо': 'Денвер', 'Коннектикут': 'Хартфорд', 'Луизиана':
'Батон-Руж', 'Массачусетс': 'Бостон', 'Миннесота': 'Сент-Пол',
'Миссисипи': 'Джэксон', 'Миссури': 'Джефферсон-Сити', 'Мичиган':
'Лансинг', 'Монтана': 'Хелена', 'Мэн': 'Огаста', 'Мэриленд':
'Аннаполис', 'Небраска': 'Линкольн', 'Невада': 'Карсон-Сити',
'Нью-Джерси': 'Трентон', 'Нью-Йорк': 'Олбани', 'Нью-Мексико':
'Санта-Фе', 'Нью-Гэмпшир': 'Конкорд', 'Огайо': 'Колумбус',
'Оклахома': 'Оклахома-Сити', 'Орегон': 'Сейлем', 'Пенсильвания':
'Гаррисберг', 'Род-Айленд': 'Провиденс', 'Северная Дакота':
```

```
    'Бисмарк', 'Северная Каролина': 'Роли', 'Теннесси': 'Нашвилл',
    'Техас': 'Остин', 'Флорида': 'Таллахасси', 'Южная Дакота': 'Пирр',
    'Южная Каролина': 'Колумбия', 'Юта': 'Солт-Лейк-Сити'}

  # Генерирование 35 файлов билетов
❸ for quizNum in range(35):
      # СДЕЛАТЬ: создать файлы билетов и ключей ответов

      # СДЕЛАТЬ: записать заголовок билета

      # СДЕЛАТЬ: перемешать порядок следования штатов

      # СДЕЛАТЬ: организовать цикл по всем 50 штатам,
      #          создавая вопрос для каждого из них
```

Программа должна располагать вопросы и ответы в случайном порядке, следовательно, необходимо импортировать модуль `random` ❶, чтобы использовать его функции. Переменная `capitals` ❷ содержит словарь, в котором штаты США играют роль ключей, а значениями служат названия столиц штатов. Поскольку мы хотим создать 35 билетов, код, который будет фактически генерировать файлы билетов и ключей ответов (пока что помечен комментариями `'СДЕЛАТЬ'`), должен быть помещен в цикл `for`, выполняющий 35 итераций ❸. (Это число можно изменить, чтобы сгенерировать любое заданное количество билетов.)

Шаг 2. Создание файлов билетов и перемешивание вопросов

Теперь пора заменить комментарии `'СДЕЛАТЬ'` реальным кодом.

Код в цикле будет повторен 35 раз — по одному разу на каждый билет. Прежде всего, необходимо создать сам файл билета. У него должно быть уникальное имя и стандартный заголовок с пустыми полями для имени, даты и группы, которые будут заполняться студентами. Далее необходимо получить список штатов, расположенных в случайном порядке, который впоследствии можно будет использовать для создания вопросов и ответов к каждому билету.

Добавьте в файл *randomQuizGenerator.py* приведенный ниже код.

```
#! python3
# randomQuizGenerator.py - создает билеты с вопросами и ответами,
# расположенными в случайном порядке, вместе с ключами ответов

-- Опущено --

# Генерирование 35 файлов билетов
for quizNum in range(35):
    # Создание файлов билетов и ключей ответов
❶   quizFile = open(f'capitalsquiz{quizNum + 1}.txt','w')
```

```
❷    answerKeyFile = open(f'capitalsquiz_answers{quizNum + 1}.txt','w')

     # Запись заголовка билета
❸    quizFile.write('Имя:\n\nДата:\n\nГруппа:\n\n')
     quizFile.write((' ' * 20) + f'Столицы штатов (билет {quizNum + 1})')
     quizFile.write('\n\n')

     # Перемешивание порядка следования штатов
     states = list(capitals.keys())
❹    random.shuffle(states)

     # СДЕЛАТЬ: организовать цикл по всем 50 штатам,
     #          создавая вопрос для каждого из них
```

Файлы билетов будут называться *capitalsquiz<N>.txt*, где *<N>* — уникальный номер билета, который берется из переменной цикла `quizNum`. Ключи ответов будут храниться в текстовых файлах *capitalsquiz_answers<N>.txt*. На каждой итерации цикла в строках `f'capitalsquiz{quizNum + 1}.txt'` и `f'capitalsquiz_answers{quizNum + 1}.txt'` выполняется подстановка значения `quizNum + 1`, поэтому файлами первого билета и ключа ответа будут *capitalsquiz1.txt* и *capitalsquiz_answers1.txt*. Эта файлы создаются вызовами функции `open()` ❶ и ❷, которой передается строка `'w'` (режим записи) в качестве второго аргумента.

Инструкции `write()` ❸ создают заголовок билета с полями, которые будут заполняться студентами. Наконец, с помощью функции `random.shuffle()` ❹, которая случайным образом переупорядочивает переданный ей список, создается рандомизированный список всех штатов.

Шаг 3. Создание вариантов ответов

Теперь необходимо сгенерировать варианты ответов для каждого вопроса, которые будут помечены буквами от `'A'` до `'Г'`. Нам понадобится еще один цикл `for` — он будет генерировать содержимое для каждого из 50 вопросов билета. Далее будет идти третий вложенный цикл `for`, предназначенный для генерирования вариантов выбора для каждого вопроса. Дополните имеющийся код, как показано ниже.

```
#! python3
# randomQuizGenerator.py - создает билеты с вопросами и ответами,
# расположенными в случайном порядке, вместе с ключами ответов

-- Опущено --

    # Организация цикла по всем 50 штатам
    # и создание вопроса для каждого из них
    for questionNum in range(50):
```

```
        # Получение правильных и неправильных ответов
❶       correctAnswer = capitals[states[questionNum]]
❷       wrongAnswers = list(capitals.values())
❸       del wrongAnswers[wrongAnswers.index(correctAnswer)]
❹       wrongAnswers = random.sample(wrongAnswers, 3)
❺       answerOptions = wrongAnswers + [correctAnswer]
❻       random.shuffle(answerOptions)

        # СДЕЛАТЬ: записать варианты вопросов
        #          и ответов в файл билета

        # СДЕЛАТЬ: записать ключ ответа в файл
```

Корректный ответ получить легко — он хранится в виде значения в словаре `capitals` ❶. Данный цикл проходит по штатам, содержащимся в перемешанном списке штатов, от `states[0]` до `states[49]`, находит каждый штат в списке `capitals` и сохраняет название его столицы в переменной `correctAnswer`.

Со списком возможных неправильных ответов дело обстоит несколько сложнее. Его можно получить, продублировав все значения из словаря `capitals` ❷, удалив правильный ответ ❸ и выбрав три случайных значения из этого списка ❹. Функция `random.sample()` упрощает такой выбор. Ее первый аргумент — это список, из которого выбираются значения; второй аргумент — это количество значений, которые необходимо выбрать. Полный список вариантов ответа представляет собой сочетание трех неправильных ответов и одного правильного ❺. Наконец, ответы следует перемешать ❻, чтобы правильный ответ не всегда соответствовал варианту `'Г'`.

Шаг 4. Запись содержимого в файлы билетов и ключей ответов

Все, что осталось сделать, — записать вопрос в файл билета, а ответ — в файл ключа ответа. Дополните код, как показано ниже.

```
#! python3
# randomQuizGenerator.py - создает билеты с вопросами и ответами,
# расположенными в случайном порядке, вместе с ключами ответов

-- Опущено --

    # Организация цикла по всем 50 штатам
    # с созданием вопроса для каждого из них
    for questionNum in range(50):
        -- Опущено --

        # Запись вариантов вопросов и ответов в файл билета
        quizFile.write(f'{questionNum + 1}. Выберите столицу штата
                       {states[questionNum]}.\n')
❶       for i in range(4):
```

```
❷            quizFile.write(f"    {'АБВГ'[i]}. {answerOptions[i]}\n")
             quizFile.write('\n')

             # Запись ключа ответа в файл
❸            answerKeyFile.write(f"{questionNum + 1}.
                 {'АБВГ'[answerOptions.index(correctAnswer)]}\n')
     quizFile.close()
     answerKeyFile.close()
```

Цикл `for`, перебирающий целые числа от 0 до 3, записывает варианты ответов из списка `answerOptions` ❶. В выражении `'АБВГ'[i]` ❷ строка `'АБВГ'` трактуется как массив с элементами `'А'`, `'Б'`, `'В'` и `'Г'`, выбираемыми на соответствующей итерации цикла.

В последней строке ❸ метод `answerOptions.index(correctAnswer)` находит целочисленный индекс правильного ответа среди случайно расположенных вариантов, а вычисление выражения `'АБВГ'[answerOptions.index(correctAnswer)]` дает буквенное обозначение правильного варианта ответа, которое записывается в файл ключа ответа.

Ниже показан примерный вид файла *capitalsquiz1.txt*, хотя, разумеется, вопросы и варианты ответов в вашем файле будут другими, в зависимости от результатов вызова функции `random.shuffle()`.

```
Имя:

Дата:

Группа:

                    Столицы штатов (билет 1)
1. Выберите столицу штата Западная Виргиния.
    А. Хартфорд
    Б. Санта-Фе
    В. Гариссберг
    Г. Чарлстон

2. Выберите столицу штата Колорадо.
    А. Роли
    Б. Гаррисберг
    В. Денвер
    Г. Линкольн

-- Опущено --
```

Соответствующий текстовый файл *capitalsquiz_answers1.txt* будет выглядеть примерно так.

```
1. Г
2. В
```

```
3. A
4. B
-- Опущено --
```

Проект: множественный буфер обмена

Давайте перепишем программу рассылки сообщений, рассмотренную в главе 6, чтобы в ней использовался модуль `shelve`. Пользователь теперь сможет сохранять новые строки для загрузки в буфер обмена, не модифицируя код программы. Мы назовем эту программу *mcb.pyw* (поскольку "mcb" короче, чем "multi-clipboard"). Расширение *.pyw* означает, что Python не будет отображать окно терминала в процессе выполнения программы (подробнее подробно об этом читайте в приложении Б).

Программа будет сохранять каждый фрагмент копируемого в буфер текста с использованием отдельного ключевого слова. Например, если вы выполните команду `py mcb.pyw save spam`, то текущее содержимое буфера обмена будет сохранено с ключевым словом `spam`. Впоследствии этот текст можно будет вновь загрузить в буфер обмена с помощью команды `py mcb.pyw spam`. А если пользователь забудет, какие ключевые слова соответствуют тем или иным текстовым фрагментам, то он всегда сможет выполнить команду `py mcb.pyw list` для копирования списка всех ключевых слов в буфер обмена.

Вот что делает данная программа:

1) проверяет аргумент командной строки, содержащий ключевое слово;
2) если этот аргумент — `save`, то содержимое буфера обмена сохраняется с данным ключевым словом;
3) если этот аргумент — `list`, то все ключевые слова копируются в буфер обмена;
4) в противном случае текст, соответствующий ключевому слову, копируется в буфер обмена.

Это означает, что программа должен выполнять следующие действия:

1) считывать аргументы командной строки из переменной `sys.argv`;
2) выполнять операции чтения и записи в буфер обмена;
3) сохранять и загружать файл хранилища.

Если вы работаете в Windows, то легко сможете запустить этот сценарий из окна **Выполнить**, создав пакетный файл *mcb.bat* со следующим содержимым:

```
@pyw.exe C:\Python34\mcb.pyw %*
```

Шаг 1. Комментарии и настройка хранилища

Начнем с создания каркаса сценария, содержащего комментарии и базовые настройки. Введите следующий код.

```
#! python3
# mcb.pyw - сохраняет и загружает фрагменты
# текста в буфер обмена
#
❶ # Использование: py.exe mcb.pyw save <ключевое_слово> - сохраняет
#                       содержимое буфера обмена с ключевым словом
#                  py.exe mcb.pyw <ключевое_слово> - загружает текст,
#                       соответствующий ключевому слову, в буфер обмена
#                  py.exe mcb.pyw list - загружает все ключевые слова
#                       в буфер обмена

❷ import shelve, pyperclip, sys

❸ mcbShelf = shelve.open('mcb')

# СДЕЛАТЬ: сохранить содержимое буфера обмена

# СДЕЛАТЬ: сформировать список ключевых слов
#          и загрузить содержимое

mcbShelf.close()
```

Общую информацию о порядке использования программы принято оформлять в виде комментариев в начале файла ❶. Если вы вдруг забудете, как запустить сценарий, прочитайте еще раз комментарии. Далее импортируются необходимые модули ❷. Для копирования и вставки текста нужен модуль `pyperclip`, а для чтения аргументов командной строки — модуль `sys`. Также потребуется модуль `shelve`: всякий раз, когда пользователь захочет сохранить новый фрагмент находящегося в буфере обмена текста, запишите этот текст в файл хранилища. Впоследствии, если пользователь захочет поместить текст обратно в буфер, откройте файл хранилища и загрузите его в программу. Имя файла хранилища будет включать префикс `mcb` ❸.

Шаг 2. Сохранение содержимого буфера обмена с ключевым словом

Программа выполняет различные действия в зависимости от того, что хочет пользователь: сохранить текст, ассоциировав его с ключевым словом, загрузить текст в буфер обмена или получить список всех имеющихся ключевых слов. Рассмотрим первый случай. Дополните код, как показано ниже.

```
#! python3
# mcb.pyw - сохраняет и загружает фрагменты
# текста в буфер обмена
-- Опущено --

# Сохранение содержимого буфера обмена
❶ if len(sys.argv) == 3 and sys.argv[1].lower() == 'save':
❷     mcbShelf[sys.argv[2]] = pyperclip.paste()
elif len(sys.argv) == 2:
❸     # СДЕЛАТЬ: сформировать список ключевых слов
      #          и загрузить содержимое

mcbShelf.close()
```

Если первый аргумент командной строки (он всегда имеет индекс 1 в списке `sys.argv`) — `'save'` ❶, то вторым аргументом будет ключевое слово для текущего содержимого буфера обмена. Это ключевое слово будет использоваться в качестве ключа для хранилища `mcbShelf`, тогда как значением будет текст, находящийся в данный момент в буфере обмена ❷.

Если передан только один аргумент командной строки, то предполагается, что это либо строка `'list'`, либо ключевое слово для загрузки содержимого в буфер обмена. Этот код будет написан далее. А пока что он заменен комментарием `'СДЕЛАТЬ'` ❸.

Шаг 3. Построение списка ключевых слов и загрузка содержимого, ассоциированного с ключевым словом

Наконец, реализуем два оставшихся пункта: загрузка в буфер обмена текста, ассоциированного с заданным ключевым словом, и получение списка всех доступных ключевых слов. Дополните код, как показано ниже.

```
#! python3
# mcb.pyw - сохраняет и загружает фрагменты
# текста в буфер обмена
-- Опущено --

# Сохранение содержимого буфера обмена
if len(sys.argv) == 3 and sys.argv[1].lower() == 'save':
    mcbShelf[sys.argv[2]] = pyperclip.paste()
elif len(sys.argv) == 2:
    # Формирование списка ключевых слов и загрузка содержимого
❶   if sys.argv[1].lower() == 'list':
❷       pyperclip.copy(str(list(mcbShelf.keys())))
    elif sys.argv[1] in mcbShelf:
❸       pyperclip.copy(mcbShelf[sys.argv[1]])

mcbShelf.close()
```

Если указан только один аргумент командной строки, то сначала необходимо проверить, не является ли этим аргументом строка `'list'` ❶. В таком случае в буфер обмена копируется строковое представление списка ключей хранилища ❷. Пользователь сможет вставить этот список в окно текстового редактора и прочитать его.

В противном случае можно считать, что аргумент командной строки представляет собой ключевое слово. Если оно является ключом хранилища `mcbShelf`, то можно загрузить соответствующее значение в буфер обмена ❸.

Вот и все! В зависимости от операционной системы эта программа может запускаться по-разному. Детали запуска программ в различных операционных системах описаны в приложении Б.

Вспомните программу автоматической рассылки сообщений, которую мы создали в главе 6. Обновление текстовых сообщений потребовало бы изменения исходного кода программы. Это далеко не лучший вариант, поскольку пользователям не нравится, если для обновления программы им приходится самостоятельно вносить изменения в код. Кроме того, при любом изменении исходного кода существует риск случайного внесения новых ошибок. Сохраняя данные для программы не в коде, а в отдельном файле, вы облегчаете использование программы другими людьми и снижаете вероятность появления в ней новых ошибок.

Резюме

Файлы хранятся в папках (другое название — каталоги), и местоположение файла описывается строкой пути. У каждой запущенной программы есть свой текущий каталог, что позволяет указывать пути относительно текущего каталога вместо того, чтобы всегда задавать полный (абсолютный) путь. Модули `pathlib` и `os.path` содержат множество функций, предназначенных для работы с путями доступа к файлам.

В программе можно непосредственно работать с содержимым текстовых файлов. Функция `open()` позволяет открывать такие файлы для чтения их содержимого в виде одной длинной строки (с помощью метода `read()`) или в виде списка строк (с помощью метода `readlines()`). Функция `open()` позволяет открывать файлы в режиме записи или добавления, что дает возможность создавать новые текстовые файлы или добавлять текст в конец существующих файлов.

В предыдущих главах мы использовали буфер обмена в качестве средства вставки готового текста, чтобы не приходилось вводить его вручную. Теперь же вы научились считывать необходимые программе данные непосредственно с жесткого диска, что очень удобно, поскольку файлы менее подвержены изменениям, чем буфер обмена.

В следующей главе вы узнаете о том, как обрабатывать сами файлы, т.е. копировать их, удалять, переименовывать, перемещать и т.п.

Контрольные вопросы

1. Относительно чего задается относительный путь?
2. С чего начинается абсолютный путь?
3. Каким будет результат операции `Path('C:/Users') / 'Al'` в Windows?
4. Каким будет результат операции `'C:/Users' / 'Al'` в Windows?
5. Каково назначение функций `os.getcwd()` и `os.chdir()`?
6. Что означают папки `.` и `..`?
7. В строке *C:\bacon\eggs\spam.txt* что является именем папки, а что — именем файла?
8. Назовите три возможных аргумента функции `open()`, задающих режим открытия файла.
9. Что происходит при открытии существующего файла в режиме записи?
10. В чем разница между методами `read()` и `readlines()`?
11. Какую структуру данных напоминает хранилище, создаваемое с помощью модуля `shelve`?

Учебные проекты

Чтобы закрепить полученные знания на практике, напишите программы для предложенных ниже задач.

Расширение возможностей множественного буфера обмена

Расширьте возможности программы для работы с буфером обмена таким образом, чтобы она поддерживала аргумент командной строки `delete <ключевое_слово>`, который позволяет удалить заданное ключевое слово и связанный с ним текст из хранилища. Если ключевое слово не указано, должны удаляться *все* ключевые слова.

Программа Mad Libs

Напишите программу Mad Libs, которая считывает текстовые файлы и дает пользователю возможность ввести собственный текст в любом месте файла, где встречается слово `'ПРИЛАГАТЕЛЬНОЕ'`, `'СУЩЕСТВИТЕЛЬНОЕ'`, `'НАРЕЧИЕ'` или `'ГЛАГОЛ'`. Например, содержимое текстового файла может быть таким.

```
ПРИЛАГАТЕЛЬНОЕ панда залезла на СУЩЕСТВИТЕЛЬНОЕ и ГЛАГОЛ. Соседний
СУЩЕСТВИТЕЛЬНОЕ не пострадал.
```

Программа найдет вхождения перечисленных слов и предложит пользователю заменить их.

```
Введите прилагательное:
глупая
Введите существительное:
забор
Введите глагол:
упала
Введите существительное:
вольер
```

В результате будет создан следующий текстовый файл.

```
Глупая панда залезла на забор и упала. Соседний
вольер не пострадал.
```

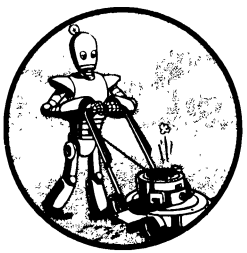
Результаты должны выводиться на экран и сохраняться в новом текстовом файле.

Поиск с помощью регулярных выражений

Напишите программу, которая просматривает все файлы с расширением *.txt* в заданной папке и выполняет поиск строк, соответствующих заданному регулярному выражению. Результаты должны выводиться на экран.

10

УПРАВЛЕНИЕ ФАЙЛАМИ



В предыдущей главе вы научились создавать и записывать в программах Python новые файлы. Но программы могут работать и с файлами, которые уже есть на диске. Возможно, вам приходилось иметь дело с папками, хранящими десятки, сотни и даже тысячи файлов, которые нужно было копировать, переименовывать, перемещать или сжимать вручную. Иногда возникают и другие специфические задачи:

- создание копий всех PDF-файлов (и *только* PDF-файлов) во всех подпапках заданной папки;
- удаление ведущих нулей из имен сотен файлов наподобие *spam001.txt*, *spam002.txt*, *spam003.txt* и т.д., хранящихся в определенной папке;
- сжатие содержимого нескольких папок в один ZIP-файл (это может требоваться в простейшей системе резервного копирования файлов).

Подобные рутинные задачи так и просятся, чтобы их автоматизировали с помощью Python. Запрограммировав компьютер для выполнения такого рода обязанностей, вы превратите его в расторопного, безошибочно функционирующего офисного клерка.

При работе с файлами полезно иметь возможность непосредственно видеть, какое расширение (*.txt*, *.pdf*, *.jpg* и др.) имеет тот или иной файл. В macOS и Linux обозреватель файлов в большинстве случаев автоматически отображает расширения файлов. В Windows расширения могут быть по умолчанию скрыты. Чтобы отобразить их, выполните команду Пуск⇨Панель управления⇨Оформление и персонализация⇨Параметры Проводника, перейдите в открывшемся окне на вкладку Вид и снимите флажок Скрывать расширения для зарегистрированных типов файлов в разделе Дополнительные параметры.

Модуль `shutil`

Модуль `shutil` (от англ. "shell utilities" – утилиты командной оболочки) содержит функции, позволяющие копировать, перемещать, переименовывать и удалять файлы. Для работы с ним необходимо выполнить инструкцию `import shutil`.

Копирование файлов и папок

С помощью модуля `shutil` можно копировать как файлы, так и целые папки.

Функция `shutil.copy(`*`источник, назначение`*`)` скопирует файл *`источник`* в папку *`назначение`*. Оба параметра функции могут быть либо строками, либо объектами `Path`. Если аргумент *`назначение`* – это имя файла, то оно будет использовано в качестве нового имени скопированного файла. Функция возвращает строку либо объект `Path` для скопированного файла.

Чтобы увидеть, как работает функция `shutil.copy()`, введите в интерактивной оболочке следующие инструкции.

```
>>> import shutil, os
>>> from pathlib import Path
>>> p = Path.home()
```

```
❶ >>> shutil.copy(p / 'spam.txt', p / 'some_folder')
  'C:\\Users\\Al\\some_folder\\spam.txt'
❷ >>> shutil.copy(p / 'eggs.txt', p / 'some_folder/eggs2.txt')
  WindowsPath('C:/Users/Al/some_folder/eggs2.txt')
```

В первом случае функция `shutil.copy()` копирует файл *C:\Users\Al\spam.txt* в папку *C:\Users\Al\some_folder*. Возвращаемым значением становится путь к скопированному файлу. Здесь второй аргумент — папка ❶, а имя копии файла совпадает с именем исходного файла: *spam.txt*. Во втором случае функция тоже копирует файл *C:\Users\Al\eggs.txt* в папку *C:\Users\Al\some_folder*, но теперь копии файла присваивается имя *eggs2.txt* ❷.

Функция `shutil.copy()` копирует одиночный файл, тогда как функция `shutil.copytree(`*`источник`*`, `*`назначение`*`)` копирует все дерево каталогов вместе со всеми папками и файлами, которые в нем содержатся. В результате папка *источник* копируется вместе со всеми находящимися в ней файлами и подпапками в папку *назначение*. Оба параметра функции являются строками. Функция возвращает строку пути к скопированной папке.

Введите в интерактивной оболочке следующие инструкции.

```
>>> import shutil, os
>>> from pathlib import Path
>>> p = Path.home()
>>> shutil.copytree(p / 'spam', p / 'spam_backup')
WindowsPath('C:/Users/Al/spam_backup')
```

В результате вызова функции `shutil.copytree()` создается новая папка `spam_backup` с таким же содержимым, как и в исходной папке `spam`. Теперь у вас есть резервная копия папки `spam`.

Перемещение и переименование файлов и папок

Функция `shutil.move(`*`источник`*`, `*`назначение`*`)` перемещает файл или папку *источник* в расположение *назначение* и возвращает строку абсолютного пути к новому расположению.

Если параметру *назначение* соответствует папка, то исходный файл перемещается в эту папку, сохраняя свое текущее имя. Например, введите в интерактивной оболочке следующие инструкции.

```
>>> import shutil
>>> shutil.move('C:\\bacon.txt', 'C:\\eggs')
'C:\\eggs\\bacon.txt'
```

Если папка *eggs* существует в каталоге *C:*, то вызов функции `shutil.move()` означает следующее: "Переместить файл *C:\bacon.txt* в папку *C:\eggs*".

Если в папке *C:\eggs* уже существует файл *bacon.txt*, то он будет заменен. Поскольку таким образом можно случайно потерять нужный файл, при использовании функции `move()` следует проявлять определенную осторожность.

Параметр *назначение* может также задавать имя файла. В следующем примере исходный файл перемещается и переименовывается.

```
>>> shutil.move('C:\\bacon.txt', 'C:\\eggs\\new_bacon.txt')
'C:\\eggs\\new_bacon.txt'
```

Эта строка кода имеет следующий смысл: "Переместить файл *C:\bacon.txt* в папку *C:\eggs* и присвоить перемещенному файлу *bacon.txt* новое имя *new_bacon.txt*".

В предыдущих примерах предполагалось, что в каталоге *C:* существует папка *eggs*. Если же это не так, то функция `move()` переименует файл *bacon.txt* в файл *eggs*.

```
>>> shutil.move('C:\\bacon.txt', 'C:\\eggs')
'C:\\eggs'
```

В данном случае функция `move()`, не обнаружив папку *eggs* в каталоге *C:*, предполагает, что путь назначения обозначает имя файла, а не папки. В результате текстовый файл *bacon.txt* переименовывается в *eggs* (тоже текстовый файл, но без расширения *.txt*) – скорее всего, это совсем не то, что вы хотели сделать! Обнаружить подобную ошибку в программе очень трудно, поскольку функция `move()` может беспрепятственно сделать то, чего вы совершенно не ожидали. Это еще одна из причин, по которым работать с функцией `move()` следует с осторожностью.

Наконец, папки, составляющие путь назначения, должны существовать, иначе Python сгенерирует исключение. Введите в интерактивной оболочке следующую инструкцию.

```
>>> shutil.move('spam.txt', 'c:\\does_not_exist\\eggs\\ham')
Traceback (most recent call last):
  -- Опущено --
FileNotFoundError: [Errno 2] No such file or directory:
'c:\\does_not_exist\\eggs\\ham'
```

Python ищет папки *eggs* и *ham* в каталоге *does_not_exist*. А поскольку такой каталог не существует, переместить файл *spam.txt* по указанному пути невозможно.

Безвозвратное удаление файлов и папок

Если для удаления одиночного файла или одиночной пустой папки можно воспользоваться функциями модуля `os`, то для удаления папки вместе со всем ее содержимым следует использовать модуль `shutil`.

- Функция `os.unlink(`*путь*`)` удаляет файл, находящийся по указанному пути.
- Функция `os.rmdir(`*путь*`)` удаляет папку, находящуюся по указанному пути. Эта папка должна быть пустой, т.е. не содержать никаких других подпапок и файлов.
- Функция `shutil.rmtree(`*путь*`)` удаляет папку, находящуюся по указанному пути, вместе со всеми содержащимися в ней подпапками и файлами.

Используя эти функции в своих программах, соблюдайте осторожность! Часто имеет смысл предварительно запустить версию программы, в которой эти вызовы закомментированы, и проконтролировать с помощью функции `print()`, какие именно файлы планируется удалять. Ниже приведен фрагмент программы, предназначенный для удаления файлов с расширением *.txt*, но из-за допущенной опечатки (выделена полужирным шрифтом) удаляющий файлы с расширением *.rxt*.

```
import os
for filename in os.listdir():
    for filename in Path.home().glob('*.rxt'):
        os.unlink(filename)
```

Если у вас есть важные файлы, имена которых заканчиваются расширением *.rxt*, то все они будут безвозвратно удалены. Вместо этого следует сначала запустить тестовую версию программы.

```
import os
for filename in os.listdir():
    for filename in Path.home().glob('*.rxt'):
        #os.unlink(filename)
        print(filename)
```

Теперь функция `os.unlink()` не будет выполняться, поскольку соответствующая строка закомментирована, и Python проигнорирует ее. Программа лишь выведет на экран имя файла, подлежащего удалению. Запустив такую версию программы, вы сразу же обнаружите, что в программе имеется ошибка, из-за которой она будет ошибочно удалять не текстовые файлы с расширением *.txt*, а файлы с расширением *.rxt*.

Убедившись в том, что программа работает так, как запланировано, удалите строку `print(filename)`, а также символ комментария в строке с вызовом `os.unlink(filename)`. После этого вновь запустите программу для удаления файлов.

Безопасное удаление с помощью модуля send2trash

Поскольку встроенная функция `shutil.rmtree()` безвозвратно удаляет файлы и папки, ее использование связано с немалым риском. Намного более безопасный способ удаления файлов и папок реализован в стороннем модуле `send2trash`. Этот модуль можно установить, выполнив в окне терминала команду `pip install --user send2trash`. (Подробнее об установке сторонних модулей рассказывается в приложении А.)

Модуль `send2trash` намного безопаснее, чем стандартные функции Python, выполняющие операцию удаления, поскольку он отправляет удаляемые файлы и папки в системную корзину, а не удаляет их безвозвратно. Если из-за ошибок в программе будут удалены файлы, которые вы не собирались удалять, но при этом использовался модуль `send2trash`, то впоследствии у вас будет возможность восстановить их из корзины.

После того как вы установите модуль `send2trash`, введите в интерактивной оболочке следующие инструкции.

```
>>> import send2trash
>>> baconFile = open('bacon.txt', 'a')     # создает файл
>>> baconFile.write('Бекон - не овощ.')
25
>>> baconFile.close()
>>> send2trash.send2trash('bacon.txt')
```

Желательно всегда использовать функцию `send2trash.send2trash()` для удаления файлов и папок, чтобы иметь последующую возможность восстановить их из корзины. Но размер свободного дискового пространства при этом не увеличивается, в отличие от безвозвратного удаления файлов. Если необходимо, чтобы программа освобождала дисковое пространство, используйте для удаления файлов и папок функции модулей `os` и `shutil`. Учтите, что функция `send2trash()` может лишь отправлять файлы в корзину, но не восстанавливать их из нее.

Обход дерева каталогов

Предположим, вы хотите переименовать все файлы, находящиеся в определенной папке, а также во всех ее подпапках. Следовательно, вам необходимо выполнить обход всего дерева каталогов, обрабатывая при этом

каждый файл. Написание соответствующей программы — задача нетривиальная; к счастью, в Python для этого есть готовая функция.

Рассмотрим структуру папки *C:\delicious* (рис. 10.1).

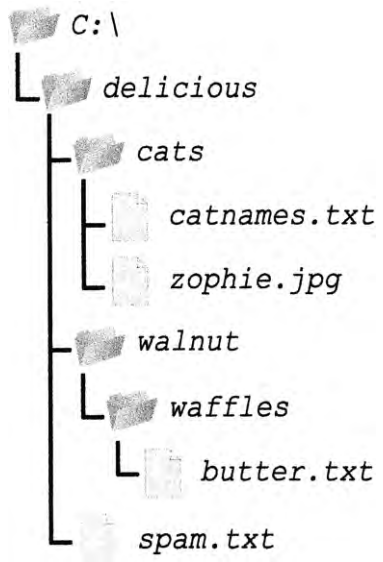


Рис. 10.1. Пример папки, содержащей три подпапки и четыре файла

Ниже приведен пример программы, в которой для обхода дерева каталогов, представленного на рис. 10.1, применяется функция `os.walk()`.

```
import os

for folderName, subfolders, filenames in os.walk('C:\\delicious'):
    print('Текущая папка - ' + folderName)

    for subfolder in subfolders:
        print('ПОДПАПКА ПАПКИ ' + folderName + ': ' + subfolder)

    for filename in filenames:
        print('ФАЙЛ В ПАПКЕ ' + folderName + ': '+ filename)

    print('')
```

В функцию `os.walk()` передается единственное строковое значение: путь к папке. Ее можно использовать в цикле `for` для обхода дерева каталогов примерно так же, как и функцию `range()` для перебора всех целых чисел из заданного диапазона. Но, в отличие от функции `range()`, функция `os.walk()` на каждой итерации цикла возвращает три значения:

- строку, содержащую текущее имя папки;
- список строк, представляющих имена подпапок, которые содержатся в текущей папке;
- список строк, представляющих имена файлов, которые содержатся в текущей папке.

(Под текущей папкой подразумевается папка, используемая на текущей итерации цикла. Применение функции `os.walk()` *не* приводит к смене текущего каталога программы.)

Подобно счетчику `i` в коде `for i in range(10):`, имена переменных для трех вышеперечисленных значений можно выбирать самостоятельно. Удобнее всего использовать имена `foldername`, `subfolders` и `filenames`.

Если вы запустите эту программу, то результат будет примерно таким.

```
Текущая папка C:\delicious
ПОДПАПКА ПАПКИ C:\delicious: cats
ПОДПАПКА ПАПКИ C:\delicious: walnut
ФАЙЛ В ПАПКЕ C:\delicious: spam.txt

Текущая папка C:\delicious\cats
ФАЙЛ В ПАПКЕ C:\delicious\cats: catnames.txt
ФАЙЛ В ПАПКЕ C:\delicious\cats: zophie.jpg

Текущая папка C:\delicious\walnut
ПОДПАПКА ПАПКИ C:\delicious\walnut: waffles

Текущая папка C:\delicious\walnut\waffles
ФАЙЛ В ПАПКЕ C:\delicious\walnut\waffles: butter.txt
```

Поскольку функция `os.walk()` возвращает списки строк для переменных `subfolders` и `filenames`, их можно использовать во вложенных циклах `for`.

Сжатие файлов с помощью модуля `zipfile`

Вы наверняка знакомы с ZIP-файлами (имеющими расширение *.zip*), в которых в сжатом виде хранится содержимое других файлов. При сжатии размер файла уменьшается, что немаловажно при передаче файлов по сети. А поскольку ZIP-файл может содержать множество файлов и папок, он представляет собой очень удобный способ архивации. Этот единственный файл, называемый *архивным*, можно, например, присоединить к сообщению электронной почты.

Программы Python могут как создавать, так и открывать (или *распаковывать*) ZIP-файлы с помощью функций модуля `zipfile`. Предположим, имеется ZIP-файл *example.zip*, содержимое которого представлено на рис. 10.2.

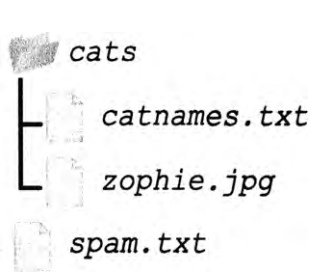


Рис. 10.2. Содержимое файла example.zip

Можете загрузить этот файл с сайта книги (см. введение) или просто использовать один из ZIP-файлов, которые уже имеются на вашем компьютере.

Чтение ZIP-файлов

Чтобы прочитать содержимое ZIP-файла, прежде всего необходимо создать объект `ZipFile` (обратите внимание на использование прописных букв `'Z'` и `'F'` в имени объекта). Объекты `ZipFile` концептуально напоминают объекты `File`, возвращаемые функцией `open()`, которая рассматривалась в предыдущей главе. Через такие объекты программа взаимодействует с файлами. Для создания объекта `ZipFile` следует вызвать функцию `zipfile.ZipFile()`, передав ей строку с именем *.zip*-файла. В данном случае `zipfile` — это имя модуля Python, а `ZipFile()` — имя функции.

Введите в интерактивной оболочке следующие инструкции.

```
>>> import zipfile, os
>>> from pathlib import Path
>>> p = Path.home()
>>> exampleZip = zipfile.ZipFile(p / 'example.zip')
>>> exampleZip.namelist()
['spam.txt', 'cats/', 'cats/catnames.txt', 'cats/zophie.jpg']
>>> spamInfo = exampleZip.getinfo('spam.txt')
>>> spamInfo.file_size
13908
>>> spamInfo.compress_size
3828
❶ >>> f'Сжатый файл в {round(spamInfo.file_size /
      spamInfo.compress_size, 2)} раза меньше!'
'Сжатый файл в 3.63 раза меньше!'
>>> exampleZip.close()
```

У объекта `ZipFile` есть метод `namelist()`, который возвращает список строк с именами всех файлов и папок, содержащихся в ZIP-архиве. Эти строки можно передать методу `getinfo()` объекта `ZipFile`, который вернет объект `ZipInfo`, содержащий информацию об указанном файле.

Объекты `ZipInfo` имеют такие атрибуты, как `file_size` и `compress_size`, определяющие соответственно размеры исходной и сжатой версии файла в байтах. Объект `ZipFile` представляет весь архивный файл, тогда как объект `ZipInfo` хранит полезную информацию об отдельном файле в сжатом архиве.

В инструкции ❶ вычисляется эффективность сжатия в файле *example.zip* путем деления размера исходного файла на размер сжатого файла.

Извлечение файлов из ZIP-архива

Метод `extractall()` объекта `ZipFile` извлекает все файлы и папки из ZIP-архива в текущий каталог.

```
>>> import zipfile, os
>>> from pathlib import Path
>>> p = Path.home()
>>> exampleZip = zipfile.ZipFile(p / 'example.zip')
❶ >>> exampleZip.extractall()
>>> exampleZip.close()
```

После выполнения этого кода содержимое файла *example.zip* будет извлечено в текущий каталог. В качестве необязательного параметра методу `extractall()` можно передать имя папки, что позволит извлекать файлы в папку, не являющуюся текущим каталогом. Если указанной папки не существует, то она будет создана. Например, если вызов ❶ заменить вызовом `exampleZip.extractall('C:\\delicious')`, то файлы будут извлечены из архива *example.zip* в новую папку *C:\delicious*.

Метод `extract()` объекта `ZipFile` извлекает одиночный файл из ZIP-архива. Продолжим выполнение примера в интерактивной оболочке.

```
>>> exampleZip.extract('spam.txt')
'C:\\spam.txt'
>>> exampleZip.extract('spam.txt', 'C:\\some\\new\\folders')
'C:\\some\\new\\folders\\spam.txt'
>>> exampleZip.close()
```

Передаваемая методу `extract()` строка должна соответствовать одной из строк в списке, возвращаемом методом `namelist()`. Методу `extract()` можно также передать необязательный второй параметр, позволяющий извлечь файлы в папку, не являющуюся текущим каталогом. Если этой папки не существует, Python создаст ее. Метод `extract()` возвращает абсолютный путь, куда был распакован данный файл.

Создание ZIP-архивов и добавление в них файлов

Чтобы создать собственный ZIP-файл, необходимо создать объект `ZipFile` в *режиме записи*, передав конструктору аргумент `'w'`. (Это аналогично открытию текстового файла в режиме записи путем передачи строки `'w'` методу `open()` в качестве второго аргумента.)

Когда вы передаете путь методу `write()` объекта `ZipFile`, Python сжимает файл, расположенный по указанному пути, и добавляет его в ZIP-файл. Первый аргумент метода `write()` — это строка с именем добавляемого файла. Второй аргумент задает *тип сжатия*, указывая, какой алгоритм следует применять для сжатия файлов. Это значение можно всегда устанавливать равным `zipfile.ZIP_DEFLATED` (данный алгоритм сжатия достаточно хорошо работает со всеми типами данных). Введите в интерактивной оболочке следующие инструкции.

```
>>> import zipfile
>>> newZip = zipfile.ZipFile('new.zip', 'w')
>>> newZip.write('spam.txt', compress_type=zipfile.ZIP_DEFLATED)
>>> newZip.close()
```

В данном случае создается новый ZIP-архив *new.zip*, содержащий файл *spam.txt*.

Имейте в виду, что, как и при обычной записи файлов, все существующее содержимое ZIP-файла в режиме записи удаляется. Если хотите добавить файлы в существующий ZIP-файл, передайте методу `zipfile.ZipFile()` в качестве второго параметра строку `'a'`, чтобы открыть ZIP-файл в режиме добавления.

Проект: переименование файлов с заменой американского формата дат европейским

Предположим, начальник перебросил вам по электронной почте тысячи файлов с просьбой переименовать их с заменой американского формата дат (ММ-ДД-ГГГГ) в их именах европейским (ДД-ММ-ГГГГ). Выполняя это поручение вручную, вы рискуете потратить на него целый день! Не лучше ли написать программу, которая сделает всю работу за вас?

Вот что должна делать эта программа:

1) искать в текущем каталоге все файлы, в имена которых содержится дата в американском формате:
2) при нахождении каждого такого файла переименовывать его, меняя местами день и месяц, чтобы привести стиль даты в соответствие с европейским форматом.

Это означает, что программа должна выполнить следующие операции:

1) создать регулярное выражение для распознавания образцов текста, соответствующих американскому формату даты:
2) вызвать функцию `os.listdir()` для создания списка всех файлов, содержащихся в текущем каталоге:
3) организовать просмотр всех имен файлов в цикле, определяя с помощью соответствующего регулярного выражения, содержат ли они даты:
4) если в имя файла входит дата, переименовать его с помощью функции `shutil.move()`.

Приступая к работе над данным проектом, откройте новое окно в файловом редакторе и сохраните его в файле *renameDates.py*.

Шаг 1. Создание регулярного выражения для поиска дат в американском формате

Вначале мы должны импортировать необходимые модули и создать регулярное выражение, способное распознавать даты в формате ММ-ДД-ГГГГ. Комментарии `'СДЕЛАТЬ'` будут напоминать о программном коде, который еще предстоит написать. Введите в файл следующий код.

```
  #! python3
  # renameDates.py - переименовывает файлы, имена которых
  # включают даты в американском формате (ММ-ДД-ГГГГ), приводя
  # их в соответствие с европейским форматом дат (ДД-ММ-ГГГГ)

❶ import shutil, os, re

  # Создание регулярного выражения, которому соответствуют имена
  # файлов, содержащие даты в американском формате
❷ datePattern = re.compile(r"""^(.*?)     # весь текст перед датой
      ((0|1)?\d)-                         # одна или две цифры месяца
      ((0|1|2|3)?\d)-                     # одна или две цифры числа
      ((19|20)\d\d)                       # четыре цифры года
      (.*?)$                              # весь текст после даты
❸     """, re.VERBOSE)

# СДЕЛАТЬ: организовать цикл по файлам в текущем каталоге

# СДЕЛАТЬ: пропустить файлы с именами, не содержащими дат

# СДЕЛАТЬ: получить отдельные фрагменты имен файлов
```

```
# СДЕЛАТЬ: сформировать имена, соответствующие европейскому
#          формату даты

# СДЕЛАТЬ: получить абсолютные пути к файлам

# СДЕЛАТЬ: переименовать файлы
```

Вы уже знаете о том, что для переименования файлов можно использовать функцию `shutil.move()`, аргументами которой служат исходное и новое имя файла. Поскольку эта функция содержится в модуле `shutil`, его необходимо импортировать ❶.

Прежде всего необходимо идентифицировать те файлы, которые подлежат переименованию. Переименовывать следует файлы, в именах которых содержатся даты, например *spam4-4-1984.txt* или *01-03-2014eggs.zip*, а такие файлы, как *littlebrother.epub*, не содержащие дат, можно игнорировать.

Для распознавания шаблона даты можно использовать регулярное выражение. Мы импортируем модуль `re` в начале файла и вызываем функцию `re.compile()` для создания объекта `Regex` ❷. Передача константы `re.VERBOSE` в качестве второго аргумента ❸ разрешает использовать пробелы и комментарии в строке регулярного выражения. Это позволяет создать удобное описание в коде.

Строка регулярного выражения начинается шаблоном `^(.*?)`, которому соответствует любой текст в имени файла, предшествующий дате. Группе `((0|1)?\d)` соответствует цифровое обозначение месяца. Первой цифрой может быть как `0`, так и `1`, так что регулярное выражение совпадет как с обозначением `12` в случае декабря, так и с обозначением `02` в случае февраля. Кроме того, эта цифра помечена как необязательная, поэтому, например, апрель будет распознан независимо от того, как он обозначен: `04` или `4`. Обозначениям дней соответствует группа `((0|1|2|3)?\d)`, в которой применяется аналогичная логика: `3`, `03` и `31` — каждый из этих вариантов является допустимым для обозначения дней. (Внимательный читатель заметит, что данному регулярному выражению будут соответствовать и некоторые недопустимые даты, такие как `4-31-2014`, `2-29-2013` или `0-15-2014`. При работе с датами следует учитывать множество подобных нюансов. Но для нашей простой программы такое регулярное выражение может считаться вполне приемлемым.)

Несмотря на то что `1885` — корректное обозначение года, мы ограничимся XX и XXI столетиями. Тем самым мы избежим случайного переименования тех файлов, в именах которых встречаются цифровые обозначения, лишь похожие на даты, такие как *10-10-1000.txt*.

Шаблону `(.*?)$` регулярного выражения соответствует любой текст, который следует за датой в имени файла.

Шаг 2. Идентификация фрагментов имен файлов, соответствующих датам

После этого программа должна просмотреть в цикле имена файлов из списка, возвращаемого функцией `os.listdir()`, и сравнить их с регулярным выражением. Любые файлы, имена которых не включают дату, должны игнорироваться. Для имен, содержащих дату, совпавший с шаблоном текст должен быть сохранен в нескольких переменных. Замените первые три комментария `'СДЕЛАТЬ'` в программе следующим кодом.

```
#! python3
# renameDates.py - переименовывает файлы, имена которых включают
# даты, указанные в американском формате (ММ-ДД-ГГГГ), приводя
# их в соответствие с европейским форматом дат (ДД-ММ-ГГГГ)

-- Опущено --

# Организация цикла по файлам в текущем каталоге
for amerFilename in os.listdir('.'):
    mo = datePattern.search(amerFilename)

    # Пропуск файлов с именами, не содержащими дат
❶   if mo == None:
❷       continue

❸   # Получение отдельных фрагментов имен файлов
    beforePart = mo.group(1)
    monthPart  = mo.group(2)
    dayPart    = mo.group(4)
    yearPart   = mo.group(6)
    afterPart  = mo.group(8)

-- Опущено --
```

Если метод `search()` возвращает значение `None` ❶, значит, строка имени файла, содержащаяся в переменной `amerFilename`, не соответствует регулярному выражению. Инструкция `continue` ❷ игнорирует оставшуюся часть цикла и осуществляет переход к следующему имени файла.

В противном случае строки, соответствующие отдельным группам в регулярном выражении, сохраняются в переменных `beforePart`, `monthPart`, `dayPart`, `yearPart` и `afterPart` ❸. Эти строки будут использованы на следующем шаге для формирования имен файлов с датами в европейском формате.

Чтобы разобраться в нумерации групп, попробуйте прочитать регулярное выражение с самого начала, прибавляя единицу всякий раз, когда встречается открывающая круглая скобка. Не думайте о коде, а просто

опишите каркас регулярного выражения. Это позволит визуализировать структуру групп.

```
datePattern = re.compile(r"""^(1)    # весь текст перед датой
    (2 (3) )-                        # одна или две цифры месяца
    (4 (5) )-                        # одна или две цифры числа
    (6 (7) )                         # четыре цифры года
    (8)$                             # весь текст после даты
    """, re.VERBOSE)
```

Здесь числа от 1 до 8 представляют собой номера групп в составленном нами регулярном выражении. Запись структуры регулярного выражения с использованием лишь круглых скобок и номеров групп поможет вам понять его смысл, прежде чем мы перейдем к написанию остальной части программы.

Шаг 3. Создание нового имени файла и переименование файлов

Последнее, что осталось сделать, — это конкатенировать строки, сохраненные в переменных на предыдущем шаге, для приведения даты к европейскому формату, в соответствии с которым число предшествует месяцу. Замените три оставшихся комментария `'СДЕЛАТЬ'` приведенным ниже кодом.

```
#! python3
# renameDates.py - переименовывает файлы, имена которых включают
# даты, указанные в американском формате (ММ-ДД-ГГГГ), приводя
# их в соответствие с европейским форматом дат (ДД-ММ-ГГГГ)

-- Опущено --

    # Создание имен, соответствующих европейскому
    # формату даты
❶   euroFilename = beforePart + dayPart + '-' + monthPart +
                   '-' + yearPart + afterPart

    # Получение абсолютных путей к файлам
    absWorkingDir = os.path.abspath('.')
    amerFilename = os.path.join(absWorkingDir, amerFilename)
    euroFilename = os.path.join(absWorkingDir, euroFilename)

    # Переименование файлов
❷   print(f'Заменяем "{amerFilename}" на "{euroFilename}"...')
❸   #shutil.move(amerFilename, euroFilename) # раскомментировать
                                              # после тестирования
```

Конкатенированная строка сохраняется в переменной `euroFilename` ❶. Исходное имя файла, сохраненное в переменной `amerFilename`, вместе

с переменной `euroFilename` передается функции `shutil.move()`, которая выполняет окончательное переименование файла ❸.

В данной версии программы вызов `shutil.move()` отключен с помощью комментария, а имена файлов, подлежащих переименованию, просто выводятся на экран ❷. Запуск программы в таком режиме позволяет дополнительно убедиться в корректности переименования файлов. После этого можно удалить символ комментария в строке с вызовом `shutil.move()` и вновь запустить программу для фактического переименования файлов.

Идеи для создания похожих программ

Необходимость в переименовании большого количества файлов может возникать по множеству других причин:

- добавление стандартного префикса в начало имен файлов (например, файл *eggs.txt* переименовывается в *spam_eggs.txt*);
- преобразование дат в именах файлов из европейского формата в американский;
- удаление ведущих нулей из имен таких файлов, как *spam0042.txt*.

Проект: создание резервной копии папки в виде ZIP-файла

Предположим, вы работаете над проектом, файлы которого хранятся в папке *C:\AlsPythonBook*. Вас волнует сохранность результатов вашей работы, и вам хотелось бы периодически создавать "моментальные снимки" проекта, сохраняя всю папку в одном ZIP-файле. При этом желательно хранить различные версии проекта в файлах с именами, содержащими номер резервной копии, который увеличивается всякий раз, когда создается новый ZIP-файл, например *AlsPythonBook_1.zip*, *AlsPythonBook_2.zip*, *AlsPythonBook_3.zip* и т.д. Это можно было бы делать и вручную, но такой подход чреват тем, что номера ZIP-файлов могут быть случайно перепутаны. Проще написать программу, которая будет выполнять всю рутинную работу вместо вас.

Приступая к работе над данным проектом, откройте новое окно в файловом редакторе и сохраните его в файле *backupToZip.py*.

Шаг 1. Определение имени, которое следует присвоить ZIP-файлу

Код программы будет помещен в функцию `backupToZip()`, что упростит его копирование и вставку в другие программы, нуждающиеся в подобной функциональности. В самом конце эта функция будет вызываться для создания резервной копии содержимого папки. Введите следующий код.

```
  #! python3
  # backupToZip.py - копирует папку вместе со всем ее содержимым
  # в ZIP-файл с инкрементируемым номером копии в имени файла

❶ import zipfile, os

  def backupToZip(folder):
      # Создание резервной копии всего содержимого
      # папки "folder" в виде ZIP-файла

      folder = os.path.abspath(folder)     # должен быть задан
                                           # абсолютный путь

      # Определяем, какое имя файла должна использовать функция,
      # исходя из имен уже существующих файлов
❷     number = 1
❸     while True:
          zipFilename = os.path.basename(folder) + '_' +
                        str(number) + '.zip'
          if not os.path.exists(zipFilename):
              break
          number = number + 1

❹     # СДЕЛАТЬ: создать ZIP-файл

      # СДЕЛАТЬ: обойти всю структуру папки и сжать файлы,
      #          содержащиеся в каждой подпапке
      print('Готово.')

  backupToZip('C:\\delicious')
```

Мы начинаем с элементарных вещей: добавляем строку сценария Python (с символами #!), описываем назначение программы и импортируем модули `zipfile` и `os` ❶.

Далее создается функция `backupToZip()`, имеющая всего один параметр: `folder`. Этот параметр представляет собой строку пути к папке, резервную копию которой необходимо создать. Сначала функция определяет имя, которое следует присвоить создаваемому ZIP-файлу, а затем создает сам файл, совершает обход содержимого папки `folder` и добавляет все ее подпапки и файлы в ZIP-файл. В исходный код включены соответствующие комментарии `'СДЕЛАТЬ'` как напоминание о том, что необходимо сделать в дальнейшем ❹.

В первой части функции, т.е. там, где ZIP-файлу присваивается имя, используется базовое имя папки. Если архивируется папка *C:\delicious*, то именем ZIP-файла будет *delicious_N.zip*, где *N* = 1 при первом запуске программы, *N* = 2 — при втором и т.д.

Можно определить, каким должно быть значение *N*, проверив, существуют ли уже файлы *delicious_1.zip*, *delicious_2.zip* и т.д. Значение *N*

хранится в переменной number ❷ и увеличивается в цикле, в котором с помощью функции os.path.exists() проверяется существование соответствующего файла ❸. Как только обнаружен несуществующий файл, цикл завершается, поскольку нам становится известно, какое имя следует присвоить новому ZIP-файлу.

Шаг 2. Создание нового ZIP-файла

Следующим шагом будет создание ZIP-файла. Дополните программу новым кодом, как показано ниже.

```
#! python3
# backupToZip.py - копирует папку вместе со всем ее содержимым
# в ZIP-файл с инкрементируемым номером копии в имени файла

-- Опущено --
    while True:
        zipFilename = os.path.basename(folder) + '_' +
                      str(number) + '.zip'
        if not os.path.exists(zipFilename):
            break
        number = number + 1

    # Создание ZIP-файла
    print(f'Создание файла {zipFilename}...')
❶   backupZip = zipfile.ZipFile(zipFilename, 'w')

    # СДЕЛАТЬ: обойти всю структуру папки и сжать файлы,
    #          содержащиеся в каждой подпапке
    print('Готово.')

backupToZip('C:\\delicious')
```

Теперь, когда имя нового ZIP-файла сохранено в переменной zipFilename, можно вызвать функцию zipfile.ZipFile() для создания ZIP-архива ❶. Не забудьте передать ей строку 'w' в качестве второго аргумента, чтобы открыть ZIP-файл в режиме записи.

Шаг 3. Обход дерева каталогов и добавление содержимого в ZIP-файл

Наконец, для обхода всех файлов и подпапок, содержащихся в данной папке, используется функция os.walk(). Дополните программу новым кодом, выделенным полужирным шрифтом.

```
#! python3
# backupToZip.py - копирует папку вместе со всем ее содержимым
# в ZIP-файл с инкрементируемым номером копии в имени файла

-- Опущено --

    # Обход всей структуры папки и сжатие файлов,
    # содержащихся в каждой подпапке
❶   for foldername, subfolders, filenames in os.walk(folder):
        print(f'Добавление файлов из папки {foldername}...')
        # Добавить в ZIP-архив текущую папку
❷       backupZip.write(foldername)

        # Добавить в ZIP-архив все файлы из данной папки
❸       for filename in filenames:
            newBase = os.path.basename(folder) + '_'
            if filename.startswith(newBase) and \
               filename.endswith('.zip'):
                continue    # не создавать резервные копии
                            # самих ZIP-файлов
            backupZip.write(os.path.join(foldername, filename))
    backupZip.close()
    print('Готово.')

backupToZip('C:\\delicious')
```

Функцию `os.walk()` можно использовать в цикле `for` ❶, и на каждой итерации цикла она будет возвращать имя текущей для данной итерации папки, а также имена всех содержащихся в ней подпапок и файлов.

В теле цикла `for` папка добавляется в ZIP-архив ❷. Обход всех файлов, имена которых содержатся в списке `filenames`, осуществляется во вложенном цикле `for` ❸. Каждый из файлов, за исключением ранее созданных ZIP-архивов, добавляется в ZIP-файл.

Запустив программу, вы получите примерно следующие результаты.

```
Создание файла delicious_1.zip...
Добавление файлов из папки C:\delicious...
Добавление файлов из папки C:\delicious\cats...
Добавление файлов из папки C:\delicious\waffles...
Добавление файлов из папки C:\delicious\walnut...
Добавление файлов из папки C:\delicious\walnut\waffles...
Готово.
```

Если запустить программу повторно, все файлы, содержащиеся в папке *C:\delicious*, будут заархивированы в ZIP-файле *delicious_2.zip* и т.д.

Идеи для создания похожих программ

Рассмотренная методика может применяться в целом ряде других программ. Например, можно написать программы для решения следующих задач:

- обход дерева каталогов и архивирование лишь файлов с конкретными расширениями, например *.txt* или *.py*;
- обход дерева каталогов и архивирование всех файлов, за исключением тех, которые имеют расширение *.txt* или *.py*;
- поиск в дереве каталогов папки, содержащей наибольшее количество файлов, или папки, занимающей наибольший объем дискового пространства.

Резюме

Даже если вы опытный пользователь, вы наверняка выполняете множество операций с файлами вручную с помощью мыши и клавиатуры. Современные файловые менеджеры упрощают работу с небольшим количеством файлов. Но иногда возникают задачи, на самостоятельное выполнение которых может уйти несколько часов.

Модули `os` и `shutil` содержат функции, позволяющие осуществлять копирование, перемещение, переименование и удаление файлов. Для удаления файлов имеет смысл пользоваться модулем `send2trash`, который позволяет перемещать файлы в корзину, а не удалять их безвозвратно. Кроме того, при написании программ, предназначенных для обработки файлов, желательно сначала закомментировать код, выполняющий опасную операцию (перемещение, переименование, удаление), добавив вместо него вызов функции `print()`. Это даст возможность убедиться в том, что программа работает с правильными файлами.

Операции подобного рода приходится выполнять не только над файлами, хранящимися в заданной папке, но и над файлами, хранящимися во вложенных папках, а также в подпапках второго и всех последующих уровней вложенности. Функция `os.walk()` может выполнить обход всей структуры папок вместо вас, позволив сконцентрироваться на выполнении конкретных операций с файлами.

С помощью модуля `zipfile` можно сжимать и извлекать файлы, хранящиеся в ZIP-архивах. В сочетании с функциями модулей `os` и `shutil` это позволяет упаковывать файлы, хранящиеся в любой папке на жестком диске. ZIP-файлы гораздо легче выгружать на сайты или пересылать по электронной почте, чем множество отдельных файлов.

В предыдущих главах вам предлагался готовый код, который было достаточно просто скопировать. При разработке собственных программ вы столкнетесь с тем, что они не всегда будут корректно работать с первого раза. В следующей главе мы рассмотрим модули Python, которые облегчают анализ и отладку программ, что поможет вам быстрее добиться их правильной работы.

Контрольные вопросы

1. Чем отличаются функции `shutil.copy()` и `shutil.copytree()`?
2. Какая функция применяется для переименования файлов?
3. Чем отличаются функции удаления файлов, предлагаемые модулями `send2trash` и `shutil`?
4. У объектов `ZipFile`, как и у объектов `File`, есть метод `close()`. Какой метод объектов `ZipFile` эквивалентен методу `open()` объектов `File`?

Учебные проекты

Чтобы закрепить полученные знания на практике, напишите программы для предложенных ниже задач.

Выборочное копирование

Напишите программу, выполняющую обход дерева каталогов с целью отбора файлов с заданным расширением (например, *.pdf* или *.jpg*). Скопируйте эти файлы из их текущего расположения в новую папку.

Удаление ненужных файлов

Нередко возникают ситуации, когда несколько ненужных файлов или папок огромного размера занимают существенную часть дискового пространства. Для освобождения места на жестком диске наибольший эффект будет достигнут от первоочередного удаления самых крупных из ненужных файлов. Но сначала их необходимо найти.

Напишите программу, которая обходит дерево папок, выполняя поиск самых больших папок и файлов, – скажем, таких, размеры которых превышают 100 Мбайт. (Вспомните, что размер файла можно определить с помощью функции `os.path.getsize()`.) Выведите абсолютные пути к этим файлам на экран.

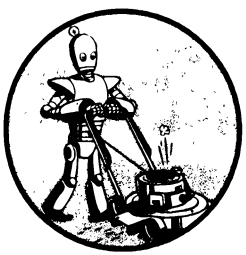
Заполнение пропусков в нумерации файлов

Напишите программу, которая ищет в папке все файлы с именами, содержащими заданный префикс, такими как *spam001.txt*, *spam002.txt* и т.д., и обнаруживает любые пропуски в нумерации файлов (например, есть файлы *spam001.txt* и *spam003.txt*, но отсутствует файл *spam002.txt*). Программа должна изменять имена файлов с бо́льшими номерами таким образом, чтобы ликвидировать имеющиеся пропуски.

В качестве дополнительного задания напишите другую программу, способную создавать пропуски в нумерации файлов.

11

ОТЛАДКА



Тех знаний, которые вы к данному моменту успели приобрести, вполне достаточно для того, чтобы приступить к написанию более сложных программ. Но вы должны быть готовы к тому, что в программном коде будут встречаться трудно обнаруживаемые ошибки. В этой главе обсуждаются инструменты и методики отладки программ, позволяющие быстро находить и устранять всевозможные “баги”.

Перефразируя расхожую шутку программистов, можно сказать так: "Программирование на 90 процентов состоит из написания кода. Оставшиеся 90 процентов приходятся на отладку".

Компьютер сделает лишь то, что вы ему прикажете. Он не способен читать мысли и исполнять ваши *намерения*. Даже профессиональные программисты иногда допускают серьезные ошибки, поэтому не стоит падать духом, если в программы обнаруживаются неполадки.

К счастью, существует целых ряд инструментов и методик, с помощью которых можно точно определить, что именно делает код и в каком месте программы произошел сбой. В первую очередь мы рассмотрим протоколирование операций и утверждения — два средства, облегчающие раннее обнаружение ошибок. По большому счету, чем раньше обнаружена ошибка, тем проще ее исправить.

Кроме того, вы познакомитесь с *отладчиком*. Это средство редактора Mu, обеспечивающее пошаговое выполнение программы, по одной инструкции за раз, что дает возможность отслеживать значения переменных и контролировать их изменение в процессе работы программы. Программа при этом выполняется медленнее, чем обычно, но зато вы получаете возможность наблюдать за фактическими значениями переменных, а не строить догадки, анализируя исходный код.

Генерирование исключений

Python генерирует исключение всякий раз, когда делается попытка выполнить недопустимый код. В главе 3 вы узнали о том, как обрабатывать исключения Python с помощью инструкций `try` и `except`, позволяющих избежать преждевременного прекращения работы программы при возникновении проблемных ситуаций, которые вы предвидели. Но в программе можно генерировать и пользовательские исключения. Появление исключения равносильно следующему приказу: "Прекрати выполнять код данной функции и перейди к выполнению инструкции `except`".

Исключения генерируются с помощью инструкции `raise`, которая состоит из следующих элементов:

- ключевое слово `raise`;
- вызов функции `Exception()`;
- строка сообщения об ошибке, передаваемая функции `Exception()`.

Введите в интерактивной оболочке следующую инструкцию.

```
>>> raise Exception('Это сообщение об ошибке.')
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
```

```
    raise Exception('Это сообщение об ошибке.')
Exception: Это сообщение об ошибке.
```

В отсутствие инструкций `try` и `except`, обрамляющих инструкцию `raise`, которая генерирует исключение, выполнение программы аварийно завершается с выводом соответствующего сообщения.

Часто исключение обрабатывается не в функции, а в коде, в котором она была вызвана. Поэтому нередко бывает так, что инструкция `raise` находится в теле функции, а связанные с ней инструкции `try` и `except` — в вызывающем коде. Например, откройте новое окно в файловом редакторе, введите в него следующий код и сохраните программу в файле *boxPrint.py*.

```
def boxPrint(symbol, width, height):
    if len(symbol) != 1:
❶       raise Exception('Переменная symbol должна содержать \
                        один символ.')
    if width <= 2:
❷       raise Exception('Значение width должно превышать 2.')
    if height <= 2:
❸       raise Exception('Значение height должно превышать 2.')

    print(symbol * width)
    for i in range(height - 2):
        print(symbol + (' ' * (width - 2)) + symbol)
    print(symbol * width)

for sym, w, h in (('*', 4, 4), ('O', 20, 5), ('x', 1, 3),
                  ('ZZ',3, 3)):
    try:
        boxPrint(sym, w, h)
❹   except Exception as err:
❺       print('Возникло исключение: ' + str(err))
```

Выполнение авторского варианта этой программы можно просмотреть на сайте `https://autbor.com/boxprint`. Мы определяем функцию `boxPrint()`, которая имеет параметр `symbol`, задающий символ, а также параметры `width` (ширина) и `height` (высота). Функция использует заданный символ для создания небольшого изображения, ширина и высота которого определяются двумя другими параметрами. Полученная прямоугольная фигура выводится на экран.

Предположим, мы хотим, чтобы параметр `symbol` мог быть только одиночным символом, а значения параметров `width` и `height` превышали 2. Для этого мы добавляем инструкции `if`, которые генерируют исключения, если эти требования не соблюдаются. Впоследствии, когда функция `boxPrint()` вызывается с различными аргументами, в блоке `try/except` обрабатываются недопустимые аргументы.

В инструкции except перехватывается исключение Exception ❹. Если в функции boxPrint() генерируется данное исключение ❶❷❸, инструкция except сохраняет его в переменной err. Далее объект исключения можно преобразовать в строку, передав его функции str() для вывода сообщения об ошибке ❺. Результаты выполнения программы *boxPrint.py* будут выглядеть так.

```
****
*  *
*  *
****
OOOOOOOOOOOOOOOOOOOO
O                  O
O                  O
O                  O
OOOOOOOOOOOOOOOOOOOO
Возникло исключение: Значение width должно превышать 2.
Возникло исключение: Переменная symbol должна содержать один символ.
```

Инструкции try и except позволяют корректно обрабатывать ошибки, препятствуя аварийному завершению программы.

Сохранение обратной трассировки стека вызовов в виде строки

Когда возникает ошибка, информация о ней подается в виде так называемой *обратной трассировки стека вызовов*. Эта информация включает текст сообщения об ошибке, номер строки в исходном коде, в которой возникла ошибка, и последовательность вызовов функций, которая привела к ошибке. Такая последовательность и называется *стеком вызовов*.

Откройте в редакторе файлов новую вкладку, введите приведенный ниже код и сохраните его в файле *errorExample.py*.

```
def spam():
    bacon()

def bacon():
    raise Exception('Это сообщение об ошибке.')

spam()
```

Запустив программу *errorExample.py*, вы получите следующее.

```
Traceback (most recent call last):
  File "errorExample.py", line 7, in <module>
    spam()
```

```
  File "errorExample.py", line 2, in spam
    bacon()
  File "errorExample.py", line 5, in bacon
    raise Exception('Это сообщение об ошибке.')
Exception: Это сообщение об ошибке.
```

На основании информации о стеке вызовов можно утверждать, что ошибка возникла в строке 5, т.е. в коде функции `bacon()`. Эта функция была вызвана в строке 2, т.е. в коде функции `spam()`, которая, в свою очередь, была вызвана в строке 7. В тех случаях, когда одна и та же функция может вызываться в нескольких местах программы, стек вызовов позволяет определить, какой именно вызов приводит к ошибке.

Python отображает стек вызовов всякий раз, когда сгенерированное исключение остается необработанным. Но его можно получить и в виде строки, вызвав функцию `traceback.format_exc()`. Эта функция пригодится в тех случаях, когда вы хотите обработать исключение и заодно получить информацию о стеке вызовов. Прежде чем вызывать данную функцию, следует импортировать модуль `traceback`.

Например, вместо того чтобы просто позволить программе аварийно завершиться сразу же после возникновения исключения, можно записать информацию о стеке вызовов в текстовый файл и продолжить выполнение программы. Впоследствии, когда вы приступите к отладке, вы сможете просмотреть содержимое текстового файла. Введите в интерактивной оболочке следующий код.

```
>>> import traceback
>>> try:
...     raise Exception('Это сообщение об ошибке.')
    except:
...     errorFile = open('errorInfo.txt', 'w')
...     errorFile.write(traceback.format_exc())
...     errorFile.close()
...     print('Стек вызовов записан в файл errorInfo.txt.')

114
Стек вызовов записан в файл errorInfo.txt.
```

Число 114 — это значение, возвращаемое методом `write()`, которое равно количеству символов, записанных в файл (включая символы новой строки).

Записанная в файл *errorInfo.txt* информация должна выглядеть так.

```
Traceback (most recent call last):
  File "<pyshell#11>", line 2, in <module>
Exception: Это сообщение об ошибке.
```

Утверждения

Утверждение — это профилактический механизм, позволяющий убедиться в правильности выполнения программы. Он основан на выполнении проверок с помощью инструкций `assert`. Если условие проверки не выполняется, то генерируется исключение `AssertionError`. Инструкция `assert` содержит следующие элементы (последние два из них необязательны):

- ключевое слово `assert`;
- условие (т.е. выражение, равное `True` или `False`);
- запятая;
- строка, которая отображается в том случае, если условие оказывается ложным.

Данную инструкцию можно трактовать так: "Я утверждаю, что условие истинно, а если нет, значит, в программе есть ошибка, поэтому ее следует немедленно остановить". Введите в интерактивной оболочке следующий код.

```
>>> ages = [26, 57, 92, 54, 22, 15, 17, 80, 47, 73]
>>> ages.sort()
>>> ages
[15, 17, 22, 26, 47, 54, 57, 73, 80, 92]
>>> assert ages[0] <= ages[-1] # проверяем, что первый возраст
                               # не превышает последний
```

В данном случае инструкция `assert` утверждает, что первый элемент списка `ages` должен быть меньше или равен последнему элементу. Это проверка безошибочности кода: если функция `sort()` не содержит ошибок и выполнила свою работу, то утверждение будет верным.

Поскольку выражение `ages[0] <= ages[-1]` истинно, инструкция `assert` в данном случае ничего не делает.

Но давайте представим, что в коде содержится ошибка. Допустим, мы случайно вызвали метод `reverse()` вместо `sort()`. Если ввести следующий код в интерактивную оболочку, то инструкция `assert` сгенерирует исключение `AssertionError`.

```
>>> ages = [26, 57, 92, 54, 22, 15, 17, 80, 47, 73]
>>> ages.reverse()
>>> ages
[73, 47, 80, 17, 15, 22, 54, 92, 57, 26]
>>> assert ages[0] <= ages[-1] # проверяем, что первый возраст
                               # не превышает последний
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError
```

В отличие от исключений, программа *не должна* обрабатывать инструкции `assert` в блоке `try/except`: в случае сбоя инструкции `assert` программа *должна аварийно завершить работу*. В результате такого "быстрого сбоя" вы сокращаете время между возникновением ошибки и точкой, где она была обнаружена. Это уменьшит объем кода, который нужно будет проверить, прежде чем найти причину ошибки.

Утверждения предназначены для выявления ошибок программиста, а не пользователя. Утверждения должны вызывать сбои только тогда, когда программа находится в стадии разработки, — пользователь никогда не должен видеть ошибки, связанные с утверждениями, в готовой программе. Для ошибок, с которыми программа может столкнуться в процессе выполнения (например, файл не найден или пользователь ввел недопустимые данные), необходимо генерировать исключения, вместо того чтобы прибегать к инструкции `assert`. Она не должна заменять механизм исключений, поскольку пользователи могут попросту отключить утверждения. Если запустить сценарий Python с помощью команды `python -O myscript.py`, а не `python myscript.py`, то Python будет пропускать инструкции `assert`. Зачастую, когда программа разрабатывается для выполнения в производственной среде, утверждения отключают, чтобы достичь максимальной производительности (впрочем, даже в этом случае их могут оставлять включенными).

Утверждения не должны рассматриваться как замена исчерпывающему тестированию. Например, если в предыдущем примере список `ages` будет содержать значения `[10, 3, 2, 1, 20]`, то утверждение `assert ages[0] <= ages[-1]` не выявит тот факт, что список не отсортирован, ведь первый элемент в нем меньше второго, а это единственное, что проверялось утверждением.

Использование утверждений в программе, имитирующей работу светофора

Предположим, требуется написать программу, имитирующую работу светофора. В качестве структуры данных, представляющей сигналы светофора на перекрестке, выбираем словарь с ключами `'ns'` и `'ew'` для секций, ориентированных вдоль направлений "север — юг" и "восток — запад" соответственно. Каждому из этих ключей могут соответствовать значения `'green'`, `'yellow'` и `'red'` (зеленый, желтый, красный). Соответствующий код будет выглядеть так.

```
market_2nd = {'ns': 'green', 'ew': 'red'}
mission_16th = {'ns': 'red', 'ew': 'green'}
```

Эти две переменные описывают светофоры на пересечении улиц "Market Street – 2nd Street" и "Mission Street – 16th Street". Приступая к проекту, необходимо написать функцию `switchLights()`, которая будет переключать сигналы светофора, получая указанный словарь в качестве аргумента.

Поначалу кажется, что функция `switchLights()` всего лишь должна последовательно переключать сигналы: после `'green'` должен быть `'yellow'`, за ним `'red'`, снова `'green'` и т.д. Соответствующий код может выглядеть так.

```
def switchLights(stoplight):
    for key in stoplight.keys():
        if stoplight[key] == 'green':
            stoplight[key] = 'yellow'
        elif stoplight[key] == 'yellow':
            stoplight[key] = 'red'
        elif stoplight[key] == 'red':
            stoplight[key] = 'green'

switchLights(market_2nd)
```

Возможно, вы уже поняли, в чем суть проблемы, но все же представим, что вы написали остальную часть симулятора светофора, насчитывающую тысячи строк, так ничего и не заметив. Когда вы запустите программу, она не потерпит крах, чего нельзя будет сказать о ваших виртуальных автомобилях!

Поскольку программа уже написана, вы совершенно не представляете, где может скрываться ошибка. Возможно, она затаилась в коде, имитирующем движение машин, или, быть может, в коде, имитирующем действия виртуальных водителей. Прежде чем вы догадаетесь, что следы ошибки ведут к самой функции `switchLights()`, может пройти немало времени.

Если бы в процессе написания функции `switchLights()` вы добавили утверждение, проверяющее, что в *любой момент времени по крайней мере один сигнал светофора красный*, то поместили бы в конце функции следующий код.

```
assert 'red' in stoplight.values(), 'Ни один из сигналов \
    не является красным! ' + str(stoplight)
```

Программа, содержащая это утверждение, завершится аварийно с выдачей следующего сообщения об ошибке.

```
  Traceback (most recent call last):
    File "carSim.py", line 14, in <module>
      switchLights(market_2nd)
    File "carSim.py", line 13, in switchLights
```

```
    assert 'red' in stoplight.values(), 'Ни один из сигналов не
    является красным! ' + str(stoplight)
❶ AssertionError: Ни один из сигналов не является красным!
    {'ns': 'yellow', 'ew': 'green'}
```

Ключевая строка здесь — `AssertionError` ❶. Несмотря то что решение допустить аварийное завершение программы нельзя назвать идеальным, оно позволило немедленно узнать о нарушении проверяемого условия: ни в одном из направлений не горит красный сигнал, а значит, движение разрешено одновременно в обе стороны. Оперативно обнаружив такую ошибку в программе, вы сэкономите усилия по ее отладке в будущем.

Протоколирование

Если вы когда-либо применяли в программе инструкцию `print()` для вывода значений интересующих вас переменных, то считайте, что с одной из форм *протоколирования* вы уже знакомы. Средства протоколирования позволяют получать информацию о том, что именно и в какой последовательности происходит в программе. Модуль `logging` в Python упрощает создание журнала подготовленных вами сообщений. Такие сообщения включают описание того, когда именно программа достигла вызова функции протоколирования, а также список значений указанных вами переменных в данный момент времени. Отсутствие сообщения в журнале свидетельствует о том, что данная часть кода была пропущена и не выполнялась.

Использование модуля `logging`

Чтобы включить вывод журнальных сообщений в процессе выполнения программы, скопируйте приведенный ниже код в начало программы (после "магической" строки сценария, начинающейся символами `#!`).

```
import logging
logging.basicConfig(level=logging.DEBUG, format=' %(asctime)s -
%(levelname)s - %(message)s')
```

Детали того, как все это работает, для вас несущественны, но вам будет полезно знать, что каждый раз, когда Python протоколирует событие, он создает объект `LogRecord`, в котором хранится информация о данном событии. Функция `basicConfig()` модуля `logging` позволяет конкретизировать, какую именно информацию об объекте `LogRecord` вы хотите видеть и в каком формате.

Предположим, вы написали функцию для вычисления *факториала* числа. Например, факториал числа 4 равен произведению 1·2·3·4, т.е. 24, а факториал числа 7 равен 1·2·3·4·5·6·7, или 5040. Откройте в файловом редакторе

новое окно и введите в нем приведенный ниже код. В программе содержится ошибка, но мы дополнительно предусмотрели несколько журнальных сообщений, которые помогут выяснить, где именно происходит сбой. Сохраните программу в файле *factorialLog.py*.

```
import logging
logging.basicConfig(level=logging.DEBUG, format=' %(asctime)s -
%(levelname)s - %(message)s')
logging.debug('Начало программы')

def factorial(n):
    logging.debug('Начало factorial(%s%%)' % (n))
    total = 1
    for i in range(n + 1):
        total *= i
        logging.debug('i = ' + str(i) + ', total = ' + str(total))
    logging.debug('Конец factorial(%s%%)' % (n))
    return total

print(factorial(5))
logging.debug('Конец программы')
```

В программе для вывода журнальной информации используется функция `logging.debug()`. Она вызывает функцию `basicConfig()`, которая выводит заданное сообщение. Формат вывода задан в функции `basicConfig()` и включает текст сообщения, переданный функции `debug()`. Вызов `print(factorial(5))` является частью исходной программы, поэтому результат будет отображаться даже тогда, когда средства протоколирования отключены.

Результаты работы программы будут примерно такими.

```
2019-05-23 16:20:12,664 - DEBUG - Начало программы
2019-05-23 16:20:12,664 - DEBUG - Начало factorial(5)
2019-05-23 16:20:12,665 - DEBUG - i = 0, total = 0
2019-05-23 16:20:12,668 - DEBUG - i = 1, total = 0
2019-05-23 16:20:12,670 - DEBUG - i = 2, total = 0
2019-05-23 16:20:12,673 - DEBUG - i = 3, total = 0
2019-05-23 16:20:12,675 - DEBUG - i = 4, total = 0
2019-05-23 16:20:12,678 - DEBUG - i = 5, total = 0
2019-05-23 16:20:12,680 - DEBUG - Конец factorial(5)
0
2019-05-23 16:20:12,684 - DEBUG - Конец программы
```

Функция `factorial()` возвращает значение `0` в качестве факториала числа `5`, что неверно. В цикле `for` значение переменной `total` должно умножаться на числа от `1` до `5`. Однако сообщения, отображаемые функцией `logging.debug()`, показывают, что начальное значение переменной `i` — `0`,

а не 1. Поскольку результатом умножения любого числа на 0 всегда будет 0, в последующих итерациях значение переменной total уже не меняется. Журнальные сообщения становятся теми самыми "хлебными крошками", идя по которым вам будет легче выяснить, в каком месте программы возникла ошибка.

Замените строку for i in range(n + 1): строкой for i in range(**1**, n + 1): и запустите программу повторно. Результат должен выглядеть так.

```
2019-05-23 17:13:40,650 - DEBUG - Начало программы
2019-05-23 17:13:40,651 - DEBUG - Начало factorial(5)
2019-05-23 17:13:40,651 - DEBUG - i = 1, total = 1
2019-05-23 17:13:40,654 - DEBUG - i = 2, total = 2
2019-05-23 17:13:40,656 - DEBUG - i = 3, total = 6
2019-05-23 17:13:40,659 - DEBUG - i = 4, total = 24
2019-05-23 17:13:40,661 - DEBUG - i = 5, total = 120
2019-05-23 17:13:40,661 - DEBUG - Конец factorial(5)
120
2019-05-23 17:13:40,666 - DEBUG - Конец программы
```

Теперь функция factorial(5) возвращает корректное значение 120. Журнальные сообщения позволили нам увидеть, что происходит в цикле, и найти причину ошибки.

Как видите, функция logging.debug() выводит не только переданную ей строку, но также метки времени и слово 'DEBUG'.

Не выполняйте отладку с помощью функции print()

Для кого-то импорт модуля logging и громоздкий вызов функции logging.basicConfig может показаться не самой удачной стратегией. Почему бы не использовать вместо этого функцию print()? Не стоит поддаваться такому искушению! Завершив отладку, вы потратите массу времени на удаление из кода вызовов print() для каждого сообщения. Более того, не исключено, что вы случайно удалите полезные вызовы print(), никак не связанные с журнальными сообщениями. Механизм протоколирования удобен тем, что в программе можно предусмотреть столько сообщений, сколько вам нужно, и впоследствии вы сможете в любой момент отключить их, добавив единственный вызов logging.disable(logging.CRITICAL). В отличие от функции print(), модуль logging упрощает переключение между режимами отображения и сокрытия сообщений.

Журнальные сообщения предназначены для программистов, а не для пользователей. Пользователя не интересует содержимое словаря, за которым вы хотите наблюдать в процессе отладки. Сообщения, адресованные пользователю, такие как "Файл не найден" или "Введите число", следует

выводить с помощью функции `print()`. Пользователя нельзя лишать этой информации после отключения журнальных сообщений.

Уровень протоколирования

Уровни протоколирования позволяют классифицировать журнальные сообщения по степени важности. Всего существует пять уровней протоколирования, перечисленных в табл. 11.1 в порядке возрастания важности. Сообщения каждого уровня выводятся с использованием разных функций.

Таблица 11.1. Уровни протоколирования в Python

Уровень	Функция протоколирования	Описание
DEBUG	`logging.debug()`	Самый низкий уровень. Предназначен для вывода малозначимой информации. Такие сообщения представляют интерес только при диагностике проблем
INFO	`logging.info()`	Предназначен для записи информации об обычных событиях, происходящих в программе, или для подтверждения нормального хода работы программы
WARNING	`logging.warning()`	Предназначен для индикации потенциально опасных ситуаций, которые не препятствуют работе программы, но могут привести к этому в будущем
ERROR	`logging.error()`	Предназначен для записи информации об ошибке, которая помешала программе выполнить требуемые действия
CRITICAL	`logging.critical()`	Наивысший уровень. Предназначен для индикации фатальных ошибок, которые привели или могут привести к аварийному завершению программы

Сообщения следует передавать функциям в строковом виде. Сами по себе уровни протоколирования — это не более чем рекомендации. В конечном счете только вы решаете, к какой категории следует отнести то или иное сообщение. Введите в интерактивной оболочке следующие инструкции.

```
>>> import logging
>>> logging.basicConfig(level=logging.DEBUG, format=' %(asctime)s -
%(levelname)s - %(message)s')
>>> logging.debug('Отладочная информация.')
2015-05-18 19:04:26,901 - DEBUG - Отладочная информация.
>>> logging.info('Работает модуль logging.')
2015-05-18 19:04:35,569 - INFO - Работает модуль logging.
>>> logging.warning('Риск получения сообщения об ошибке.')
2015-05-18 19:04:56,843 - WARNING - Риск получения сообщения
об ошибке.
>>> logging.error('Произошла ошибка.')
2015-05-18 19:05:07,737 - ERROR - Произошла ошибка.
>>> logging.critical('Программа не может выполняться.')
2015-05-18 19:05:45,794 - CRITICAL - Программа не может выполняться.
```

Преимущество уровней протоколирования в том, что у вас есть возможность задать граничный приоритет сообщений, подлежащих отслеживанию. Если передать функции `basicConfig()` значение `logging.DEBUG` в качестве аргумента `level`, то будут отображаться сообщения всех уровней (поскольку `DEBUG` — самый низкий уровень). На последующих этапах разработки программы вас уже будет интересовать только информация об ошибках. В таком случае вы сможете передать функции `basicConfig()` аргумент `logging.ERROR`. В результате будут отображаться лишь сообщения категорий `ERROR` и `CRITICAL`, а сообщения категорий `DEBUG`, `INFO` и `WARNING` будут игнорироваться.

Отключение протоколирования

После завершения отладки программы вам уже будут не нужны журнальные сообщения, захламляющие экран. Функция `logging.disable()` позволяет отключить их, не внося изменения в программный код. Вы просто сообщаете ей требуемый уровень протоколирования, и она подавляет вывод сообщений, относящихся к этому и более низким уровням. Таким образом, чтобы полностью отключить режим протоколирования, достаточно добавить в программу вызов `logging.disable(logging.CRITICAL)`. Например, введите в интерактивной оболочке следующие инструкции.

```
>>> import logging
>>> logging.basicConfig(level=logging.INFO, format=' %(asctime)s -
%(levelname)s - %(message)s')
>>> logging.critical('Критическая ошибка!')
2015-05-22 11:10:48,054 - CRITICAL - Критическая ошибка!
>>> logging.disable(logging.CRITICAL)
>>> logging.critical('Критическая ошибка!')
>>> logging.error('Ошибка! Ошибка!')
```

Поскольку функция `logging.disable()` отключает все последующие инструкции протоколирования, ее нужно вставить после строки `import logging`. Благодаря этому вы сможете быстро находить ее, чтобы при необходимости закомментировать или раскомментировать включения или отключения режима протоколирования.

Запись сообщений в файл журнала

Вместо того чтобы отображать журнальные сообщения на экране, их можно записывать в текстовый файл. Для этого следует воспользоваться функцией `logging.basicConfig()`, которая поддерживает именованный аргумент `filename`.

```
import logging
logging.basicConfig(filename='myProgramLog.txt', level=logging.DEBUG, \
format='%(asctime)s - %(levelname)s - %(message)s')
```

В данном случае сообщения сохраняются в файле *myProgramLog.txt*. Какими бы полезными ни были журнальные сообщения, они могут затруднять просмотр результатов работы программы. Запись сообщений в файл журнала позволяет не захламлять экран, а ознакомиться с текстом сообщений можно будет уже после выполнения программы. Просмотреть файл журнала можно в любом текстовом редакторе, таком как Блокнот или TextEdit.

Отладчик Mu

Отладчик – это средство Mu, IDLE или другого редактора, которое позволяет управлять работой программы в пошаговом режиме. Отладчик выполняет одну строку кода и переходит в состояние ожидания, пока не получит команду продолжить выполнение. Запустив программу в отладчике, вы сможете наблюдать за значениями переменных в любом ее месте. Отладчик незаменим для выявления ошибок в программе.

Чтобы запустить программу в отладчике Mu, щелкните на кнопке Debug (Отладка), находящейся в верхнем ряду кнопок, справа от кнопки Run (Выполнить). Наряду с обычной панелью вывода внизу, вдоль правой границы окна появится панель Debug Inspector (Инспектор отладки), в которой отображаются текущие значения переменных. Как показано на рис. 11.1, отладчик приостановил программу перед выполнением первой строки кода. В файловом редакторе эта строка подсвечена.

В режиме отладки на панели инструментов появятся следующие новые кнопки: Continue (Продолжить), Step Over (Шаг с обходом), Step In (Шаг с заходом) и Step Out (Шаг с выходом). Также доступна обычная кнопка Stop (Остановить).

Кнопка Continue

При щелчке на кнопке Continue (Продолжить) программа будет выполняться до полного завершения или до тех пор, пока не встретится *точка останова* (они будут описаны далее.) Если вы завершили отладку и хотите, чтобы программа продолжила выполняться в обычном режиме, щелкните на кнопке Continue.

Кнопка Step In

Щелчок на кнопке Step In (Шаг с заходом) приводит к выполнению следующей строки кода и приостановке программы. Если следующей строкой

кода окажется вызов функции, то отладчик выполнит "шаг с заходом", т.е. перейдет в функцию и остановится на первой строке ее кода.

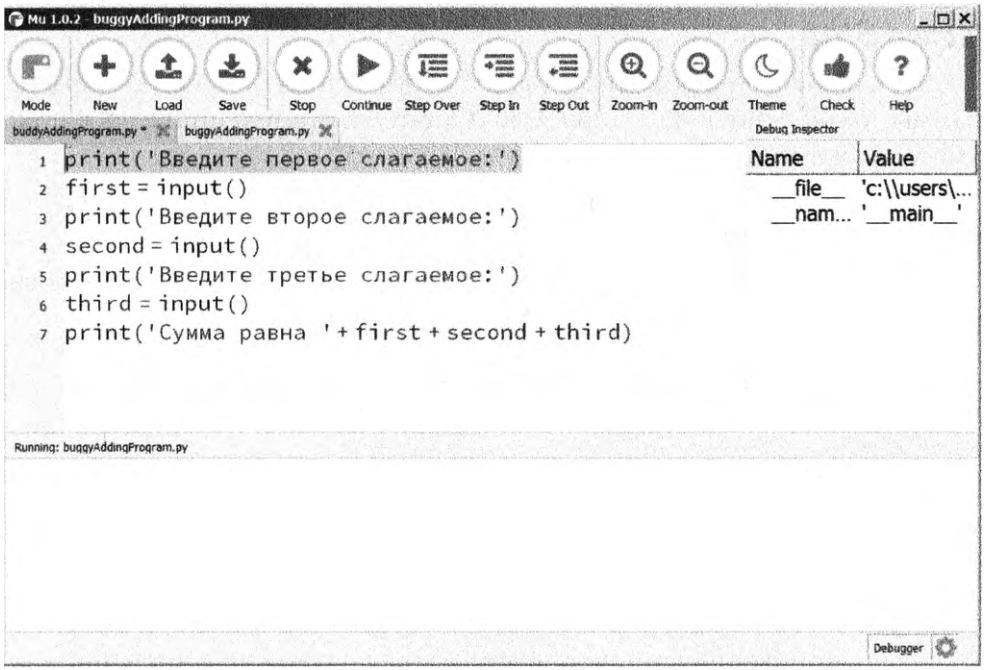


Рис. 11.1. Выполнение программы в отладчике Mu

Кнопка Step Over

Щелчок на кнопке Step Over (Шаг с обходом) приводит к выполнению следующей строки кода, как и после щелчка на кнопке Step In. Но если следующей строкой кода окажется вызов функции, то отладчик сделает "шаг с обходом", т.е. выполнит сразу весь код функции и остановится сразу же после ее завершения. Например, если в следующей строке кода содержится вызов функции `spam()`, то нас интересует лишь вывод переданной ей строки на экран, а не сам код функции. Поэтому обычно кнопкой Step Over пользуются чаще, чем кнопкой Step In.

Кнопка Step Out

Щелчок на кнопке Step Out (Шаг с выходом) приводит к выполнению программы в обычном режиме до тех пор, пока не завершится текущая функция. Если перед этим вы выполнили шаг с заходом в функцию с помощью кнопки Step In, а теперь просто хотите продолжить выполнение

инструкций вплоть до возврата из функции, то щелкните на кнопке Step Out, чтобы выйти из текущей функции.

Кнопка Stop

Если вы хотите прекратить отладку, не выполняя остальную часть программы, то щелкните на кнопке Stop (Остановить), что приведет к немедленному завершению программы.

Отладка программы сложения чисел

Откройте в файловом редакторе новое окно и введите в нем следующий код.

```
print('Введите первое слагаемое:')
first = input()
print('Введите второе слагаемое:')
second = input()
print('Введите третье слагаемое:')
third = input()
print('Сумма равна ' + first + second + third)
```

Сохраните текст программы в файле *buggyAddingProgram.py* и выполните ее сначала при отключенном отладчике. Вы должны получить примерно такие результаты.

```
Введите первое слагаемое:
5
Введите второе слагаемое:
3
Введите третье слагаемое::
42
Сумма равна 5342
```

Программа не завершилась аварийно, однако полученная сумма явно неправильна. Включите режим отладки и вновь запустите программу, на этот раз под управлением отладчика.

После щелчка на кнопке Debug программа останавливается в строке 1 (это строка кода, которую он собирается выполнить). Окно редактора Mu должно выглядеть так, как было показано на рис. 11.1.

Щелкните на кнопке Step Over один раз, чтобы выполнить первый вызов функции `print()`. Мы используем кнопку Step Over, а не Step In, поскольку не хотим входить в код функции `print()`. (Впрочем, отладчик Mu воспрепятствует переходу к встроенным функциям Python.) Отладчик переходит к строке 2 и выделяет ее в редакторе файлов, как показано на рис. 11.2. Тем

самым показывается, где в данный момент находится точка выполнения программы.

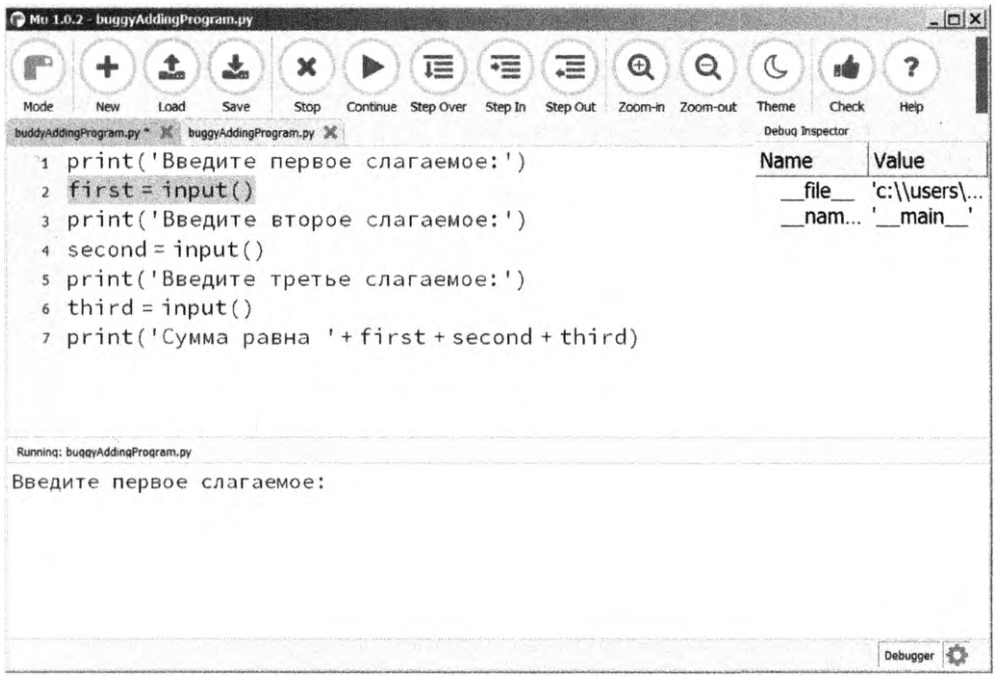


Рис. 11.2. Так выглядит окно редактора Mu после щелчка на кнопке Step Over

Щелкните на кнопке Step Over еще раз, чтобы выполнить вызов функции `input()`. Подсветка исчезнет, пока отладчик Mu ждет, чтобы вы что-то ввели для вызова функции `input()` на панели вывода. Введите **5** и нажмите клавишу <Enter>. Снова отобразится подсветка.

Продолжайте щелкать на кнопке Step Over и введите **3** и **42** в качестве следующих двух чисел. Когда отладчик достигает строки 7 (последний вызов функции `print()` в программе), окно редактора Mu должно выглядеть так, как показано на рис. 11.3.

На панели диспетчера Debug Inspector видно, что переменным `first`, `second` и `third` присвоены строковые значения `'5'`, `'3'` и `'42'`, а не целочисленные значения `5`, `3` и `42`. По достижении последней строки кода вместо сложения трех чисел выполняется конкатенация строк, что приводит к ошибочному результату.

Пошаговое выполнение программы с помощью отладчика чрезвычайно информативно, хоть и замедляет работу программы. Часто необходимо, чтобы программа выполнялась в обычном режиме, пока не достигнет определенной строки кода. Можно сконфигурировать отладчик для такого режима работы, используя точки останова.

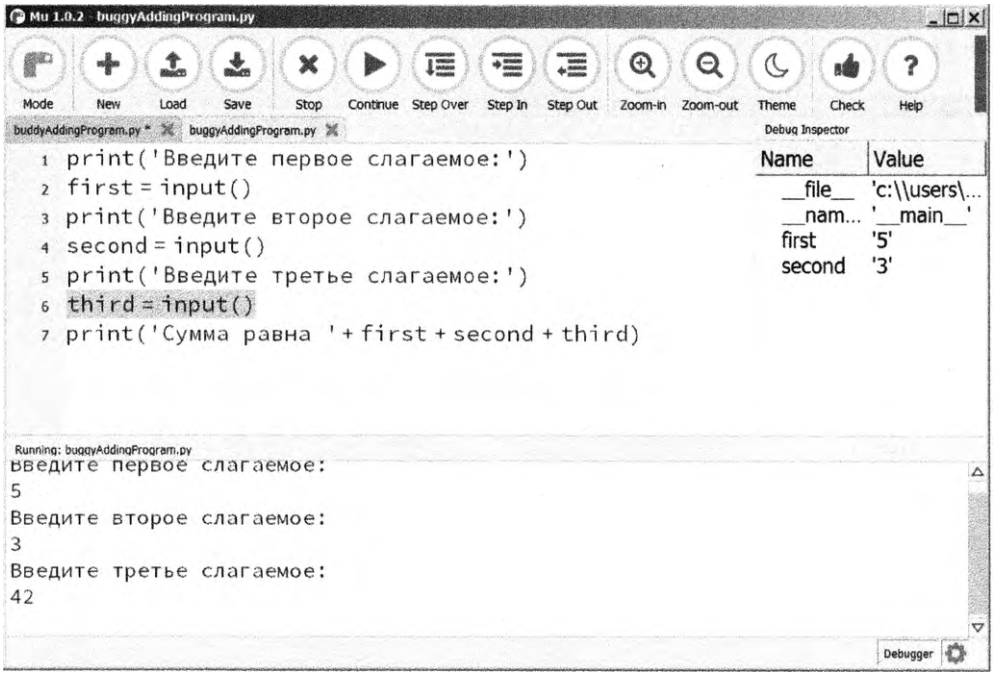


Рис. 11.3. Благодаря панели Debug Inspector справа можно увидеть, что ошибка возникает из-за переменных, которым присвоены строковые, а не целочисленные значения

Точки останова

Точка останова, находящаяся в определенной строке кода, сообщает отладчику о том, что по достижении данной строки выполнение программы следует приостановить. Откройте в файловом редакторе новое окно и введите следующую программу, имитирующую подбрасывание монеты 1000 раз. Сохраните программу в файле *coinFlip.py*.

```
import random
heads = 0
for i in range(1, 1001):
❶     if random.randint(0, 1) == 1:
          heads = heads + 1
      if i == 500:
❷         print('Полпути пройдено!')
print('Орел выпал ' + str(heads) + ' раз.')
```

Функция `random.randint(0, 1)` ❶ в половине случаев будет возвращать `0`, а в половине — `1`. Этот факт можно использовать для имитации подбрасывания монеты с равной вероятностью выпадения "орла" и "решки"; в данном случае "орлу" соответствует значение `1`. Запустив программу без отладчика, вы очень быстро получите примерно следующее.

```
Полпути пройдено!
Орел выпал 490 раз.
```

Если же запустить программу под управлением отладчика, то вам придется щелкать на кнопке Step Over тысячи раз, пока программа завершится. Чтобы узнать, сколько раз выпал "орел", когда программа выполнилась наполовину, т.е. произошло 500 подбрасываний монеты из 1000 возможных, задайте точку останова в строке `print('Полпути пройдено!')` ❷. Для этого щелкните на номере строки в файловом редакторе, после чего она будет помечена красным маркером (рис. 11.4).

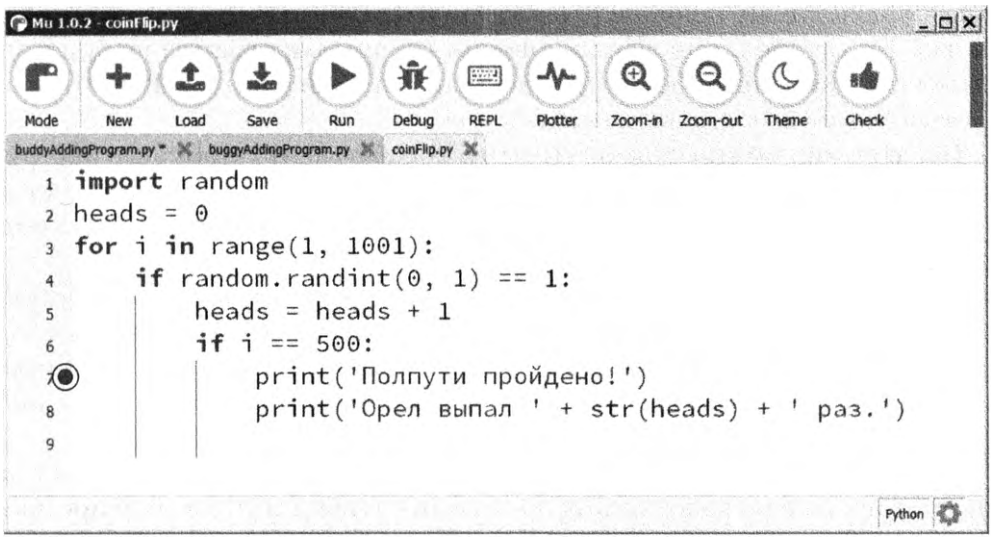


Рис. 11.4. Задание точки останова

Задавать точку останова для строки с инструкцией `if` не следует, поскольку она выполняется на каждой итерации цикла. А вот если задать точку останова внутри блока `if`, то отладчик будет прерывать выполнение только в тех случаях, когда программа будет входить в данный блок.

Строка, для которой задана точка останова, подсвечивается в файловом редакторе желтым цветом. Когда вы запускаете программу под управлением отладчика, ее выполнение начинается с состояния ожидания в первой строке. Но если щелкнуть на кнопке Continue, то программа начнет выполнять инструкцию за инструкцией до тех пор, пока не достигнет строки, для которой задана точка останова. Далее можете щелкать на кнопках Continue, Step Over, Step, Step In и Step Out, чтобы продолжать выполнение в соответствующем режиме.

Чтобы удалить точку останова, щелкните на номере строки еще раз. Красный маркер исчезнет, и отладчик не будет делать паузу на этой строке в будущем.

Резюме

Утверждения, исключения, протоколирование и отладка — незаменимые инструменты, предназначенные для выявления ошибок в программах. Механизм утверждений, обеспечиваемый инструкцией `assert`, — отличное средство реализации профилактических проверок, обеспечивающее раннее обнаружение потенциальных ошибок, если не выполняются те или иные необходимые условия. Утверждения предназначены для выявления только тех ошибок, при возникновении которых программа не будет пытаться восстановить работу и должна аварийно завершиться. В противном случае следует использовать исключения.

Исключения можно перехватывать и обрабатывать с помощью инструкций `try` и `except`. Модуль `logging` предназначен для наблюдения за состоянием программы в процессе ее выполнения. Это удобнее, чем использовать функцию `print()`.

Отладчик позволяет пошагово выполнять программу по одной инструкции за раз. Кроме того, можно выполнять программу в обычном режиме до строки кода, для которой задана точка останова. Используя отладчик, можно наблюдать за значениями любых переменных в любой точке программы.

Использование перечисленных средств и методик отладки облегчает написание правильно работающих программ. Ошибки есть в любой программе, независимо от того, каким опытом программирования вы обладаете.

Контрольные вопросы

1. Напишите инструкцию `assert`, которая генерирует исключение `AssertionError`, если переменная `spam` содержит целое число, меньшее `10`.
2. Напишите инструкцию `assert`, которая генерирует исключение `AssertionError`, если переменные `eggs` и `bacon` содержат одинаковые строки без учета регистра (например, `'hello'` и `'hello'` или `'goodbye'` и `'GOODbye'`).
3. Напишите инструкцию `assert`, которая *всегда* генерирует исключение `AssertionError`.
4. Какие две строки кода должна содержать программа для того, чтобы иметь возможность вызывать функцию `logging.debug()`?

5. Какие две строки кода должна содержать программа для того, чтобы функция `logging.debug()` записывала журнальные сообщения в файл *programLog.txt*?
6. Назовите пять уровней протоколирования.
7. Какую строку кода можно добавить для того, чтобы отключить все журнальные сообщения в программе?
8. Почему для отображения одного и того же отладочного текста лучше использовать журнальные сообщения, а не вызовы функции `print()`?
9. В чем разница между командами Step In, Step Over и Step Out в отладчике?
10. Когда отладчик прервет выполнение программы, если щелкнуть на кнопке Continue?
11. Что такое точка останова?
12. Как задать точку останова для строки кода в отладчике Mu?

Учебный проект

Чтобы закрепить полученные знания на практике, реализуйте предложенную ниже задачу.

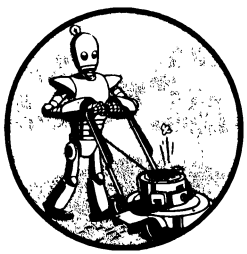
Отладка программы, имитирующей подбрасывание монеты

Приведенная ниже программа предназначена для имитации игры в подбрасывание монеты. Игроку предлагаются два варианта ответа (это простая игра). Однако в программе есть несколько ошибок. Запустите программу несколько раз для того, чтобы обнаружить ошибки, препятствующие ее правильной работе.

```
import random
guess = ''
while guess not in ('орел', 'решка'):
    print('Угадайте результат! Введите "орел" или "решка":')
    guess = input()
toss = random.randint(0, 1)    # 0 - решка, 1 - орел
if toss == guess:
    print('Угадали!')
else:
    print('Увы! Попробуйте снова!')
    guesss = input()
    if toss == guess:
        print('Угадали!')
    else:
        print('Нет. Вам не везет в этой игре.')
```

12

ВЕБ-СКРЕЙПИНГ



В те редкие пугающие минуты, когда остаешься без Wi-Fi, отчетливо осознаешь, насколько то, что ты делаешь с помощью компьютера, связано с Интернетом. Я часто замечаю, что по привычке меня так и тянет проверить электронную почту, прочитать сообщения друзей в Твиттере или ответить на вопросы типа "Снимались ли Брэд Питт и Леонардо Ди Каприо вместе до фильма *Однажды в Голливуде*?"[1]

[1] Ответ: нет.

Поскольку множество действий, выполняемых на компьютере, связано с выходом в Интернет, было бы замечательно, чтобы и ваши программы поддерживали такую возможность. *Веб-скрейпинг* — это термин, обозначающий использование программы для загрузки и обработки содержимого веб-страниц. К примеру, Google выполняет множество таких программ, индексируя веб-страницы для своей поисковой системы. В этой главе вы познакомитесь с несколькими модулями, которые облегчают сбор данных с веб-страниц с помощью Python.

- **`webbrowser`**. Входит в состав Python и предназначен для открытия браузера на определенной веб-странице.
- **`requests`**. Предназначен для загрузки файлов и веб-страниц из Интернета.
- **`Bs4`**. Предназначен для парсинга (синтаксического анализа) HTML — языка, на котором написаны веб-страницы.
- **`selenium`**. Предназначен для запуска браузера и управления его работой. Поддерживает заполнение веб-форм и имитацию щелчков мыши в браузере.

Проект: программа `mapIt.py` с модулем `webbrowser`

Функция `open()` модуля `webbrowser` запускает браузер, передавая ему указанный URL-адрес. Введите в интерактивной оболочке следующие инструкции.

```
>>> import webbrowser
>>> webbrowser.open('http://inventwithpython.com/')
```

В браузере откроется веб-страница с адресом `https://inventwithpython.com/`. Это почти единственное, что может делать модуль `webbrowser`. Тем не менее функция `open()` позволяет реализовывать интересные вещи. Например, вставка почтового адреса в приложение Google Карты через буфер обмена — довольно утомительная задача. Можно сэкономить время, написав простой сценарий, который будет автоматически открывать карту в браузере, используя содержимое буфера обмена. Благодаря этому вам останется лишь скопировать адрес в буфер обмена и запустить сценарий, который и загрузит карту.

Вот что должна делать такая программа:

1) получать почтовый адрес из командной строки или буфера обмена;
2) открывать в браузере карту Google, соответствующую указанному адресу.

Это означает, что программа должна выполнять следующие операции:

1) считывать аргументы командной строки из списка `sys.argv`;
2) считывать содержимое буфера обмена;
3) вызывать функцию `webbrowser.open()` для открытия браузера.

Откройте в файловом редакторе новое окно и сохраните программу в файле *mapIt.py*.

Шаг 1. Определение URL-адреса

Руководствуясь инструкциями, приведенными в приложении Б, настройте файл *mapIt.py* таким образом, чтобы при его запуске, например, с помощью следующей команды:

```
C:\> mapit 870 Valencia St, San Francisco, CA 94110
```

сценарий использовал в качестве почтового адреса аргументы командной строки, а не содержимое буфера обмена. Если же аргументы командной строки не заданы, то программа будет знать, что в этом случае нужно использовать буфер обмена.

Прежде всего, необходимо определить, какой URL-адрес следует использовать для указанного почтового адреса. Если открыть сайт `https://maps.google.com/` и выполнить поиск по интересующему вас почтовому адресу, то URL-адрес, отображаемый в строке браузера, будет выглядеть примерно так:

```
https://www.google.com/maps/place/870+Valencia+St/@37.7590311,
122.4215096,17z/data=!3m1!4b1!4m2!3m1!1s0x808f7e3dadc07a37:
0xc86b0b2bb93b73d8
```

Веб-сайты часто вставляют в URL-адреса дополнительные данные для отслеживания посетителей или настройки страниц. Если исключить эти данные и попытаться перейти по упрощенному адресу `https://www.google.com/maps/place/870+Valencia+St+San+Francisco+CA/`, то выясняется, что открывается та же самая страница. Следовательно, нам достаточно направить браузер по адресу `'https://www.google.com/maps/place/`*адресная_строка*`'`, где *адресная_строка* — это почтовый адрес, для которого должна быть открыта карта.

Шаг 2. Обработка аргументов командной строки

Введите следующий код.

```
#! python3
# mapIt.py - открывает карту в браузере, используя почтовый адрес
#            из командной строки или буфера обмена

import webbrowser, sys
if len(sys.argv) > 1:
    # Получение почтового адреса из командной строки
    address = ' '.join(sys.argv[1:])

# СДЕЛАТЬ: получить почтовый адрес из буфера обмена
```

Первая строка сценария всегда начинается символами `#!`. Далее мы импортируем модули `webbrowser` (необходим для запуска браузера) и `sys` (необходим для чтения аргументов командной строки). В переменной `sys.argv` хранится список, включающий имя программы и аргументы командной строки. Если этот список включает что-то кроме имени файла, то функция `len(sys.argv)` вернет значение, большее `1`, указывающее на то, что в командной строке имеются дополнительные аргументы.

Обычно аргументы командной строки разделяются пробелами, но в данном случае мы хотим интерпретировать все аргументы как единую строку. Поскольку `sys.argv` — это список строк, его можно передать методу `join()`, который вернет результат в виде одной строки. Имя программы в этой строке не представляет для нас интереса, поэтому мы передаем методу `join()` не весь список `sys.argv`, а его срез `sys.argv[1:]`, тем самым исключая ненужный элемент списка. Результирующая строка сохраняется в переменной `address`.

Если для запуска программы использовать следующую команду:

```
mapit 870 Valencia St, San Francisco, CA 94110
```

то переменная `sys.argv` будет содержать такой список:

```
['mapIt.py', '870', 'Valencia', 'St, ', 'San', 'Francisco, ', 'CA',
 '94110']
```

В то же время в переменной `address` будет содержаться строка `'870 Valencia St, San Francisco, CA 94110'`.

Шаг 3. Обработка содержимого буфера обмена и запуск браузера

Добавьте в программу код, выделенный полужирным шрифтом.

```
#! python3
# mapIt.py - запускает карту в браузере, используя почтовый адрес
#            из командной строки или буфера обмена

import webbrowser, sys, pyperclip
if len(sys.argv) > 1:
    # Получение почтового адреса из командной строки
    address = ' '.join(sys.argv[1:])
else:
    # Получение почтового адреса из буфера обмена
    address = pyperclip.paste()

webbrowser.open('https://www.google.com/maps/place/' + address)
```

Если аргументы командной строки не предоставлены, то программа предполагает, что адрес хранится в буфере обмена. Мы можем получить содержимое буфера обмена с помощью функции `pyperclip.paste()` и сохранить его в переменной `address`. Наконец, для запуска браузера и открытия сайта Google Карты вызывается функция `webbrowser.open()`.

Обычно мы пишем программы, которые экономят уйму времени, выполняя трудоемкие задачи. Но не стоит недооценивать и программы, позволяющие сэкономить пару секунд каждый раз, когда вы выполняете такие простые действия, как отображение интересующего вас места на карте Google. В табл. 12.1 сравниваются действия, которые приходится выполнять для отображения карты с помощью и без помощи программы *mapIt.py*.

Таблица 12.1. Получение карты с помощью и без помощи программы *mapIt.py*

Получение карты вручную	Использование программы *mapIt.py*
1. Выделение адреса	1. Выделение адреса
2. Копирование адреса	2. Копирование адреса
3. Открытие браузера	3. Запуск программы *mapIt.py*
4. Переход на сайт `http://maps.google.com/`	
5. Щелчок в поле для ввода адреса	
6. Вставка адреса	
7. Нажатие клавиши <Enter>	

Согласитесь, задача немного упростилась, не так ли?

Идеи для создания похожих программ

Модуль `webbrowser` избавляет от необходимости самостоятельно открывать браузер и переходить на нужный веб-сайт. Это может потребоваться для решения следующих задач:

- открытие всех ссылок на веб-странице в отдельных вкладках браузера;
- открытие браузера на странице с прогнозом погоды по вашему региону;
- открытие нескольких сайтов социальных сетей, которые вы регулярно посещаете.

Загрузка файлов из Интернета с помощью модуля `requests`

Модуль `requests` упрощает загрузку файлов из Интернета, позволяя не думать о таких вещах, как ошибки сети, проблемы подключения и сжатие данных. Этот модуль не входит в состав Python и должен быть предварительно установлен. Выполните в командной строке команду **`pip install requests`**. (Информация об установке сторонних модулей приведена в приложении А.)

Далее убедитесь в том, что модуль `requests` корректно установлен:

```
>>> import requests
```

Отсутствие сообщений об ошибке будет означать, что установка модуля `requests` прошла успешно.

Загрузка веб-страницы с помощью функции `requests.get()`

Функция `requests.get()` получает строку URL-адреса, с которого должна осуществляться загрузка. Вызвав функцию `type()` для значения, возвращаемого функцией `requests.get()`, вы увидите, что это объект `Response`, в котором содержится ответ сервера на ваш запрос. С объектом `Response` мы познакомимся позже, а пока введите в интерактивной оболочке следующие инструкции, предварительно убедившись в том, что компьютер подключен к Интернету.

```
  >>> import requests
❶ >>> res =
  requests.get('https://automatetheboringstuff.com/files/rj.txt')
  >>> type(res)
  <class 'requests.models.Response'>
❷ >>> res.status_code == requests.codes.ok
```

```
True
>>> len(res.text)
178981
>>> print(res.text[:250])

The Project Gutenberg EBook of Romeo and Juliet, by William
Shakespeare. This eBook is for the use of anyone anywhere at no
cost and with almost no restrictions whatsoever. You may copy
it, give it away or re-use it under the terms of the Proje
```

По указанному URL-адресу находится веб-страница, содержащая полный текст пьесы "Ромео и Джульетта" ❶. Проверить успешность выполнения запроса можно с помощью атрибута `status_code` объекта `Response`. Значение `requests.codes.ok` означает, что запрос был успешно выполнен ❷. В протоколе HTTP успешному запросу, т.е. "OK", соответствует код 200. Вы наверняка уже сталкивались с кодом состояния 404 "Not Found" (Не найдено). Полный список кодов состояния HTTP доступен по следующему адресу:

```
https://en.wikipedia.org/wiki/List_of_HTTP_status_codes
```

В случае успешного выполнения запроса загруженная страница сохраняется в виде строки в переменной `text` объекта `Response`. В этой переменной хранится полный текст пьесы в виде одной длинной строки. Функция `len(res.text)` сообщает о том, что данная строка содержит более 178 000 символов. Наконец, функция `print(res.text[:250])` отображает первые 250 символов строки.

Если запрос не выполнен и появляется сообщение об ошибке, например "Failed to establish a new connection" (Не удалось установить новое подключение) или "Max retries exceeded" (Превышено максимальное количество попыток), проверьте подключение к Интернету. Взаимодействие с сервером — достаточно сложная тема, и список возможных проблем в данном случае слишком велик. Попробуйте выяснить общие причины появления ошибки, выполнив поиск сообщения об ошибке в Интернете.

Проверка ошибок

Как вам уже известно, объект `Response` имеет атрибут `status_code`, сравнение которого со значением `requests.codes.ok` позволяет судить о том, была ли загрузка успешной. Однако для такой проверки есть и более простой способ, который заключается в том, чтобы вызвать метод `raise_for_status()` объекта `Response`. Данный метод генерирует исключение, если в процессе загрузки файла произошла ошибка, и не совершает никаких действий в случае успешной загрузки. Введите в интерактивной оболочке следующие инструкции.

```
>>> res =
requests.get('https://inventwithpython.com/page_that_does_not_exist')
>>> res.raise_for_status()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>

  File "C:\Users\Al\AppData\Local\Programs\Python\Python37\lib\
sitepackages\requests\models.py", line 940, in raise_for_status
    raise HTTPError(http_error_msg, response=self)
requests.exceptions.HTTPError: 404 Client Error: Not Found for url:
https://inventwithpython.com/page_that_does_not_exist.html
```

Метод `raise_for_status()` — это эффективный способ гарантированно остановить программу в случае неудачной загрузки, ведь вы сами заинтересованы в том, чтобы при возникновении неожиданных ошибок выполнение программы как можно быстрее прекращалось. Если же неудачная загрузка *не* является препятствием для дальнейшего выполнения программы, то можно заключить строку кода `raise_for_status()` в блок `try/except`, чтобы обработать эту ошибку, не допуская аварийной остановки программы.

```
import requests
res = requests.get('https://inventwithpython.com/
                   page_that_does_not_exist')
try:
    res.raise_for_status()
except Exception as exc:
    print('Возникла проблема: %s' % (exc))
```

В результате вызова метода `raise_for_status()` программа выведет следующую информацию:

```
Возникла проблема: 404 Client Error: Not Found for url:
https://inventwithpython.com/page_that_does_not_exist.html
```

Целесообразно всегда вызывать метод `raise_for_status()` после функции `requests.get()`. Это позволит убедиться в том, что загрузка действительно прошла успешно, прежде чем продолжить выполнение программы.

Сохранение загруженных файлов на жестком диске

В этом разделе речь пойдет о том, как сохранить веб-страницу в файле на жестком диске с помощью стандартной функции `open()` и метода `write()`. Но здесь есть один нюанс. Дело в том, что файл необходимо открыть в режиме *бинарной записи*, передав функции `open()` строку `'wb'` в качестве второго аргумента. Даже если страница содержит простой текст

(например, загруженный ранее текст пьесы "Ромео и Джульетты"), для поддержки кодировки Unicode необходимо записывать бинарные данные, а не текстовые.

Кодировка Unicode

Рассмотрение кодировки Unicode выходит за рамки книги. Для получения более подробной информации по этой теме обратитесь к следующим ресурсам в Интернете:

- *Joel on Software: The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)*:

  `https://www.joelonsoftware.com/articles/Unicode.html`

- *Pragmatic Unicode*:

  `https://nedbatchelder.com/text/unipain.html`

Записать веб-страницу в файл можно в цикле `for`, используя метод `iter_content()` объекта `Response`.

```
>>> import requests
>>> res =
requests.get('https://automatetheboringstuff.com/files/rj.txt')
>>> res.raise_for_status()
>>> playFile = open('RomeoAndJuliet.txt', 'wb')
>>> for chunk in res.iter_content(100000):
        playFile.write(chunk)

100000
78981
>>> playFile.close()
```

Метод `iter_content()` на каждой итерации цикла возвращает фрагмент содержимого. Каждая порция данных — это данные байтового типа; соответственно, нужно указать, каким должен быть размер фрагмента в байтах. В общем случае сто тысяч байтов — вполне подходящий размер, поэтому мы передаем функции `iter_content()` значение `100000` в качестве аргумента.

Теперь в текущем каталоге появится файл *RomeoAndJuliet.txt*. Обратите внимание на то, что имя, под которым файл сохраняется на жестком диске, отличается от имени файла, используемого на веб-странице (*rj.txt*). Загрузкой содержимого веб-страниц управляет модуль `requests`. Как только страница загружена, она становится простым набором данных в программе. Даже если после загрузки страницы соединение с Интернетом будет разорвано, ее содержимое останется в памяти компьютера.

Метод `write()` возвращает количество байтов, записанных в файл. В предыдущем примере первая порция включала 100 000 байт, а оставшаяся часть файла заняла всего 78 981 байт.

Опишем пошаговый процесс загрузки и сохранения файла.

1. Вызов функции `requests.get()` для загрузки файла.
2. Вызов функции `open()` с аргументом `'wb'` для создания нового файла в режиме бинарной записи.
3. Цикл по возвращаемому значению метода `iter_content()` объекта `Response`.
4. Вызов метода `write()` на каждой итерации для записи содержимого файла.
5. Вызов метода `close()` для закрытия файла.

Это все, что следует знать о модуле `requests`! Использование цикла `for` и метода `iter_content()` может показаться более сложной процедурой по сравнению с технологией, основанной на применении цепочки вызовов `open()/write()/close()`, которую мы использовали для записи текстовых файлов. Однако такой подход гарантирует, что модуль `requests` не будет потреблять слишком много памяти даже в случае загрузки огромных файлов. Более подробная информация о других возможностях модуля `requests` доступна по адресу `https://requests.readthedocs.org/`.

HTML

Прежде чем приступать к анализу веб-страниц, следует познакомиться с основами HTML и узнать о том, как получить доступ к мощным средствам веб-разработки, имеющимся в самом браузере.

Ресурсы для изучения HTML

HTML (Hypertext Markup Language – язык гипертекстовой разметки) применяется для записи веб-страниц. Предполагается, что вы уже владеете определенными навыками работы с HTML, но если вы нуждаетесь в руководстве для новичков, то порекомендую следующие сайты:

- `https://developer.mozilla.org/ru/docs/Learn/HTML/`
- `https://htmldog.com/guides/html/beginner/`
- `https://www.codecademy.com/learn/learn-html`

Краткие сведения об HTML

Если раньше вы сталкивались с HTML лишь эпизодически, то будет не лишним вспомнить основы языка разметки. HTML-файл – это обычный

текстовый файл с расширением *.html*. Текст в таких файлах заключается в *теги* (дескрипторы), а каждый тег представляет собой слово в угловых скобках. Теги сообщают браузеру, как следует форматировать веб-страницу. Начальный и конечный (открывающий и закрывающий) теги могут обрамлять определенный текст, вместе с которым они образуют *элемент*.

Текст (внутренняя HTML-разметка) — это содержимое, находящееся между открывающим и закрывающим тегами. Например, следующая HTML-строка отображает в браузере фразу `'Здравствуй, мир!'`, в которой слово `'Здравствуй'` выделено полужирным шрифтом:

```
<strong>Здравствуй</strong>, мир!
```

Вид страницы в браузере показан на рис. 12.1.

Рис. 12.1. Фраза `'Здравствуй, мир!'`, отображаемая в браузере

Открывающий тег `<strong>` указывает на то, что последующий текст должен отображаться полужирным шрифтом. Закрывающий тег `</strong>` обозначает, где заканчивается полужирный текст.

В HTML имеется множество тегов. У некоторых из них есть дополнительные свойства в виде *атрибутов*, записываемых в угловых скобках. Например, тег `<a>` содержит текст гиперссылки, при этом URL-адрес гиперссылки задается атрибутом `href`. Рассмотрим пример.

```
<a href="http://inventwithpython.com">Книги по Python</a>,
бесплатно предоставляемые на сайте Эла.
```

Вид страницы в браузере показан на рис. 12.2.

У некоторых элементов есть атрибут `id`, который используется в качестве уникального идентификатора элемента на веб-странице. В своих программах вам часто придется выполнять поиск элемента по идентификатору, поэтому нахождение данного атрибута с помощью инструментов

разработчика, доступных в браузере, — одна из самых распространенных задач при написании программ для веб-скрейпинга.

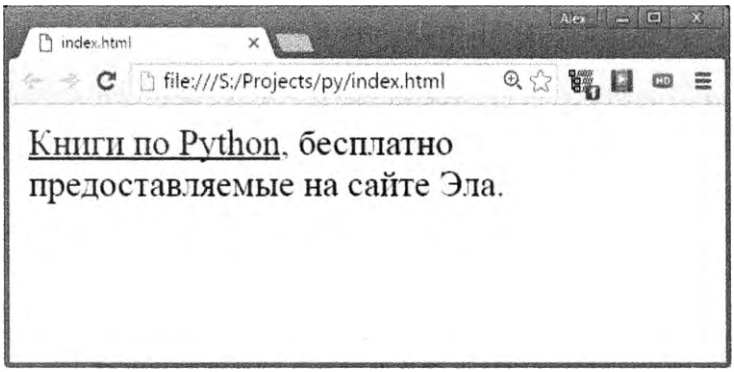


Рис. 12.2. Гиперссылка, отображаемая в браузере

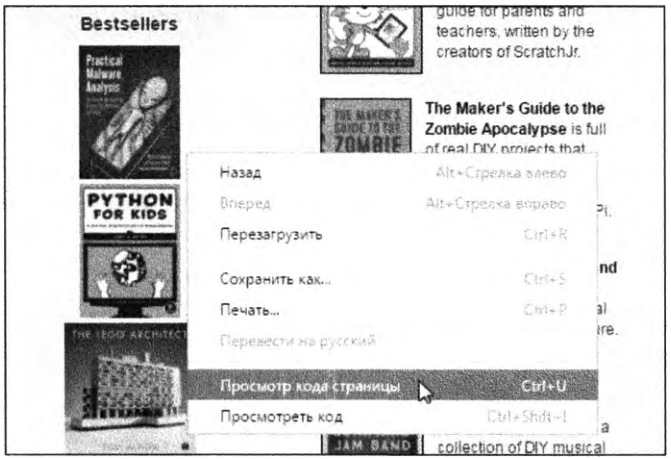
Просмотр HTML-кода веб-страницы

Так или иначе вам понадобится просматривать исходный HTML-код веб-страниц, с которыми будут работать ваши программы. Чтобы увидеть разметку, щелкните правой кнопкой мыши (или левой кнопкой при нажатой клавише <Ctrl> в macOS) на любой веб-странице в браузере и выберите в контекстном меню пункт Просмотреть код (View Source) или Просмотр кода страницы (View page source), как показано на рис. 12.3. Именно с этим текстом работает браузер. Браузеру известно, как отображать (*визуализировать*) веб-страницы на основе HTML-кода.

Рекомендую потратить какое-то время на изучение HTML-кода ваших любимых сайтов. Если при просмотре HTML-кода вам не все будет понятно, не расстраивайтесь. Чтобы писать простые программы для веб-скрейпинга, не нужно быть искусным мастером HTML. В конце концов, речь не идет о разработке полноценного веб-сайта. Вам достаточно лишь знать, как организовать сбор данных с существующих сайтов.

Открытие окна инструментов веб-разработки в браузере

HTML-код веб-страницы можно просматривать с помощью инструментов веб-разработки, имеющихся в браузере. В браузерах Chrome и Internet Explorer для Windows инструменты разработчика уже установлены, и для доступа к ним достаточно нажать клавишу <F12> (рис. 12.4). Повторное нажатие клавиши <F12> приводит к закрытию окна веб-разработки. В браузере Chrome можно также выполнить команду Настройка и управление Google Chrome⇨Инструменты⇨Инструменты разработчика. В macOS для этого следует нажать комбинацию клавиш <⌘+Option+I>.



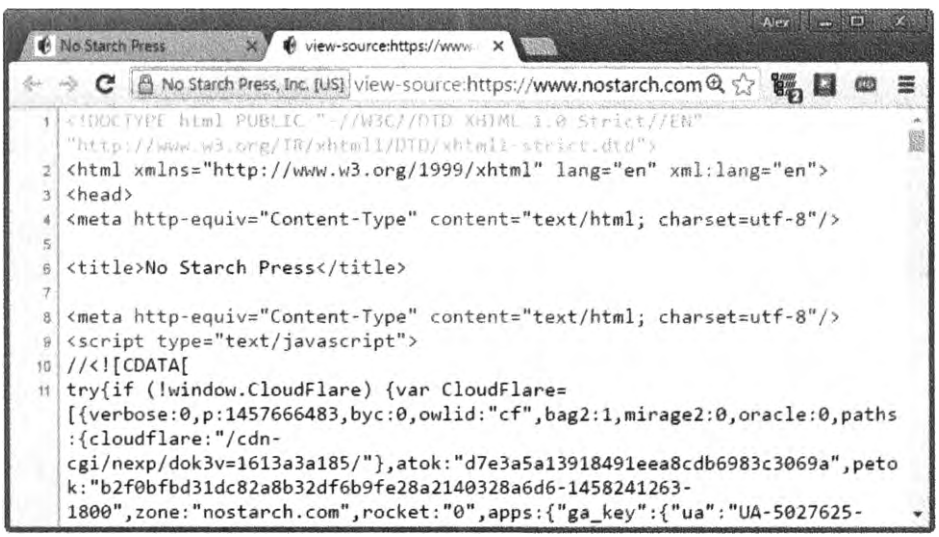


Рис. 12.3. Просмотр исходного кода веб-страницы

В браузере Mozilla Firefox можно нажать комбинацию клавиш <Ctrl+Shift+C> в Windows и Linux или комбинацию клавиш <⌘+Option+C> в macOS. Окно инструментов веб-разработки будет выглядеть почти так же, как и в Chrome.

В браузере Safari следует открыть окно Preferences (Установки) и установить флажок Show Develop menu in the menu bar (Показывать меню инструментов разработчика в строке меню) на панели Advanced (Дополнительно). После этого можно вызвать окно инструментов разработчика, нажав комбинацию клавиш <⌘+Option+I>.

Активизировав инструменты разработчика в браузере, щелкните правой кнопкой мыши в любой части веб-страницы и выберите в контекстном

меню пункт Inspect Element (Исследовать элемент), чтобы выделить HTML-разметку, связанную с данным элементом страницы. Такая возможность окажется очень полезной, когда вы приступите к синтаксическому анализу (парсингу) HTML-документов в программах веб-скрейпинга.

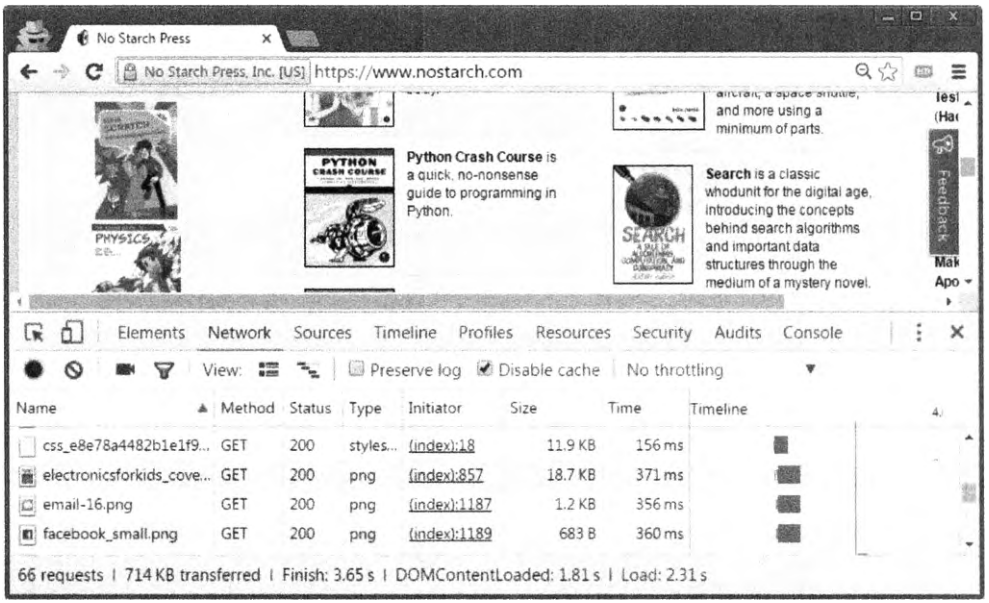


Рис. 12.4. Окно инструментов разработчика в Google Chrome

Не пытайтесь использовать регулярные выражения для парсинга HTML-разметки

Казалось бы, что может быть эффективнее регулярных выражений, когда речь идет о нахождении определенных HTML-элементов в строке? Однако поступать так не рекомендуется. Существует множество способов корректного оформления HTML-кода, и попытки описать все возможные варианты с помощью регулярных выражений потребуют чрезмерных усилий и будут чреваты ошибками. Лучше воспользоваться одним из модулей, специально разработанных для парсинга HTML-разметки, например `bs4`.

Использование инструментов веб-разработки для поиска HTML-элементов

Как только программа загрузит веб-страницу с помощью модуля `requests`, в вашем распоряжении окажется ее HTML-содержимое в виде одной строки. Дальнейшие действия заключаются в том, чтобы определить, какая часть HTML-содержимого соответствует интересующему вас элементу.

И здесь на помощь придут инструменты разработчика, предоставляемые браузером. Предположим, вы хотите написать программу, осуществляющую сбор данных о прогнозах погоды на сайте `https://weather.gov/`. Прежде чем приступать к написанию кода, проведите небольшой эксперимент. Посетите указанный сайт и выполните поиск по ZIP-коду (почтовому индексу) 94105. Вы получите прогноз погоды для данного региона.

На появившейся странице вас интересуют только данные о погоде. Щелкните правой кнопкой мыши (или левой кнопкой мыши при нажатой клавише <Control> в macOS) в соответствующем месте страницы и выберите в контекстном меню пункт Inspect Element (Исследовать элемент). В открывшемся окне инструментов разработчика отобразится HTML-код, ответственный за создание данного фрагмента страницы. На рис. 12.5 показано окно инструментов веб-разработки с соответствующей HTML-разметкой. (Учтите, что при изменении дизайна сайта вам придется исследовать другие элементы.)

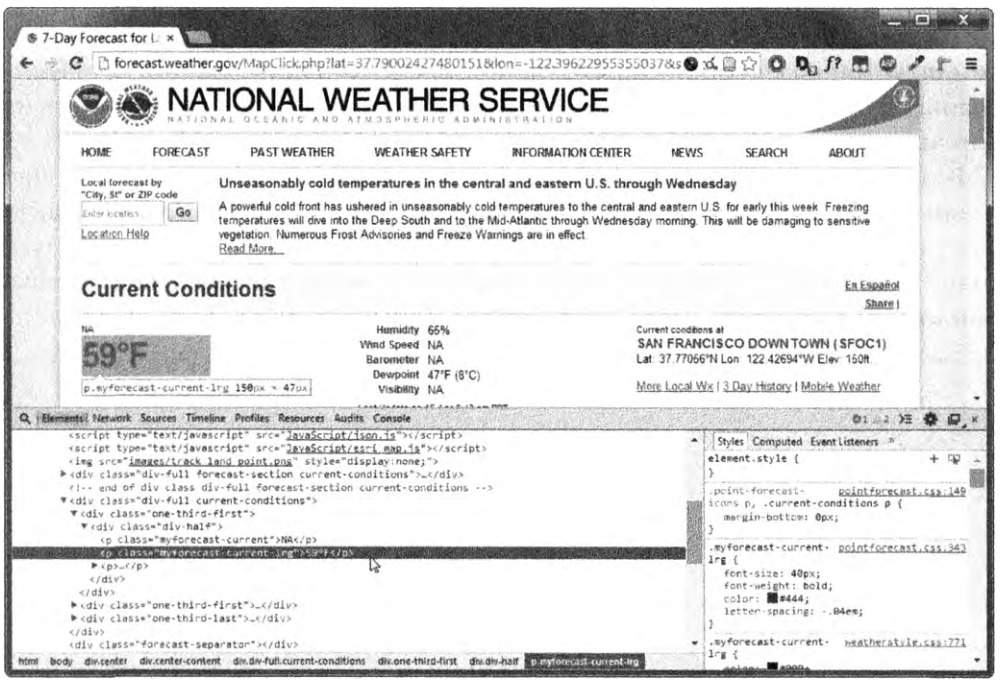


Рис. 12.5. Инспектирование HTML-разметки с помощью инструментов веб-разработки

Окно инструментов разработчика позволяет увидеть, что за отображение прогноза погоды отвечает следующий код.

```
<div class="col-sm-10 forecast-text" >Sunny, with a high near 64.
West wind 11 to 16 mph, with gusts as high as 21 mph.</div>
```

Это именно то, что вы искали! Нетрудно заметить, что информация о погоде содержится в элементе , который относится к CSS-классу forecast-text. Щелкните правой кнопкой мыши на этом элемент в консоли разработчика и выберите в контекстном меню пункт Копировать⇨CSS-селектор. Это приведет к копированию строки вида div.row-odd:nth-child(1) > div:nth-child(2) в буфер обмена. Данную строку можно использовать в функции select() модуля Beautiful Soup или в методе find_element_by_css_selector() модуля selenium, как будет описано далее. Теперь, когда вы уже знаете, что именно вам требуется, модуль Beautiful Soup поможет вам найти искомый элемент в строке.

Парсинг HTML-разметки с помощью модуля bs4

Beautiful Soup — это модуль, предназначенный для извлечения информации из HTML-страницы (он намного удобнее, чем регулярные выражения). Имя модуля в Python — bs4. Для установки модуля необходимо выполнить в командной строке команду pip install --user beautifulsoup4. (Инструкции по установке сторонних модулей приведены в приложении А.) Несмотря на то что при установке используется имя beautifulsoup4, модуль импортируется с помощью инструкции import bs4.

В примерах этой главы мы будем выполнять *парсинг* (синтаксический анализ) HTML-файла, хранящегося на жестком диске. Откройте в файловом редакторе новое окно, введите в нем приведенный ниже текст и сохраните его в файле *example.html*. Можете также воспользоваться готовым файлом, который содержится в каталоге примеров книги (см. введение).

```
<!-- Это файл example.html -->

<html><head><title>Заголовок веб-сайта</title></head>
<body>
<p>Загрузите мои книги по <strong>Python</strong>
с <a href="http://inventwithpython.com">моего сайта</a>.</p>
<p class="slogan">Простой подход к изучению Python!</p>
<p>Автор <span id="author">Эл Свейгарт</span></p>
</body></html>
```

Как видите, даже простой HTML-файл включает множество тегов и атрибутов, количество которых быстро увеличивается при переходе к сложным веб-сайтам. К счастью, модуль Beautiful Soup существенно упрощает работу с HTML-разметкой.

Создание объекта BeautifulSoup на основе HTML-разметки

Необходимо вызвать функцию `bs4.BeautifulSoup()`, передав ей строку, которая содержит анализируемый HTML-код. Данная функция возвращает объект `BeautifulSoup`. Введите в интерактивной оболочке следующие инструкции, предварительно убедившись в том, что компьютер подключен к Интернету.

```
>>> import requests, bs4
>>> res = requests.get('https://nostarch.com')
>>> res.raise_for_status()
>>> noStarchSoup = bs4.BeautifulSoup(res.text, 'html.parser')
>>> type(noStarchSoup)
<class 'bs4.BeautifulSoup'>
```

Сначала функция `requests.get()` загружает главную страницу сайта No Starch Press, после чего атрибут `text` ответа передается функции `bs4.BeautifulSoup()`. Возвращаемый этой функцией объект `BeautifulSoup` сохраняется в переменной `noStarchSoup`.

Можно также загрузить HTML-файл с жесткого диска, передав функции `bs4.BeautifulSoup()` объект `File` вместе со вторым аргументом, который сообщает Beautiful Soup о том, какой парсер используется для анализа HTML-разметки.

Введите в интерактивной оболочке следующие инструкции (предварительно убедившись в том, что файл *example.html* находится в текущем каталоге).

```
>>> exampleFile = open('example.html')
>>> exampleSoup = bs4.BeautifulSoup(exampleFile, 'html.parser')
>>> type(exampleSoup)
<class 'bs4.BeautifulSoup'>
```

Парсер `html.parser` входит в состав Python. Можно также использовать более быстрый парсер `'lxml'`, если в системе установлен сторонний модуль `lxml`. (Инструкции по установке сторонних модулей приведены в приложении А. В данном случае нужно выполнить команду `pip install --user lxml`.) При отсутствии второго аргумента вы получите предупреждение `"UserWarning: No parser was explicitly specified"`.

После получения объекта `BeautifulSoup` можно использовать его методы для поиска нужных фрагментов HTML-документа.

Поиск элемента с помощью метода select()

Чтобы получить элемент веб-страницы из объекта `BeautifulSoup`, можно вызвать метод `select()` и передать ему строку *CSS-селектора*

искомого элемента. Селекторы напоминают регулярные выражения: они тоже задают шаблон поиска, но только в данном случае поиск осуществляется в HTML-страницах, а не в текстовых строках.

Рассмотрение синтаксиса CSS-селекторов выходит за рамки книги (официальное руководство доступно по адресу `https://www.w3.org/TR/CSS/#css`), поэтому мы ограничимся лишь кратким их обзором. Примеры наиболее часто используемых селекторов приведены в табл. 12.2.

Таблица 12.2. Примеры CSS-селекторов

Селектор, передаваемый методу `select()`	Чему соответствует
`soup.select('div')`	Все элементы `<div>`
`soup.select('#author')`	Элемент, атрибут `id` которого равен `"author"`
`soup.select('.notice')`	Все элементы, атрибут `class` которых равен `"notice"`
`soup.select('div span')`	Все элементы `<span>`, вложенные в элементы `<div>`
`soup.select('div > span')`	Все элементы `<span>`, вложенные *непосредственно* в элементы `<div>`, без каких бы то ни было элементов между ними
`soup.select('input[name]')`	Все элементы `<input>`, имеющие атрибут *name*, независимо от его значения
`soup.select('input[type="button"]')`	Все элементы `<input>`, имеющие атрибут `type` со значением `"button"`

Различные селекторы могут сочетаться, образуя сложные шаблоны. Например, вызову `soup.select('p #author')` будет соответствовать любой элемент, атрибут `id` которого равен `author`, при условии, что этот элемент вложен в элемент `<p>`. Вместо самостоятельного создания селектора можно щелкнуть правой кнопкой мыши на элементе в окне браузера и выбрать в контекстном меню пункт **Просмотреть код**. После открытия консоли разработчика щелкните правой кнопкой на HTML-коде элемента и выполните команду **Копировать⇨CSS-селектор**, чтобы скопировать строку селектора в буфер обмена и вставить ее в исходный код.

Метод `select()` возвращает список объектов `Tag`, представляющих HTML-элементы. Этот список будет содержать по одному объекту `Tag` для каждого найденного совпадения в HTML-коде объекта `BeautifulSoup`. Объекты `Tag` можно передавать функции `str()` для отображения соответствующих тегов HTML. У этих объектов есть также атрибут `attrs`, представляющий все HTML-атрибуты данного тега в виде словаря. Используя рассмотренный ранее файл *example.html*, введите в интерактивной оболочке следующие инструкции.

```
>>> import bs4
>>> exampleFile = open('example.html')
>>> exampleSoup = bs4.BeautifulSoup(exampleFile.read(), 'html.parser')
>>> elems = exampleSoup.select('#author')
>>> type(elems)     # elems -- список объектов Tag
<class 'list'>
>>> len(elems)
1
>>> type(elems[0])
<class 'bs4.element.Tag'>
>>> str(elems[0])     # преобразование объекта Tag в строку
'<span id="author">Эл Свейгарт</span>'
>>> elems[0].getText()
'Эл Свейгарт'
>>> elems[0].attrs
{'id': 'author'}
```

В данном случае мы извлекаем элемент с идентификатором `"author"` из нашего примера HTML-разметки. Функция `select('#author')` возвращает список всех элементов, удовлетворяющих данному условию. Мы сохраняем этот список объектов `Tag` в переменной `elems`, и значение, возвращаемое функцией `len(elems)`, указывает на то, что в списке имеется только один такой элемент. Вызов метода `getText()` для этого элемента возвращает содержащийся в нем текст, т.е. внутреннюю HTML-разметку. Текст элемента — это содержимое, заключенное между его открывающим и закрывающим тегами. В данном случае это строка `'Эл Свейгарт'`.

Функция `str()` возвращает строку, включающую открывающий и закрывающий теги вместе с текстом элемента. Наконец, переменная `attrs` содержит словарь, включающий имя атрибута, `'id'`, и его значение, `'author'`.

Также можно извлечь из объекта `BeautifulSoup` все элементы `<p>`. Введите в интерактивной оболочке следующие инструкции.

```
>>> pElems = exampleSoup.select('p')
>>> str(pElems[0])
'<p>Загрузите мои книги по <strong>Python</strong>
с <a href="http://inventwithpython.com">моего сайта</a>.</p>'
>>> pElems[0].getText()
'Загрузите мои книги по Python с моего сайта.'
>>> str(pElems[1])
'<p class="slogan">Простой подход к изучению Python!</p>'
>>> pElems[1].getText()
'Простой подход к изучению Python!'
>>> str(pElems[2])
'<p>Автор <span id="author">Эл Свейгарт</span></p>'
>>> pElems[2].getText()
'Автор Эл Свейгарт'
```

На этот раз метод `select()` возвращает список с тремя элементами, который сохраняется в переменной `pElems`. Поочередно передавая функции `str()` элементы `pElems[0]`, `pElems[1]` и `pElems[2]`, мы отображаем каждый из этих элементов в виде строки, а вызов метода `getText()` для элементов позволяет получить их текстовое содержимое.

Получение данных из атрибутов элемента

Метод `get()` объекта `Tag` упрощает доступ к значениям атрибутов соответствующего элемента. Метод получает строку с именем атрибута и возвращает его значение. Используя файл *example.html*, введите в интерактивной оболочке следующие инструкции.

```
>>> import bs4
>>> soup = bs4.BeautifulSoup(open('example.html'), 'html.parser')
>>> spanElem = soup.select('span')[0]
>>> str(spanElem)
'<span id="author">Эл Свейгарт</span>'
>>> spanElem.get('id')
'author'
>>> spanElem.get('несуществующий_адрес') == None
True
>>> spanElem.attrs
{'id': 'author'}
```

Мы используем метод `select()` для нахождения элементов `<span>` и сохранения первого из них в переменной `spanElem`. Передав методу `get()` имя атрибута `'id'`, мы получаем соответствующее значение: `'author'`.

Проект: открытие всех результатов поиска

Выполняя поиск в Google, я никогда не начинаю сразу же просматривать все полученные ссылки одну за другой. Вместо этого я открываю несколько первых ссылок в отдельных вкладках, чтобы просмотреть их позже. Поскольку я довольно часто пользуюсь поиском в Google, описанный порядок действий — открытие браузера, поиск темы и последующее открытие ссылок — оказывается достаточно утомительным. Было бы неплохо, если бы я мог просто ввести поисковый термин в командной строке, а компьютер автоматически открыл в браузере отдельные вкладки с первыми результатами поиска. Давайте напишем соответствующий сценарий, работающий со страницей индекса пакетов Python (`https://pypi.org/`). Подобную программу можно адаптировать ко множеству других сайтов, хотя Google и DuckDuckGo часто принимают меры, которые затрудняют веб-скрейпинг их поисковых страниц.

Вот что должна делать такая программа:

1) получать из аргументов командной строки ключевые слова, по которым должен быть выполнен поиск;
2) извлекать страницу с результатами поиска;
3) открывать каждый результат в отдельной вкладке браузера.

Это означает, что программа должна будет выполнять следующие операции:

1) читать аргументы командной строки из списка `sys.argv`;
2) извлекать страницу с результатами поиска с помощью модуля `requests`;
3) находить ссылки для каждого результата поиска;
4) вызывать функцию `webbrowser.open()` для открытия браузера.

Откройте в редакторе файлов новую вкладку и сохраните программу в файле *searchpypi.py*.

Шаг 1. Получение аргументов командной строки и запрос поисковой страницы

Прежде чем приступить к написанию кода, необходимо определить URL-адрес страницы с результатами поиска. Взглянув на адресную строку браузера после выполнения поиска в Google, вы увидите там URL-адрес вида `https://pypi.org/search?q=`*ПОИСКОВЫЙ_ТЕРМИН*. Модуль `requests` загрузит эту страницу, после чего вы сможете использовать Beautiful Soup для нахождения ссылок на результаты поиска в HTML-коде. В конце используется модуль `webbrowser` для открытия найденных ссылок в отдельных вкладках браузера.

Введите в созданный файл следующий код.

```
#! python3
# searchpypi.py - открывает несколько результатов поиска

import requests, sys, webbrowser, bs4
print('Поиск...')    # отображается при загрузке страницы
                     # результатов поиска
res = requests.get('https://pypi.org/search/?q=' + \
                   ' '.join(sys.argv[1:]))
res.raise_for_status()

# СДЕЛАТЬ: извлечь первые несколько найденных ссылок

# СДЕЛАТЬ: открыть отдельную вкладку для каждого результата
```

Поисковые термины будут предоставляться пользователем с помощью аргументов командной строки при запуске программы. Эти аргументы хранятся в виде строк в списке `sys.argv`.

Шаг 2. Поиск всех результатов

Теперь настал черед использовать модуль Beautiful Soup для извлечения нескольких первых поисковых ссылок из загруженного HTML-документа. Но как определить, какой селектор использовать для этого? Например, нельзя просто отобрать все теги `<a>`, поскольку в полученном HTML-документе будет множество ссылок, не представляющих для вас интереса. Вместо этого необходимо проинспектировать страницу с результатами поиска с помощью инструментов веб-разработки, доступных в браузере, чтобы определить селектор, который отбирает только нужные нам ссылки.

Используя для поиска строку *Beautiful Soup* в качестве поискового термина, откройте окно инструментов разработчика и исследуйте некоторые из элементов, содержащих ссылки. Эти элементы могут выглядеть следующим образом.

```
<a class="package-snippet" href="HYPERLINK "
view-source:https://pypi.org/project/xml-parser/
"/project/xml-parser/">
```

Пусть вас не смущает вид элемента. Нам лишь нужно определить шаблон, который является общим для всех ссылок.

Добавьте в программу код, выделенный полужирным шрифтом.

```
#! python3
# searchpypi.py - открывает несколько результатов поиска

import requests, sys, webbrowser, bs4
-- Опущено --
# Извлечение первых найденных ссылок
soup = bs4.BeautifulSoup(res.text, 'html.parser')
# Открытие отдельной вкладки для каждого результата
linkElems = soup.select('.package-snippet')
```

Изучив элемент `<a>`, вы заметите, что все поисковые ссылки относятся к классу `package-snippet`. Просмотр оставшейся части HTML-кода позволяет предположить, что этот класс используется только для таких ссылок. Вам необязательно знать, что собой представляет CSS-класс `package-snippet` и что он делает. Вы просто будете использовать его в качестве маркера элемента `<a>`, который нужно найти. Можно создать объект `BeautifulSoup` из HTML-текста загруженной страницы, а затем

использовать селектор '.package-snippet' для поиска всех элементов <a>, которые вложены в элемент, имеющий CSS-класс package-snippet. Учтите, что если структура сайта PyPI изменится, то придется обновить программу, передав методу soup.select() новую строку селектора CSS. Остальная часть программы останется неизменной.

Шаг 3. Открытие браузера для каждого из результатов поиска

Наконец, нам необходимо, чтобы программа открыла в браузере отдельные вкладки для каждого из результатов поиска. Добавьте в конец программы следующий код.

```
#! python3
# searchpypi.py - открывает несколько результатов поиска

import requests, sys, webbrowser, bs4
-- Опущено --
# Открытие отдельной вкладки для каждого результата
linkElems = soup.select('.package-snippet')
numOpen = min(5, len(linkElems))
for i in range(numOpen):
    urlToOpen = 'https://pypi.org' + linkElems[i].get('href')
    print('Открытие ', urlToOpen)
    webbrowser.open(urlToOpen)
```

По умолчанию мы открываем с помощью модуля webbrowser новые вкладки для пяти результатов поиска. Однако пользователь мог выполнить поиск, дающий менее пяти результатов. Функция soup.select() возвращает список всех элементов, соответствующих селектору '.package-snippet', поэтому количество открываемых вкладок либо будет равно 5, либо будет определяться длиной указанного списка (в зависимости от того, что меньше).

Встроенная функция min() возвращает наименьшее из переданных ей целых или вещественных чисел. (Также существует встроенная функция max(), которая возвращает наибольший из аргументов.) Мы используем функцию min() для того, чтобы выяснить, содержит ли список менее пяти ссылок, и сохранить количество ссылок, подлежащих открытию, в переменной numOpen. После этого выполняется цикл for, в котором вызывается функция range(numOpen).

На каждой итерации цикла мы открываем новую вкладку в браузере с помощью функции webbrowser.open(). Следует учитывать, что атрибут href в возвращаемых элементах <a> не содержит начальную часть URL-адреса https://pypi.org, поэтому необходимо конкатенировать ее со строкой атрибута href.

Теперь вы сможете сразу открыть первые пять результатов поиска на сайте PyPI, выполненного, скажем, для поискового термина "boring stuff", введя в командной строке команду `searchpypi boring stuff`!

Идеи для создания похожих программ

Преимуществом описанного подхода к выполнению поиска является то, что он позволяет легко открывать ссылки в новых вкладках для последующего просмотра. Программа, автоматически открывающая сразу несколько ссылок, поможет вам сэкономить время, выполняя следующие операции:

- открытие страниц всех заинтересовавших вас товаров после поиска на сайте какого-либо интернет-магазина, например Amazon;
- открытие ссылок на все обзоры, посвященные одному и тому же продукту;
- открытие ссылок на фотографии после поиска на каком-либо фотосайте, таком как Flickr или Imgur.

Проект: загрузка всех комиксов на сайте XKCD

В блогах и на других регулярно обновляемых сайтах обычно есть главная страница с последней публикацией и кнопка Previous (Предыдущая), которая позволяет просмотреть предыдущую публикацию. Предыдущий пост тоже снабжен кнопкой Previous, что дает возможность последовательно просматривать публикации на сайте от последней к первой. Если требуется скопировать содержимое сайта, чтобы прочесть его позднее в офлайн-режиме, то можно вручную пройтись по всем страницам и сохранить их. Но это довольно утомительное занятие. Давайте лучше напишем программу, которая все сделает за нас.

XKCD — это популярный веб-комикс (рис. 12.6), сайт которого вписывается в описанную схему. На главной странице сайта `https://xkcd.com/` есть кнопка Prev, щелкая на которой пользователь может переходить к предыдущим комиксам. Загрузка всех комиксов вручную длилась бы целую вечность, но можно написать сценарий, который выполнит эту работу за пару минут.

Вот что должна делать программа:

1) загружать главную страницу сайта XKCD;
2) сохранять изображение комикса, отображаемого на данной странице;
3) выполнять переход по ссылке Prev;
4) повторять описанные действия, пока не будет достигнут самый первый комикс.

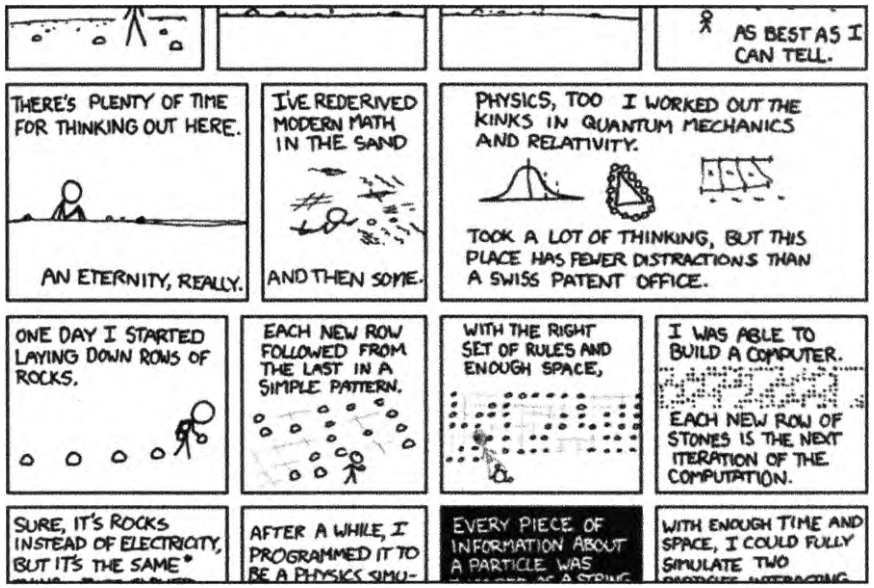


Рис. 12.6. XKCD — это "веб-комикс о романтике, сарказме, математике и языке"

Это означает, что программа должна выполнять следующие операции:

1) загружать веб-страницы с помощью модуля `requests`;
2) находить URL-адрес изображения комикса для веб-страницы, используя модуль Beautiful Soup;
3) загружать и сохранять изображение комикса на жестком диске, используя метод `iter_content()`.
4) находить URL-адрес ссылки Prev и повторять действия.

Откройте в файловом редакторе новое окно и сохраните программу в файле *downloadXkcd.py*.

Шаг 1. Проектирование программы

Открыв в браузере окно инструментов разработчика и проинспектировав элементы страницы, вы обнаружите следующее:

- URL-адрес файла с изображением комикса задается атрибутом `href` элемента `<img>`;
- элемент `<img>` вложен в элемент `<div id="comic">`;
- кнопка Prev имеет HTML-атрибут `rel` со значением `prev`;
- с кнопкой Prev первого комикса связан URL-адрес `https://xkcd.com/#`, указывающий на отсутствие других предыдущих страниц.

Введите в файл следующий код.

```
#! python3
# downloadXkcd.py - загружает все комиксы XKCD

import requests, os, bs4

url = 'https://xkcd.com'    # начальный URL-адрес
os.makedirs('xkcd', exist_ok=True)    # сохраняем комикс
                                      # в папке ./xkcd
while not url.endswith('#'):
    # СДЕЛАТЬ: загрузить страницу

    # СДЕЛАТЬ: найти URL-адрес изображения комикса

    # СДЕЛАТЬ: загрузить изображение

    # СДЕЛАТЬ: сохранить изображение в папке ./xkcd

    # СДЕЛАТЬ: получить URL-адрес кнопки Prev

print('Готово.')
```

У нас есть переменная `url` с начальным значением `'https://xkcd.com'`, которое будет регулярно обновляться (в цикле `for`) URL-адресом кнопки Prev текущей страницы. На каждом шаге цикла мы загружаем комикс, находящийся по адресу, который хранится в переменной `url`. Мы будем знать, что цикл следует завершить, когда встретится значение `url`, заканчивающееся символом `'#'`.

Файлы изображений будут загружаться в подпапку `xkcd` текущего каталога. Функция `os.makedirs()` гарантирует существование этой папки, а именованный аргумент `exist_ok=True` предотвращает появление исключения в том случае, если эта папка уже существует. Остальная часть кода пока что — комментарии, описывающие, что предстоит сделать.

Шаг 2. Загрузка веб-страницы

Приступим к реализации кода для загрузки страницы. Добавьте в программу код, выделенный полужирным шрифтом.

```
#! python3
# downloadXkcd.py - загружает все комиксы XKCD

import requests, os, bs4

url = 'http://xkcd.com'    # начальный URL-адрес
os.makedirs('xkcd', exist_ok=True)    # сохраняем комикс
                                      # в папке ./xkcd
while not url.endswith('#'):
    # Загрузка страницы
```

```
    print('Загружается страница %s...' % url)
    res = requests.get(url)
    res.raise_for_status()

    soup = bs4.BeautifulSoup(res.text, 'html.parser')

    # СДЕЛАТЬ: найти URL-адрес изображения комикса

    # СДЕЛАТЬ: загрузить изображение

    # СДЕЛАТЬ: сохранить изображение в папке ./xkcd

    # СДЕЛАТЬ: получить URL-адрес кнопки Prev

print('Готово.')
```

Прежде всего мы выводим значение `url`, информируя пользователя о том, по какому URL-адресу сейчас будет осуществляться загрузка. Для последующей загрузки изображения используется функция `request.get()`. Как всегда, если в процессе загрузки что-то пойдет не так, метод `raise_for_status()` объекта `Response` сгенерирует исключение и завершит работу программы. В противном случае из текста загруженной страницы будет создан объект `BeautifulSoup`.

Шаг 3. Поиск и загрузка изображения комикса

Добавьте в программу код, выделенный полужирным шрифтом.

```
#! python3
# downloadXkcd.py - загружает все комиксы XKCD

import requests, os, bs4

-- Опущено --

    # Поиск URL-адреса изображения комикса
    comicElem = soup.select('#comic img')
    if comicElem == []:
        print('Не удалось найти изображение комикса.')
    else:
        comicUrl = 'https:' + comicElem[0].get('src')
        # Загрузить изображение
        print('Загружается изображение %s...' % (comicUrl))
        res = requests.get(comicUrl)
        res.raise_for_status()

    # СДЕЛАТЬ: сохранить изображение в папке ./xkcd

    # СДЕЛАТЬ: получить URL-адрес кнопки Prev

print('Готово.')
```

Как показывают результаты инспектирования главной страницы сайта XKCD с помощью инструментов разработчика, элемент <img> для изображения комикса помещен в элемент , атрибут id которого равен comic. Поэтому для извлечения нужного элемента <img> из объекта Beautiful Soup следует использовать селектор '#comic img'.

Некоторые страницы сайта XKCD содержат специальный контент, не являющийся файлом изображения. Никаких сложностей это не представляет — мы просто пропускаем эти страницы. Если селектор не найдет вообще никаких элементов, то функция soup.select('#comic img') вернет пустой список. В таком случае программа выведет сообщение об ошибке и продолжит работу, пропустив загрузку изображения.

В противном случае селектор вернет список, содержащий один элемент <img>. Мы можем получить значение атрибута src этого элемента и передать его функции requests.get() для загрузки файла изображения комикса.

Шаг 4. Сохранение изображения и поиск предыдущего комикса

Добавьте в программу код, выделенный полужирным шрифтом.

```
#! python3
# downloadXkcd.py - загружает все комиксы XKCD

import requests, os, bs4

-- Опущено --

        # Сохранение изображения в папке ./xkcd
        imageFile = open(os.path.join('xkcd', \
                         os.path.basename(comicUrl)), 'wb')
        for chunk in res.iter_content(100000):
            imageFile.write(chunk)
        imageFile.close()

    # Получение URL-адреса кнопки Prev
    prevLink = soup.select('a[rel="prev"]')[0]
    url = 'https://xkcd.com' + prevLink.get('href')

print('Готово.')
```

К этому моменту файл изображения комикса хранится в переменной res. Нам нужно записать данные изображения в файл на жестком диске.

Функции open() необходимо передать имя локального файла изображения. Переменная comicUrl будет иметь значение наподобие 'https://imgs.xkcd.com/comics/heartbleed_explanation.png', которое, как вы, должно быть, заметили, во многом напоминает путь к файлу. Действительно, можно вызвать функцию os.path.basename(), передав ей значение comicUrl в качестве аргумента, и она вернет лишь последнюю часть URL-

адреса, 'heartbleed_explanation.png'. Эту строку можно использовать в качестве имени файла при сохранении изображения на жестком диске. Полученное имя можно объединить с именем папки xkcd с помощью функции os.path.join(), что позволяет использовать в строке пути символы обратной косой черты при работе в Windows и косой черты при работе в macOS и Linux. Получив имя файла, мы можем вызвать функцию open() для открытия нового файла в режиме 'wb' (запись бинарных данных).

Как уже говорилось, для сохранения загруженных файлов с помощью модуля requests следует организовать цикл по возвращаемому значению метода iter_content(). Содержащийся в цикле код записывает порции данных изображения (не более 100 000 байт в каждой порции) в файл, после чего мы закрываем файл. Теперь изображение сохранено на жестком диске.

Затем селектор 'a[rel="prev"]' находит элемент <a>, атрибут rel которого равен prev. Атрибут href этого элемента используется для получения URL-адреса предыдущего комикса, который сохраняется в переменной url. После этого цикл while повторяет весь процесс загрузки для данного комикса.

Результаты работы программы будут выглядеть примерно так.

```
Загружается страница https://xkcd.com...
Загружается изображение
 https://imgs.xkcd.com/comics/phone_alarm.png...
Загружается страница https://xkcd.com/1358/...
Загружается изображение https://imgs.xkcd.com/comics/nro.png...
Загружается страница https://xkcd.com/1357/...
Загружается изображение
 https://imgs.xkcd.com/comics/free_speech.png...
Загружается страница https://xkcd.com/1356/...
Загружается изображение
 https://imgs.xkcd.com/comics/orbital_mechanics.png...
Загружается страница https://xkcd.com/1355/...
Загружается изображение
 https://imgs.xkcd.com/comics/airplane_message.png...
Загружается страница https://xkcd.com/1354/...
Загружается изображение
 https://imgs.xkcd.com/comics/heartbleed_explanation.png...
-- Опущено --
```

Этот проект представляет собой пример программы, которая способна автоматически выполнять переходы по ссылкам для сбора больших объемов данных в Интернете. Чтобы узнать о других возможностях модуля Beautiful Soup, обратитесь к документации, которая доступна по следующему адресу:

https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Идеи для создания похожих программ

Загрузка веб-страниц и переходы по ссылкам — две ключевые операции в программах, собирающих данные в Интернете. Аналогичные программы могут также выполнять следующие задачи:

- резервное копирование всего сайта путем обхода всех его ссылок;
- копирование всех сообщений, опубликованных на форуме;
- дублирование каталога товаров интернет-магазина.

Модули `requests` и `bs4` оказываются очень полезными при условии, что вы знаете URL-адрес, который необходимо передать функции `requests.get()`. Но иногда определить нужный адрес не так-то просто. Кроме того, может оказаться, что сайт, на который вы хотите перейти в программе, требует предварительной регистрации. В таких случаях на помощь приходит модуль `selenium`.

Управление браузером с помощью модуля `selenium`

Модуль `selenium` позволяет непосредственно управлять браузером из кода Python путем программной имитации щелчков на ссылках и автоматической регистрации на сайтах, как если бы сам пользователь взаимодействовал с веб-страницей. Этот модуль предлагает гораздо более гибкие методы работы с сайтами, чем модули `requests` и `bs4`, но поскольку он запускает браузер, работа ведется немного медленнее, и ее сложнее выполнять в фоновом режиме, если вам, например, нужно всего лишь скачать несколько файлов из Интернета.

И тем не менее при взаимодействии с веб-страницей, на которой выполняется код JavaScript, обновляющий страницу, требуется использовать модуль `selenium`, а не `requests`. Это связано с тем, что на ведущих торговых сайтах, таких как Amazon, почти всегда имеются программные системы для выявления трафика от скриптов, собирающих информацию с сайта или регистрирующихся в нескольких бесплатных аккаунтах. Такие сайты спустя какое-то время могут начать блокировать ваши скрипты, и у модуля `selenium` намного больше перспектив продолжить работу в подобных условиях, чем у модуля `requests`.

Для сайта основной "подсказкой" о том, что вы используете скрипт, служит строка `user-agent`, которая идентифицирует веб-браузер и включается во все HTTP-запросы. Например, строка `user-agent` для модуля `requests` выглядит примерно так: `'python-reports/2.21.0'`. Есть специальные сайты, такие как `https://www.whatsmyua.info/`, где можно узнать содержимое строки `user-agent`. Используя модуль `selenium`, вы с большей вероятностью "сойдете за своего". Во-первых, строка `user-agent` для Selenium такая

же, как и у обычного браузера (например, 'Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:65.0).) Gecko/20100101 Firefox/65.0'). А во-вторых, модуль сохраняет привычную структуру трафика: браузер, работающий под управлением Selenium, будет загружать изображения, поддерживать рекламу, куки-файлы и трекеры. Впрочем, Selenium выявляется некоторыми крупными сайтами, занимающимися продажей билетов и электронной коммерцией. Они часто блокируют браузеры, управляемые модулем selenium, для предотвращения сбора информации с их веб-страниц.

Запуск браузера под управлением Selenium

В примерах этого раздела будет показано, как управлять браузером Firefox, который бесплатно доступен на сайте https://getfirefox.com/. Чтобы установить модуль selenium, выполните в командной строке команду pip install --user selenium (дополнительные сведения об установке сторонних модулей приведены в приложении А).

Импорт модулей для Selenium происходит не совсем стандартным способом. Вместо инструкции import selenium необходимо применять инструкцию from selenium import webdriver (объяснение причин, почему это так, выходит за рамки книги). Теперь вы сможете запускать браузер Firefox с помощью Selenium. Введите в интерактивной оболочке следующие инструкции.

```
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> type(browser)
<class 'selenium.webdriver.firefox.webdriver.WebDriver'>
>>> browser.get('https://inventwithpython.com')
```

Вы увидите, что вызов webdriver.Firefox() приводит к запуску браузера Firefox. Передав возвращаемое значение функции type(), мы видим, что тип данных этого значения — WebDriver. Последующий вызов browser.get('https://inventwithpython.com') перенаправляет браузер на сайт https://inventwithpython.com/. Окно браузера должно выглядеть примерно так, как показано на рис. 12.7.

Если появится сообщение об ошибке 'geckodriver executable needs to be in PATH', значит, необходимо вручную загрузить веб-драйвер для Firefox, прежде чем использовать модуль selenium для управления этим браузером. Поддерживаются и другие браузеры, если для них установлены веб-драйверы.

В случае Firefox перейдите по адресу https://github.com/mozilla/geckodriver/releases и скачайте драйвер Gecko для своей операционной системы (Gecko — браузерный движок Firefox). Например, в Windows это

будет файл *geckodriver-v0.29.0-win64.zip*, а в macOS — *geckodriver-v0.29.0-macos.tar.gz*. В ZIP-архиве содержится файл *geckodriver.exe* (Windows) или *geckodriver* (macOS и Linux), который нужно поместить в папку, прописанную в системной переменной `PATH`. Дополнительные сведения о переменной `PATH` приведены в приложении Б; также рекомендую прочитать обсуждение на сайте `https://stackoverflow.com/q/40208051/1893164`.

Рис. 12.7. Запуск браузера Firefox из Python

В случае Chrome перейдите по адресу `https://sites.google.com/a/chromium.org/chromedriver/downloads` и скачайте ZIP-файл для своей операционной системы. В ZIP-архиве будет содержаться файл *chromedriver.exe* (Windows) или *chromedriver* (macOS и Linux), который нужно поместить в папку, прописанную в системной переменной `PATH`.

Веб-драйверы доступны и для других популярных браузеров. Их можно найти в Интернете по запросу "*имя_браузера* веб-драйвер".

Если остаются проблемы с запуском браузера под управлением Selenium, это может быть связано с тем, что текущая версия браузера несовместима с модулем `selenium`. Одним из решений будет установка более старой версии браузера или, что намного проще, более старой версии модуля `selenium`. Список номеров версий `selenium` приведен по адресу `https://pypi.org/project/selenium/#history`. К сожалению, совместимость между версиями `selenium` и браузером иногда нарушается, и решение проблемы приходится искать в Интернете. В приложении А приведена дополнительная информация о том, как установить конкретную версию `selenium` с помощью

утилиты pip. (Например, может понадобиться выполнить команду pip install --user -U selenium == 3.14.1.)

Поиск элементов на веб-странице

У объектов WebDriver имеется достаточно большое количество методов, предназначенных для поиска элементов на веб-страницах. Эти методы условно делятся на две группы: find_element_*() и find_elements_*(). Методы первой группы возвращают одиночный объект WebElement, который представляет первый из найденных элементов, соответствующих запросу. Методы второй группы возвращают список объектов WebElement_*, в который входят *все* элементы, соответствующие запросу.

В табл. 12.3 перечислен ряд методов find_element_*() и find_elements_*() объекта WebDriver, сохраненного в переменной browser.

Таблица 12.3. Методы объекта WebDriver для поиска элементов

Имя метода	Возвращаемый объект (список объектов) WebElement
browser.find_element_by_class_name(*имя*) browser.find_elements_by_class_name(*имя*)	Элементы, использующие CSS-класс с указанным именем
browser.find_element_by_css_selector (*селектор*) browser.find_elements_by_css_selector (*селектор*)	Элементы, соответствующие указанному селектору CSS
browser.find_element_by_id(*id*) browser.find_elements_by_id(*id*)	Элементы с указанным идентификатором
browser.find_element_by_link_text(*текст*) browser.find_elements_by_link_text(*текст*)	Элементы <a>, полностью совпадающие с указанным текстом
browser.find_element_by_partial_link_ text(*текст*) browser.find_elements_by_partial_link_ text(*текст*)	Элементы <a>, содержащие указанный текст
browser.find_element_by_name(*имя*) browser.find_elements_by_name(*имя*)	Элементы, содержащие атрибут с указанным именем
browser.find_element_by_tag_name(*имя*) browser.find_elements_by_tag_name(*имя*)	Элементы с указанным именем тега (без учета регистра); например, тегу <a> будут соответствовать имена 'a' и 'A'

За исключением группы методов *_by_tag_name(), аргументы всех методов нечувствительны к регистру. Если на веб-странице нет искомых элементов, модуль selenium сгенерирует исключение NoSuchElement. Чтобы предотвратить аварийное завершение программы, используйте инструкции try и except.

Информацию о полученном объекте WebElement можно извлечь с помощью его атрибутов и методов (табл. 12.4).

Таблица 12.4. Атрибуты и методы объекта `WebElement`

Атрибут или метод	Описание
`tag_name`	Имя тега, например `'a'` в случае элемента `<a>`
`get_attribute(`*имя*`)`	Значение атрибута с указанным именем для данного элемента
`text`	Текст, содержащийся в элементе, например `'hello'` в случае элемента `<span>hello</span>`
`clear()`	В случае текстового поля или текстовой области удаляет введенный текст
`is_displayed()`	Возвращает `True`, если элемент видимый, в противном случае возвращается `False`
`is_enabled()`	Для элементов ввода возвращает `True`, если элемент активизирован, в противном случае возвращается `False`
`is_selected()`	Для флажков или переключателей возвращает `True`, если элемент выбран, в противном случае возвращается `False`
`location`	Словарь с ключами `'x'` и `'y'` для позиции элемента на веб-странице

Откройте в файловом редакторе новое окно и введите следующий код.

```
from selenium import webdriver
browser = webdriver.Firefox()
browser.get('https://inventwithpython.com')
try:
    elem = browser.find_element_by_class_name('bookcover')
    print('Найден элемент <%s> с данным именем класса!' %
          (elem.tag_name))
except:
    print('Не удалось найти элемент с данным именем класса.')
```

Программа запускает Firefox и направляет браузер по заданному URL-адресу. На открывшейся странице выполняется поиск элемента с классом `'bookcover'`, и если такой элемент обнаружен, то на экран выводится имя тега, определяемое атрибутом `tag_name`. В противном случае выводится другое сообщение.

Результат работы программы будет таким:

```
Найден элемент <img> с данным именем класса!
```

Мы нашли элемент с именем класса `'bookcover'` и именем тега `'img'`.

Щелчок на веб-странице

У объектов `WebElement`, возвращаемых методами `find_element_*()` и `find_elements_*()`, есть метод `click()`, имитирующий щелчок мыши на элементе. Этот метод можно использовать для перехода по ссылке, установки переключателя, щелчка на кнопке Submit (Отправить) или

инициирования любого другого действия, которое может быть запущено щелчком на элементе. Например, введите в интерактивной оболочке следующие инструкции.

```
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> browser.get('https://inventwithpython.com')
>>> linkElem = browser.find_element_by_link_text('Read for Free')
>>> type(linkElem)
<class 'selenium.webdriver.remote.webelement.WebElement'>
>>> linkElem.click()    # перейти по ссылке "Read for Free"
```

В данном случае мы направляем Firefox на сайт `https://inventwithpython.com/`, получаем объект `WebElement` для элемента `<a>` с текстом "Read for Free", а затем имитируем щелчок на этом элементе. Все происходит так, как если бы вы сами щелкнули на ссылке, заставляя браузер открыть соответствующую страницу.

Заполнение и отправка веб-форм

Отправка нажатий клавиш, когда фокус ввода находится в текстовом поле, сводится к нахождению на странице элемента `<input>` или `<textarea>`, соответствующего данному полю, и последующему вызову метода `send_keys()`. Например, введите в интерактивной оболочке следующие инструкции.

```
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> browser.get('https://login.metafilter.com')
>>> userElem = browser.find_element_by_id('user_name)
>>> userElem.send_keys('ваше_имя_пользователя')

>>> passwordElem = browser.find_element_by_id('user_pass')
>>> passwordElem.send_keys('ваш_пароль')
>>> passwordElem.submit()
```

Если на сайте MetaFilter не поменялись идентификаторы текстовых полей Username и Password, то данные инструкции заполнят эти поля предоставленным вами текстом. (Для проверки идентификатора всегда можно воспользоваться инспектором браузера.) Вызов метода `submit()` для любого элемента будет иметь тот же результат, что и щелчок на кнопке Submit для формы, в которой находится элемент. (Можно было бы вызвать метод `userElem.submit()`, и результат был бы тем же.)

Предупреждение

По возможности избегайте указания паролей в исходном коде. Если программа хранится на жестком диске в незашифрованном виде, злоумышленники могут легко получить доступ к паролям. Программа должна всегда предлагать пользователю ввести пароль с клавиатуры; для этого предназначена функция `pyinputplus.inputPassword()` *(см. главу 8).*

Отправка кодов специальных клавиш

В модуле `selenium` содержится модуль `selenium.webdriver.common.keys`, предназначенный для обработки нажатий клавиш, которые нельзя ввести в строковом виде. Поскольку имя модуля довольно длинное, проще выполнить в начале программы инструкцию `from selenium.webdriver.common.keys import Keys`, после чего можно будет использовать короткое имя `Keys`. В табл. 12.5 перечислены чаще всего применяемые переменные модуля `Keys`.

Таблица 12.5. Часто используемые переменные модуля `selenium.webdriver.common.keys`

Переменная	Описание
`Keys.DOWN`, `Keys.UP`, `Keys.LEFT`, `Keys.RIGHT`	Клавиши со стрелками
`Keys.ENTER`, `Keys.RETURN`	Клавиши <Enter> и <Return>
`Keys.HOME`, `Keys.END`, `Keys.PAGE_DOWN`, `Keys.PAGE_UP`	Клавиши <Home>, <End>, <PageDown> и <PageUp>
`Keys.ESCAPE`, `Keys.BACK_SPACE`, `Keys.DELETE`	Клавиши <Esc>, <Backspace> и <Delete>
`Keys.F1`, `Keys.F2`, ..., `Keys.F12`	Клавиши от <F1> до <F12>
`Keys.TAB`	Клавиша <Tab>

Например, если курсор в данный момент не находится в текстовом поле, то нажатие клавиш <Home> и <End> приводит к прокрутке веб-страницы в начало или в конец соответственно. Введите в интерактивной оболочке следующие инструкции и обратите внимание на то, как вызовы метода `send_keys()` приводят к прокрутке страницы.

```
>>> from selenium import webdriver
>>> from selenium.webdriver.common.keys import Keys
>>> browser = webdriver.Firefox()
>>> browser.get('https://nostarch.com')
>>> htmlElem = browser.find_element_by_tag_name('html')
>>> htmlElem.send_keys(Keys.END)     # прокрутка в конец
>>> htmlElem.send_keys(Keys.HOME)    # прокрутка в начало
```

Тег <html> — корневой в HTML-файлах: все содержимое HTML-файла заключено между тегами <html> и </html>. Вызов browser.find_element_by_tag_name('html') позволяет отправлять коды клавиш целой веб-странице. Это может оказаться полезным, если, например, новое содержимое загружается сразу же, как только вы прокрутили страницу до конца.

Щелчки на кнопках браузера

Модуль selenium также позволяет имитировать щелчки на различных кнопках браузера с помощью следующих методов:

- browser.back() — щелчок на кнопке Back (Назад);
- browser.forward() — щелчок на кнопке Forward (Вперед);
- browser.refresh() — щелчок на кнопке Refresh (Обновить)/Reload (Перезагрузить);
- browser.quit() — щелчок на кнопке Close Window (Закрыть окно).

Получение дополнительной информации о модуле selenium

Возможности модуля selenium гораздо шире, чем описано здесь. Он поддерживает изменение куки-файлов браузера, получение моментальных снимков веб-страниц и выполнение пользовательских скриптов JavaScript. Документация к модулю доступна на сайте https://selenium-python.readthedocs.org/.

Резюме

Рутинные задачи, которые приходится выполнять на компьютере, связаны не только с обработкой файлов. Другая полезная возможность — программная загрузка веб-страниц. Модуль requests упрощает процесс загрузки веб-контента, а с помощью модуля BeautifulSoup можно выполнять парсинг (синтаксический анализ) загруженных страниц, обладая лишь базовыми знаниями HTML-тегов и CSS-селекторов.

Но для полноценной автоматизации веб-операций требуется непосредственно управлять браузером, что реализуется с помощью модуля selenium, который поддерживает автоматическую регистрацию на сайтах и заполнение веб-форм. Этот модуль станет незаменимым инструментом в арсенале веб-разработчика.

Контрольные вопросы

1. Вкратце опишите различия между модулями webbrowser, requests, bs4 и selenium.

2. Объект какого типа возвращается функцией `requests.get()`? Как получить доступ к загруженному содержимому в виде строкового значения?
3. Какой метод модуля `requests` позволяет проверить успешность загрузки?
4. Как получить код состояния HTTP из ответа на запрос модуля `requests`?
5. Как сохранить в файле ответ на запрос модуля `requests`?
6. Какая комбинация клавиш предназначена для открытия окна инструментов веб-разработки в браузере?
7. Как в окне инструментов веб-разработки просмотреть HTML-код конкретного элемента на веб-странице?
8. Какая строка CSS-селектора ищет элемент, атрибут `id` которого равен `'main'`?
9. Какая строка CSS-селектора ищет элементы, относящиеся к CSS-классу `highlight`?
10. Какая строка CSS-селектора ищет все элементы `<div>`, вложенные в другой элемент `<div>`?
11. Какая строка CSS-селектора ищет элемент `<button>`, атрибут `value` которого равен `"favorite"`?
12. Предположим, для элемента `<div>Hello world!</div>` имеется объект `Tag` модуля Beautiful Soup, сохраненный в переменной `spam`. Как получить строку `'Hello world!'` из объекта `Tag`?
13. Как получить все атрибуты объекта `Tag` модуля Beautiful Soup, сохраненного в переменной `linkElem`?
14. Инструкция `import selenium` не работает. Как правильно импортировать модуль `selenium`?
15. В чем разница между методами `find_element_*()` и `find_elements_*()`?
16. Какие методы объекта `WebElement` модуля `selenium` имитируют щелчки мышью и нажатия клавиш?
17. Чтобы выполнить отправку веб-формы с помощью модуля `selenium`, можно вызвать метод `send_keys(Keys.ENTER)` для объекта `WebElement` кнопки Submit. Существует ли более простой способ сделать то же самое?
18. Как сымитировать щелчки на кнопках браузера Forward (Вперед), Back (Назад) и Refresh (Обновить) с помощью модуля `selenium`?

Учебные проекты

Чтобы закрепить полученные знания на практике, напишите программы для предложенных ниже задач.

Программа для отправки электронной почты из командной строки

Напишите программу, которая получает адрес электронной почты и строку текста в командной строке, а затем, используя модуль `selenium`, входит в вашу учетную запись электронной почты и отправляет строку сообщения по указанному адресу. (Возможно, для этой программы целесообразно завести отдельную учетную запись электронной почты.)

Это отличный способ дополнить свои программы средствами рассылки уведомлений. Можно написать аналогичную программу для отправки сообщений из учетных записей Facebook или Твиттера.

Загрузчик изображений из Интернета

Напишите программу, которая перенаправляет браузер на какой-либо фотосайт, например Flickr или Imgur, выполняет на этом сайте поиск фотографий определенной категории и загружает все найденные изображения. Можно написать программу, способную работать с любым фотосайтом, предоставляющим средства поиска.

2048

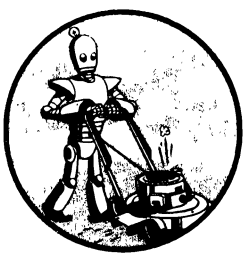
2048 — это простая браузерная игра, в которой игрок перемещает плитки с помощью клавиш управления курсором. Когда две плитки с одинаковыми цифрами соприкасаются, они сливаются в одну. Напишите программу, которая запускает игру на сайте `https://gabrielecirulli.github.io/2048/`, а затем отправляет коды клавиш управления курсором, соответствующих перемещениям вверх, вниз, вправо и влево, автоматически поддерживая процесс игры.

Верификация гиперссылок

Напишите программу, которая получает URL-адрес веб-страницы, а затем пытается загрузить каждую страницу, на которую там имеется ссылка. Программа должна помечать все страницы, для которых получен код состояния 404 “Страница не найдена”, и выводить на экран информацию о неработающих ссылках.

13

РАБОТА С ТАБЛИЦАМИ EXCEL



Мы не так часто думаем об электронных таблицах как о средствах программирования, но почти каждый из нас использует их для работы с двухмерными структурами данных, выполнения расчетов и вывода результатов в виде диаграмм. В следующих двух главах мы интегрируем Python с двумя приложениями электронных таблиц: Microsoft Excel и Google Таблицы.

Excel — популярная программа для работы с электронными таблицами в среде Windows. Модуль `openpyxl` позволяет приложениям Python читать и изменять файлы Excel. Например, можно скопировать требуемые данные из одной таблицы в другую или выбрать среди тысяч строк те, которые соответствуют определенному критерию. Такого рода рутинные задачи легко автоматизируются с помощью Python.

Excel — коммерческий продукт компании Microsoft, но есть и бесплатные альтернативы для Windows, macOS и Linux. Приложения LibreOffice Calc и OpenOffice Calc поддерживают используемый в Excel формат *.xlsx*, поэтому модуль `openpyxl` может работать с их электронными таблицами. Эти программы доступны для загрузки на сайтах `www.libreoffice.org` и `www.openoffice.org` соответственно. Тем не менее примеры данной главы сделаны в Excel 2010.

Документы Excel

Прежде всего, определимся с терминами. Документ Excel называется *рабочей книгой*. Она хранится в файле с расширением *.xlsx*. Каждая книга может содержать произвольное количество *рабочих листов*. Лист, просматриваемый пользователем в данный момент, называется *активным*.

Лист состоит из *столбцов* (адресуемых с помощью букв, начиная с A) и *строк* (адресуемых с помощью чисел, начиная с 1). Прямоугольная область, образуемая пересечением столбца и строки, называется *ячейкой*. Каждая ячейка таблицы может содержать числовое или текстовое значение. Совокупность ячеек вместе с содержащимися в них данными образует рабочий лист.

Установка модуля `openpyxl`

Модуль `openpyxl` не входит в состав Python, поэтому его нужно инсталлировать. Следуйте инструкциям по установке сторонних модулей, приведенным в приложении A.

В данной главе используется модуль версии 2.6.2. Установите именно эту версию, выполнив команду `pip install --user -U openpyxl == 2.6.2`, поскольку более новые версии модуля несовместимы с рассматриваемыми примерами. Чтобы проверить наличие модуля, введите в интерактивной оболочке следующую инструкцию:

```
>>> import openpyxl
```

Если модуль установлен корректно, то инструкция не выдаст никаких сообщений. Не забывайте импортировать модуль перед началом работы

с примерами в интерактивной оболочке, иначе вы получите сообщение `'NameError: name 'openpyxl' is not defined'`.

Документация по модулю `openpyxl` доступна на сайте `https://openpyxl.readthedocs.org/`.

Чтение документов Excel

В этой главе мы будем работать с электронной таблицей *example.xlsx*, находящейся в текущей папке. Можете либо самостоятельно создать этот файл, либо взять его из архива примеров книги (см. введение). На рис. 13.1 показаны вкладки трех стандартных листов, которые Excel автоматически добавляет во все создаваемые рабочие книги. (Количество создаваемых по умолчанию листов может различаться в зависимости от операционной системы и конкретного приложения электронных таблиц.)

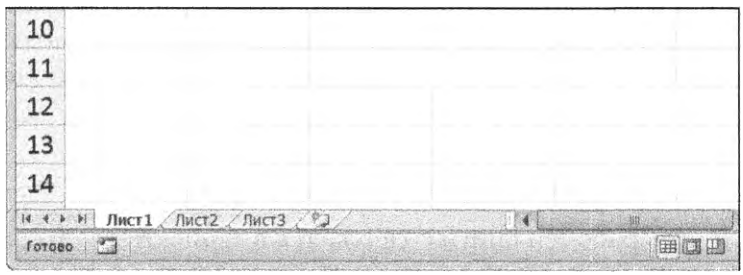


Рис. 13.1. Вкладки листов рабочей книги, расположенные в левом нижнем углу окна Excel

Содержимое листа `Лист1` приведено в табл. 13.1.

Таблица 13.1. Электронная таблица в файле *example.xlsx*

	A	B	C
1	05.04.2015 13:34	Яблоки	73
2	05.04.2015 3:41	Вишни	85
3	06.04.2015 12:46	Груши	14
4	08.04.2015 8:59	Апельсины	52
5	10.04.2015 2:07	Яблоки	152
6	10.04.2015 18:10	Бананы	23
7	10.04.2015 2:40	Клубника	98

Рассмотрим, как работать с такой таблицей с помощью модуля `openpyxl`.

Открытие документов Excel с помощью модуля openpyxl

Для открытия файла Excel следует вызвать функцию openpyxl.load_workbook(). Введите в интерактивной оболочке следующие инструкции.

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> type(wb)
<class 'openpyxl.workbook.workbook.Workbook'>
```

Функция openpyxl.load_workbook() получает имя файла в качестве аргумента и возвращает значение типа Workbook. Объект Workbook представляет файл Excel, подобно тому, как объект File представляет открытый текстовый файл.

Не забывайте: для того чтобы можно было работать с файлом *example.xlsx*, он должен находиться в текущем каталоге. Чтобы определить, какая именно папка является текущим каталогом, импортируйте модуль os и вызовите функцию os.getcwd(). Если потребуется сменить рабочий каталог, воспользуйтесь функцией os.chdir().

Получение списка листов рабочей книги

Список листов рабочей книги содержится в атрибуте sheetnames. Введите в интерактивной оболочке следующие инструкции.

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> wb.sheetnames    # имена листов книги
['Лист1', 'Лист2', 'Лист3']
>>> sheet = wb['Лист3']    # конкретный лист
>>> sheet
<Worksheet "Лист3">
>>> type(sheet)
<class 'openpyxl.worksheet.worksheet.Worksheet'>
>>> sheet.title    # заголовок листа
'Лист3'
>>> anotherSheet = wb.active    # активный лист
>>> anotherSheet
<Worksheet "Лист1">
```

Каждый лист представлен объектом Worksheet, который можно получить, указав в квадратных скобках имя листа, подобно ключу словаря. Также можно использовать атрибут active объекта Workbook для получения активного листа книги. Активный лист — это лист, находящийся в начале списка при открытии книги в Excel. Название листа содержится в атрибуте title объекта Worksheet.

Получение ячеек рабочих листов

Имея в своем распоряжении объект `Worksheet`, можно получать доступ к объектам `Cell` по имени. Введите в интерактивной оболочке следующие инструкции.

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb['Лист1']    # получение листа книги
>>> sheet['A1']    # получение ячейки листа
<Cell Лист1.A1>
>>> sheet['A1'].value     # получение значения ячейки
datetime.datetime(2015, 4, 5, 13, 34, 2)
>>> c = sheet['B1']    # получение другой ячейки
>>> c.value
'Яблоки'
>>> # Получение строки, столбца и значения из ячейки
>>> 'Строка %s, Столбец %s : %s' % (c.row, c.column, c.value)
'Строка 1, Столбец B : Яблоки'
>>> 'Ячейка %s : %s' % (c.coordinate, c.value)
'Ячейка B1 : Яблоки'
>>> sheet['C1'].value
73
```

У объекта `Cell` есть атрибут `value`, в котором хранится значение ячейки. Также имеются атрибуты `row`, `column` и `coordinate`, которые содержат информацию о расположении данной ячейки в таблице.

В данном случае при обращении к атрибуту `value` объекта `Cell` для ячейки B1 мы получаем строку `'Яблоки'`. Атрибут `row` содержит целочисленное значение `1`, атрибут `column` — значение `'B'`, а атрибут `coordinate` — значение `'B1'`.

Модуль OpenPyXL автоматически интерпретирует даты в столбце A и возвращает их в виде значений типа `datetime`, а не в виде строк. Более подробно о типе данных `datetime` рассказывается в главе 17.

Адресация столбца с помощью буквенных обозначений может вызывать определенные затруднения, поскольку после столбца Z обозначения становятся двухбуквенными: AA, AB, AC и т.д. В качестве альтернативы можно обращаться к ячейке, используя метод `cell()` объекта `Sheet`, передавая ему целочисленные значения именованных аргументов `row` и `column`. Первому столбцу или первой строке соответствует целое число `1`, а не `0`. Введите следующие инструкции.

```
>>> sheet.cell(row=1, column=2)
<Cell Лист1.B1>
>>> sheet.cell(row=1, column=2).value
'Яблоки'
>>> for i in range(1, 8, 2):    # проходим каждую вторую строку
```

```
...     print(i, sheet.cell(row=i, column=2).value)
...
1 Яблоки
3 Груши
5 Яблоки
7 Клубника
```

Как видите, вызывая метод `cell()` объекта `Sheet` с аргументами `row = 1` и `column = 2`, мы получаем объект `Cell` для ячейки B1. Это аналогично обращению `sheet['B1']`. Далее мы используем метод `cell()` в цикле `for` для вывода значений нескольких ячеек.

Предположим, мы хотим смещаться вниз по столбцу B и выводить значения, содержащиеся в ячейках с нечетными номерами строк. Передав значение `2` параметру `step` функции `range()`, мы получим ячейки из каждой второй строки (в данном случае — из каждой нечетной). Именованному аргументу `row` метода `cell()` передается счетчик цикла `for`, тогда как именованному аргументу `column` все время передается значение `2` (заметьте: не строка `'B'`).

Размер листа можно определить с помощью атрибутов `max_row` и `max_column` объекта `Worksheet`. Введите в интерактивной оболочке следующие инструкции.

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb['Лист1']
>>> sheet.max_row    # наибольший номер строки
7
>>> sheet.max_column    # наибольший номер столбца
3
```

Обратите внимание на то, что атрибут `max_column` содержит целое число, а не буквенное обозначение, которое появляется в Excel.

Преобразование буквенных и числовых обозначений столбцов

Чтобы преобразовать буквенное обозначение столбца в цифровое, следует вызвать функцию `openpyxl.cell.column_index_from_string()`. Если необходимо преобразовать цифровое обозначение столбца в буквенное, вызовите функцию `openpyxl.cell.get_column_letter()`. Введите в интерактивной оболочке следующие инструкции.

```
>>> import openpyxl
>>> from openpyxl.cell import get_column_letter,
                              column_index_from_string
>>> get_column_letter(1)
'A'
```

```
>>> get_column_letter(2)
'B'
>>> get_column_letter(27)
'AA'
>>> get_column_letter(900)
'AHP'
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb['Лист1']
>>> get_column_letter(sheet.max_column)
'C'
>>> column_index_from_string('A')
1
>>> column_index_from_string('AA')
27
```

Мы можем вызвать функцию `get_column_letter()` и передать ей целочисленное значение, например `27`, чтобы выяснить, какое буквенное обозначение соответствует столбцу с этим номером. Функция `column_index_string()` решает обратную задачу: вы передаете ей буквенное имя столбца, а она возвращает его номер. Для вызова этих функций не требуется загруженная рабочая книга. Но при желании можете загрузить файл Excel, получить объект `Worksheet` и использовать один из его атрибутов, например `max_column`, для получения целочисленного результата, а затем передать это число функции `get_column_letter()`.

Получение строк и столбцов рабочих листов

Используя срезы объектов `Worksheet`, можно получать все объекты `Cell`, принадлежащие к определенной строке, столбцу или прямоугольной области электронной таблицы. После этого можно организовать цикл по всем ячейкам среза. Введите в интерактивной оболочке следующие инструкции.

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb['Лист1']
>>> tuple(sheet['A1':'C3'])    # все ячейки от A1 до C3
((<Cell Лист1.A1>, <Cell Лист1.B1>, <Cell Лист1.C1>),
(<Cell Лист1.A2>, <Cell Лист1.B2>, <Cell Лист1.C2>),
(<Cell Лист1.A3>, <Cell Лист1.B3>, <Cell Лист1.C3>))
❶ >>> for rowOfCellObjects in sheet['A1':'C3']:
❷ ...     for cellObj in rowOfCellObjects:
...         print(cellObj.coordinate, cellObj.value)
...     print('--- КОНЕЦ СТРОКИ ---')

A1 2015-04-05 13:34:02
B1 Яблоки
C1 73
--- КОНЕЦ СТРОКИ ---
A2 2015-04-05 03:41:23
```

```
B2 Вишни
C2 85
--- КОНЕЦ СТРОКИ ---
A3 2015-04-06 12:46:51
B3 Груши
C3 14
--- КОНЕЦ СТРОКИ ---
```

Здесь мы указываем, что нас интересуют ячейки прямоугольной области таблицы, левый верхний и правый нижний углы которой определяются ячейками A1 и C3, и получаем объект `Generator`, который содержит объекты `Cell`, принадлежащие к указанной области. Чтобы было легче понять, что именно представляет собой данный объект `Generator`, можно воспользоваться функцией `tuple()` и отобразить ячейки в виде кортежа.

Данный кортеж сам состоит из трех кортежей: по одному для каждой строки интересующей нас области в порядке следования сверху вниз. Каждый из трех внутренних кортежей содержит объекты `Cell`, принадлежащие к одной строке, в порядке следования слева направо. Таким образом, срез листа содержит все ячейки прямоугольной области таблицы, углы которой определяются ячейками A1 и C3.

Для вывода значений всех ячеек данной области мы используем два цикла `for`. Внешний цикл перебирает все строки в срезе ❶, тогда как вложенный цикл перебирает все ячейки текущей строки ❷.

Для доступа к ячейкам конкретной строки или столбца можно также воспользоваться атрибутами `rows` и `columns` объекта `Worksheet`. Введите в интерактивной оболочке следующие инструкции.

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb.active
>>> sheet.columns[1]    # ячейки второго столбца
(<Cell Лист1.B1>, <Cell Лист1.B2>, <Cell Лист1.B3>,
 <Cell Лист1.B4>, <Cell Лист1.B5>, <Cell Лист1.B6>,
 <Cell Лист1.B7>)
>>> for cellObj in sheet.columns[1]:
        print(cellObj.value)

Яблоки
Вишни
Груши
Апельсины
Яблоки
Бананы
Клубника
```

В атрибуте `rows` объекта `Worksheet` хранится кортеж кортежей. Каждый из внутренних кортежей представляет собой строку электронной таблицы

и содержит ее ячейки в виде объектов `Cell`. Атрибут `columns` также хранит кортеж кортежей, причем каждый из внутренних кортежей содержит объекты `Cell`, принадлежащие к определенному столбцу. В случае таблицы *example.xlsx*, имеющей 7 строк и 3 столбца, атрибут `rows` содержит кортеж, состоящий из 7 вложенных кортежей (каждый из которых содержит по 3 объекта `Cell`), а атрибут `columns` содержит кортеж, состоящий из 3 вложенных кортежей (каждый из которых содержит по 7 объектов `Cell`).

Чтобы получить доступ к определенному кортежу, можно сослаться на него по индексу в охватывающем кортеже. Например, для получения кортежа, представляющего столбец B, следует использовать элемент `list(sheet.columns)[1]`. Чтобы получить кортеж, содержащий объекты `Cell` столбца A, следует использовать элемент `list(sheet.columns)[0]`. Как только в вашем распоряжении оказывается кортеж, представляющий строку или столбец таблицы, можно организовать цикл по содержащимся в нем объектам `Cell` и вывести их значения.

Рабочие книги, листы и ячейки

Подводя итог, опишем процесс чтения содержимого ячейки электронной таблицы.

1. Импортируйте модуль `openpyxl`.
2. Вызовите функцию `openpyxl.load_workbook()`.
3. Получите объект `Workbook`.
4. Используйте атрибуты `active` или `sheetnames` объекта `Workbook`.
5. Получите объект `Worksheet`.
6. Используйте индексирование или метод `cell()` объекта `Sheet`, передав ему именованные аргументы `row` и `column`.
7. Получите объект `Cell`.
8. Прочитайте значение атрибута `value` объекта `Cell`.

Проект: чтение данных электронной таблицы

Предположим, имеется электронная таблица, содержащая данные переписи населения США за 2010 год, и вам предстоит утомительный просмотр тысяч строк для определения численности населения и количества переписных районов по округам. (Переписной район — это географическая область, определенная для целей переписи населения.) Каждая строка таблицы представляет один переписной район. Наш файл электронной таблицы будет называться *censuspopdata.xlsx*, и его можно загрузить из архива примеров книги (см. введение). Вид таблицы показан на рис. 13.2.

	A	B	C	D	E
1	Переписной район	Штат	Округ	POP2010	
9841	06075010500	CA	San Francisco	2685	
9842	06075010600	CA	San Francisco	3894	
9843	06075010700	CA	San Francisco	5592	
9844	06075010800	CA	San Francisco	4578	
9845	06075010900	CA	San Francisco	4320	
9846	06075011000	CA	San Francisco	4827	
9847	06075011100	CA	San Francisco	5164	

Население по районам

Готово

Рис. 13.2. Электронная таблица censuspopdata.xlsx

Конечно, Excel без проблем рассчитает сумму выделенных ячеек, но вам необходимо самостоятельно выбирать ячейки для каждого из более чем 3000 округов. Даже если на расчет численности населения округа у вас уйдет лишь несколько секунд, обработка всей таблицы займет много часов.

В этом проекте мы напишем программу, которая будет считывать файл электронной таблицы с данными переписи населения и вычислять данные по всем округам за несколько секунд.

Вот что должна делать программа:

1) считывать данные из электронной таблицы Excel;
2) подсчитывать количество переписных районов в каждом округе;
3) подсчитывать численность населения, проживающего в каждом округе;
4) выводить результаты.

Это означает, что программа будет выполнять следующие операции:

1) открывать документ Excel и читать содержимое ячеек электронной таблицы с помощью модуля `openpyxl`;
2) собирать статистику, касающуюся количества переписных районов и численности населения, и сохранять ее в структуре данных;
3) записывать структуру данных в текстовый файл с расширением *.py* с помощью модуля `pprint`.

Шаг 1. Чтение электронной таблицы

В электронной таблице *censuspopdata.xlsx* имеется только один рабочий лист — `'Население по районам'`, в каждой строке которого хранятся данные, относящиеся к одному переписному району. Столбцами таблицы являются номер переписного района (A), сокращенное название штата (B), название округа (C) и численность населения в переписном районе (D).

Откройте в файловом редакторе новое окно, введите в нем следующий код и сохраните программу в файле *readCensusExcel.py*.

```
#! python3
# readCensusExcel.py - формирует таблицу с данными о численности
# населения и количестве переписных районов в каждом округе

❶ import openpyxl, pprint
print('Открытие рабочей книги...')
❷ wb = openpyxl.load_workbook('censuspopdata.xlsx')
❸ sheet = wb['Население по районам']
countyData = {}

# СДЕЛАТЬ: заполнить словарь countyData данными о численности
#          населения и переписных районах округов

print('Чтение строк...')
❹ for row in range(2, sheet.max_row + 1):
    # В каждой строке электронной таблицы содержатся
    # данные для одного переписного района
    state = sheet['B' + str(row)].value
    county = sheet['C' + str(row)].value
    pop = sheet['D' + str(row)].value

# СДЕЛАТЬ: открыть новый текстовый файл и записать
#          в него содержимое словаря countyData
```

Программа импортирует модуль `openpyxl`, а также модуль `pprint`, который применяется для вывода окончательных данных по округу ❶. Далее мы открываем файл *censuspopdata.xlsx* ❷, получаем лист с данными переписи ❸ и проходим по его строкам ❹.

Обратите внимание на создание переменной `countyData`, которая будет содержать данные по численности населения и количеству переписных районов, рассчитанные для каждого округа. Но прежде чем сохранять что-либо, необходимо понять, как должны быть структурированы данные в словаре.

Шаг 2. Заполнение структуры данных

В качестве структуры данных, сохраняемой в переменной `countyData`, мы выбираем словарь с сокращенными названиями штатов в качестве ключей. Каждому ключу штата будет соответствовать другой словарь, ключами которого служат названия округов данного штата. В свою очередь, каждому названию округа будет соответствовать словарь, содержащий всего два ключа: `'pop'` и `'tracts'`. Этим ключам соответствуют численность населения округа и количество переписных районов. Словарь будет выглядеть примерно так.

```
{'AK': {'Aleutians East': {'pop': 3141, 'tracts': 1},
        'Aleutians West': {'pop': 5561, 'tracts': 2},
        'Anchorage': {'pop': 291826, 'tracts': 55},
        'Bethel': {'pop': 17013, 'tracts': 3},
        'Bristol Bay': {'pop': 997, 'tracts': 1},
-- Опущено --
```

Если бы в переменной `countyData` был сохранен показанный выше словарь, то мы получили бы следующие результаты.

```
>>> countyData['AK']['Anchorage']['pop']
291826
>>> countyData['AK']['Anchorage']['tracts']
55
```

Ключи словаря `countyData` будут иметь следующий синтаксис.

```
countyData[аббревиатура_штата][округ]['tracts']
countyData[аббревиатура_штата][округ]['pop']
```

Теперь, когда вы знаете, как будут структурированы данные в переменной `countyData`, можно написать код, который заполняет эту структуру данными округа. Добавьте в конце программы код, выделенный полужирным шрифтом.

```
  #! python 3
  # readCensusExcel.py - формирует таблицу с данными о численности
  # населения и количестве переписных районов в каждом округе

-- Опущено --

  for row in range(2, sheet.max_row + 1):
      # В каждой строке электронной таблицы содержатся
      # данные для одного переписного района
      state  = sheet['B' + str(row)].value
      county = sheet['C' + str(row)].value
      pop    = sheet['D' + str(row)].value

      # Гарантируем наличие ключа для данного штата
❶     countyData.setdefault(state, {})
      # Гарантируем наличие ключа для данного округа штата
❷     countyData[state].setdefault(county, {'tracts': 0, 'pop': 0})

      # Каждая строка представляет один переписной район,
      # поэтому увеличиваем результат на единицу
❸     countyData[state][county]['tracts'] += 1
      # Увеличение численности населения округа на количество
      # жителей переписного района
```

```
❹     countyData[state][county]['pop'] += int(pop)

      # СДЕЛАТЬ: открыть новый текстовый файл и записать
      #          в него содержимое словаря countyData
```

Последние две инструкции выполняют фактические вычисления, инкрементируя значение ключа `tracts` ❸ и увеличивая значение ключа `pop` ❹ для текущего округа на каждой итерации цикла `for`.

Остальной код нужен из-за того, что мы не можем добавить словарь округа в качестве значения для ключа штата до тех пор, пока сам ключ штата не будет создан в переменной `countyData`. (Другими словами, инструкция `countyData['AK']['Anchorage']['tracts'] += 1` вызовет ошибку, если ключ `'AK'` еще не существует.) Чтобы гарантировать существование ключа штата в структуре данных, следует вызвать метод `setdefault()` и установить значение ключа, если для данного штата оно еще не существует ❶.

В свою очередь, каждый из словарей округов, вложенных в словарь штата, тоже должен быть инициализирован значениями по умолчанию ❷. В нем должны содержаться название округа в качестве ключа и вложенный словарь с ключами `'tracts'` и `'pop'`, значения которых начинаются с `0`. (Если запутаетесь, обратитесь к примеру словаря в начале раздела.)

Если ключ существует, метод `setdefault()` не будет выполнять никаких действий, поэтому его можно свободно вызывать на каждой итерации цикла `for`.

Шаг 3. Запись результатов в файл

Когда цикл `for` завершится, словарь `countyData` будет содержать всю информацию о численности населения и количестве переписных районов, структурированную с помощью ключей по округам и штатам. Можно было бы написать код для записи структурированной информации в текстовый файл или другую электронную таблицу Excel. Мы же ограничимся использованием функции `pprint.pformat()` для записи словаря `countyData` в виде одной большой строки в файл *census2010.py*. Добавьте в конец программы код, выделенный полужирным шрифтом (он должен находиться вне цикла, поэтому проследите за тем, чтобы он был введен без отступов).

```
#! python 3
# readCensusExcel.py - формирует таблицу с данными о численности
# населения и количестве переписных районов в каждом округе

-- Опущено --

for row in range(2, sheet.max_row() + 1):
-- Опущено --
```

```
# Открытие нового текстового файла и запись
# в него содержимого словаря countyData
print('Запись результатов...')
resultFile = open('census2010.py', 'w')
resultFile.write('allData = ' + pprint.pformat(countyData))
resultFile.close()
print('Готово.')
```

Функция `pprint.pformat()` создает строку, отформатированную в виде корректного кода на языке Python. Выводя ее в текстовый файл *census2010.py*, вы генерируете другую программу Python из своей собственной программы! Это может показаться ненужным усложнением, но зато вы сможете импортировать файл *census2010.py* подобно любому другому модулю Python. В интерактивной оболочке поменяйте текущий каталог на папку, в которой находится только что созданный файл *census2010.py*, и импортируйте его.

```
>>> import os

>>> os.chdir('C:\\Python37')
>>> import census2010
>>> census2010.allData['AK']['Anchorage']
{'pop': 291826, 'tracts': 55}
>>> anchoragePop = census2010.allData['AK']['Anchorage']['pop']
>>> print('Население округа Анкоридж в 2010 году - ' +
    str(anchoragePop))
Население округа Анкоридж в 2010 году - 291826
```

Программа *readCensusExcel.py* теперь больше не нужна: всякий раз, когда вам понадобятся данные, хранящиеся в файле *census2010.py*, вы сможете просто импортировать модуль `census2010`.

Расчет этих данных вручную занял бы у вас несколько часов, тогда как программа справилась с задачей за несколько секунд. Используя модуль OpenPyXL, вы легко сможете извлекать данные из электронных таблиц Excel и выполнять вычисления над ними. Готовый код программы содержится в архиве примеров книги (см. введение).

Идеи для создания похожих программ

Excel широко применяется во многих компаниях и организациях для хранения табличных данных, и нередко электронные таблицы разрастаются настолько, что становятся громоздкими и неудобными. Любая программа, работающая с данными в формате Excel, будет иметь похожую структуру: она должна загрузить файл электронной таблицы, подготовить соответствующие переменные или структуры данных, а затем организовать обработку строк таблицы в цикле. Такие программы могут использоваться для решения следующих задач:

- сравнение данных, хранящихся в нескольких строках электронной таблицы;
- открытие нескольких файлов Excel и сравнение данных, хранящихся в различных таблицах;
- поиск пустых строк или недопустимых данных в ячейках электронной таблицы и вывод предупреждающих сообщений в случае их обнаружения;
- чтение данных из электронной таблицы и их использование в качестве входных данных для программ на языке Python.

Запись документов Excel

Модуль `openpyxl` также позволяет записывать данные, а это означает, что в программе можно создавать и изменять файлы электронных таблиц. С помощью Python можно легко создать электронную таблицу, насчитывающую тысячи строк.

Создание и сохранение документов Excel

Для создания пустого объекта `Workbook` необходимо вызвать функцию `openpyxl.Workbook()`. Введите в интерактивной оболочке следующие инструкции.

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()     # создаем пустую рабочую книгу
>>> wb.sheetnames     # она содержит один лист
['Sheet']
>>> sheet = wb.active
>>> sheet.title
'Sheet'
>>> sheet.title = 'Spam Bacon Eggs Sheet'     # меняем название листа
>>> wb.sheetnames
['Spam Bacon Eggs Sheet']
```

Рабочая книга создается с одним рабочим листом `Sheet`, имя которого можно изменить, сохранив в атрибуте `title` новую строку.

Любые изменения объекта `Workbook` или его листов и ячеек не будут сохранены в файле электронной таблицы до тех пор, пока вы не вызовете метод `save()` для этого объекта. Введите в интерактивной оболочке следующие инструкции (предполагается, что файл *example.xlsx* находится в текущем каталоге).

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb.active
```

```
>>> sheet.title = 'Spam Spam Spam'
>>> wb.save('example_copy.xlsx')    # сохраняем рабочую книгу
```

Здесь мы переименовываем рабочий лист, после чего сохраняем это изменение, передавая методу `save()` строку с новым именем файла. Если задается другое имя файла, отличающееся от первоначального, например `'example_copy.xlsx'`, то будет сохранена копия электронной таблицы.

Всякий раз, когда вы вносите изменения в электронную таблицу, загруженную из файла, сохраняйте ее в файле, имя которого отличается от первоначального. Это будет гарантией того, что в случае записи некорректных данных из-за ошибок в программе вы всегда сможете вернуться к исходному файлу.

Создание и удаление рабочих листов

Для добавления и удаления листов рабочей книги предназначены методы `create_sheet()` и `remove_sheet()` соответственно. Введите в интерактивной оболочке следующие инструкции.

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> wb.sheetnames
['Sheet']
>>> wb.create_sheet()    # добавляем новый лист
<Worksheet "Sheet1">
>>> wb.sheetnames
['Sheet', 'Sheet1']
>>> wb.create_sheet(index=0, title='Первый лист')
<Worksheet "Первый лист">
>>> wb.sheetnames
['Первый лист', 'Sheet', 'Sheet1']
>>> wb.create_sheet(index=2, title='Средний лист')
<Worksheet "Средний лист">
>>> wb.sheetnames
['Первый лист', 'Sheet', 'Средний лист', 'Sheet1']
```

Метод `create_sheet()` возвращает новый объект `Worksheet` с именем `SheetX`, который по умолчанию становится последним листом книги. При желании можно с помощью именованных аргументов `index` и `title` задать не только имя, но и индекс создаваемого листа.

Продолжите предыдущий пример и введите следующие инструкции.

```
>>> wb.sheetnames
['Первый лист', 'Sheet', 'Средний лист', 'Sheet1']
>>> del wb['Средний лист']
>>> del wb['Sheet1']
>>> wb.sheetnames
['Первый лист', 'Sheet']
```

Для удаления листа из книги можно использовать инструкцию `del`, как и при удалении пар "ключ — значение" из словаря.

Не забывайте вызывать метод `save()` для сохранения изменений после добавления или удаления рабочих листов.

Запись значений в ячейки

Запись значений в ячейки во многом напоминает запись значений в ключи словаря. Введите в интерактивной оболочке следующие инструкции.

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb['Sheet']
>>> sheet['A1'] = 'Здравствуй, мир!' # редактируем содержимое ячейки
>>> sheet['A1'].value
'Здравствуй, мир!'
```

Если у вас имеются координаты ячейки, значение которой нужно изменить, в виде строки, используйте эту строку в качестве ключа словаря.

Проект: обновление электронной таблицы

В этом проекте мы напишем программу, которая обновляет ячейки электронной таблицы, содержащей данные об объемах продаж. Программа будет просматривать электронную таблицу, находить конкретные виды продукции и обновлять их цены. Электронная таблица *produceSales.xlsx*, которую мы будем использовать (рис. 13.3), содержится в архиве примеров книги (см. введение).

	A	B	C	D	E
1	НАИМЕНОВАНИЕ	ЦЕНА (за 1 кг)	ПРОДАНО (кг)	ВЫРУЧКА	
2	Картофель	0,86	21,6	18,58	
3	Бамия	2,26	38,6	87,24	
4	Бобы	2,69	32,8	88,23	
5	Арбузы	0,66	27,3	18,02	
6	Чеснок	1,19	4,9	5,83	
7	Пастернак	2,27	1,1	2,5	
8	Спаржа	2,49	37,9	94,37	
9	Авокадо	3,23	9,2	29,72	
10	Сельдерей	3,07	28,9	88,72	
11	Бамия	2,26	40	90,4	

Лист

Готово 100%

Рис. 13.3. Электронная таблица с данными об объемах продаж

Каждая строка представляет отдельную операцию продажи. Столбцами являются название проданного продукта (A), цена за килограмм (B), проданное количество в килограммах (C) и выручка от продажи (D). В столбце `'ВЫРУЧКА'` содержится формула `=ОКРУГЛ(B3*C3, 2)`, в соответствии с которой стоимость одного килограмма продукта умножается на количество проданного товара и результат округляется с точностью до сотых. Благодаря формуле значения ячеек в столбце `'ВЫРУЧКА'` будут автоматически обновляться при изменении значений ячеек в столбцах B и C.

А теперь представьте, что цены на чеснок, сельдерей и лимоны были введены неправильно, и вам предстоит утомительный просмотр тысяч строк таблицы для исправления данных о цене во всех строках, относящихся к данным продуктам. Воспользоваться операцией поиска и замены с использованием только значения цены нельзя, поскольку цена какого-то другого продукта может случайно оказаться такой же, и в результате будут внесены неправильные изменения. На выполнение этой работы вручную у вас уйдет много часов. Но можно написать программу, которая справится с этим за несколько секунд.

Программа должна делать следующее:

1) просматривать все строки в цикле;
2) изменять значения цены для чеснока, сельдерея и лимонов.

Это означает, что в коде нужно выполнять следующие операции:

1) открывать файл электронной таблицы;
2) проверять для каждой строки, не содержит ли столбец A значение `'Лимоны'`, `'Сельдерей'` или `'Чеснок'`;
3) в случае положительного результата проверки изменять цену в столбце B;
4) сохранять электронную таблицу в новом файле (в качестве страховки, чтобы не потерять таблицу с прежними значениями).

Шаг 1. Создание структуры, содержащей данные для обновления

Ниже приведены новые цены, которые должны быть внесены в электронную таблицу.

Лимоны	1,27
Сельдерей	1,19
Чеснок	3,07

Соответственно, можно было бы написать такой код.

```
if produceName == 'Лимоны':
    cellObj = 1.27
```

```
if produceName == 'Сельдерей':
    cellObj = 1.19
if produceName == 'Чеснок':
    cellObj = 3.07
```

Однако такой подход считается не самым удачным. Если придется вновь обновлять цены, причем для других продуктов, придется вносить в программу много изменений, и каждый раз это чревато появлением новых ошибок.

Более гибкий подход заключается в том, чтобы сохранить информацию об изменяемых ценах в виде словаря и написать код, использующий эту структуру данных. Откройте в файловом редакторе новое окно и введите следующий код.

```
#! python3
# updateProduce.py - корректирует цены в таблице продаж

import openpyxl

wb = openpyxl.load_workbook('produceSales.xlsx')
sheet = wb['Лист']

# Названия товаров и их обновленные цены
PRICE_UPDATES = {'Лимоны': 3.07,
                 'Сельдерей': 1.19,
                 'Чеснок': 1.27}

# СДЕЛАТЬ: создать цикл по строкам и обновить цены
```

Сохраните программу в файле *updateProduce.py*. Если вновь потребуется обновить электронную таблицу, достаточно будет внести изменения только в словарь `PRICE_UPDATES`, а остальной код останется прежним.

Шаг 2. Проверка всех строк и обновление некорректных цен

Далее в программе организуется цикл по всем строкам электронной таблицы. Добавьте в конец файла *updateProduce.py* код, выделенный полужирным шрифтом.

```
  #! python3
  # updateProduce.py - корректирует цены в таблице продаж

  -- Опущено --

  # Создание цикла по строкам и обновление цен
❶ for rowNum in range(2, sheet.max_row):    # пропуск первой строки
❷     produceName = sheet.cell(row=rowNum, column=1).value
❸         if produceName in PRICE_UPDATES:
```

```
            sheet.cell(row=rowNum, column=2).value = \
                PRICE_UPDATES[produceName]

❹ wb.save('updatedProduceSales.xlsx')
```

Цикл по строкам электронной таблицы начинается со строки 2, поскольку строка 1 — это заголовок ❶. Ячейка из столбца 1 (т.е. столбца A) сохраняется в переменной `produceName` ❷. Если ключ `produceName` имеется в словаре `PRICE_UPDATES` ❸, значит, это и есть строка, цена в которой подлежит исправлению. Правильное значение цены хранится в элементе словаря `PRICE_UPDATES[produceName]`.

Обратите внимание на то, насколько аккуратнее стал выглядеть код благодаря использованию словаря `PRICE_UPDATES`. Для обновления всех цен, нуждающихся в исправлении, потребовалась всего лишь одна инструкция `if`, а не три инструкции наподобие `if produceName == 'Чеснок':`. А поскольку вместо жестко закодированных названий и обновленных цен продуктов теперь используется словарь `PRICE_UPDATES`, при внесении последующих изменений в электронную таблицу достаточно будет модифицировать лишь словарь, а не основной код программы.

По завершении цикла мы сохраняем объект `Workbook` в новом файле *updatedProduceSales.xlsx* ❹. Тем самым мы избегаем затирания старого файла электронной таблицы, который может понадобиться нам, если из-за ошибок в программе внесенные в таблицу исправления оказались некорректными. Впоследствии, когда вы убедитесь в правильности обновленного варианта электронной таблицы, прежний файл можно будет удалить.

Готовый код программы содержится в архиве примеров книги (см. введение).

Идеи для создания похожих программ

Поскольку многим офисным сотрудникам приходится постоянно работать с электронными таблицами Excel, любая программа, способная автоматизировать процесс редактирования и записи Excel-документов, может принести реальную пользу. Вот несколько примеров задач, которые может понадобиться решать.

- Чтение данных из одной электронной таблицы и их запись в другую таблицу.
- Чтение данных с веб-сайтов, из текстовых файлов или буфера обмена и их запись в электронную таблицу.
- Автоматическое форматирование данных в электронной таблице. Например, программа может использовать регулярные выражения для

чтения телефонных номеров в различных форматах и приведения их к единому стандартному формату.

Настройка шрифтов ячеек

С помощью стилевого оформления определенных ячеек, строк или столбцов можно выделять важные области электронной таблицы. Например, в предыдущей электронной таблице программа может выделить полужирным шрифтом строки, содержащие данные о чесноке, сельдерее и лимонах. Или, например, вы захотите выделить курсивом все строки, в которых цена продукта превышает 5 долларов. Стилевое оформление большой электронной таблицы — очень трудоемкая задача, но программы Python справляются с этим за считанные секунды.

Для настройки шрифтов, используемых в ячейках, необходимо импортировать функцию `Font()` из модуля `openpyxl.styles`.

```
from openpyxl.styles import Font
```

Данная инструкция позволяет использовать короткий вызов `Font()` вместо более длинного `openpyxl.styles.Font()` (см. главу 2.)

Ниже приведен пример создания рабочей книги, в которой для шрифта ячейки A1 устанавливается курсивное начертание и размер шрифта 24 пункта. Введите в интерактивной оболочке следующие инструкции.

```
>>> import openpyxl
>>> from openpyxl.styles import Font, Style
>>> wb = openpyxl.Workbook()
>>> sheet = wb['Лист']
❶ >>> italic24Font = Font(size=24, italic=True)     # объект шрифта
❷ >>> sheet['A'].font = italic24Font     # применение шрифта к ячейке A1
>>> sheet['A1'] = 'Здравствуй мир!'
>>> wb.save('styles.xlsx')
```

В данном примере функция `Font(size=24, italic=True)` возвращает объект `Font`, который сохраняется в переменной `italic24Font` ❶. Именованные аргументы `size` и `italic` функции `Font()` позволяют сконфигурировать стилевые атрибуты объекта `Font` (размер шрифта и начертание). Когда в атрибут `sheet['A'].font` записывается значение `italic24Font` ❷, вся информация о шрифте применяется к ячейке A1.

Объекты Font

Атрибуты шрифта задаются путем передачи именованных аргументов функции `Font()` (табл. 13.2).

Таблица 13.2. Именованные аргументы объекта `Font`

Именованный аргумент	Тип данных	Описание
`name`	`String`	Название шрифта, например `'Calibri'` или `'Times New Roman'`
`size`	`Integer`	Размер шрифта
`bold`	`Boolean`	`True` для полужирного начертания
`italic`	`Boolean`	`True` для курсивного начертания

С помощью функции `Font()` можно создать объект `Font` и сохранить его в переменной, которую затем можно передать атрибуту `font` объекта `Cell`. Попробуйте поэкспериментировать с различными вариантами шрифтов.

```
>>> import openpyxl
>>> from openpyxl.styles import Font
>>> wb = openpyxl.Workbook()
>>> sheet = wb['Sheet']

>>> fontObj1 = Font(name='Times New Roman', bold=True)
>>> sheet['A1'].font = fontObj1
>>> sheet['A1'] = 'Bold Times New Roman'

>>> fontObj2 = Font(size=24, italic=True)
>>> sheet['B3'].font = fontObj2
>>> sheet['B3'] = '24 pt Italic'

>>> wb.save('styles.xlsx')
```

В данном случае мы сохраняем объект `Font` в переменной `fontObj1` и присваиваем значение этой переменной атрибуту `font` объекта `Cell` ячейки A1. Затем процесс повторяется с использованием другого объекта `Font` для задания стиля второй ячейки. В результате для ячеек A1 и B3 электронной таблицы будут установлены заданные нами стили шрифта, как показано на рис. 13.4.

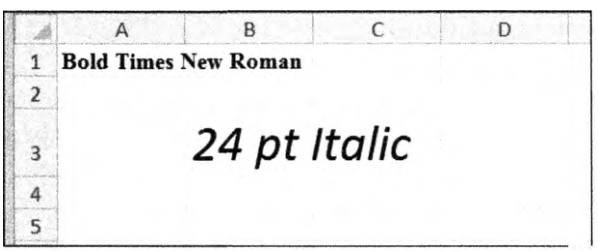


Рис. 13.4. Электронная таблица с модифицированными стилями шрифтов

В случае ячейки A1 мы устанавливаем для атрибута `name` значение `'Times New Roman'`, а для атрибута `bold` — значение `true`. В результате текст в ячейке отображается с использованием полужирного шрифта Times New Roman. Поскольку размер шрифта не указан, для него используется значение `11`, установленное модулем `openpyxl` по умолчанию. В случае ячейки B3 текст выводится с использованием курсивного начертания и с размером шрифта, равным `24`. Поскольку название шрифта не указано, используется шрифт Calibri, установленный модулем `openpyxl` по умолчанию.

Формулы

Формулы Excel, начинающиеся со знака равенства, позволяют устанавливать для ячеек значения, рассчитываемые на основе содержимого других ячеек. В этом разделе мы будем использовать модуль `openpyxl` для программного добавления формул в ячейки, например:

```
>>> sheet['B9'] = '=СУММ(B1:B8)'
```

Эта инструкция сохранит строку `'=СУММ(B1:B8)'` в качестве значения ячейки B9. Тем самым для ячейки B9 задается формула, которая суммирует значения, хранящиеся в ячейках B1–B8 (рис. 13.5).

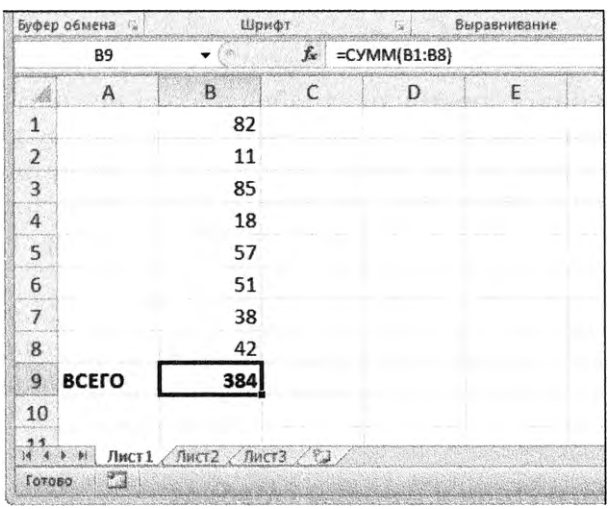


Рис. 13.5. В ячейке B9 содержится формула `'=СУММ(B1:B8)'`, которая суммирует содержимое ячеек B1–B8

Формула задается подобно любому другому текстовому значению в ячейке. Введите в интерактивной оболочке следующие инструкции.

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.active
>>> sheet['A1'] = 200
>>> sheet['A2'] = 300
>>> sheet['A3'] = '=SUM(A1:A2)'     # задаем формулу
>>> wb.save('writeFormula.xlsx')
```

Для ячеек A1 и A2 устанавливаются значения 200 и 300 соответственно. Значение в ячейке A3 определяется формулой, суммирующей значения ячеек A1 и A2. Если открыть эту электронную таблицу в Excel, то в качестве значения ячейки A3 отобразится 500.

Формулы Excel добавляют определенный уровень автоматизации в электронные таблицы, но в сложных задачах они быстро становятся слишком громоздкими. Например, даже для эксперта Excel не так-то легко понять смысл следующей формулы:

```
=ЕСЛИОШИБКА(СЖПРОБЕЛЫ(ЕСЛИ(ДЛСТР(ВПР(F7, Лист2!$A$1:$B$10000,
2, FALSE)) > 0, ПОДСТАВИТЬ(ВПР(F7, Лист2!$A$1:$B$10000, 2, FALSE),
" ", ""),"")), "")
```

Согласитесь, читать код Python гораздо легче.

Настройка строк и столбцов

Изменить размер строки или столбца в Excel не составляет никакого труда. Для этого достаточно перетащить мышью границу заголовка строки или столбца в нужную позицию. Но если необходимо установить размеры строк и столбцов в зависимости от содержимого ячеек или задать их размеры сразу в нескольких файлах, то проще написать программу на языке Python, которая сделает все за вас.

Кроме того, строки и столбцы можно скрывать из таблицы. Их также можно "заморозить", чтобы они всегда оставались на экране при прокрутке листа и появлялись на каждой странице при выводе таблицы на печать (это удобно в отношении заголовков).

Настройка высоты строк и ширины столбцов

У объекта `Worksheet` есть атрибуты `row_dimensions` и `column_dimensions`, которые управляют высотой строк и шириной столбцов. Введите в интерактивной оболочке следующие инструкции.

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.active
```

```
>>> sheet['A1'] = 'Высокая строка'
>>> sheet['B2'] = 'Широкий столбец'
>>> # Настройка высоты и ширины
>>> sheet.row_dimensions[1].height = 70
>>> sheet.column_dimensions['B'].width = 20
>>> wb.save('dimensions.xlsx')
```

Атрибуты `row_dimensions` и `column_dimensions` рабочего листа напоминают словари. Первый из них содержит объекты `RowDimension`, а второй — объекты `ColumnDimension`. Доступ к объектам в атрибуте `row_dimensions` осуществляется с использованием номера строки (в данном случае — `1`), а в атрибуте `column_dimensions` — с использованием буквы столбца (в данном случае — `B`).

Электронная таблица *dimensions.xlsx* показана на рис. 13.6.

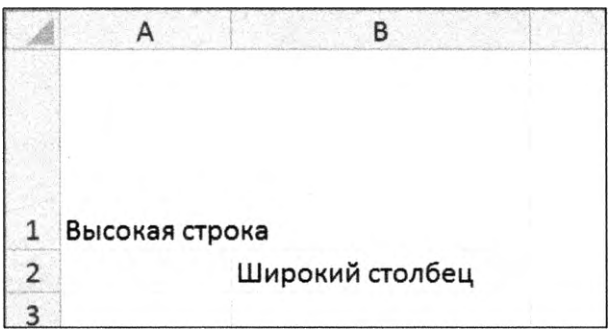


Рис. 13.6. Для строки 1 и столбца B установлены большие значения высоты и ширины

С помощью объекта `RowDimension` можно задать высоту строки, а с помощью объекта `ColumnDimension` — ширину столбца. Значение высоты строки может быть целочисленным или вещественным и должно находиться в диапазоне от `0` до `409`. Единицами измерения служат *пункты* (1 пункт равен 1/72 дюйма). По умолчанию высота строк устанавливается равной `12.75`. Значение ширины столбца может быть целочисленным или вещественным и должно находиться в диапазоне от `0` до `255`. Это значение определяет доступное количество символов в ячейке для заданного по умолчанию размера шрифта (11 пунктов). По умолчанию ширина столбца равна `8.43`. Столбцы с нулевой шириной и строки с нулевой высотой невидимы для пользователя.

Объединение и отмена объединения ячеек

Ячейки, занимающие прямоугольную область, могут быть объединены в одну ячейку с помощью метода `merge_cells()` рабочего листа. Введите в интерактивной оболочке следующие инструкции.

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.active
>>> sheet.merge_cells('A1:D3')    # объединение всей группы ячеек
>>> sheet['A1'] = 'Объединены двенадцать ячеек.'
>>> sheet.merge_cells('C5:E5')    # объединение трех ячеек
>>> sheet['C5'] = 'Объединены три ячейки.'
>>> wb.save('merged.xlsx')
```

Аргументом метода `merge_cells()` служит строка, задающая левую верхнюю и правую нижнюю ячейки прямоугольной области. Относящиеся к этой области ячейки будут объединены в одну: строке `'A1:D3'` соответствует блок из 12 ячеек. Чтобы задать значение для данной группы ячеек, достаточно установить значение для левой верхней ячейки.

Созданный файл *merged.xlsx* показан на рис. 13.7.

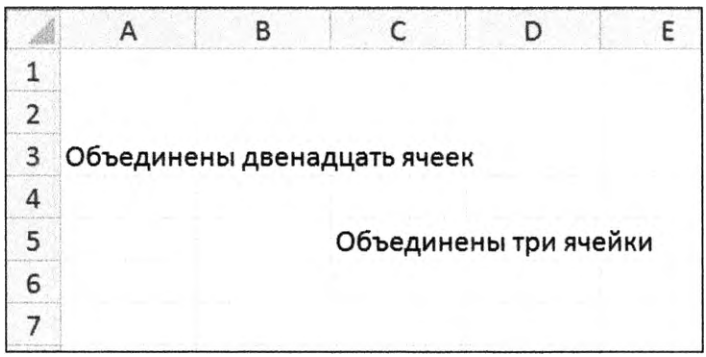


Рис. 13.7. Объединение ячеек в электронной таблице

Чтобы отменить объединение ячеек, необходимо вызвать метод `unmerge_cells()` рабочего листа. Введите в интерактивной оболочке следующие инструкции.

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('merged.xlsx')
>>> sheet = wb.active
>>> sheet.unmerge_cells('A1:D3')    # отмена объединения ячеек
>>> sheet.unmerge_cells('C5:E5')
>>> wb.save('merged.xlsx')
```

Закрепление областей

Если электронная таблица настолько большая, что ее нельзя увидеть целиком, можно заблокировать несколько верхних строк или крайних слева столбцов в их позициях на экране. В этом случае пользователь всегда будет видеть заблокированные заголовки столбцов или строк, даже если

прокручивает электронную таблицу на экране. Такие заблокированные в своих позициях ячейки называют *закрепленными областями*. В модуле `openpyxl` у каждого объекта `Worksheet` имеется атрибут `freeze_panes`, значением которого может быть объект `Cell` или строка с координатами ячейки. Все строки и столбцы, расположенные соответственно выше и левее, будут закреплены, однако строка и столбец, в которых расположена сама указанная ячейка, в их число не войдут.

Чтобы отменить закрепление областей, достаточно установить значение атрибута `freeze_panes` равным `None` или `'A1'`. В табл. 13.3 показано, какие строки и столбцы будут закреплены при тех или иных значениях атрибута `freeze_panes`.

Таблица 13.3. Примеры закрепления областей

Настройка атрибута `freeze_panes`	Закрепленные строки и столбцы
`sheet.freeze_panes = 'A2'`	Строка 1
`sheet.freeze_panes = 'B1'`	Столбец A
`sheet.freeze_panes = 'C1'`	Столбцы A и B
`sheet.freeze_panes = 'C2'`	Строка 1 и столбцы A и B
`sheet.freeze_panes = 'A1'` или `sheet.freeze_panes = None`	Закрепленные области отсутствуют

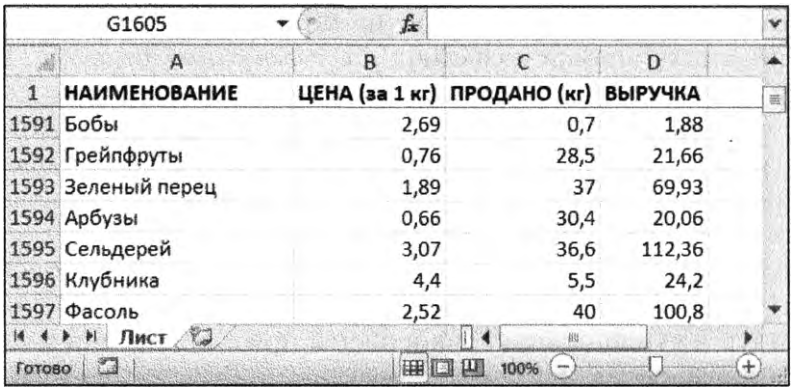
Убедившись в том, что в текущем каталоге находится рассмотренный ранее файл *produceSales.xlsx* с данными о продажах, и введите в интерактивной оболочке следующие инструкции.

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('produceSales.xlsx')
>>> sheet = wb.active
>>> sheet.freeze_panes = 'A2'    # закрепление строк над ячейкой A2
>>> wb.save('freezeExample.xlsx')
```

Если установить для атрибута `freeze_panes` значение `'A2'`, то строка 1 всегда будет оставаться видимой, независимо от прокрутки электронной таблицы (рис. 13.8).

Диаграммы

Модуль `openpyxl` поддерживает создание гистограмм, графиков, а также точечных и круговых диаграмм с использованием данных, хранящихся в электронной таблице. Чтобы создать диаграмму, необходимо выполнить следующие действия:



	A	B	C	D
1	НАИМЕНОВАНИЕ	ЦЕНА (за 1 кг)	ПРОДАНО (кг)	ВЫРУЧКА
1591	Бобы	2,69	0,7	1,88
1592	Грейпфруты	0,76	28,5	21,66
1593	Зеленый перец	1,89	37	69,93
1594	Арбузы	0,66	30,4	20,06
1595	Сельдерей	3,07	36,6	112,36
1596	Клубника	4,4	5,5	24,2
1597	Фасоль	2,52	40	100,8

Рис. 13.8. При установке атрибута `freeze_panes` равным `'A2'` строка 1 всегда будет оставаться видимой, независимо от прокрутки электронной таблицы на экране

1) создать объект `Reference` на основе ячеек в пределах выделенной прямоугольной области;
2) создать объект `Series`, передав ему объект `Reference`;
3) создать объект `Chart`;
4) присоединить объект `Series` к объекту `Chart`;
5) добавить объект `Chart` в объект `Worksheet`, указав координаты левого верхнего угла диаграммы (задавать координаты не обязательно).

Следует сказать несколько слов об объекте `Reference`. Такие объекты создаются путем вызова функции `openpyxl.charts.Reference()`, которой передаются три аргумента.

1. Объект `Worksheet`, содержащий данные диаграммы.
2. Кортеж, состоящий из двух целых чисел, которые представляют левую верхнюю ячейку выделенной прямоугольной области, где содержатся данные диаграммы: первое число задает строку, а второе – столбец. Учтите, что первой строке соответствует `1`, а не `0`.
3. Кортеж, состоящий из двух целых чисел, которые представляют правую нижнюю ячейку выделенной прямоугольной области, где содержатся данные диаграммы: первое число задает строку, а второе – столбец.

На рис. 13.9 приведено несколько примеров.

Введите в интерактивную оболочку следующие инструкции, чтобы создать гистограмму и добавить ее в электронную таблицу.

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.active
>>> for i in range(1, 11):     # создание данных в столбце A
```

```
...     sheet['A' + str(i)] = i
...
>>> refObj = openpyxl.chart.Reference(sheet, min_col=1,
                  min_row=1, max_col=1, max_row=10)
>>> seriesObj = openpyxl.chart.Series(refObj, title='Ряд 1')

>>> chartObj = openpyxl.chart.BarChart()
>>> chartObj.title = 'Моя диаграмма'
>>> chartObj.append(seriesObj)

>>> sheet.add_chart(chartObj, 'C5')
>>> wb.save('sampleChart.xlsx')
```

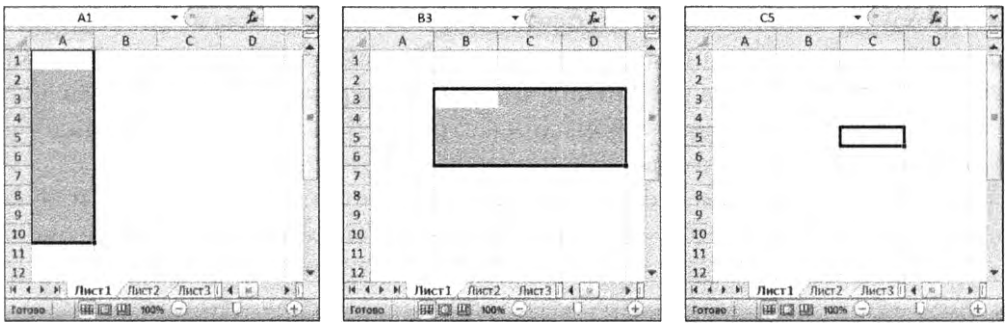


Рис. 13.9. Слева направо: (1, 1), (10, 1); (3, 2), (6, 4); (5, 3), (5, 3)

Созданная электронная таблица показана на рис. 13.10.

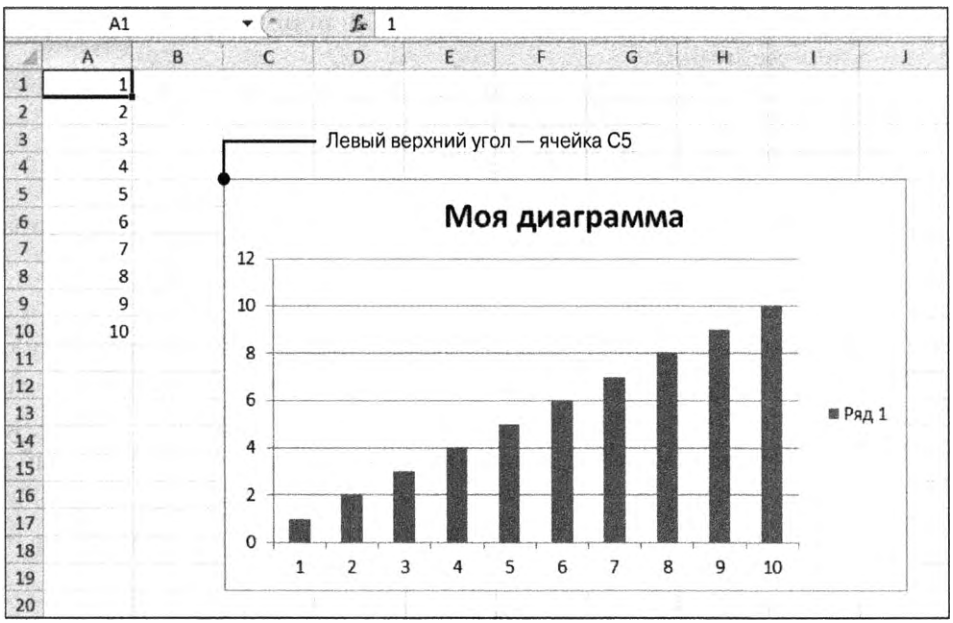


Рис. 13.10. Электронная таблица с добавленной диаграммой

Мы создали гистограмму с помощью метода `openpyxl.chart.BarChart()`. Аналогичным образом можно создавать графики, точечные и круговые диаграммы, вызывая методы `openpyxl.chart.LineChart()`, `openpyxl.chart.ScatterChart()` и `openpyxl.chart.PieChart()`.

Резюме

В процессе обработки информации зачастую самое сложное — это получить данные в подходящем формате. Но как только электронная таблица будет загружена в программу, вы сможете извлекать табличные данные и манипулировать ими гораздо быстрее, чем если бы делали это вручную.

Кроме того, электронные таблицы могут генерироваться программой в качестве выходных данных. Поэтому, если вашим коллегам понадобится, чтобы текстовый файл (или PDF-документ), содержащий данные о тысячах сделок, был преобразован в формат электронной таблицы, вам не придется тратить время на то, чтобы переносить данные вручную в Excel.

Теперь, когда вы имеете в своем распоряжении модуль `openpyxl` и обладаете определенными навыками программирования, обработка даже очень больших электронных таблиц не составит для вас никакого труда.

Контрольные вопросы

В ответах на контрольные вопросы предполагается, что объект `Workbook` хранится в переменной `wb`, объект `Worksheet` — в переменной `sheet`, объект `Cell` — в переменной `cell`, объект `Comment` — в переменной `comm` и объект `Image` — в переменной `img`.

1. Что возвращает функция `openpyxl.load_workbook()`?
2. Что содержит атрибут `wb.sheetnames` объекта `Workbook`?
3. Как получить объект `Worksheet` для рабочего листа `'Sheet1'`?
4. Как получить объект `Worksheet` для активного листа рабочей книги?
5. Как получить значение ячейки C5?
6. Как установить значение `"Hello"` для ячейки C5?
7. Как получить номера строки и столбца ячейки в виде целых чисел?
8. Что хранят атрибуты `max_column` и `max_row` рабочего листа и к какому типу данных относятся эти значения?
9. Какую функцию следует вызвать, чтобы получить целочисленный индекс столбца `'M'`?
10. Какую функцию следует вызвать, чтобы получить строковое имя столбца `14`?
11. Как получить кортеж всех объектов `Cell` для ячеек от A1 до F1?
12. Как сохранить рабочую книгу в файле *example.xlsx*?

13. Как задать формулу в ячейке?
14. Как задать высоту строки 5 равной `100`?
15. Как скрыть столбец C?
16. Что такое закрепленная область?
17. Какие пять функций и методов необходимо вызвать для создания гистограммы?

Учебные проекты

Чтобы закрепить полученные знания на практике, напишите программы для предложенных ниже задач.

Генератор таблиц умножения

Напишите программу *multiplicationTable.py*, которая получает число *N* из командной строки и создает таблицу умножения размером *N*×*N* в электронной таблице Excel. Например, если запустить программу с помощью команды `py multiplicationTable.py 6`, то она должна создать электронную таблицу, которая выглядит так, как показано на рис. 13.11.

	A	B	C	D	E	F	G
1		**1**	**2**	**3**	**4**	**5**	**6**
2	**1**	1	2	3	4	5	6
3	**2**	2	4	6	8	10	12
4	**3**	3	6	9	12	15	18
5	**4**	4	8	12	16	20	24
6	**5**	5	10	15	20	25	30
7	**6**	6	12	18	24	30	36

Рис. 13.11. Таблица умножения, сгенерированная в электронной таблице

В строке 1 и столбце A должны содержаться заголовки, отображаемые полужирным шрифтом.

Программа для вставки пустых строк

Создайте программу *blankRowInserter.py*, которая получает два целых числа и строку с именем файла в качестве аргументов командной строки. Обозначим первое число буквой *N*, второе — буквой *M*. Программа должна вставлять в электронную таблицу *M* пустых строк, начиная со строки *N*. Например, если вызвать программу с помощью команды `python blankRowInserter.py 3 2 myProduce.xlsx`, то электронная таблица должна быть преобразована так, как показано на рис. 13.12.

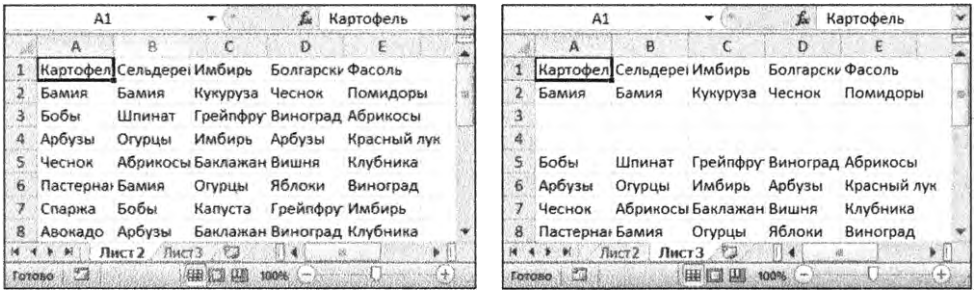


Рис. 13.12. Электронная таблица до (слева) и после (справа) вставки двух пустых строк в строке 3

В первую очередь программа должна прочитать содержимое электронной таблицы. Затем в процессе записи новой таблицы программа должна использовать цикл `for` для копирования первых *N* строк. Для каждой из оставшихся строк программа должна прибавлять *M* к номеру строки в результирующей электронной таблице.

Транспонирование электронной таблицы

Напишите программу, меняющую строки и столбцы электронной таблицы местами. Например, она должна переводить значение ячейки из столбца 3 строки 5 в столбец 5 строки 3 (и наоборот). Аналогичным образом должны быть транспонированы все ячейки электронной таблицы. Электронная таблица до и после транспонирования будет выглядеть так, как показано на рис. 13.13.

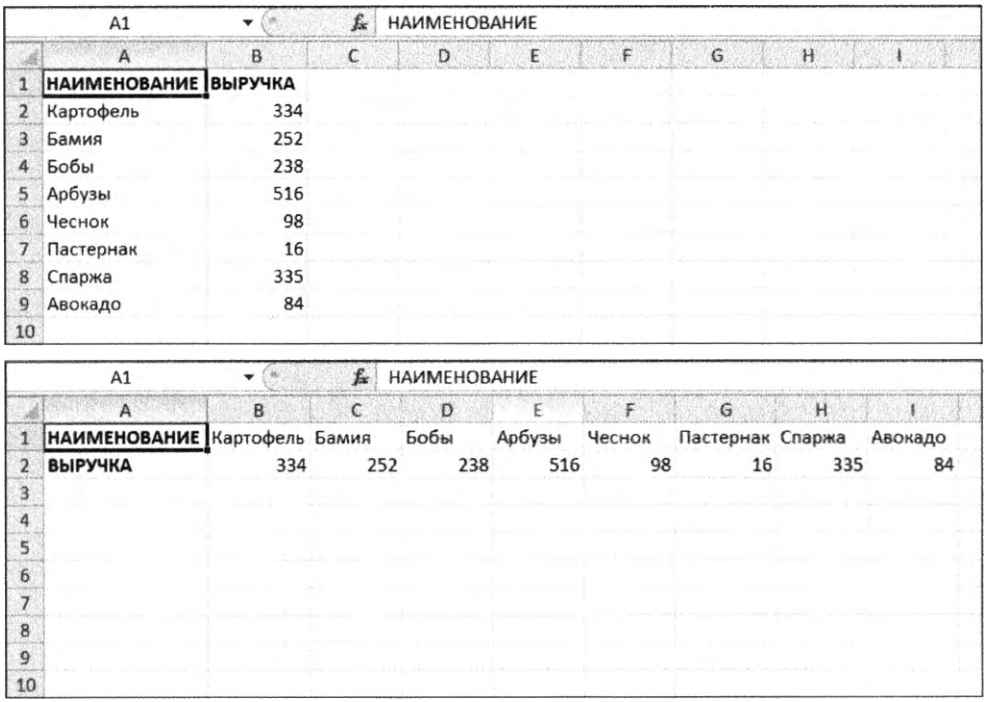
Используйте вложенные циклы `for` для переноса данных электронной таблицы в структуру типа список списков. В этой структуре ячейка из столбца `y` строки `x` будет храниться в элементе `sheetData[x][y]`. При записи данных в новую электронную таблицу поместите эту ячейку в элемент `sheetData[y][x]`.

Преобразование текстовых файлов в электронную таблицу

Напишите программу, которая будет читать содержимое нескольких текстовых файлов (можете подготовить их самостоятельно) и вставлять его в электронную таблицу по одной строке текста в одну строку таблицы. Строки первого текстового файла будут заполнять ячейки столбца A, второго – ячейки столбца B и т.д.

Используйте метод `readlines()` объекта `File` для получения списка строк файла. В случае первого файла первая текстовая строка должна помещаться в столбец 1 строки 1. Вторую текстовую строку следует записать в столбец 1 строки 2 и т.д. Содержимое следующего файла, прочитанное

с помощью метода `readlines()`, будет записываться в столбец 2, следующего — в столбец 3 и т.д.



A1 НАИМЕНОВАНИЕ

	A	B	C	D	E	F	G	H	I
1	НАИМЕНОВАНИЕ	ВЫРУЧКА							
2	Картофель	334							
3	Бамия	252							
4	Бобы	238							
5	Арбузы	516							
6	Чеснок	98							
7	Пастернак	16							
8	Спаржа	335							
9	Авокадо	84							
10									

A1 НАИМЕНОВАНИЕ

	A	B	C	D	E	F	G	H	I
1	НАИМЕНОВАНИЕ	Картофель	Бамия	Бобы	Арбузы	Чеснок	Пастернак	Спаржа	Авокадо
2	ВЫРУЧКА	334	252	238	516	98	16	335	84
3									
4									
5									
6									
7									
8									
9									
10									

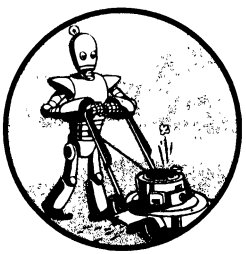
Рис. 13.13. Вид электронной таблицы до (вверху) и после (внизу) транспонирования

Преобразование электронной таблицы в текстовые файлы

Напишите программу, которая будет выполнять операции предыдущей программы в обратном порядке. Она должна открывать электронную таблицу и записывать содержимое ячеек столбца A в один текстовый файл, содержимое ячеек столбца B — в другой текстовый файл и т.д.

14

РАБОТА С ПРИЛОЖЕНИЕМ GOOGLE ТАБЛИЦЫ



Google Таблицы (Google Sheets) — бесплатное веб-приложение для работы с электронными таблицами, доступное любому, у кого есть аккаунт Google или Gmail. Это приложение обладает массой полезных функций и стало достойным конкурентом Excel. У приложения Google Таблицы есть собственный программный интерфейс, но он достаточно сложен для изучения. В этой главе мы рассмотрим сторонний модуль EZSheets, доступный по адресу `https://ezsheets.readthedocs.io/`. По сравнению с официальным API приложения Google Таблицы он не настолько функционален, зато он упрощает решение типовых задач, возникающих у пользователей электронных таблиц.

Установка и настройка модуля EZSheets

Чтобы установить модуль EZSheets, выполните в окне терминала команду `pip install --user ezsheets`. В процессе инсталляции будут также установлены модули `google-api-python-client`, `google-auth-httplib2` и `google-auth-oauthlib`, которые позволяют программам регистрироваться на серверах Google и делать API-запросы. Взаимодействие с этими модулями скрыто от пользователя, так что вам не придется разбираться в том, как они работают.

Получение файлов учетных данных и токенов

Перед использованием модуля EZSheets вы должны включить программные интерфейсы приложений Google Таблицы и Google Диск в своей учетной записи Google. Посетите следующие два сайта и щелкните на кнопке Enable в каждом из них:

- `https://console.developers.google.com/apis/library/sheets.googleapis.com/`
- `https://console.developers.google.com/apis/library/drive.googleapis.com/`

Также необходимо получить три файла, которые должны храниться в той же папке, что и сценарий Python, использующий модуль EZSheets:

- файл учетных данных *credentials-sheets.json*;
- токен для приложения Google Таблицы *token-sheets.pickle*;
- токен для хранилища Google Диск *token-drive.pickle*.

Файл учетных данных сгенерирует файлы токенов. Самый простой способ получить данный файл — перейти на страницу Google Sheets Python Quickstart, доступную по адресу `https://developers.google.com/sheets/api/quickstart/python/`, и щелкнуть на синей кнопке Enable the Google Sheets API (Включить API Google Таблицы), как показано на рис. 14.1. Чтобы получить доступ к этой странице, потребуется войти в свою учетную запись Google.

Щелкните на указанной кнопке, чтобы открыть окно со ссылкой Download Client Configuration (Загрузить клиентскую конфигурацию), которая позволит загрузить файл *credentials.json*. Переименуйте этот файл в *credentials-sheets.json* и поместите в ту же папку, в которой находятся сценарии Python.

После загрузки файла *credentials-sheets.json* выполните команду `import ezsheets`. При первом импорте модуля EZSheets появится окно браузера с приглашением войти в учетную запись Google. Щелкните на кнопке Allow (Разрешить).

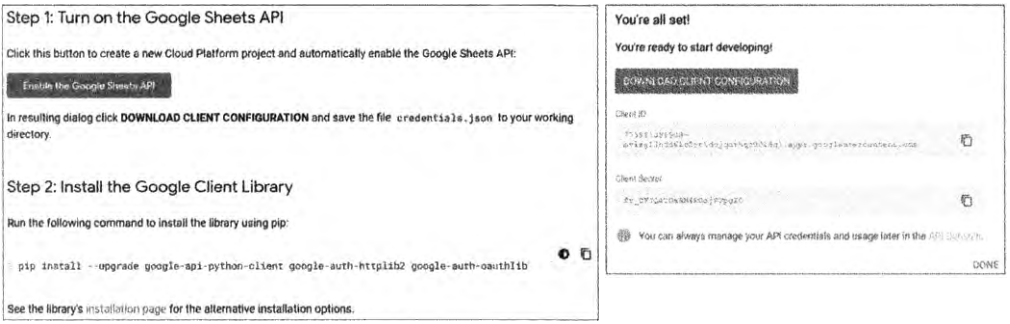


Рис. 14.1. Получение файла учетных данных

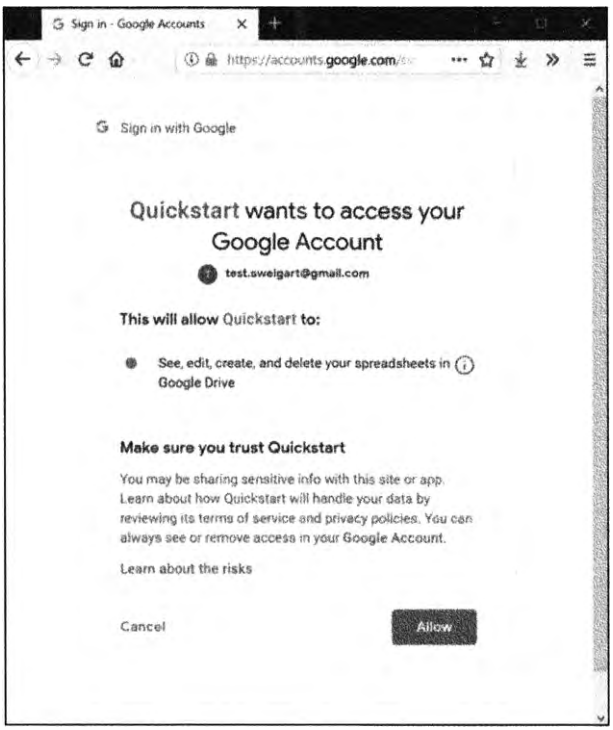


Рис. 14.2. Разрешение доступа к учетной записи Google

Упоминание службы Quickstart связано с тем, что вы загрузили файл учетных данных со страницы Google Sheets Python Quickstart. Учтите, что это окно откроется *дважды*: сначала для доступа к приложению Google Таблицы, а затем для доступа к Google Диск. Хранилище Google используется для загрузки, выгрузки и удаления электронных таблиц.

После входа в систему браузер предложит вам закрыть его окно, и файлы *token-sheet.pickle* и *token-drive.pickle* появятся в той же папке, что и файл

credentials-sheet.json. Этот процесс придется пройти лишь один раз — при первом выполнении инструкции `import ezsheets`.

Если после щелчка на кнопке Allow возникает ошибка и страница зависает, убедитесь, что вы включили программные интерфейсы приложений Google Таблицы и Google Диск, воспользовавшись ссылками в начале раздела. Серверам Google понадобится несколько минут, чтобы зарегистрировать изменение вашей учетной записи, поэтому, возможно, придется немного подождать, прежде чем использовать модуль EZSheets.

Никому не передавайте файлы учетных данных и токенов — относитесь к ним как к паролям.

Отзыв файла учетных данных

Если вы случайно передадите другим пользователям файлы учетных данных или токенов, они не смогут изменить пароль вашей учетной записи Google, зато получат доступ к вашим таблицам. Вы сможете отозвать эти файлы, перейдя на страницу консоли разработчика Google Cloud Platform по адресу `https://console.developers.google.com/`. Вам понадобится войти в свою учетную запись Google, чтобы просмотреть эту страницу. Щелкните на ссылке Учетные данные на боковой панели, а затем — на значке корзины рядом с файлом учетных данных, которым вы случайно поделились (рис. 14.3).

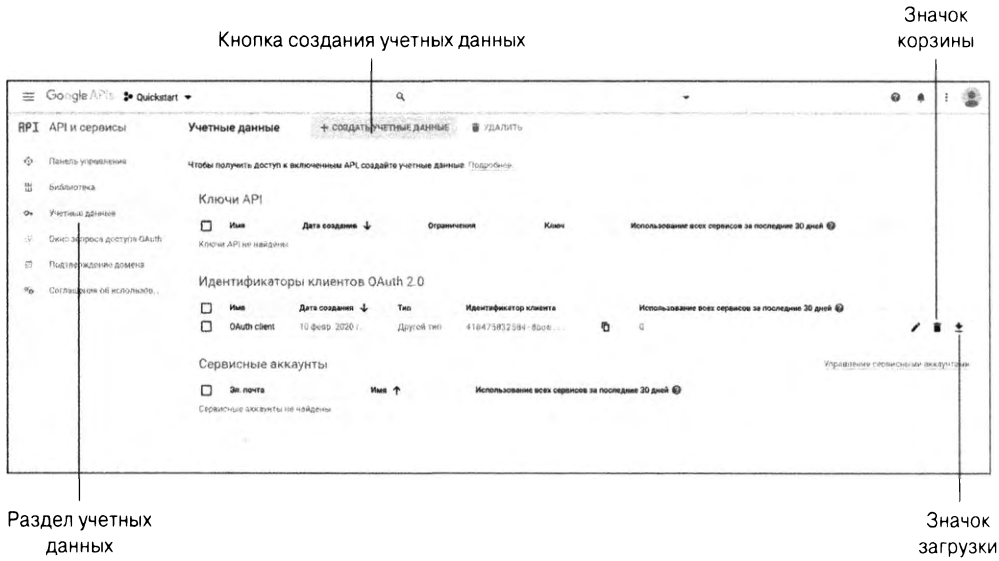


Рис. 14.3. Страница учетных данных консоли разработчика Google Cloud Platform

Чтобы сгенерировать новый файл учетных данных, щелкните на кнопке Создать учетные данные и выберите идентификатор клиента OAuth.

В качестве типа приложения выберите Другой и дайте файлу любое подходящее имя. На странице появится новый файл учетных данных; для его загрузки щелкните на соответствующем значке. Загруженный файл будет иметь длинное и сложное имя, поэтому присвойте ему имя по умолчанию, которое модуль EZSheets пытается загрузить: *credentials-sheet.json*. Можно также создать новый файл учетных данных, щелкнув на кнопке Enable the Google Sheets API, упомянутой в предыдущем разделе.

Объекты `Spreadsheet`

В приложении Google Таблицы электронная таблица может содержать несколько рабочих листов, напоминающих листы Excel. На рис. 14.4 показана электронная таблица "Education Data", содержащая три листа: "Студенты", "Классы" и "Ресурсы". Первый столбец каждого листа обозначен буквой A, а первая строка — 1.

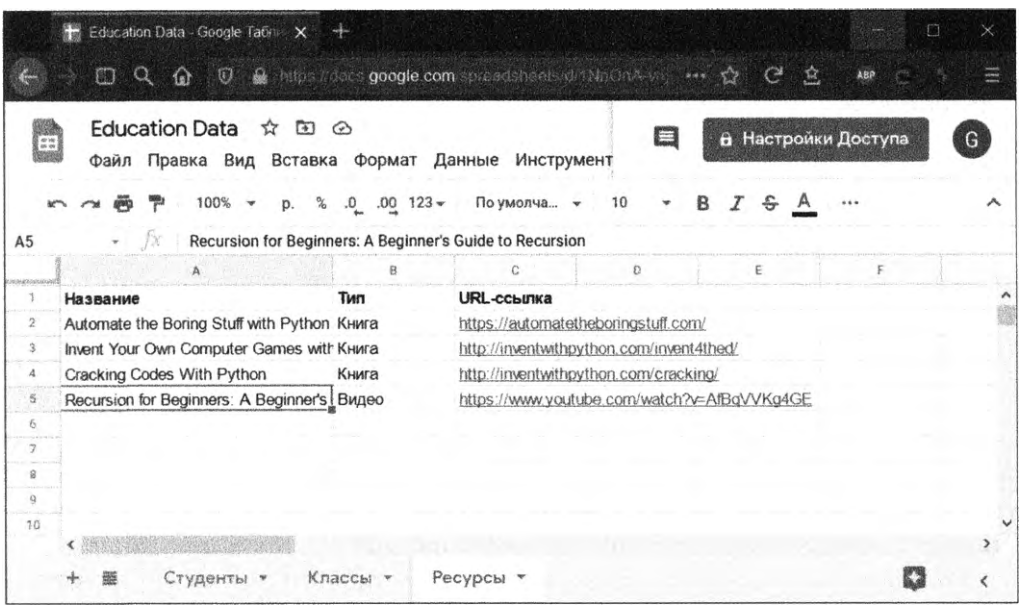


Рис. 14.4. Электронная таблица "Education Data" с тремя листами

В основном мы будем работать с объектами `Sheet`, но можно также изменять объекты `Spreadsheet`, как будет показано в следующем разделе.

Создание, выгрузка и отображение электронных таблиц

Можно создать новый объект `Spreadsheet` на основе существующей, пустой или загруженной электронной таблицы. Чтобы создать объект `Spreadsheet` на основе существующей таблицы приложения Google

Таблицы, необходимо знать ее идентификатор. Он содержится в URL-адресе после строки `spreadsheets/d/` и перед строкой `/edit`. Например, если электронная таблица, показанная на рис. 14.4, доступна по следующему адресу:

```
https://docs.google.com/spreadsheets/d/1J-Jx6Ne2K_vqI9J2SO-
TAXOFbxx_9tUjwnkPC22LjeU/edit#gid=151537240/
```

то ее идентификатор — `1J-Jx6Ne2K_vqI9J2SO-TAXOFbxx_9tUjwnkPC22LjeU`.

Примечание

Идентификаторы электронных таблиц, используемые в этой главе, предназначены для учетной записи автора книги. Они не будут работать, если вы введете их в своей интерактивной оболочке. Перейдите на сайт `https://sheets.google.com/`, *чтобы создать электронные таблицы в своей учетной записи, и вы увидите идентификаторы в строке адреса.*

Передайте идентификатор своей электронной таблицы в виде строки в функцию `ezsheets.Spreadsheet()`, чтобы получить объект `Spreadsheet`.

```
>>> import ezsheets
>>> ss =
ezsheets.Spreadsheet('1J-Jx6Ne2K_vqI9J2SO-TAXOFbxx_9tUjwnkPC22LjeU')
>>> ss
Spreadsheet(spreadsheetId='1J-Jx6Ne2K_vqI9J2SO-TAXOFbxx_
9tUjwnkPC22LjeU')
>>> ss.title
'Education Data'
```

Можно также получить объект `Spreadsheet` существующей электронной таблицы, передав в функцию ее полный URL-адрес. Или же, если в вашей учетной записи Google есть только одна электронная таблица с таким названием, можете передать в функцию это название.

Чтобы создать новую пустую электронную таблицу, вызовите функцию `ezsheets.createSpreadsheet()` и передайте ей название новой электронной таблицы. Например, введите в интерактивной оболочке следующие инструкции.

```
>>> import ezsheets
>>> ss = ezsheets.createSpreadsheet('Название новой таблицы')
>>> ss.title
'Название новой таблицы'
```

Чтобы загрузить существующую электронную таблицу формата Excel, OpenOffice, CSV или TSV в приложение Google Таблицы, передайте имя файла электронной таблицы функции `ezsheets.upload()`. Введите в интерактивной оболочке следующие инструкции, заменив *my_spreadsheet.xlsx* именем файла вашей электронной таблицы.

```
>>> import ezsheets
>>> ss = ezsheets.upload('my_spreadsheet.xlsx')
>>> ss.title
'my_spreadsheet'
```

Чтобы вывести список электронных таблиц, загруженных в учетную запись Google, воспользуйтесь функцией `listSpreadsheets()`.

```
>>> ezsheets.listSpreadsheets()
{'1J-Jx6Ne2K_vqI9J2SO-TAXOFbxx_9tUjwnkPC22LjeU': 'Education Data'}
```

Функция `listSpreadsheets()` возвращает словарь, ключами которого будут идентификаторы электронных таблиц, а значениями — названия этих таблиц.

После того как объект `Spreadsheet` получен, можно будет использовать его атрибуты и методы для работы с электронной таблицей, загруженной в приложение Google Таблицы.

Атрибуты объекта `Spreadsheet`

У объекта `Spreadsheet` есть несколько атрибутов, предназначенных для управления самой электронной таблицей: `title`, `spreadsheetId`, `url`, `sheetTitles` и `sheets`. Введите в интерактивной оболочке следующие инструкции.

```
>>> import ezsheets
>>> ss =
ezsheets.Spreadsheet('1J-Jx6Ne2K_vqI9J2SO-TAXOFbxx_9tUjwnkPC22LjeU')
>>> ss.title    # заголовок электронной таблицы
'Education Data'
>>> ss.title = 'Данные класса'    # изменение заголовка
>>> ss.spreadsheetId    # уникальный идентификатор (только для чтения)
'1J-Jx6Ne2K_vqI9J2SO-TAXOFbxx_9tUjwnkPC22LjeU'
>>> ss.url    # исходная URL-ссылка (только для чтения)
'https://docs.google.com/spreadsheets/d/1J-Jx6Ne2K_vqI9J2SO-
TAXOFbxx_9tUjwnkPC22LjeU'
>>> ss.sheetTitles    # заголовки всех объектов Sheet
('Студенты', 'Классы', 'Ресурсы')
>>> ss.sheets    # объекты Sheet в этой таблице
(<Sheet sheetId=0, title='Студенты', rowCount=1000, columnCount=26>,
 <Sheet sheetId=1669384683, title='Классы', rowCount=1000,
```

```
  columnCount=26>, <Sheet sheetId=151537240, title='Ресурсы',
  rowCount=1000, columnCount=26>)
>>> ss[0]     # первый объект Sheet в таблице
<Sheet sheetId=0, title='Студенты', rowCount=1000, columnCount=26>
>>> ss['Студенты']    # Доступ к листу по заголовку
<Sheet sheetId=0, title='Студенты', rowCount=1000, columnCount=26>
>>> del ss[0]    # Удаление первого объекта Sheet из таблицы
>>> ss.sheetTitles    # Лист 'Студенты' удален
('Классы', 'Ресурсы')
```

Если кто-то изменит электронную таблицу в приложении Google Таблицы, программа может обновить объект `Spreadsheet`, чтобы он соответствовал данным в Интернете. Для этого предназначен метод `refresh()`:

```
>>> ss.refresh()
```

Метод `refresh()` обновляется не только атрибуты объекта `Spreadsheet`, но и данные в объектах `Sheet`, которые он содержит. Изменения, внесенные в объект `Spreadsheet`, будут отражены в электронной таблице в режиме реального времени.

Загрузка и выгрузка электронных таблиц

Электронную таблицу можно загрузить из приложения Google Таблицы в различных форматах: Excel, OpenOffice, CSV, TSV и PDF. Можно также загрузить ее в виде ZIP-архива, содержащего HTML-файлы табличных данных. Для каждого формата в модуле EZSheets предусмотрена соответствующая функция.

```
>>> import ezsheets
>>> ss =
ezsheets.Spreadsheet('1J-Jx6Ne2K_vqI9J2SO-TAXOFbxx_9tUjwnkPC22LjeU')
>>> ss.title
'Данные о классе'
>>> ss.downloadAsExcel()    # загрузка таблицы в формате Excel
'Class_Data.xlsx'
>>> ss.downloadAsODS()    # загрузка таблицы в формате OpenOffice
'Class_Data.ods'
>>> ss.downloadAsCSV()    # загрузка первого листа в формате CSV
'Class_Data.csv'
>>> ss.downloadAsTSV()    # загрузка первого листа в формате TSV
'Class_Data.tsv'
>>> ss.downloadAsPDF()   # загрузка таблицы в формате PDF
'Class_Data.pdf'
>>> ss.downloadAsHTML()  # загрузка HTML-файлов в ZIP-архиве
'Class_Data.zip'
```

Учтите, что файлы в форматах CSV и TSV могут содержать лишь один лист, поэтому, если вы загрузите электронную таблицу Google Таблицы в таком формате, то получите только первый лист. Чтобы загрузить другие листы, укажите индекс листа.

Все функции загрузки возвращают строку с именем загруженного файла. Можно также указать собственное имя файла для электронной таблицы, передав его в функцию загрузки.

```
>>> ss.downloadAsExcel('a_different_filename.xlsx')
'a_different_filename.xlsx'
```

Функция вернет новое имя файла.

Удаление электронной таблицы

Для удаления электронной таблицы следует вызвать метод `delete()`.

```
>>> import ezsheets
>>> ss = ezsheets.createSpreadsheet('Удали меня') # создание таблицы
>>> ezsheets.listSpreadsheets()    # проверка создания
{'1aCw2NNJSZblDbhygVv77kPsL3djmgV5zJZllSOZ_mRk': 'Удали меня'}
>>> ss.delete()    # удаление таблицы
>>> ezsheets.listSpreadsheets()
{}
```

Метод `delete()` перемещает электронную таблицу в корзину Google Диск. Чтобы просмотреть содержимое корзины, перейдите по адресу `https://drive.google.com/drive/trash`. Если требуется окончательно удалить таблицу, передайте именованному аргументу `permanent` значение `True`:

```
>>> ss.delete(permanent=True)
```

В целом безвозвратное удаление электронной таблицы — не лучшая идея, поскольку вы не сможете восстановить таблицу, которая была случайно удалена вследствие ошибки в программе. Даже в бесплатных аккаунтах Google Диск доступно хранилище объемом порядка 15 Гбайт, поэтому вам, скорее всего, не придется беспокоиться об освобождении места на диске.

Объекты `Sheet`

Объект `Spreadsheet` может содержать один или несколько объектов `Sheet`, которые представляют строки и столбцы данных каждого листа. Можно получить доступ к листам, используя оператор `[]` и целочисленный

индекс. Атрибут sheets объекта Spreadsheet содержит кортеж объектов Sheet в порядке их отображения в электронной таблице. Введите в интерактивной оболочке следующие инструкции.

```
>>> import ezsheets
>>> ss =
ezsheets.Spreadsheet('1J-Jx6Ne2K_vqI9J2SO-TAXOFbxx_9tUjwnkPC22LjeU')
>>> ss.sheets    # объекты Sheet в таблице (упорядочены)
(<Sheet sheetId=1669384683, title='Классы', rowCount=1000,
        columnCount=26>, <Sheet sheetId=151537240, title='Ресурсы',
        rowCount=1000, columnCount=26>)
>>> ss.sheets[0]    # получение первого объекта Sheet
<Sheet sheetId=1669384683, title='Классы', rowCount=1000,
       columnCount=26>
>>> ss[0]    # получение первого объекта Sheet
<Sheet sheetId=1669384683, title='Классы', rowCount=1000,
       columnCount=26>
```

Можно также получить объект Sheet с помощью оператора [] и строки с именем листа. Атрибут sheetTitles объекта Spreadsheet содержит кортеж всех заголовков листов. Например, введите в интерактивной оболочке следующие инструкции.

```
>>> ss.sheetTitles    # заголовки всех объектов Sheet
('Классы', 'Ресурсы')
>>> ss['Классы']    # Доступ к листам возможен и по заголовку
<Sheet sheetId=1669384683, title= 'Классы', rowCount=1000,
       columnCount=26>
```

Чтение и запись данных

Как и в Excel, рабочие листы приложения Google Таблицы содержат ячейки с данными. С помощью оператора [] можно считывать данные из ячеек, а также записывать данные в эти ячейки. Например, чтобы создать новую электронную таблицу и добавить в нее данные, введите в интерактивной оболочке следующие инструкции.

```
>>> import ezsheets
>>> ss = ezsheets.createSpreadsheet('Моя таблица')
>>> sheet = ss[0]    # получить первый лист таблицы
>>> sheet.title
'Лист1'
>>> sheet = ss[0]
>>> sheet['A1'] = 'Имя'    # установить значение в ячейке A1
>>> sheet['B1'] = 'Возраст'
>>> sheet['C1'] = 'Любимый фильм'
>>> sheet['A1']    # чтение значения в ячейке A1
```

```
'Имя'
>>> sheet['A2']     # пустые ячейки возвращают пустую строку
''
>>> sheet[2, 1]     # столбец 2, строка 1 - тот же адрес, что и B1
'Возраст'
>>> sheet['A2'] = 'Алиса'
>>> sheet['B2'] = 30
>>> sheet['C2'] = 'Робокоп'
```

Эти инструкции создают электронную таблицу Google Таблицы, показанную на рис. 14.5.

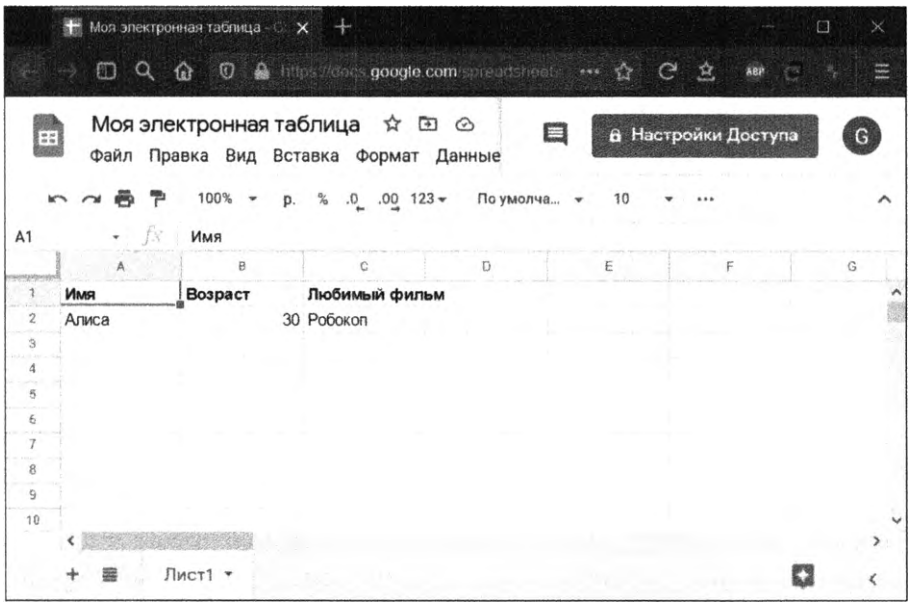


Рис. 14.5. Эта электронная таблица создана с помощью приведенных выше инструкций (выделение полужирным в строке заголовка добавлено вручную)

Несколько пользователей могут обновлять лист одновременно. Чтобы обновить локальные данные в объекте `Sheet`, вызовите его метод `refresh()`:

```
>>> sheet.refresh()
```

Все данные в объекте `Sheet` загружаются при первой загрузке объекта `Spreadsheet`, поэтому данные считываются мгновенно. Однако запись значений в онлайн-таблицу требует сетевого подключения и может занять около секунды. Если в таблице тысячи ячеек для обновления, их последовательное обновление может проходить довольно медленно.

Адресация строк и столбцов

Ячейки в приложении Google Таблицы адресуются так же, как и в Excel. В отличие от списков Python, где индексация ведется с 0, в Google Таблицы индексы столбцов и строк начинаются с 1: первый столбец или строка имеет индекс 1, а не 0. Преобразовать адрес в виде строки `'A2'` в адрес кортежа вида (*столбец*, *строка*) можно с помощью функции `convertAddress()`. Функции `getColumnLetterOf()` и `getColumnNumberOf()` также преобразуют адрес столбца из букв в цифры и наоборот. Введите в интерактивной оболочке следующие инструкции.

```
>>> import ezsheets
>>> ezsheets.convertAddress('A2')     # преобразование адреса
(1, 2)
>>> ezsheets.convertAddress(1, 2)     # обратное преобразование
'A2'
>>> ezsheets.getColumnLetterOf(2)
'B'
>>> ezsheets.getColumnNumberOf('B')
2
>>> ezsheets.getColumnLetterOf(999)
'ALK'
>>> ezsheets.getColumnNumberOf('ZZZ')
18278
```

Адреса вида `'A2'` удобны, если вводить их в исходный код. Адреса в виде кортежей (*столбец*, *строка*) удобны, если вы выполняете цикл с перебором диапазона адресов и нуждаетесь в числовом номере столбца. Функции `convertAddress()`, `getColumnLetterOf()` и `getColumnNumberOf()` позволяют быстро выполнить требуемое преобразование между двумя форматами.

Чтение и запись столбцов и строк целиком

Как уже упоминалось, последовательная запись ячеек по одной за раз может занимать слишком много времени. К счастью, в модуле EZSheets у объекта `Sheet` есть методы, предназначенные для чтения и записи столбцов и строк целиком: `getColumn()`, `getRow()`, `updateColumn()` и `updateRow()`. Эти методы отправляют запросы на серверы приложения Google Таблицы для обновления электронной таблицы, поэтому они требуют подключения к Интернету. В примере данного раздела мы загрузим в приложение Google Таблицы файл *produceSales.xlsx*, созданный в предыдущей главе. Первые восемь строк показаны в табл. 14.1.

Таблица 14.1. Первые восемь строк электронной таблицы *produceSales.xlsx*

	A	B	C	D
1	**НАИМЕНОВАНИЕ**	**ЦЕНА (за 1 кг)**	**ПРОДАНО (кг)**	**ВЫРУЧКА**
2	Картофель	0,86	21,6	18,58
3	Бамия	2,26	38,6	87,24
4	Бобы	2,69	32,8	88,23
5	Арбузы	0,66	27,3	18,02
6	Чеснок	1,19	4,9	5,83
7	Пастернак	2,27	1,1	2,5
8	Спаржа	2,49	37,9	94,37

Чтобы загрузить эту электронную таблицу, введите в интерактивной оболочке следующие инструкции.

```
>>> import ezsheets
>>> ss = ezsheets.upload('produceSales.xlsx')
>>> sheet = ss[0]
>>> sheet.getRow(1)    # первая строка - 1, а не 0
['НАИМЕНОВАНИЕ', 'ЦЕНА (за 1 кг)', 'ПРОДАНО (кг)', 'ВЫРУЧКА', '', '']
>>> sheet.getRow(2)
['Картофель', '0,86', '21,6', '18,58', '', '']
>>> columnOne = sheet.getColumn(1)
>>> sheet.getColumn(1)
['НАИМЕНОВАНИЕ', 'Картофель', 'Бамия', 'Бобы', 'Арбузы', 'Чеснок',
-- Опущено --
>>> sheet.getColumn('A')    # тот же результат, что и getColumn(1)
['НАИМЕНОВАНИЕ', 'Картофель', 'Бамия', 'Бобы', 'Арбузы', 'Чеснок',
-- Опущено --
>>> sheet.getRow(3)
['Бамия', '2.26', '38.6', '87.24', '', '']
>>> sheet.updateRow(3, ['Тыква', '11.50', '20', '230'])
>>> sheet.getRow(3)
['Тыква', '11.50', '20', '230', '', '']
>>> columnOne = sheet.getColumn(1)
>>> for i, value in enumerate(columnOne):
...     # Создание списка Python со строками в верхнем регистре
...     columnOne[i] = value.upper()
...
>>> sheet.updateColumn(1, columnOne)    # Обновление всего столбца
```

Функции `getRow()` и `getColumn()` извлекают данные из каждой ячейки в определенной строке или столбце в виде списка значений. Обратите внимание на то, что пустые ячейки становятся пустыми строками в списке. Можно передать методу `getColumn()` номер столбца или букву, чтобы он смог извлечь данные из определенного столбца. В предыдущем примере

было показано, что методы `getColumn(1)` и `getColumn('A')` возвращают один и тот же список.

Функции `updateRow()` и `updateColumn()` перезапишут все данные в строке или столбце соответственно, воспользовавшись полученным списком значений. В этом примере третья строка изначально содержит информацию о бамии, но вызов метода `updateRow()` заменяет ее данными о тыкве. Вызовите метод `sheet.getRow(3)` еще раз, чтобы просмотреть новые значения в третьей строке.

Далее мы обновляем электронную таблицу `produceSales`. Обновление ячеек по одной происходит медленно, если таких ячеек много. Намного быстрее получить столбец или строку в виде списка, обновить список, а затем обновить весь столбец или всю строку с помощью нового списка, поскольку все изменения могут быть внесены за один запрос.

Чтобы получить все строки одновременно, вызовите метод `getRows()`, который возвращает список списков. Внутренние списки представляют одну строку листа. Мы можем изменить значения в этой структуре данных, чтобы отредактировать наименование продукта, количество проданного товара и общую выручку в некоторых строках. Обновленный список необходимо передать методу `updateRows()`. Введите в интерактивной оболочке следующие инструкции.

```
>>> rows = sheet.getRows()     # получение всех строк таблицы
>>> rows[0]    # проверка значений в первой строке
['НАИМЕНОВАНИЕ', 'ЦЕНА (за 1 кг)', 'ПРОДАНО (кг)', 'ВЫРУЧКА', '', '']
>>> rows[1]
['КАРТОФЕЛЬ', '0,86', '21,6', '18,58', '', '']
>>> rows[1][0] = 'ТЫКВА'    # изменение названия продукта
>>> rows[1]
['ТЫКВА', '0,86', '21,6', '18,58', '', '']
>>> rows[10]
['Бамия', '2,26', '40', '90,4', '', '']
>>> rows[10][2] = '400'    # изменение количества проданных товаров
>>> rows[10][3] = '904'    # изменение выручки
>>> rows[10]
['Бамия', '2.26', '400', '904', '', '']
>>> sheet.updateRows(rows)    # обновление электронной таблицы
```

Можно обновить весь лист за один запрос, передав методу `updateRows()` список списков, полученный от метода `getRows()` и дополненный изменениями, внесенными в строки 1 и 10.

Обратите внимание на то, что строки в приложении Google Таблицы завершаются пустыми значениями. Это связано с тем, что в загруженном листе количество столбцов равно 6, а столбцов с данными всего 4. Можно определить количество строк и столбцов на листе с помощью атрибутов

rowCount и columnCount соответственно. Отредактировав эти значения, мы изменим размеры листа.

```
>>> sheet.rowCount     # количество строк на листе
23758
>>> sheet.columnCount     # количество столбцов на листе
6
>>> sheet.columnCount = 4     # изменить количество столбцов на 4
>>> sheet.columnCount     # теперь количество столбцов равно 4
4
```

Эти инструкции удаляют пятый и шестой столбцы таблицы produce Sales, как показано на рис. 14.6.

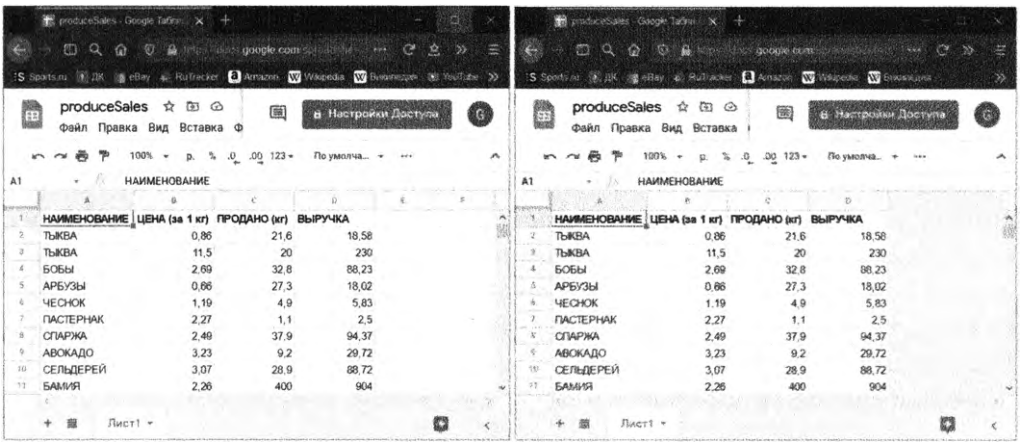


Рис. 14.6. Лист до (слева) и после (справа) изменения количества столбцов

Согласно статье https://support.google.com/drive/answer/37603?hl=ru/, в приложении Google Таблицы может содержаться до 5 млн ячеек. Однако рекомендуется делать листы большими настолько, насколько это необходимо, чтобы минимизировать время, необходимое для обновления данных.

Создание и удаление листов

Все электронные таблицы приложения Google Таблицы создаются с одним листом Лист1. Можно добавлять дополнительные листы в конец таблицы с помощью метода createSheet(), которому следует передать строку заголовка нового листа. Необязательный второй аргумент задает целочисленный индекс нового листа. Введите в интерактивной оболочке следующие инструкции, чтобы создать электронную таблицу и добавить в нее новые листы.

```
>>> import ezsheets
>>> ss = ezsheets.createSpreadsheet('Несколько листов')
>>> ss.sheetTitles
('Лист1',)
>>> ss.createSheet('Тушенка')    # создание нового листа в конце
<Sheet sheetId=2032744541, title='Тушенка', rowCount=1000,
    columnCount=26>
>>> ss.createSheet('Яйца')     # создание еще одного нового листа
<Sheet sheetId=417452987, title='Яйца', rowCount=1000,
   columnCount=26>
>>> ss.sheetTitles
('Лист1', 'Тушенка', 'Яйца')
>>> ss.createSheet('Бекон', 0)     # создание листа с индексом 0
<Sheet sheetId=814694991, title='Бекон', rowCount=1000,
   columnCount=26>
>>> ss.sheetTitles
('Бекон', 'Лист1', 'Тушенка', 'Яйца')
```

Эти инструкции добавляют в электронную таблицу три новых листа: `'Бекон'`, `'Тушенка'` и `'Яйца'` (в дополнение к листу `Лист1`, созданному по умолчанию). Листы в таблице упорядочены, и новые добавляются в конец списка, если только не передать методу `createSheet()` второй аргумент, определяющий индекс листа. В данном случае лист `'Бекон'` создается с индексом 0, что делает его первым в таблице, а остальные три листа смещаются на одну позицию вправо. Это напоминает поведение спискового метода `insert()`.

Новые листы на вкладках в нижней части окна показаны на рис. 14.7.

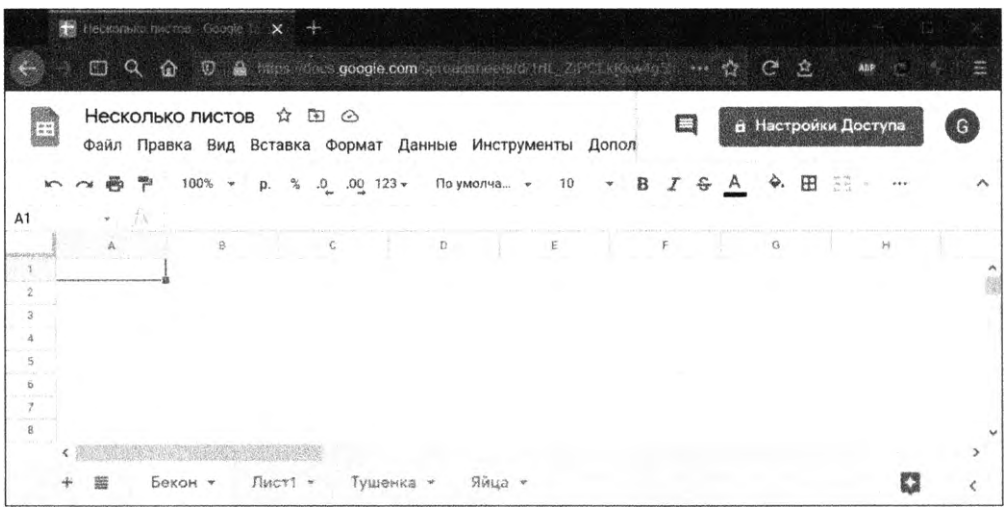


Рис. 14.7. Электронная таблица после добавления листов `'Тушенка'`, `'Яйца'` и `'Бекон'`

Метод `delete()` объекта `Sheet` удаляет лист из электронной таблицы. Если нужно сохранить лист, а удалить только данные, которые он содержит, вызовите метод `clear()`, чтобы очистить все ячейки листа, сделав его пустым. Введите в интерактивной оболочке следующие инструкции.

```
>>> ss.sheetTitles
('Бекон', 'Лист1', 'Тушенка', 'Яйца')
>>> ss[0].delete()    # удаление листа с индексом 0, т.е. 'Бекон'
>>> ss.sheetTitles
('Лист1', 'Тушенка', 'Яйца')
>>> ss['Тушенка'].delete()    # удаление листа 'Тушенка'
>>> ss.sheetTitles
('Лист1', 'Яйца')
>>> sheet = ss['Яйца']    # запись листа 'Яйца' в переменную
>>> sheet.delete()    # удаление листа 'Яйца'
>>> ss.sheetTitles
('Лист1',)
>>> ss[0].clear()    # очистка всех ячеек листа 'Лист1'
>>> ss.sheetTitles    # лист 'Лист1' имеется, но он пуст
('Лист1',)
```

Удаление листов — необратимая операция, и нет никакого способа восстановить данные. Но можно скопировать листы в другую электронную таблицу с помощью метода `copyTo()`, как описано в следующем разделе.

Копирование листов

Каждый объект `Spreadsheet` содержит упорядоченный список объектов `Sheet`, который можно использовать, чтобы изменить порядок листов (см. предыдущий раздел) или скопировать их в другие электронные таблицы. Если требуется скопировать объект `Sheet` в другой объект `Spreadsheet`, вызовите метод `copyTo()` и передайте ему целевой объект `Spreadsheet` в качестве аргумента. Введите в интерактивной оболочке следующие инструкции, чтобы создать две электронные таблицы и скопировать данные листа из первой таблицы во вторую.

```
>>> import ezsheets
>>> ss1 = ezsheets.createSpreadsheet('Первая электронная таблица')
>>> ss2 = ezsheets.createSpreadsheet('Вторая электронная таблица')
>>> ss1[0]
<Sheet sheetId=0, title='Лист1', rowCount=1000, columnCount=26>
>>> ss1[0].updateRow(1, ['Данные', 'в', 'первой', 'строке'])
>>> ss1[0].copyTo(ss2)    # копирование листа Лист1 из таблицы ss1
                          # в таблицу ss2
>>> ss2.sheetTitles    # таблица ss2 теперь содержит копию листа
                       # Лист1 таблицы ss1
('Лист1', 'Копия Лист1')
```

Поскольку в таблице ss2 уже есть лист с именем Лист1, скопированный лист будет называться Копия Лист1. Скопированные листы появляются в конце списка листов целевой таблицы. При желании можно изменить их атрибут index, чтобы поменять порядок листов в новой таблице.

Квоты приложения Google Таблицы

Поскольку приложение Google Таблицы доступно в Интернете, можно легко обмениваться листами между несколькими пользователями, которым предоставляется одновременный доступ к листам. Но это также означает, что чтение и обновление листов будет выполняться медленнее, чем чтение и обновление файлов Excel, хранящихся локально на жестком диске. Кроме того, в приложении Google Таблицы существуют ограничения на количество выполняемых операций чтения и записи.

Согласно рекомендациям Google для разработчиков, пользователи могут создавать 250 новых электронных таблиц в день, а в бесплатных учетных записях Google можно выполнять до 100 запросов на чтение и 100 запросов на запись в течение каждых 100 секунд. Попытка превысить эту квоту вызовет исключение googleapiclient.errors.HttpError — "Quota exceeded for quota group" (Превышена квота для группы квот). Модуль EZSheets автоматически перехватит это исключение и повторит запрос. Когда такое происходит, вызовы функций, выполняющих чтение или запись данных, приостановятся на несколько секунд (а возможно, даже на минуту-другую), прежде чем будет получен результат. Если сбои запроса продолжатся (возможно, другой сценарий, использующий те же учетные данные, тоже выполняет запросы в настоящий момент), EZSheets повторно сгенерирует указанное исключение.

Это означает, что иногда методы модуля EZSheets будут выполняться по нескольку секунд. Если хотите просмотреть свою статистику использования Google API или увеличить квоту, перейдите на страницу IAM & Admin Quotas по адресу https://console.developers.google.com/quotas/, чтобы узнать, как оплатить повышенный уровень использования. Если вы предпочитаете обрабатывать исключения HttpError, установите для параметра ezsheets.IGNORE_QUOTA значение True.

Резюме

Google Таблицы — это популярное браузерное приложение для работы с электронными таблицами. Используя сторонний модуль EZSheets, вы сможете загружать, создавать, читать и редактировать электронные таблицы, хранящиеся в Интернете. В модуле EZSheets электронным таблицам соответствуют объекты Spreadsheet, каждый из которых содержит

упорядоченный список объектов `Sheet`. В каждом листе есть столбцы и строки данных, которые можно читать и обновлять.

Несмотря на то что приложение Google Таблицы упрощает обмен данными и совместное редактирование таблиц, основным его недостатком является низкое быстродействие: таблицы обновляются с помощью веб-запросов, выполнение которых может занять несколько секунд. Но для большинства задач это ограничение не слишком влияет на сценарии Python, использующие модуль EZSheets. Приложение Google Таблицы также ограничивает частоту внесения изменений в таблицы.

Полная документация к модулю EZSheets доступна на сайте `https://ezsheets.readthedocs.io/`.

Контрольные вопросы

1. Какие три файла нужны модулю EZSheets для доступа к приложению Google Таблицы?
2. Какие два типа объектов имеются в модуле EZSheets?
3. Каким образом можно создать файл Excel на основе электронной таблицы Google Таблицы?
4. Каким образом можно создать таблицу Google Таблицы из файла Excel?
5. Переменная `ss` содержит объект `Spreadsheet`. Как прочитать данные из ячейки B2 на листе `'Студенты'`?
6. Как найти буквы столбца для столбца 999?
7. Как узнать, сколько строк и столбцов содержит лист?
8. Как удалить электронную таблицу? Можно ли отменить удаление?
9. Какие функции создают новый объект `Spreadsheet` и новый объект `Sheet`?
10. Что произойдет, если, делая частые запросы на чтение и запись с помощью модуля EZSheets, вы превысите квоту своей учетной записи Google?

Учебные проекты

Для практики напишите программы, выполняющие следующие задачи.

Загрузка данных из приложения Google Формы

Приложение Google Формы позволяет создавать простые веб-формы, которые облегчают сбор информации от пользователей. Информация, вводимая в форму, хранится в приложении Google Таблицы. Напишите

программу, которая будет автоматически загружать информацию из формы, заполненной пользователем. Перейдите на сайт `https://docs.google.com/forms/` и создайте пустую форму. Добавьте в нее поля, в которые пользователь должен ввести свое имя и адрес электронной почты. Затем щелкните на кнопке Отправить в правом верхнем углу, чтобы получить ссылку вида `https://goo.gl/forms/QZsq5sC2Qe4fYO592/`. Попробуйте ввести в форму несколько значений.

На вкладке Ответы щелкните на зеленой кнопке Создать таблицу, чтобы создать электронную таблицу приложения Google Таблицы, в которой будут храниться данные, введенные пользователями. В первых строках этой таблицы вы увидите свои примеры ответов. Напишите сценарий Python, использующий модуль EZSheets для получения списка адресов электронной почты из этой таблицы.

Преобразование электронных таблиц в другие форматы

Приложение Google Таблицы можно использовать для преобразования файла электронной таблицы в другие форматы. Напишите сценарий, который передает заданный файл методу `upload()`. После того как электронная таблица будет выгружена в приложение Google Таблицы, загрузите ее обратно, используя функции `downloadAsExcel()`, `downloadAsODS()` и т.п., чтобы получить копию таблицы в других форматах.

Поиск ошибок в электронной таблице

После долгого дня в офисе я завершил создание электронной таблицы с итогами подсчетов и загрузил ее в приложение Google Таблицы. Это общедоступная таблица (ее, правда, нельзя редактировать). Доступ к ней можно получить с помощью следующих инструкций.

```
>>> import ezsheets
>>> ss =
ezsheets.Spreadsheet('1jDZEdvSIh4TmZxccyy0ZXrH-ELlrwq8_YYiZrEOB4jg')
```

Можете также просмотреть эту таблицу в браузере, перейдя по следующей ссылке:

```
https://docs.google.com/spreadsheets/d/
1jDZEdvSIh4TmZxccyy0ZXrH-ELlrwq8_YYiZrEOB4jg/edit?usp=sharing/
```

Таблица содержит один лист со столбцами `'Beans per Jar'`, `'Jars'` и `'Total Beans'`. Столбец `'Total Beans'` представляет собой произведение чисел в столбцах `'Beans per Jar'` и `'Jars'`. Но в одной из 15 000 строк на этом листе есть ошибка. Ручная проверка затруднительна из-за слишком

большого количества строк. К счастью, можно написать сценарий, который проверяет итоговые значения.

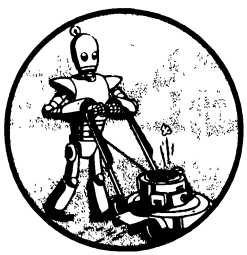
В качестве подсказки: можно получить доступ к отдельным ячейкам в строке с помощью вызова `ss[0].getRow(`*НомерСтроки*`)`, где `ss` — объект `Spreadsheet`. Помните о том, что нумерация строк в приложении Google Таблицы начинается с 1, а не с 0. Значения ячеек будут строками, поэтому их нужно преобразовать в целые числа, чтобы программа могла с ними работать. Следующее выражение истинно, если в строке содержится верное итоговое значение.

```
int(ss[0].getRow(2)[0]) * int(ss[0].getRow(2)[1]) ==
    int(ss[0].getRow(2)[2])
```

Поместите этот код в цикл, чтобы выяснить, в какой строке листа содержится неверное итоговое значение.

15

РАБОТА С ДОКУМЕНТАМИ PDF И WORD



Документы в форматах PDF и Word представляют собой бинарные файлы, поэтому работать с ними сложнее, чем с простыми текстовыми файлами. Помимо текста в них содержится масса дополнительной информации о шрифтах, используемых цветах и стилях абзацев. Если в программе требуется читать и записывать файлы PDF или Word, недостаточно передавать имена файлов функции `open()`.

К счастью, в Python имеются модули, упрощающие обработку документов в форматах PDF и Word. В этой главе будут описаны два таких модуля: PyPDF2 и Python-Docx.

PDF-документы

Файлы PDF (Portable Document Format – формат переносимых документов) имеют расширение *.pdf*. В этой главе мы рассмотрим две операции, которые выполняются чаще всего: чтение текстового содержимого из PDF-файлов и создание новых PDF-документов на основе существующих документов.

Для работы с PDF-документами мы будем использовать модуль PyPDF2. Чтобы инсталлировать его, выполните в командной строке команду `pip install --user PyPDF2==1.26.0`. Важно установить именно версию 1.26.0, так как будущие версии могут быть несовместимы с рассматриваемым кодом. Имя модуля чувствительно к регистру, поэтому проследите, чтобы только буква `'y'` была в нижнем регистре, а все остальные буквы были введены в верхнем регистре. (Более подробно процедура установки сторонних модулей описана в приложении А.) Признаком того, что модуль установлен корректно, является отсутствие сообщений об ошибках при выполнении команды `import PyPDF2` в интерактивной оболочке.

Проблематичность формата PDF

PDF-файлы удобны для чтения и вывода на печать, но программам не так-то легко выполнять их синтаксический анализ с целью преобразования в простой текст. Как следствие, модуль PyPDF2 иногда допускает ошибки при извлечении текста из PDF-файлов, а некоторые файлы ему вообще не удается открыть. К сожалению, с этим ничего нельзя поделать. Тем не менее подобное случается крайне редко.

Извлечение текста из PDF-файлов

Модуль PyPDF2 не умеет извлекать изображения, диаграммы и другие мультимедийные данные из PDF-документов, но способен извлекать текст и возвращать его в виде строки Python. Мы будем работать с PDF-документом, показанным на рис. 15.1.

Загрузите этот PDF-документ из архива примеров книги (см. введение) и введите в интерактивной оболочке следующие инструкции.

```
>>> import PyPDF2
>>> pdfFileObj = open('meetingminutes.pdf', 'rb')
>>> pdfReader = PyPDF2.PdfFileReader(pdfFileObj)
❶ >>> pdfReader.numPages
19
```

```
❷ >>> pageObj = pdfReader.getPage(0)
❸ >>> pageObj.extractText()
'OOFFFFIICCIIAALL BBOOAARRDD MMIINNUUTTEESS Meeting of March 7,
2015    \n      The Board of Elementary and Secondary Education
shall provide leadership and create policies for education that
expand opportunities for children, empower families and
communities, and advance Louisiana in an increasingly
competitive global market. BOARD of ELEMENTARY and SECONDARY
EDUCATION '
>> pdfFileObj.close()
```

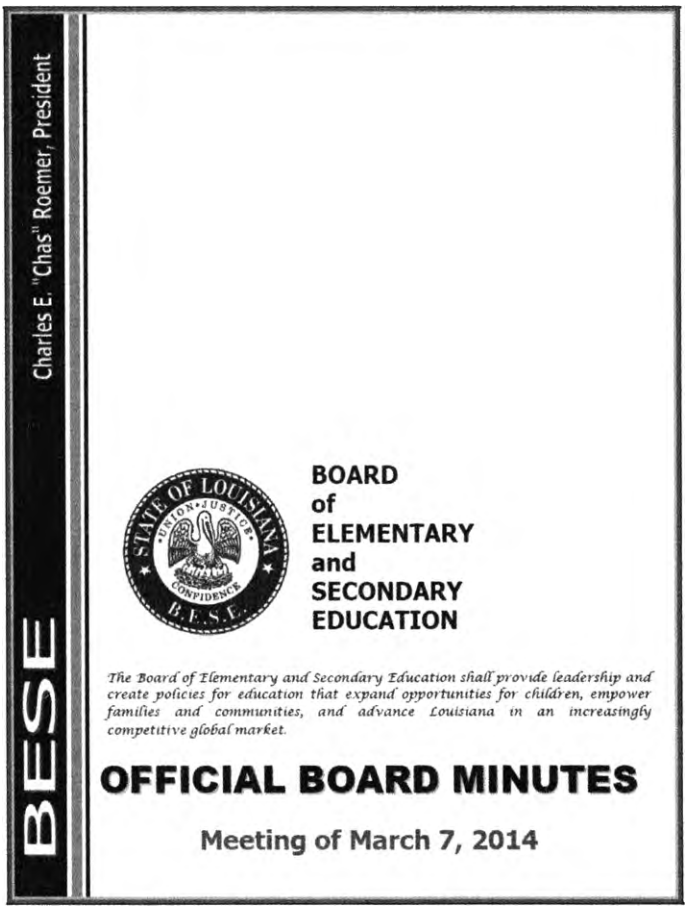


Рис. 15.1. Страница PDF-документа, из которой мы будем извлекать текст

Сначала мы импортируем модуль PyPDF2, после чего открываем файл *meetingminutes.pdf* в режиме чтения бинарных данных и сохраняем его содержимое в переменной `pdfFileObj`. Чтобы получить объект `PdfFileReader`, который представляет PDF-документ, необходимо вызвать метод `PyPDF2`.

PdfFileReader() и передать ему объект pdfFileObj. Объект PdfFileReader сохраняется в переменной pdfReader.

Количество страниц в документе хранится в атрибуте numPages объекта PdfFileReader ❶. В данном случае документ содержит 19 страниц, но мы извлечем из него только текст первой страницы.

Чтобы извлечь текст страницы, необходимо получить объект Page, который представляет отдельную страницу PDF-файла. Для этого следует вызвать метод getPage() объекта PdfFileReader ❷ для объекта PdfFileReader, передав ему номер требуемой страницы (в данном случае 0).

В модуле PyPDF2 индексация страниц ведется *с нуля*: первая страница имеет номер 0, вторая — 1 и т.д. Такой порядок нумерации соблюдается всегда, даже в тех случаях, когда нумерация страниц в самом документе иная. Например, предположим, что PDF-документ представляет собой трехстраничную выдержку из длинного отчета, и его страницы имеют номера 42, 43 и 44. Чтобы получить первую страницу такого документа, следует вызвать метод pdfReader.getPage(0), а не getPage(42) или getPage(1).

Получив объект Page, мы вызываем его метод extractText(), который возвращает строку, содержащую текст страницы ❸. Извлечение текста не проходит идеально: некоторые строки не были прочитаны, а в некоторых местах нарушены интервалы между словами. Тем не менее этого может оказаться вполне достаточно для программы.

Дешифровка PDF-документов

Некоторые PDF-документы могут быть зашифрованы, и для чтения такого документа необходимо указать пароль. Введите в интерактивной оболочке следующие инструкции, чтобы открыть PDF-документ *encrypted.pdf* (содержится в архиве примеров книги; см. введение), который зашифрован паролем 'rosebud'.

```
  >>> import PyPDF2
  >>> pdfReader = PyPDF2.PdfFileReader(open('encrypted.pdf', 'rb'))
❶ >>> pdfReader.isEncrypted
  True
  >>> pdfReader.getPage(0)
❷ Traceback (most recent call last):
    File "<pyshell#173>", line 1, in <module>
      pdfReader.getPage()
    -- Опущено --
    File "C:\Python34\lib\site-packages\PyPDF2\pdf.py", line 1173,
        in getObject
        raise utils.PdfReadError("file has not been decrypted")
  PyPDF2.utils.PdfReadError: file has not been decrypted
  >>> pdfReader = PyPDF2.PdfFileReader(open('encrypted.pdf', 'rb'))
```

```
❸ >>> pdfReader.decrypt('rosebud')
  1
  >>> pageObj = pdfReader.getPage(0)
```

У всех объектов `PdfFileReader` есть атрибут `isEncrypted`, который имеет значение `True`, если PDF-документ зашифрован, и `False` — в противном случае ❶. Любая попытка вызвать функцию, пытающуюся прочесть файл, прежде чем он будет дешифрован с использованием правильного пароля, приведет к ошибке ❷.

Примечание

Из-за ошибки в модуле PyPDF2 версии 1.26.0 вызов функции `getPage()` для зашифрованного PDF-файла перед вызовом функции `decrypt()` приводит к сбою последующих вызовов функции `getPage()` с появлением сообщения об ошибке `'IndexError: list index out of range'` (индекс списка за пределами диапазона). Вот почему мы повторно открываем файл, получая новый объект `PdfFileReader`.

Чтобы прочитать зашифрованный PDF-документ, вызовите функцию `decrypt()`, передав ей пароль в виде строки ❸. Если пароль правильный, то вызов метода `getPage()` больше не будет сопровождаться появлением сообщения об ошибке. В случае неверного пароля функция `decrypt()` вернет `0`, и метод `getPage()` не сможет выполниться. Следует отметить, что метод `decrypt()` дешифрует только объект `PdfFileReader`, но не сам PDF-файл. После того как программа завершит работу, файл на жестком диске останется зашифрованным. При следующем запуске программа должна будет снова вызвать функцию `decrypt()`.

Создание PDF-документов

В модуле PyPDF2 объекты `PdfFileReader` дополняются объектами `PdfFileWriter`, которые могут создавать новые PDF-файлы. Но модуль не поддерживает запись произвольного текста в формате PDF, как это делает Python с простыми текстовыми файлами. Возможности модуля в отношении записи PDF-документов ограничены копированием страниц из других PDF-файлов, поворотом и наложением страниц, а также шифрованием файлов.

Модуль PyPDF2 не позволяет непосредственно редактировать PDF-документ. Вместо этого нужно создать новый PDF-файл, а затем скопировать содержимое из существующего документа. В примерах этого раздела мы будем следовать такой процедуре:

1) открыть один или несколько существующих PDF-файлов (исходных PDF-документов) в объектах `PdfFileReader`;
2) создать новый объект `PdfFileWriter`;
3) скопировать страницы из объектов `PdfFileReader` в объект `PdfFileWriter`;
4) использовать объект `PdfFileWriter` для записи выходного PDF-документа.

При создании объекта `PdfFileWriter` вы лишь получаете структуру, которая представляет PDF-документ в Python. Сам PDF-файл будет создан, только когда вы вызовете метод `write()` объекта `PdfFileWriter`.

Методу `write()` передается обычный объект `File`, открытый в режиме записи бинарных данных. Такой объект можно получить, вызвав функцию `open()` с двумя аргументами: строкой с именем PDF-файла и строкой `'wb'`, задающей режим записи файла.

Копирование страниц

Модуль PyPDF2 можно использовать для копирования страниц из одного PDF-документа в другой. Это позволяет объединять несколько PDF-файлов, переупорядочивать страницы или удалять ненужные.

Загрузите файлы *meetingminutes.pdf* и *meetingminutes2.pdf* из архива примеров книги (см. введение) и поместите их в текущий каталог, после чего введите в интерактивной оболочке следующие инструкции.

```
  >>> import PyPDF2
  >>> pdf1File = open('meetingminutes.pdf', 'rb')
  >>> pdf2File = open('meetingminutes2.pdf', 'rb')
❶ >>> pdf1Reader = PyPDF2.PdfFileReader(pdf1File)
❷ >>> pdf2Reader = PyPDF2.PdfFileReader(pdf2File)
❸ >>> pdfWriter = PyPDF2.PdfFileWriter()

  >>> for pageNum in range(pdf1Reader.numPages):
❹         pageObj = pdf1Reader.getPage(pageNum)
❺         pdfWriter.addPage(pageObj)

  >>> for pageNum in range(pdf2Reader.numPages):
❹         pageObj = pdf2Reader.getPage(pageNum)
❺         pdfWriter.addPage(pageObj)

❻ >>> pdfOutputFile = open('combinedminutes.pdf', 'wb')
  >>> pdfWriter.write(pdfOutputFile)
  >>> pdfOutputFile.close()
  >>> pdf1File.close()
  >>> pdf2File.close()
```

Мы открываем оба PDF-файла в режиме чтения бинарных данных и сохраняем результирующие объекты `File` в переменных `pdf1File` и `pdf2File`, после чего получаем объект `PdfFileReader` для файла *meetingminutes.pdf*, вызвав функцию `PyPDF2.PdfFileReader()` и передав ей переменную `pdf1File` ❶. Эта же функция вызывается с переменной `pdf2File`, чтобы получить объект `PdfFileReader` для файла *meetingminutes2.pdf* ❷. Затем создается новый объект `PdfFileWriter`, представляющий пустой PDF-документ ❸.

Далее все страницы копируются из двух исходных PDF-файлов и добавляются в объект `PdfFileWriter`. Мы получаем объект `Page`, вызывая метод `getPage()` для объекта `PdfFileReader` ❹, и передает этот объект `Page` методу `addPage()` объекта `PdfFileWriter` ❺. Указанные действия выполняются сначала для объекта `pdf1Reader`, а затем — для объекта `pdf2Reader`. Завершив копирование страниц, мы записываем новый PDF-файл *combinedminutes.pdf*, передавая объект `File` методу `write()` объекта `PdfFileWriter` ❻.

Примечание

Модуль PyPDF2 не позволяет вставлять страницы в середину документа, представляемого объектом `PdfFileWriter`. Метод `addPage()` способен добавлять страницы лишь в конец документа.

Мы создали новый PDF-файл, объединяющий страницы из файлов *meetingminutes.pdf* и *meetingminutes2.pdf* в один документ. Не забывайте о том, что объект `File`, передаваемый функции `PyPDF2.PdfFileReader()`, должен быть открыт в режиме чтения бинарных данных. Для этого в качестве второго аргумента функции `open()` должна быть задана строка `'rb'`. Аналогичным образом объект `File`, передаваемый функции `PyPDF2.PdfFileWriter()`, должен быть открыт в режиме записи бинарных данных с помощью строки `'wb'`.

Поворот страниц

Страницы PDF-документа можно поворачивать на углы, кратные 90°, по часовой стрелке и против часовой стрелки с помощью методов `rotateClockwise()` и `rotateCounterClockwise()` соответственно. В качестве аргументов эти методы получают целые числа `90`, `180` и `270`. Убедившись в наличии файла *meetingminutes.pdf* в текущем каталоге, введите в интерактивной оболочке следующие инструкции.

```
>>> import PyPDF2
>>> minutesFile = open('meetingminutes.pdf', 'rb')
>>> pdfReader = PyPDF2.PdfFileReader(minutesFile)
❶ >>> page = pdfReader.getPage(0)
❷ >>> page.rotateClockwise(90)
{'/Contents': [IndirectObject(961, 0), IndirectObject(962, 0),
```

```
  -- Опущено --
  }
  >>> pdfWriter = PyPDF2.PdfFileWriter()
  >>> pdfWriter.addPage(page)
❸ >>> resultPdfFile = open('rotatedPage.pdf', 'wb')
  >>> pdfWriter.write(resultPdfFile)
  >>> resultPdfFile.close()
  >>> minutesFile.close()
```

Мы вызываем метод `getPage(0)` для выбора первой страницы PDF-документа ❶, а затем поворачиваем эту страницу на 90° по часовой стрелке с помощью вызова `rotateClockwise(90)` ❷. Повернутую страницу мы записываем в новый PDF-документ и сохраняем его в файле *rotatedPage.pdf* ❸.

Результирующий документ будет содержать одну исходную страницу, повернутую на 90° по часовой стрелке (рис. 15.2). Значения, возвращаемые методами `rotateClockwise()` и `rotateCounterClockwise()`, содержат дополнительную информацию, которую можно игнорировать.

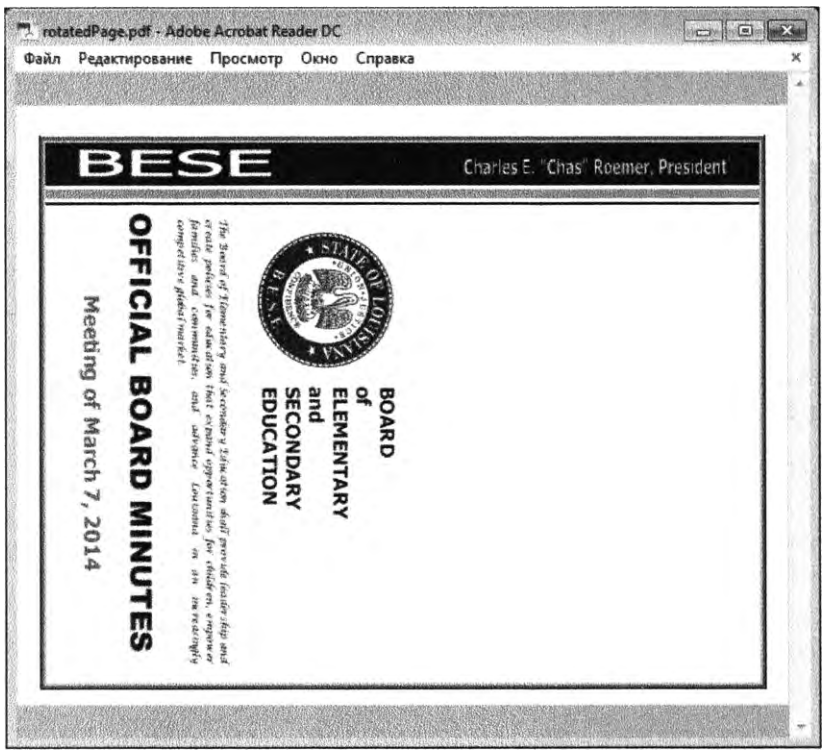


Рис. 15.2. Сохраненная в файле rotatedPage.pdf страница, повернутая на 90° по часовой стрелке

Наложение страниц

Модуль PyPDF2 позволяет накладывать содержимое одной страницы поверх другой, что можно использовать для добавления в документ логотипа, метки времени или водяного знака. С помощью Python можно легко добавить водяные знаки в несколько файлов, и только на те страницы, которые будут указаны в программе.

Загрузите файл *watermark.pdf*, доступный в архиве примеров книги (см. введение), и поместите этот PDF-документ в текущий каталог вместе с файлом *meetingminutes.pdf*. После этого введите в интерактивной оболочке следующие инструкции.

```
  >>> import PyPDF2
  >>> minutesFile = open('meetingminutes.pdf', 'rb')
❶ >>> pdfReader = PyPDF2.PdfFileReader(minutesFile)
❷ >>> minutesFirstPage = pdfReader.getPage(0)
❸ >>> pdfWatermarkReader =
          PyPDF2.PdfFileReader(open('watermark.pdf', 'rb'))
❹ >>> minutesFirstPage.mergePage(pdfWatermarkReader.getPage(0))
❺ >>> pdfWriter = PyPDF2.PdfFileWriter()
❻ >>> pdfWriter.addPage(minutesFirstPage)

❼ >>> for pageNum in range(1, pdfReader.numPages):
          pageObj = pdfReader.getPage(pageNum)
          pdfWriter.addPage(pageObj)

  >>> resultPdfFile = open('watermarkedCover.pdf', 'wb')
  >>> pdfWriter.write(resultPdfFile)
  >>> minutesFile.close()
  >>> resultPdfFile.close()
```

Мы создаем объект `PdfFileReader` на основе файла *meetingminutes.pdf* ❶. Далее мы вызываем метод `getPage(0)`, чтобы получить объект `Page` для первой страницы и сохранить его в переменной `minutesFirstPage` ❷. Затем создается объект `PdfFileReader` для файла *watermark.pdf* ❸ и вызывается метод `mergePage()` для объекта, сохраненного в переменной `minutesFirstPage` ❹. Аргументом, который мы передаем методу `mergePage()`, служит объект `Page` для первой страницы файла *watermark.pdf*.

Теперь, когда для переменной `minutesFirstPage` был вызван метод `mergePage()`, она представляет первую страницу файла с добавленным водяным знаком. В следующей строке мы создаем объект `PdfFileWriter` ❺ и добавляем в него эту страницу ❻. Затем мы выполняем цикл по оставшимся страницам файла *meetingminutes.pdf* и тоже добавляем их в объект `PdfFileWriter` ❼. Наконец, мы открываем новый PDF-файл *watermarkedCover.pdf* и записываем в него содержимое объекта `PdfFileWriter`.

Результат представлен на рис. 15.3. В нашем новом PDF-документе *watermarkedCover.pdf* хранится все содержимое документа *meetingminutes.pdf* с первой страницей, помеченной водяным знаком.

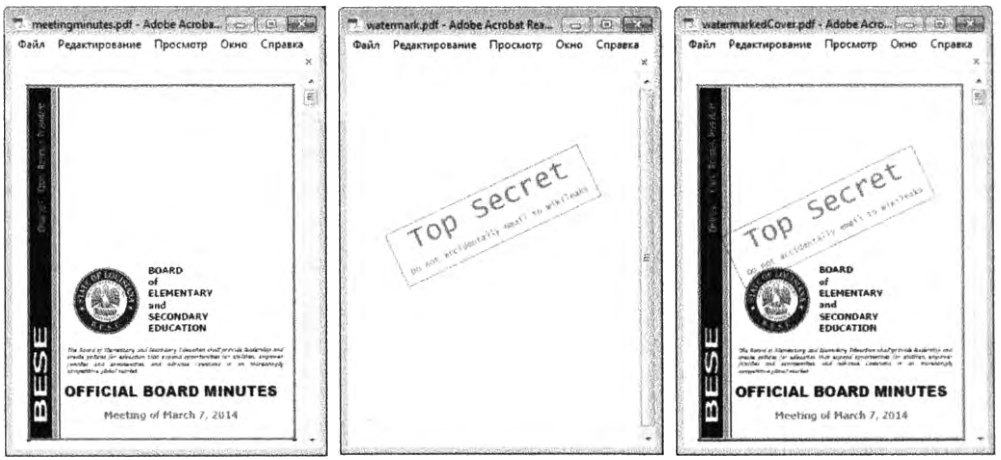


Рис. 15.3. Оригинальный PDF-документ (слева), водяной знак (в центре) и объединенный PDF-документ (справа)

Шифрование PDF-документов

Объект `PdfFileWriter` также позволяет шифровать PDF-документы. Введите в интерактивной оболочке следующие инструкции.

```
>>> import PyPDF2
>>> pdfFile = open('meetingminutes.pdf', 'rb')
>>> pdfReader = PyPDF2.PdfFileReader(pdfFile)
>>> pdfWriter = PyPDF2.PdfFileWriter()
>>> for pageNum in range(pdfReader.numPages):
        pdfWriter.addPage(pdfReader.getPage(pageNum))

❶ >>> pdfWriter.encrypt('swordfish')
>>> resultPdf = open('encryptedminutes.pdf', 'wb')
>>> pdfWriter.write(resultPdf)
>>> resultPdf.close()
```

Прежде чем вызывать метод `write()` для сохранения файла, вызовите метод `encrypt()` и передайте ему строку пароля ❶. PDF-документы могут иметь *пароль пользователя* (позволяющий просматривать документ) и *пароль владельца* (позволяющий устанавливать разрешения для вывода документа на печать, снабжения его комментариями и извлечения текста и т.п.). Пароли пользователя и владельца задаются соответственно в качестве первого и второго аргументов метода `encrypt()`. Если передать методу `encrypt()` только одну строку, то она будет использована для обоих паролей.

В этом примере мы скопировали страницы документа *meetingminutes.pdf* в объект `PdfFileWriter`. Далее мы зашифровали объект `PdfFileWriter` с помощью пароля `'swordfish'`, открыли новый PDF-файл *encryptedminutes.pdf* и записали в него содержимое зашифрованного объекта. Если пользователь захочет просмотреть содержимое файла *encryptedminutes.pdf*, он должен будет ввести этот пароль. После того как вы убедитесь в том, что копия зашифрована корректно, оригинальный незашифрованный файл *meetingminutes.pdf* можно будет удалить.

Проект: объединение выбранных страниц из многих PDF-документов

Предположим, вам предстоит выполнить утомительную работу по слиянию десятков PDF-документов в один PDF-файл. Каждый из документов начинается с титульного листа на первой странице, но вы не хотите, чтобы первые страницы повторялись в итоговом документе. Несмотря на то что существует много бесплатных программ, позволяющих объединять PDF-документы, все, на что они способны, — это слияние исходных файлов в единый файл. Мы же напишем программу, позволяющую выбирать страницы, которые должны включаться в результирующий PDF-документ.

Программа должна выполнять следующие действия:

1) находить все PDF-файлы в текущем каталоге;
2) сортировать файлы по именам, чтобы файлы добавлялись в определенном порядке;
3) записывать каждую страницу исходного PDF-файла, за исключением первой, в выходной файл.

Это означает, что программа будет выполнять следующие операции:

1) вызывать функцию `os.listdir()` для нахождения всех файлов в текущем каталоге и удалять из списка все, кроме файлов в формате PDF;
2) вызывать списковый метод `sort()` для сортировки имен файлов в алфавитном порядке;
3) создавать объект `PdfFileWriter` для выходного PDF-файла;
4) организовывать цикл по всем PDF-файлам, создавая объект `PdfFileReader` для каждого из них;
5) организовывать цикл по всем страницам (за исключением первой) каждого PDF-файла;
6) добавлять страницы в выходной PDF-файл;
7) записывать выходной PDF-файл в файл *allminutes.pdf*.

Откройте в файловом редакторе новое окно и сохраните программу в файле *combinePdfs.py*.

Шаг 1. Поиск всех PDF-файлов

В первую очередь программа должна получить список всех файлов с расширением *.pdf* в текущем каталоге и отсортировать этот список. Введите следующий код.

```
#! python3
# combinePdfs.py - объединяет все PDF-файлы, находящиеся
# в текущем каталоге, в единый PDF-документ

❶ import PyPDF2, os

# Получение списка всех PDF-файлов
pdfFiles = []
for filename in os.listdir('.'):
    if filename.endswith('.pdf'):
❷       pdfFiles.append(filename)
❸ pdfFiles.sort(key=str.lower)

❹ pdfWriter = PyPDF2.PdfFileWriter()

# СДЕЛАТЬ: организовать цикл по всем PDF-файлам

# СДЕЛАТЬ: организовать цикл по всем страницам (кроме первой),
#          добавляя их в результирующий документ

# СДЕЛАТЬ: сохранить результирующий PDF-документ в файле
```

Вслед за строкой сценария и комментарием, описывающим назначение программы, в программе импортируются модули `os` и `PyPDF2` ❶. Функция `os.listdir('.')` возвращает список всех файлов, находящихся в текущем каталоге. Программа просматривает этот список в цикле и добавляет в новый список `pdfFiles` лишь файлы с расширением *.pdf* ❷. После этого список `pdfFiles` сортируется в алфавитном порядке, который задается именованным аргументом `key=str.lower` при вызове метода `sort()` ❸.

Для хранения объединенных PDF-страниц создается объект `PdfFileWriter` ❹. Остальные части программы предстоит написать.

Шаг 2. Открытие PDF-файлов

Теперь программа должна прочитать каждый из PDF-файлов, входящих в список `pdfFiles`. Добавьте в программу код, выделенный полужирным шрифтом.

```
#! python3
# combinePdfs.py - объединяет все PDF-файлы, находящиеся
# в текущем каталоге, в единый PDF-документ

import PyPDF2, os

# Получение списка всех PDF-файлов
pdfFiles = []
-- Опущено --

# Организация цикла по всем PDF-файлам
for filename in pdfFiles:
    pdfFileObj = open(filename, 'rb')
    pdfReader = PyPDF2.PdfFileReader(pdfFileObj)
    # СДЕЛАТЬ: организовать цикл по всем страницам (кроме первой),
    #          добавляя их в результирующий документ

# СДЕЛАТЬ: сохранить результирующий PDF-документ в файле
```

В цикле каждый PDF-файл открывается в режиме чтения бинарных данных путем вызова метода `open()`, которому передается строка `'rb'` в качестве второго аргумента. Метод `open()` возвращает объект `File`, который передается функции `PyPDF2.PdfFileReader()`, создающей объект `PdfFileReader` для данного PDF-файла.

Шаг 3. Добавление страниц

Для каждого из PDF-файлов необходимо отобрать в цикле все страницы, за исключением первой. Добавьте в программу код, выделенный полужирным шрифтом.

```
#! python3
# combinePdfs.py - объединяет все PDF-файлы, находящиеся
# в текущем каталоге, в единый PDF-документ

import PyPDF2, os

-- Опущено --

# Организация цикла по всем PDF-файлам
for filename in pdfFiles:
-- Опущено --
    # Организация цикла по всем страницам (за исключением
    # первой), которые добавляются в результирующий документ
❶   for pageNum in range(1, pdfReader.numPages):
        pageObj = pdfReader.getPage(pageNum)
        pdfWriter.addPage(pageObj)

# СДЕЛАТЬ: сохранить результирующий PDF-документ в файле
```

Код в цикле копирует каждый объект `Page` по отдельности в объект `PdfFileWriter`. Вспомните, что мы хотим пропускать первую страницу. Поскольку в модуле PyPDF2 первой странице соответствует индекс `0`, цикл должен выполняться в диапазоне индексов от `1` до `pdfReader.numPages` (последнее значение не включается в диапазон) ❶.

Шаг 4. Сохранение результатов

После выполнения вложенных циклов `for` переменная `pdfWriter` будет содержать объект `PdfFileWriter`, включающий страницы всех объединяемых PDF-документов. Последний шаг заключается в том, чтобы записать это содержимое в файл. Добавьте в программу код, выделенный полужирным шрифтом.

```
#! python3
# combinePdfs.py - объединяет все PDF-файлы, находящиеся
# в текущем каталоге, в единый PDF-документ

import PyPDF2, os

-- Опущено --

# Организация цикла по всем PDF-файлам
for filename in pdfFiles:
-- Опущено --
    # Организация цикла по всем страницам (за исключением
    # первой), которые добавляются в результирующий документ
    for pageNum in range(1, pdfReader.numPages):
     -- Опущено --

# Сохранение результирующего PDF-документа в файле
pdfOutput = open('allminutes.pdf', 'wb')
pdfWriter.write(pdfOutput)
pdfOutput.close()
```

Чтобы открыть выходной файл *allminutes.pdf* в режиме записи бинарных данных, мы передаем функции `open()` строку `'wb'`. В результате последующей передачи результирующего объекта `File` методу `write()` создается сам PDF-файл. В конце вызывается метод `close()`.

Идеи для создания похожих программ

Возможность создавать PDF-документы на основе страниц, извлекаемых из других PDF-документов, можно применить для написания программ, которые решают следующие задачи:

- удаление указанных страниц из PDF-документов;
- реорганизация страниц в PDF-документах;

- создание PDF-документа на основе только тех страниц, которые содержат заданный текст, определяемый вызовом метода `extractText()`.

Документы Word

С помощью модуля Python-Docx можно создавать и изменять документы Word с расширением *.docx*. Чтобы установить этот модуль, выполните команду `install --user -U python-docx==0.8.10`. (Более подробно процедура установки сторонних модулей описана в приложении А.)

Примечание

Используя команду `pip` для установки модуля Python-Docx, обязательно вводите `install python-docx`, а не `docx`. Имя `docx` относится к другому модулю, который в данной книге не рассматривается. В то же время при импорте модуля в программе следует использовать инструкцию `import docx`, а не `import python-docx`.

Если у вас нет приложения Microsoft Word, воспользуйтесь его бесплатными альтернативами LibreOffice Writer и OpenOffice Writer (доступны для операционных систем Windows, macOS и Linux), которые позволяют открывать файлы *.docx*. Их можно скачать на сайтах `https://www.libreoffice.org` и `https://openoffice.org` соответственно. Документация к модулю Python-Docx доступна на сайте `https://python-docx.readthedocs.org/`. Несмотря на то что существует версия Word для macOS, в этой главе мы будем работать с версией Word для Windows.

В отличие от простых текстовых файлов, файлы с расширением *.docx* имеют сложную внутреннюю структуру. В модуле Python-Docx эта структура представлена тремя типами данных. На самом верхнем уровне объект `Document` представляет весь документ. Он содержит список объектов `Paragraph`, которые представляют абзацы документа. (Новый абзац начинается всякий раз, когда пользователь нажимает клавишу <Enter> или <Return> при вводе текста в Word.) Каждый из абзацев содержит список, состоящий из одного или нескольких объектов `Run`, представляющих фрагменты текста с различными стилями форматирования. Абзац, показанный на рис. 15.4, состоит из четырех таких фрагментов.

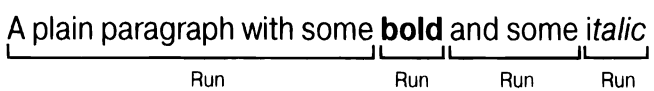


Рис. 15.4. Объекты `Run`, определенные в объекте `Paragraph`

Текст в документах Word — это не просто текстовая строка. Он включает информацию, описывающую тип, размер и цвет шрифта, а также другую информацию, связанную с форматированием. Коллекция этих атрибутов образует *стиль* документа Word. Объект `Run` представляет непрерывный фрагмент текста, оформленный с использованием одного и того же стиля. Каждой смене стиля соответствует новый объект `Run`.

Чтение документов Word

Поэкспериментируем с модулем `docx`. Загрузите файл *demo.docx*, хранящийся в архиве примеров книги (см. введение), и сохраните документ в текущем каталоге, после чего введите в интерактивной оболочке следующие инструкции.

```
>>> import docx
❶ >>> doc = docx.Document('demo.docx')
❷ >>> len(doc.paragraphs)
7
❸ >>> doc.paragraphs[0].text
'Document Title'
❹ >>> doc.paragraphs[1].text
'A plain paragraph with some bold and some italic'
❺ >>> len(doc.paragraphs[1].runs)
4
❻ >>> doc.paragraphs[1].runs[0].text
'A plain paragraph with some '
❼ >>> doc.paragraphs[1].runs[1].text
'bold'
❽ >>> doc.paragraphs[1].runs[2].text
' and some '
❾ >>> doc.paragraphs[1].runs[3].text
'italic'
```

Сначала мы открываем файл *.docx*, вызывая функцию `docx.Document()` и передавая ей имя файла *demo.docx* ❶. Эта функция возвращает объект `Document`, атрибут `paragraphs` которого представляет собой список объектов `Paragraph`. Значение `7`, возвращаемое методом `len()` для списка `doc.paragraphs` ❷, указывает на то, что в документе содержится семь объектов `Paragraph`. Каждый из этих объектов `Paragraph` имеет атрибут `text`, содержащий строку текста данного абзаца (без информации о стиле). В данном случае первый атрибут `text` содержит строку `'Document Title'` ❸, а второй — строку `'A plain paragraph with some bold and some italic'` ❹.

Кроме того, каждый объект `Paragraph` имеет атрибут `runs`, который представляет собой список объектов `Run`. У объектов `Run` тоже есть атрибут `text`, который содержит лишь текст данного фрагмента форматирования. Обратимся к атрибутам `text` второго объекта `Paragraph` с текстом `'A plain`

paragraph with some bold and some italic'. Вызов метода len() для этого объекта сообщает о том, что объект включает четыре объекта Run ❺. Строке 'A plain paragraph with some ' соответствует первый объект Run ❻. Затем к тексту применяется полужирное начертание, поэтому строке 'bold' соответствует новый объект Run ❼. Далее идет текст с обычным форматированием ' and some ', которому соответствует третий объект Run ❽.Четвертый объект Run содержит строку 'italic', к которой применено курсивное начертание ❾.

Благодаря модулю Python-Docx программы смогут читать текст из файлов с расширением *.docx* и работать с ним как с обычными строками.

Получение всего текста из файла .docx

Если в документе Word вас интересует только текст без информации о форматировании, можно написать функцию getText(), которая получает имя файла *.docx* в качестве аргумента и возвращает строку, содержащую полный текст файла. Откройте в файловом редакторе новое окно, введите в нем следующий код и сохраните программу в файле *readDocx.py*.

```
#! python3

import docx

def getText(filename):
    doc = docx.Document(filename)
    fullText = []
    for para in doc.paragraphs:
        fullText.append(para.text)
    return '\n'.join(fullText)
```

Функция getText() открывает документ Word и просматривает в цикле все объекты Paragraph в списке paragraphs, присоединяя содержащийся в них текст к списку fullText. По завершении цикла все строки, содержащиеся в списке fullText, объединяются с использованием символа новой строки в качестве разделителя.

Программу *readDocx.py* можно импортировать подобно любому другому модулю. Если все, что вам нужно, — это извлечь текст из документа Word, введите следующие инструкции.

```
>>> import readDocx
>>> print(readDocx.getText('demo.docx'))
Document Title
A plain paragraph with some bold and some italic
Heading, level 1
Intense quote
```

```
first item in unordered list
first item in ordered list
```

Можно сделать так, чтобы функция `getText()` изменяла строку, прежде чем возвращать ее. Например, чтобы выделить каждый абзац отступом, замените вызов `append()` в программе *readDocx.py* следующим его вариантом:

```
fullText.append('    ' + para.text)
```

Чтобы добавить удвоенный междустрочный интервал между абзацами, замените вызов `join()` следующим:

```
return '\n\n'.join(fullText)
```

Как видите, для написания функции, которая считывает файл *.docx* и возвращает строку с его содержимым, достаточно всего нескольких строк кода.

Стилевое оформление абзаца и объекты `Run`

В Word для Windows список стилей можно увидеть, нажав комбинацию клавиш <Ctrl+Alt+Shift+S> для отображения панели Стили (рис. 15.5). В macOS следует выбрать пункты меню View⇨Styles (Вид⇨Стили).

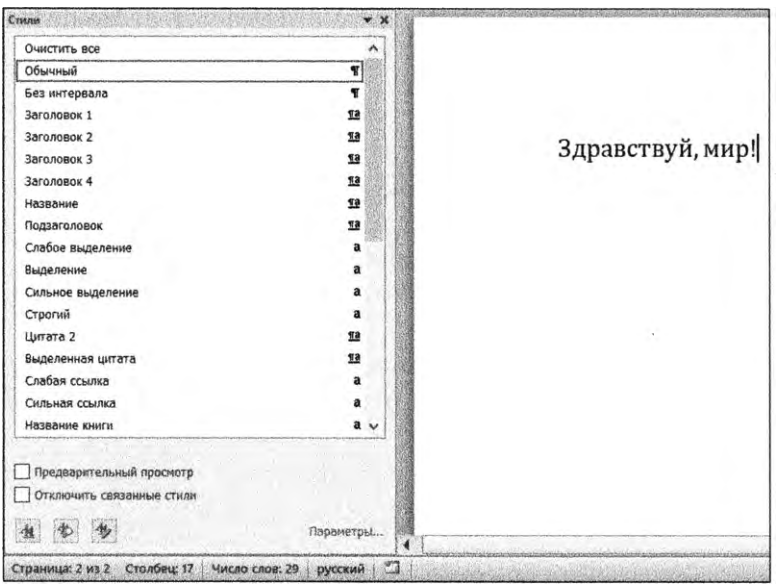


Рис. 15.5. Панель стилей

Стили используются в Word и других текстовых процессорах для придания документу единообразного внешнего вида. Возможно, вы хотите, чтобы для абзацев основного текста использовался шрифт Times New Roman размером 11 пунктов, а сам текст выравнивался по левому краю. Можно создать стиль с этими параметрами и назначить его всем абзацам документа. Впоследствии, если понадобится изменить внешний вид абзацев по всему документу, достаточно будет изменить лишь стиль, и все абзацы автоматически поменяются.

В документах Word существуют три типа стилей: *стили абзацев*, которые могут применяться к объектам `Paragraph`, *стили символов*, которые могут применяться к объектам `Run`, и *связанные стили*, которые могут применяться к обоим типам объектов. Как объектам `Paragraph`, так и объектам `Run` можно назначать стили, присваивая атрибуту `style` название стиля. Если этот атрибут равен `None`, то у объекта `Paragraph` или `Run` не будет связанного с ним стиля.

Ниже приведены имена стилей Word, используемых по умолчанию.

```
'Normal'          'Heading 5'          'List Bullet'          'List Paragraph'
'Body Text'       'Heading 6'          'List Bullet 2'        'MacroText'
'Body Text 2'     'Heading 7'          'List Bullet 3'        'No Spacing'
'Body Text 3'     'Heading 8'          'List Continue'        'Quote'
'Caption'         'Heading 9'          'List Continue 2'      'Subtitle'
'Heading 1'       'Intense Quote'      'List Continue 3'      'TOC Heading'
'Heading 2'       'List'               'List Number'          'Title'
'Heading 3'       'List 2'             'List Number 2'
'Heading 4'       'List 3'             'List Number 3'
```

Если к объекту `Run` применяется связанный стиль, то к его имени следует присоединить строку `'Char'`. Например, для объекта `Paragraph` связанный стиль `'Quote'` задается инструкцией `paragraphObj.style = 'Quote'`, но в случае объекта `Run` требуется инструкция `runObj.style = 'QuoteChar'`.

В текущей версии модуля Python-Docx (0.8.10) единственные стили, которые можно использовать, — это заданные по умолчанию стили Word и стили, существующие в открытом документе *.docx*. Возможность создания новых стилей в настоящее время отсутствует, хотя в будущих версиях модуля Python-Docx ситуация может измениться.

Создание документов Word с нестандартными стилями

Если требуется создавать документы Word, в которых используются нестандартные стили, прежде всего откройте в Word пустой документ и создайте стили самостоятельно, щелкнув на кнопке Создать стиль в нижней части панели Стили (рис. 15.6).

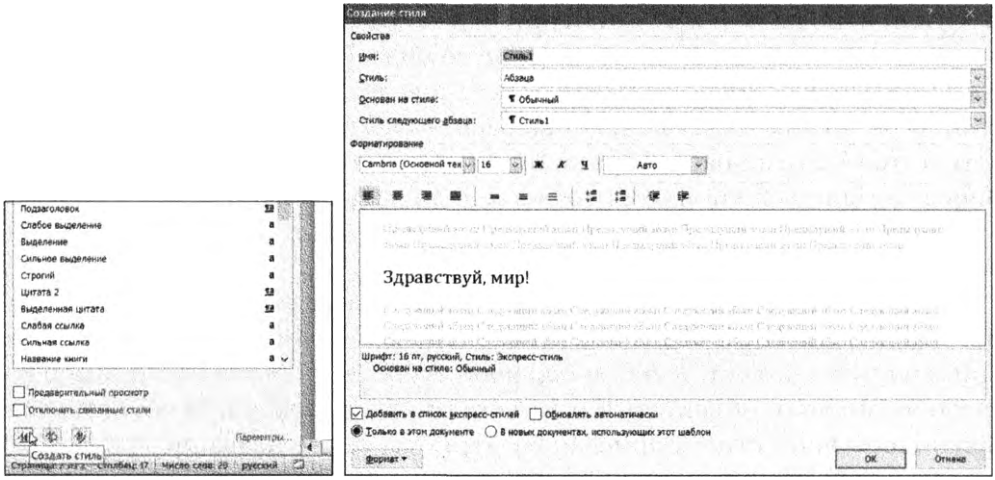


Рис. 15.6. Кнопка Создать стиль *(слева) и диалоговое окно* Создание стиля *(справа)*

В результате откроется диалоговое окно Создание стиля, в котором можно настроить новый стиль. После этого вернитесь в интерактивную оболочку, откройте документ с помощью функции `docx.Document()` и используйте его в качестве основы для других документов. Теперь имя, которое вы присвоили новому стилю, станет доступно для функций модуля Python-Docx.

Атрибуты объекта Run

К фрагментам текста, представленным объектами `Run`, можно применять дополнительное форматирование с помощью атрибутов `text`. Каждый из этих атрибутов может иметь одно из трех значений: `True` (атрибут активен постоянно, независимо от применения к данному фрагменту других стилей), `False` (атрибут всегда отключен) и `None` (применяется стиль, установленный для данного объекта `Run`).

Атрибуты `text`, которые могут назначаться объектам `Run`, перечислены в табл. 15.1.

Таблица 15.1. Атрибуты `text` объекта `Run`

Атрибут	Описание
`bold`	Полужирный текст
`italic`	Курсив
`underline`	Подчеркивание
`strike`	Зачеркивание
`double_strike`	Двойное зачеркивание
`all_caps`	Все прописные

Окончание табл. 15.1

Атрибут	Описание
small_caps	Отображение текста малыми прописными буквами (капитель), размер которых на два пункта больше размера строчных букв
shadow	Текст с тенью
outline	Контурный текст
rtl	Направление текста справа налево
imprint	Утопленный текст
emboss	Приподнятый текст

Поэкспериментируйте с изменением стилей форматирования текста из файла *demo.docx*, введя в интерактивной оболочке следующие инструкции.

```
>>> import docx
>>> doc = docx.Document('demo.docx')
>>> doc.paragraphs[0].text
'Document Title'
>>> doc.paragraphs[0].style    # идентификатор может быть другим
_ParagraphStyle('Title') id: 3095631007984
>>> doc.paragraphs[0].style = 'Normal'
>>> doc.paragraphs[1].text
'A plain paragraph with some bold and some italic'
>>> (doc.paragraphs[1].runs[0].text, doc.paragraphs[1].runs[1].text,
doc.paragraphs[1].runs[2].text, doc.paragraphs[1].runs[3].text)
('A plain paragraph with some ', 'bold', ' and some ', 'italic')
>>> doc.paragraphs[1].runs[0].style = 'QuoteChar'
>>> doc.paragraphs[1].runs[1].underline = True
>>> doc.paragraphs[1].runs[3].underline = True
>>> doc.save('restyled.docx')
```

В этом примере показано, как исследовать содержимое абзацев документа с помощью атрибутов `text` и `style`. Абзац можно легко разбить на фрагменты, чтобы получать доступ к ним по отдельности. В данном случае мы извлекаем первый, второй и четвертый фрагменты второго абзаца, применяем к ним стили и сохраняем результат в новом документе.

К строке "Document Title" в начале документа *restyled.docx* вместо стиля `'Title'` применен стиль `'Normal'`, к тексту "A plain paragraph with some" применен стиль `'QuoteChar'`, а атрибутам `underline` двух объектов `Run` для слов "bold" и "italic" присвоено значение `True`. Результат показан на рис. 15.7.

Более подробная информация о применении стилей форматирования в документах Word с помощью модуля Python-Docx доступна по следующему адресу:

```
https://python-docx.readthedocs.io/en/latest/
user/styles-understanding.html
```

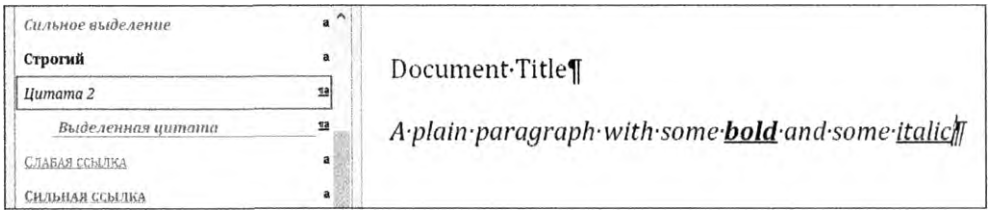


Рис. 15.7. Измененный документ restyled.docx

Запись документов Word

Введите в интерактивной оболочке следующие инструкции.

```
>>> import docx
>>> doc = docx.Document()
>>> doc.add_paragraph('Здравствуй, мир!')
<docx.text.Paragraph object at 0x0000000003B56F60>
>>> doc.save('helloworld.docx')
```

Чтобы создать собственный файл *.docx*, необходимо вызвать функцию `docx.Document()`, которая возвращает пустой объект `Document`. Метод `add_paragraph()` этого объекта добавляет новый абзац текста в документ и возвращает ссылку на объект `Paragraph`. Когда работа с текстом будет завершена, сохраните документ в файле, передав строку с именем файла методу `save()` объекта `Document`.

В данном примере в текущем каталоге создается файл *helloworld.docx*, вид которого показан на рис. 15.8.

Для добавления абзацев необходимо повторно вызывать метод `add_paragraph()`. Кроме того, можно добавить текст в конец существующего абзаца, вызвав для него метод `add_run()` и указав строку текста. Введите в интерактивной оболочке следующие инструкции.

```
>>> import docx
>>> doc = docx.Document()
>>> doc.add_paragraph('Здравствуй, мир!')
<docx.text.Paragraph object at 0x000000000366AD30>
>>> paraObj1 = doc.add_paragraph('Это второй абзац.')
>>> paraObj2 = doc.add_paragraph('Это еще один абзац.')
>>> paraObj1.add_run(' Этот текст добавляется во второй абзац.')
<docx.text.Run object at 0x0000000003A2C860>
>>> doc.save('multipleParagraphs.docx')
```

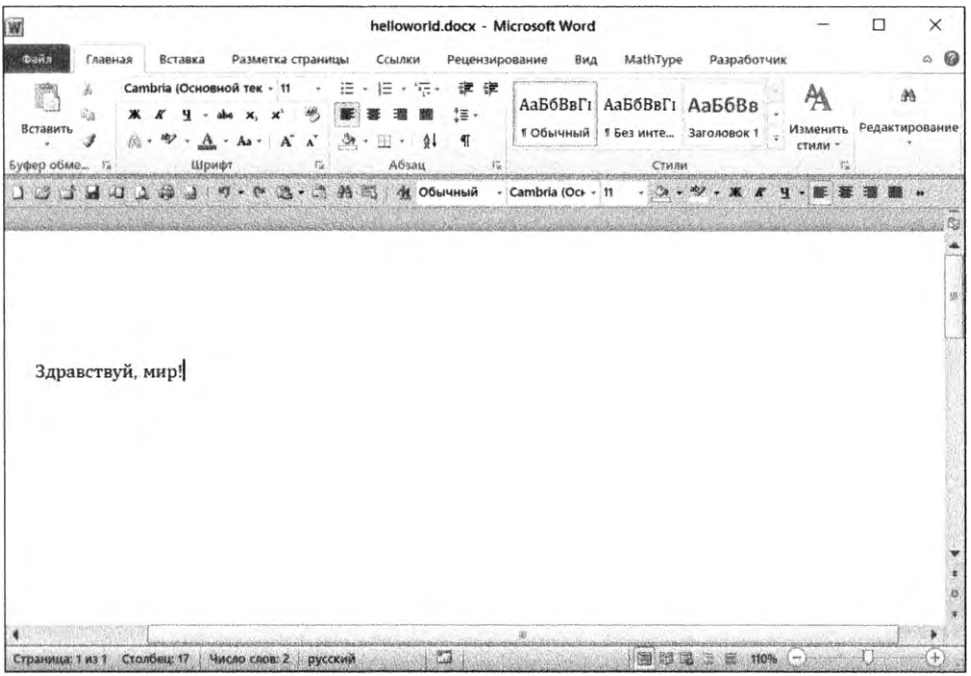


Рис. 15.8. Документ Word, созданный с помощью вызова `add_paragraph('Здравствуй, мир!')`

Результат показан на рис. 15.9. Обратите внимание на то, что строка `'Этот текст добавляется во второй абзац.'` добавляется в переменную `paraObj1`, которая представляет второй из абзацев документа. Функции `add_paragraph()` и `add_run()` возвращают объекты `Paragraph` и `Run` соответственно, что избавляет вас от лишних забот по извлечению необходимых фрагментов текста.

Имейте в виду, что в версии Python-Docx 0.8.10 новые объекты `Paragraph` можно добавлять только в конец документа, а новые объекты `Run` — только в конец абзаца.

Метод `save()` можно вызывать повторно для сохранения дополнительных изменений.

Оба метода, `add_paragraph()` и `add_run()`, поддерживают необязательный второй аргумент, содержащий строку с названием стиля объекта `Paragraph` или `Run` соответственно:

```
>>> doc.add_paragraph('Здравствуй, мир!', 'Title')
```

Эта инструкция добавляет абзац с текстом `'Здравствуй мир!'`, которые форматируется с использованием стиля `'Title'`.

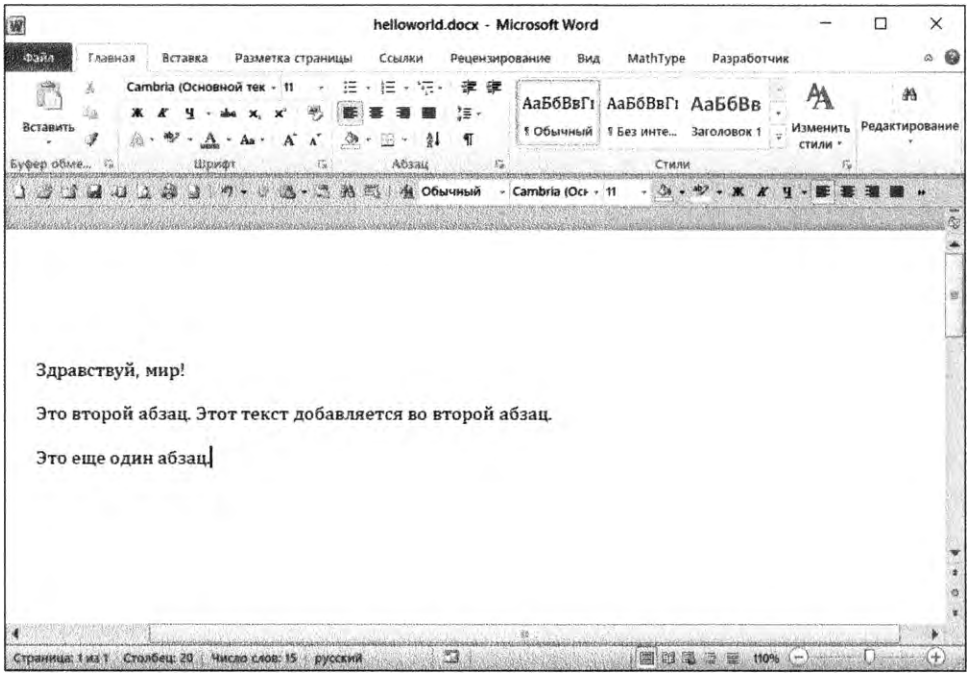


Рис. 15.9. Документ с несколькими добавленными объектами `Paragraph` *и* `Run`

Добавление заголовков

Вызов метода `add_heading()` приводит к добавлению абзаца, отформатированного с помощью одного из возможных стилей заголовков. Введите в интерактивной оболочке следующие инструкции.

```
>>> doc = docx.Document()
>>> doc.add_heading('Header 0', 0)
<docx.text.Paragraph object at 0x00000000036CB3C8>
>>> doc.add_heading('Header 1', 1)
<docx.text.Paragraph object at 0x00000000036CB630>
>>> doc.add_heading('Header 2', 2)
<docx.text.Paragraph object at 0x00000000036CB828>
>>> doc.add_heading('Header 3', 3)
<docx.text.Paragraph object at 0x00000000036CB2E8>
>>> doc.add_heading('Header 4', 4)
<docx.text.Paragraph object at 0x00000000036CB3C8>
>>> doc.save('headings.docx')
```

Аргументами метода `add_heading()` служат строка заголовка и целое число в диапазоне от `0` до `4`. Значению `0` соответствует стиль заголовка `Title`, используемый в начале документа в качестве заголовка верхнего уровня. Целым числам от `1` до `4` соответствуют различные уровни заголовков

в порядке убывания значимости. Функция `add_heading()` возвращает объект `Paragraph`, избавляя вас от необходимости извлекать абзацы из документа отдельной операцией.

Вид полученного документа *headings.docx* показан на рис. 15.10.

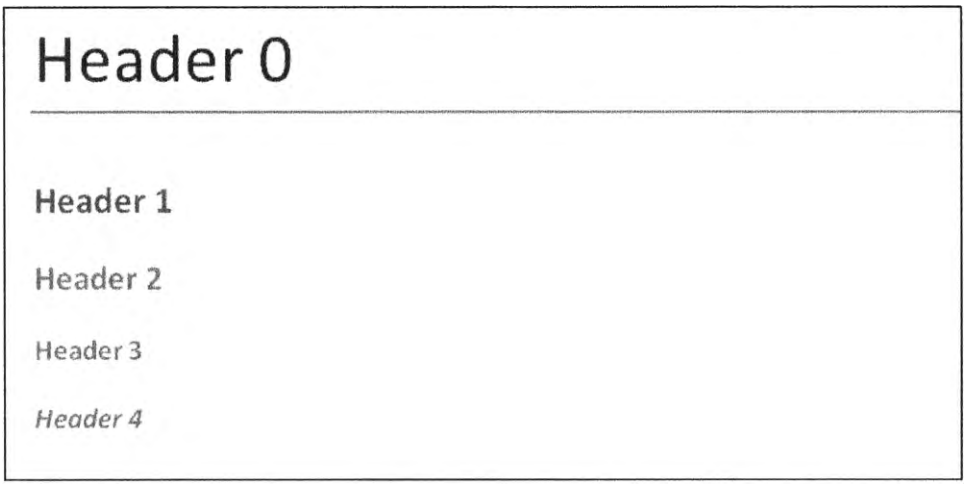


Рис. 15.10. Документ headings.docx с заголовками, соответствующими уровням от 0 до 4

Добавление разрывов строк и страниц

Чтобы добавить разрыв строки (а не начинать новый абзац), можно вызвать метод `add_break()` для того объекта `Run`, после которого требуется вставить разрыв строки. Если же необходимо добавить разрыв страницы, то функции `add_break()` следует передать аргумент `docx.enum.text.WD_BREAK.PAGE`, как показано ниже.

```
>>> doc = docx.Document()
>>> doc.add_paragraph('Текст на первой странице')
<docx.text.Paragraph object at 0x0000000003785518>
❶ >>> doc.paragraphs[0].runs[0].add_break(docx.enum.text.WD_BREAK.PAGE)
>>> doc.add_paragraph('Текст на второй странице')
<docx.text.Paragraph object at 0x00000000037855F8>
>>> doc.save('twoPage.docx')
```

В результате создается двухстраничный документ Word со строкой `'Текст на первой странице'` на первой странице и строкой `'Текст на второй странице'` — на второй. Несмотря на то что на первой странице остается еще много свободного места, мы принудительно начинаем следующий абзац с новой страницы, вставляя разрыв страницы после первого объекта `Run` первого абзаца ❶.

Добавление изображений

Метод `add_picture()` объекта `Document` позволяет добавить изображение в конец документа. Предположим, в текущем каталоге находится файл *zophie.png*. Чтобы добавить в конец документа изображение *zophie.png* шириной 1 дюйм и высотой 4 сантиметра (Word распознает как метрические, так и неметрические единицы измерения), введите следующие инструкции.

```
>>> doc.add_picture('zophie.png', width=docx.shared.Inches(1),
    height=docx.shared.Cm(4))
<docx.shape.InlineShape object at 0x00000000036C7D30>
```

Первый аргумент — это строка, задающая имя файла изображения. Необязательные именованные аргументы `width` и `height` задают ширину и высоту изображения в документе. Если их опустить, то значения этих аргументов будут определяться размерами самого изображения.

Если требуется указать высоту и ширину изображения в привычных для вас единицах — дюймах или сантиметрах, то используйте функции `docx.shared.Inches()` и `docx.shared.Cm()`.

Создание документов PDF на основе документов Word

Модуль PyPDF2 не позволяет создавать документы PDF напрямую, но можно генерировать PDF-файлы с помощью Python, если вы работаете в Windows и у вас установлено приложение Microsoft Word. Вам понадобится пакет Pywin32, который следует установить с помощью команды `pip install --user -U pywin32==224`. С помощью этого пакета и модуля `docx` можно создавать документы Word, а затем преобразовывать их в PDF-файлы, используя приведенный ниже сценарий.

Откройте в файловом редакторе новую вкладку, введите следующий код и сохраните программу в файле *convertWordToPDF.py*.

```
# Этот сценарий выполняется только в Windows, и у вас
# должно быть установлено приложение Microsoft Word
import win32com.client     # устанавливается командой
                           # "pip install pywin32==224"
import docx
wordFilename = 'ваш_документ_word.docx'
pdfFilename  = 'ваш_документ_pdf.pdf'

doc = docx.Document()
# Здесь должен быть код для создания документа Word
doc.save(wordFilename)

wdFormatPDF = 17    # числовой код Word для документов PDF
wordObj = win32com.client.Dispatch('Word.Application')
```

```
docObj = wordObj.Documents.Open(wordFilename)
docObj.SaveAs(pdfFilename, FileFormat=wdFormatPDF)
docObj.Close()
wordObj.Quit()
```

Чтобы написать программу, которая создает PDF-файлы с вашим собственным содержимым, необходимо воспользоваться модулем `docx` для создания документа Word, а затем использовать модуль `win32com.client` из пакета Pywin32 для преобразования документа в PDF-файл. Замените строку комментария

```
# Здесь должен быть код для создания документа Word
```

вызовами функций модуля `docx`, которые создают ваше собственное содержимое в документе Word для PDF-файла.

Такой способ создания PDF-файлов выглядит достаточно сложным, но для профессиональных программных решений это не удивительно.

Резюме

Текстовая информация хранится не только в простых текстовых файлах. Чаще приходится работать с документами PDF и Word. Для чтения PDF-файлов можно использовать модуль PyPDF2. К сожалению, ввиду сложности формата PDF текст таких файлов не всегда распознается правильно, а некоторые PDF-документы вообще невозможно прочесть. Остается лишь надеяться, что в будущих версиях модуля PyPDF2 будет обеспечена более полная поддержка формата PDF.

В этом смысле документы Word более надежны, и их можно читать с помощью модуля `docx` из пакета Python-Docx. Для редактирования текста в документах Word предназначены объекты `Paragraph` и `Run`. Им можно назначать стили форматирования, хотя возможности выбора стилей ограничиваются лишь стандартным набором, а также стилями, уже существующими в документе. Разрешается добавлять новые абзацы, заголовки, разрывы строк и страниц, а также изображения, но только в конец документа.

Многие ограничения при работе с документами PDF и Word обусловлены тем, что эти форматы предназначены в первую очередь для удобства чтения и не ориентированы на анализ текста программными средствами. В следующей главе мы рассмотрим два других популярных формата, используемых для хранения информации: JSON и CSV. Вы увидите, что работать с этими форматами в Python гораздо легче.

Контрольные вопросы

1. В функцию `PdfFileReader()` модуля PyPDF2 *не* передается строка с именем PDF-файла. Что же в таком случае ей передается?
2. В каких режимах должны открываться объекты `File` для функций `PdfFileReader()` и `PdfFileWriter()`?
3. Как получить объект `Page` для страницы 5 из объекта `PdfFileReader`?
4. В какой переменной объекта `PdfFileReader` хранится количество страниц PDF-документа?
5. Если PDF-документ объекта `PdfFileReader` зашифрован с помощью пароля `'swordfish'`, то что нужно сделать, прежде чем вы сможете получить из него объекты `Page`?
6. Какие методы предназначены для поворота страницы PDF-документа?
7. Какой метод вернет объект `Document` для файла *demo.docx*?
8. В чем разница между объектами `Paragraph` и `Run`?
9. Как получить список объектов `Paragraph` для объекта `Document`, который хранится в переменной `doc`?
10. У какого объекта есть атрибуты `bold`, `underline`, `italic`, `strike` и `outline`?
11. В чем разница между значениями `True`, `False` и `None` атрибута `bold`?
12. Как создать объект `Document` для нового документа Word?
13. Как добавить абзац с текстом `'Hello, there!'` в объект `Document`, хранящийся в переменной `doc`?
14. Какие целочисленные значения представляют уровни заголовков, доступные в документах Word?

Учебные проекты

Чтобы закрепить полученные знания на практике, напишите программы для предложенных ниже задач.

PDF-паранойя

Используя функцию `os.walk()` из главы 10, напишите сценарий, который выбирает все PDF-файлы в папке (и всех ее подпапках) и защищает их паролей, переданным в командной строке. Сохраните каждый зашифрованный PDF-файл, добавляя к исходному имени файла суффикс *_encrypted.pdf*. Прежде чем удалять исходный файл, попытайтесь прочитать и дешифровать результирующий файл, чтобы убедиться в корректности применения пароля.

Затем напишите программу, которая находит все зашифрованные PDF-файлы в папке (и всех ее подпапках) и создает дешифрованную копию каждого из них, используя предоставленный пароль. В случае, если пароль не подходит, программа должна выводить предупреждающее сообщение и переходить к обработке следующего файла.

Персонализированные приглашения в виде документов Word

Предположим, имеется текстовый файл *guests.txt* со списком гостей. Каждое имя в этом файле записано в отдельной строке.

```
Prof. Plum
Miss Scarlet
Col. Mustard
Al Sweigart
RoboCop
```

Напишите программу, которая генерирует документ Word, содержащий персонализированные приглашения наподобие того, которое показано на рис. 15.11.

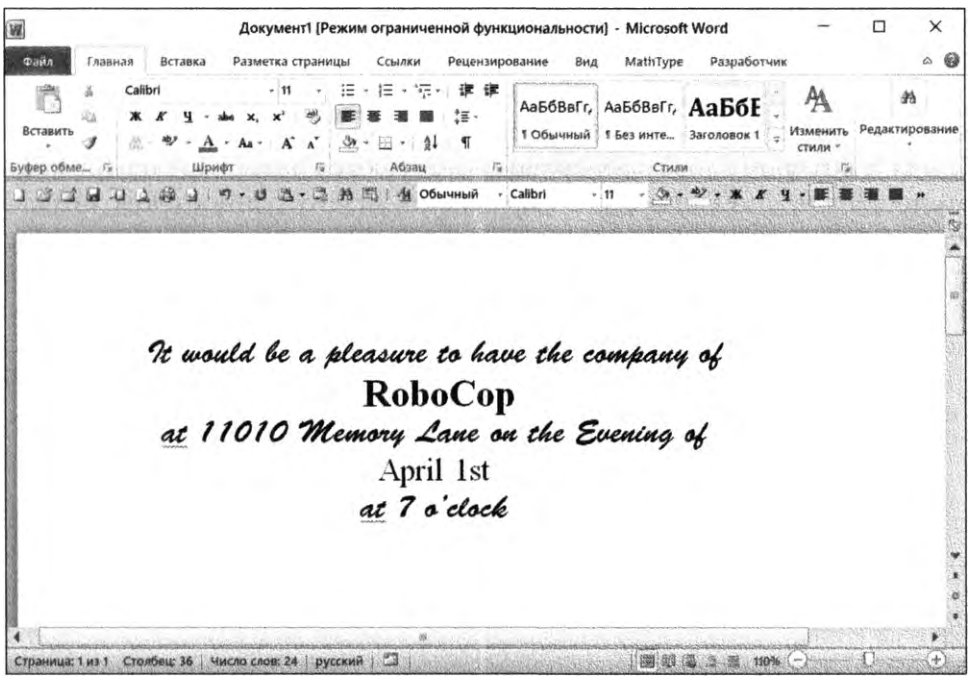


Рис. 15.11. Документ Word, полученный с помощью сценария, генерирующего персонализированные приглашения

Поскольку модуль Python-Docx может использовать только те стили, которые уже существуют в документе Word, вам придется сначала добавить эти стили в пустой файл Word, а затем открыть файл с помощью модуля Python-Docx. Результирующий документ Word должен содержать по одному приглашению на страницу, поэтому вызывайте метод `add_break()` для добавления разрывов страниц за последним абзацем каждого приглашения. В таком случае, чтобы напечатать сразу все приглашения, понадобится открыть всего один документ Word.

Готовый образец файла *guests.txt* содержится в архиве примеров книги (см. введение).

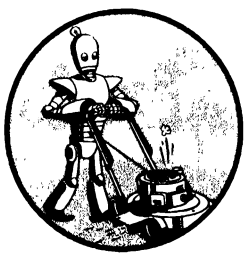
Взлом паролей PDF-файлов методом грубой силы

Предположим, имеется зашифрованный PDF-файл, пароль доступа к которому вы забыли, но помните, что это какое-то слово на английском языке. Угадывание забытого пароля — довольно утомительная задача. Вместо этого можно написать программу, которая дешифрует PDF-файл, перебирая все возможные слова до тех пор, пока не будет найдено слово, совпадающее с паролем. Такой подход к взлому паролей называют *атакой методом грубой силы*. Загрузите текстовый файл *dictionary.txt*, который находится в архиве примеров книги (см. введение). В этом словаре содержится свыше 44 тысяч английских слов, по одному слову в строке.

Используя свои навыки в чтении файлов, приобретенные в главе 9, создайте список слов, прочитав содержимое файла словаря. Затем организуйте цикл, в котором слова поочередно извлекаются из словаря и передаются методу `decrypt()`. Если метод возвращает `0`, значит, данное слово не совпадает с паролем, и программа должна переходить к следующему слову. Если же метод `decrypt()` возвращает значение `1`, то программа должна выйти из цикла и вывести строку пароля. Каждое слово из словаря следует проверять в верхнем и нижнем регистрах. (На ноутбуке автора перебор всех 88 тысяч вариантов пароля в верхнем и нижнем регистрах занимает около двух минут. Вот почему нельзя использовать в качестве паролей простые слова.)

16

РАБОТА С CSV-ФАЙЛАМИ И ДАННЫМИ В ФОРМАТЕ JSON



В главе 15 вы научились извлекать текст из документов Word и PDF. Файлы этого типа имеют бинарный формат, и для доступа к содержащимся в них данным приходится применять специальные модули Python. В то же время файлы CSV и JSON хранятся в виде простого текста. Их содержимое можно просматривать в любом текстовом редакторе, в том числе в редакторе Mu. Тем не менее в Python имеются специальные модули `csv` и `json`, которые содержат функции, упрощающие работу с этими форматами.

CSV (Comma-Separated Values) — формат несложных электронных таблиц, хранящихся в обычных текстовых файлах. Модуль `csv` позволяет выполнять синтаксический анализ (парсинг) CSV-файлов.

JSON (JavaScript Object Notation) — это формат, предназначенный для хранения кода JavaScript в обычных текстовых файлах. Для работы с файлами JSON не нужно знать JavaScript, но знакомство с этим форматом будет для вас полезным, так как он применяется во многих веб-приложениях.

Модуль csv

Каждая строка CSV-файла представляет строку электронной таблицы, ячейки в которой разделены запятыми. Например, электронная таблица, хранящаяся в файле *example.xlsx* (доступен в архиве примеров книги; см. введение), будет иметь следующий вид в формате CSV.

```
05.04.2015 13:34,Яблоки,73
05.04.2015 3:41,Вишни,85
06.04.2015 12:46,Груши,14
08.04.2015 8:59,Апельсины,52
10.04.2015 2:07,Яблоки,152
10.04.2015 18:10,Бананы,23
10.04.2015 2:40,Клубника,98
```

Эту таблицу мы будем использовать для выполнения примеров в интерактивной оболочке. Можете либо загрузить готовый файл из архива примеров книги, либо самостоятельно ввести текст в каком-нибудь текстовом редакторе и сохранить его в файле *example.csv*.

CSV — достаточно простой формат, в котором нет многих возможностей, доступные при работе с электронными таблицами Excel:

- отсутствуют типы значений — все значения являются строками;
- нет настроек размера и цвета шрифта;
- нельзя иметь несколько рабочих листов;
- нельзя задавать ширину и высоту ячеек;
- нельзя объединять ячейки;
- нельзя внедрять изображения и диаграммы.

Достоинство CSV-файлов — их простота. Они поддерживаются во многих приложениях, их можно просматривать в текстовых редакторах (включая Mu), и они позволяют легко представлять табличные данные. По сути, формат CSV — это обычный текстовый файл, значения в котором разделены запятыми.

Поскольку CSV-файлы хранятся в текстовом виде, может возникнуть соблазн читать их содержимое в строковом виде, а затем обрабатывать

полученную строку, используя методики, изученные в главе 9. Например, поскольку столбцы данных разделены в CSV-файле запятыми, можно попытаться извлекать значения, содержащиеся в каждой строке, с помощью метода `split(',')`. Но не всякая запятая в CSV-файле соответствует границе между двумя ячейками. Кроме того, в CSV-файлах могут встречаться собственные экранированные символы, позволяющие включать запятые *непосредственно в строковые значения*. Такие экранированные символы не обрабатываются методом `split()`. С учетом этих потенциальных ловушек лучше всегда применять модуль `csv` для чтения и записи CSV-файлов.

Объекты reader

Чтобы прочитать данные из CSV-файла, необходимо создать объект `reader`, который служит итератором строк CSV-файла. Убедитесь в том, что в текущем каталоге находится файл *example.csv*, и введите в интерактивной оболочке следующие инструкции.

```
❶ >>> import csv
❷ >>> exampleFile = open('example.csv')
❸ >>> exampleReader = csv.reader(exampleFile)
❹ >>> exampleData = list(exampleReader)
❺ >>> exampleData
  [['05.04.2015 13:34', 'Яблоки', '73'],
  ['05.04.2015 3:41', 'Вишни', '85'],
  ['06.04.2015 12:46', 'Груши', '14'],
  ['08.04.2015 8:59', 'Апельсины', '52'],
  ['10.04.2015 2:07', 'Яблоки', '152'],
  ['10.04.2015 18:10', 'Бананы', '23'],
  ['10.04.2015 2:40', 'Клубника', '98']]
```

Модуль `csv` входит в стандартную библиотеку Python, поэтому мы можем просто импортировать его ❶ без предварительной установки.

Прежде чем читать CSV-файл с помощью модуля `csv`, необходимо открыть его с помощью функции `open()` ❷, как это делается при открытии любого текстового файла. Но вместо того чтобы вызывать метод `read()` или `readlines()` для объекта `File`, возвращаемого функцией `open()`, мы передаем этот объект функции `csv.reader()` ❸. Она возвращает объект `reader`, который мы будем использовать в дальнейшем. Обратите внимание на то, что функции `csv.reader()` передается объект, а не просто строка с именем файла.

Самый простой способ получить доступ к значениям в объекте `reader` — передать этот объект функции `list()` ❹, которая преобразует его в обычный список Python. В результате мы получаем список списков, который можно сохранить в переменной `exampleData`. Чтобы просмотреть содержимое списка, введите его имя в интерактивной оболочке ❺.

Теперь, когда у вас есть CSV-файл в виде списка списков, можно обращаться к значениям таблицы с помощью выражения `exampleData[`*строка*`][`*столбец*`]`, где *строка* — индекс одного из списков в объекте `exampleData`, а *столбец* — индекс нужного элемента из этого списка. Введите в интерактивной оболочке следующие инструкции.

```
>>> exampleData[0][0]
'05.04.2015 13:34'
>>> exampleData[0][1]
'Яблоки'
>>> exampleData[0][2]
'73'
>>> exampleData[1][1]
'Вишни'
>>> exampleData[6][1]
'Клубника'
```

Выражение `exampleData[0][0]` возвращает первую строку первого списка, выражение `exampleData[0][2]` — третью строку в этом же списке и т.д.

Чтение данных из объекта `reader` в цикле `for`

В случае больших CSV-файлов удобнее использовать объект `reader` в цикле `for`. Это позволяет не загружать сразу весь файл в память компьютера. Например, введите в интерактивной оболочке следующие инструкции.

```
>>> import csv
>>> exampleFile = open('example.csv')
>>> exampleReader = csv.reader(exampleFile)
>>> for row in exampleReader:
        print('Строка #' + str(exampleReader.line_num) + ' ' +
              str(row))

Строка #1 ['05.04.2015 13:34', 'Яблоки', '73']
Строка #2 ['05.04.2015 3:41', 'Вишни', '85']
Строка #3 ['06.04.2015 12:46', 'Груши', '14']
Строка #4 ['08.04.2015 8:59', 'Апельсины', '52']
Строка #5 ['10.04.2015 2:07', 'Яблоки', '152']
Строка #6 ['10.04.2015 18:10', 'Бананы', '23']
Строка #7 ['10.04.2015 2:40', 'Клубника', '98']
```

Импортировав модуль `csv` и создав объект `reader` из CSV-файла, мы организуем цикл по строкам в объекте `reader`. Каждая строка — это список значений, каждое из которых представляет отдельную ячейку таблицы.

С помощью функции `print()` мы выводим номер строки и ее содержимое. Для получения номера текущей строки используется переменная `line_num` объекта `reader`.

Цикл по объекту `reader` может выполняться только один раз. Для повторного чтения CSV-файла необходимо заново создать объект `reader`, вызвав функцию `csv.reader()`.

Объекты `writer`

Объект `writer` позволяет записывать данные в CSV-файл. Для создания объекта `writer` предназначена функция `csv.writer()`. Введите в интерактивной оболочке следующие инструкции.

```
>>> import csv
❶ >>> outputFile = open('output.csv', 'w', newline='')
❷ >>> outputWriter = csv.writer(outputFile)
>>> outputWriter.writerow(['тушенка', 'яйца', 'бекон', 'ветчина'])
28
>>> outputWriter.writerow(['Привет, мир!', 'яйца', 'бекон', 'ветчина'])
35
>>> outputWriter.writerow([1, 2, 3.141592, 4])
16
>>> outputFile.close()
```

Прежде всего необходимо вызвать функцию `open()` и передать ей аргумент `'w'` для открытия файла в режиме записи ❶. В результате создается файловый объект, который затем передается функции `csv.writer()` ❷ для создания объекта `writer`.

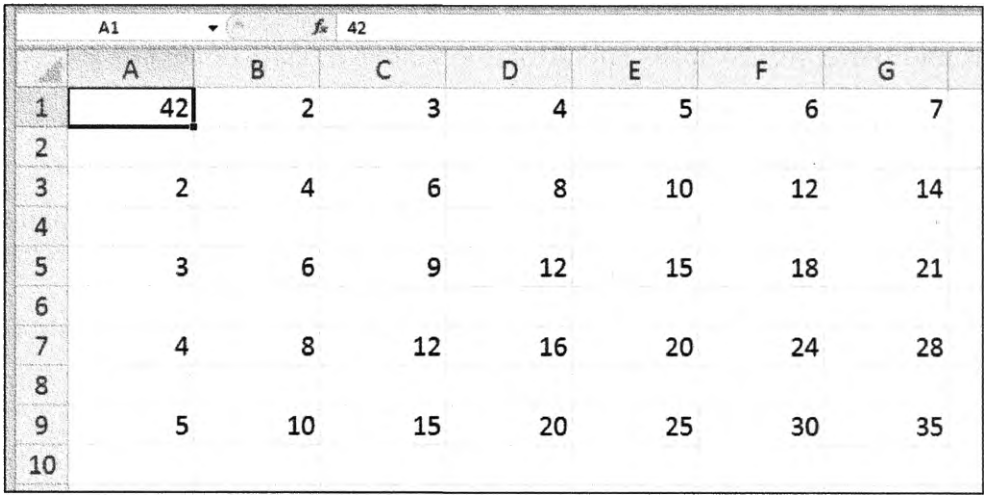
Если вы работаете в Windows, то в качестве значения именованного аргумента `newline` функции `open()` следует передавать пустую строку. Если этого не сделать, то в силу технических причин, обсуждение которых выходит за рамки книги, в файле *output.csv* появятся лишние пустые строки (рис. 16.1).

Метод `writerow()` объекта `writer` получает аргумент в виде списка. Каждое значение этого списка помещается в отдельную ячейку выходного CSV-файла. Метод возвращает число символов, записанных в файл для данной строки таблицы (включая символы новой строки).

Вот как выглядит содержимое файла *output.csv*, созданного в данном примере.

```
тушенка,яйца,бекон,ветчина
"Привет, мир!",яйца,бекон,ветчина
1,2,3.141592,4
```

Обратите внимание на то, как объект `writer` автоматически экранирует кавычками запятую в строке `'Привет, мир!'` CSV-файла. Модуль `csv` избавляет вас от самостоятельной обработки подобных специальных случаев.



	A	B	C	D	E	F	G
1	42	2	3	4	5	6	7
2							
3	2	4	6	8	10	12	14
4							
5	3	6	9	12	15	18	21
6							
7	4	8	12	16	20	24	28
8							
9	5	10	15	20	25	30	35
10							

Рис. 16.1. Если вы забудете передать именованный аргумент `newline=''` функции `open()`, то при выводе содержимого CSV-файла появятся лишние строки

Именованные аргументы `delimiter` и `lineterminator`

Предположим, вы хотите использовать в качестве разделителя ячеек не запятую, а символ табуляции и при этом удвоить междустрочный интервал. Введите в интерактивной оболочке следующие инструкции.

```
>>> import csv
>>> csvFile = open('example.tsv', 'w', newline='')
>>> csvWriter = csv.writer(csvFile, delimiter='\t',
                           lineterminator='\n\n')
>>> csvWriter.writerow(['яблоки', 'апельсины', 'виноград'])
27
>>> csvWriter.writerow(['яйца', 'бекон', 'ветчина'])
20
>>> csvWriter.writerow(['тушенка', 'тушенка', 'тушенка',
                        'тушенка', 'тушенка', 'тушенка'])
49
>>> csvFile.close()
```

В результате разделители данных и строк в файле изменятся. *Разделитель данных* — это символ, используемый для разделения значений в строке. По умолчанию в CSV-файлах в качестве разделителя используется запятая. *Разделитель строк* — это символ, добавляемый в конце строки таблицы. По умолчанию в качестве разделителя строк используется символ новой строки. Вместо этих символов можно использовать другие, передав соответствующие значения в функцию `csv.writer()` в качестве именованных аргументов `delimiter` и `lineterminator`.

В результате передачи аргументов `delimeter='\t'` и `lineterminator='\n\n'` ❶ разделителем данных становится символ табуляции, а разделителем строк — удвоенный символ новой строки. После этого мы трижды вызываем функцию `writerow()`, записывая в таблицу три строки.

Запустив программу, мы получим файл *example.tsv* следующего вида.

```
яблоки  апельсины       виноград

яйца    бекон   ветчина

тушенка тушенка тушенка тушенка тушенка тушенка
```

В связи с тем, что ячейки таблицы теперь разделены символами табуляции, мы используем для файла расширение *.tsv*.

Объекты DictReader и DictWriter

В случае CSV-файлов, содержащих строки заголовков, удобнее работать с объектами `DictReader` и `DictWriter`, а не с объектами `reader` и `writer`.

Объекты `reader` и `writer` читают и записывают строки CSV-файла с помощью списков. Объекты `DictReader` и `DictWriter` делают то же самое, но вместо списков в них используются словари, а первая строка CSV-файла содержит ключи этих словарей.

Загрузите из архива примеров книги (см. введение) файл *exampleWithHeader.csv*. Это такой же файл, как *example.csv*, за исключением того, что в первой строке содержатся заголовки столбцов `'Время'`, `'Фрукт'` и `'Количество'`. Чтобы прочитать файл, введите в интерактивной оболочке следующие инструкции.

```
>>> import csv
>>> exampleFile = open('exampleWithHeader.csv')
>>> exampleDictReader = csv.DictReader(exampleFile)
>>> for row in exampleDictReader:
...     print(row['Время'], row['Фрукт'], row['Количество'])
...
4/5/2015 13:34 Яблоки 73
4/5/2015 3:41 Вишни 85
4/6/2015 12:46 Груши 14
4/8/2015 8:59 Апельсины 52
4/10/2015 2:07 Яблоки 152
4/10/2015 18:10 Бананы 23
4/10/2015 2:40 Клубника 98
```

В цикле объект `DictReader` записывает в переменную `row` объект словаря с ключами, полученными из заголовков в первой строке. (Технически переменной `row` присваивается объект `OrderedDict`, который можно

использовать так же, как и словарь.) Благодаря объекту DictReader вам не нужен дополнительный код для пропуска первой строки заголовка, так как объект DictReader делает это за вас.

Если попытаться использовать объекты DictReader с файлом *example.csv*, у которого нет заголовков в первой строке, то ключами словаря станут строки '5/5/2015 13:34', 'Яблоки' и '73'. Чтобы избежать этого, можно предоставить функции DictReader() второй аргумент, содержащий придуманные нами названия столбцов.

```
>>> import csv
>>> exampleFile = open('example.csv')
>>> exampleDictReader = csv.DictReader(exampleFile,
                                       ['Время', 'Фрукт', 'Количество'])
>>> for row in exampleDictReader:
...     print(row['Время'], row['Фрукт'], row['Количество'])
...
4/5/2015 13:34 Яблоки 73
4/5/2015 3:41 Вишни 85
4/6/2015 12:46 Груши 14
4/8/2015 8:59 Апельсины 52
4/10/2015 2:07 Яблоки 152
4/10/2015 18:10 Бананы 23
4/10/2015 2:40 Клубника 98
```

Поскольку первая строка файла *example.csv* не содержит никакого заголовка в каждом столбце, мы создаем собственные заголовки: 'Время', 'Фрукт' и 'Количество'.

Объекты DictWriter используют словари для создания CSV-файлов.

```
>>> import csv
>>> outputFile = open('output.csv', 'w', newline='')
>>> outputDictWriter = csv.DictWriter(outputFile,
...       ['Имя', 'Домашний питомец', 'Телефон'])
>>> outputDictWriter.writeheader()
>>> outputDictWriter.writerow({'Имя': 'Алиса',
...       'Домашний питомец': 'кот', 'Телефон': '555-1234'}))
20
>>> outputDictWriter.writerow({'Имя': 'Боб',
...       'Телефон': '555-9999'}))
15
>>> outputDictWriter.writerow({'Телефон': '555-5555',
...       'Имя': 'Кэрол', 'Домашний питомец': 'собака'}))
23
>>> outputFile.close()
```

Если нужно, чтобы файл содержал строку заголовка, запишите эту строку, вызвав метод writeheader(). В противном случае пропустите вызов,

чтобы не включать строку заголовка в файл. Затем нужно записать каждую строку CSV-файла с помощью метода `writerow()`, передавая ему словарь, который использует заголовки в качестве ключей и содержит данные, предназначенные для записи в файл.

Файл *output.csv*, создаваемый программой, выглядит следующим образом.

```
Имя,Домашний питомец,Телефон
Алиса,кот,555-1234
Боб,,555-9999
Кэрол,собака,555-5555
```

Обратите внимание на то, что порядок пар "ключ — значение" в словарях, передаваемых методу `writerow()`, не имеет значения: они записываются в порядке ключей, переданных функции `DictWriter()`. Например, в последней строке мы передали ключ и значение `'Телефон'` перед ключами и значениями `'Имя'` и `'Домашний питомец'`, номер телефона по-прежнему отображается в строке последним.

Также следует отметить, что для отсутствующих ключей в CSV-файл вставляются пустые строки.

Проект: удаление заголовков из CSV-файла

Предположим, вам предстоит выполнить рутинную работу по удалению первой строки из нескольких сотен CSV-файлов. Возможно, вы собираетесь передавать эти файлы какой-то программе, которой требуются только данные без заголовков столбцов. Можно было бы открыть каждый файл в Excel, удалить первую строку таблицы и заново сохранить файл, но на это ушло бы несколько часов. Лучше написать программу, которая проделает всю работу вместо вас.

Программа должна будет открывать все файлы с расширением *.csv* в текущем каталоге, читать содержимое каждого CSV-файла и перезаписывать его без первой строки в файл с тем же именем. В результате старое содержимое CSV-файла будет заменено новым, в котором заголовки столбцов таблицы отсутствуют.

Предупреждение

Всякий раз, когда вы пишете программу, изменяющую файлы, не забывайте создавать их резервные копии хотя бы для того, чтобы застраховать себя на случай, если что-то пойдет не так. Так вы застрахуете себя от ошибочного удаления исходных файлов.

Программа должна делать следующее:

1) находить всех CSV-файлы в текущем каталоге;
2) считывать содержимое каждого файла;
3) записывать содержимое без первой строки в новый CSV-файл.

Это означает, что программа будет выполнять следующие операции:

1) проходить в цикле по списку файлов, возвращаемому функцией `os.listdir()`, оставляя только CSV-файлы;
2) создавать объект `reader` и читать содержимое файла, используя атрибут `line_num` для определения того, какую строку следует пропустить;
3) создавать объект `writer` и записывать прочитанные данные в новый файл.

Откройте в файловом редакторе новое окно и сохраните программу в файле *removeCsvHeader.py*.

Шаг 1. Цикл по всем CSV-файлам

Первое, что должна сделать программа, — организовать цикл по всем CSV-файлам, находящимся в текущем каталоге. Введите в файл *removeCsvHeader.py* следующий код.

```
#! python3
# removeCsvHeader.py - удаляет заголовки
# из всех CSV-файлов в текущем каталоге

import csv, os

os.makedirs('headerRemoved', exist_ok=True)

# Цикл по всем файлам в текущем каталоге
for csvFilename in os.listdir('.'):
    if not csvFilename.endswith('.csv'):
❶       continue    # оставляем только CSV-файлы

    print('Удаление заголовка из файла ' + csvFilename + '...')

    # СДЕЛАТЬ: прочитать CSV-файл (без первой строки)

    # СДЕЛАТЬ: записать CSV-файл
```

Функция `os.makedirs()` создает подпапку *headerRemoved*, в которую будут записаны все CSV-файлы без заголовков. Цикл `for` по элементам списка `os.listdir('.')` проходит по *всем* файлам в текущем каталоге, поэтому в начале цикла необходимо добавить код, обеспечивающий пропуск

файлов, не имеющих расширение *.csv*. Если в цикле встречается такой файл, то инструкция `continue` ❶ обеспечивает переход к следующему файлу.

Далее идет вызов функции `print()`, которая выводит на экран имя текущего CSV-файла, что позволяет контролировать ход выполнения программы. Комментарии `'СДЕЛАТЬ'` напоминают о том, что нам предстоит написать.

Шаг 2. Чтение CSV-файла

В действительности программа не удаляет первую строку из CSV-файла. Вместо этого она создает копию файла, но уже без первой строки. Поскольку имя файла-копии совпадает с именем исходного файла, он перезаписывает оригинал.

В программе необходимо отслеживать, является ли текущая строка в цикле первой. Добавьте в файл *removeCsvHeader.py* код, выделенный полужирным шрифтом.

```
#! python3
# removeCsvHeader.py - удаляет заголовки
# из всех CSV-файлов в текущем каталоге

-- Опущено --

    # Прочитать CSV-файл (без первой строки)
    csvRows = []
    csvFileObj = open(csvFilename)
    readerObj = csv.reader(csvFileObj)
    for row in readerObj:
        if readerObj.line_num == 1:
            continue    # пропустить первую строку
        csvRows.append(row)
    csvFileObj.close()

    # СДЕЛАТЬ: записать CSV-файл
```

Для отслеживания номера строки CSV-файла, которая читается в данный момент, можно использовать атрибут `line_num` объекта `reader`. Другой цикл `for` проходит по строкам, возвращаемым объектом `reader`, и все строки, кроме первой, присоединяются к списку `csvRows`.

В цикле `for` программа проверяет, является ли значение атрибута `readerObj.line_num` равным `1`. Если это так, то инструкция `continue` осуществляет переход к следующей строке, не присоединяя текущую строку к списку `csvRows`. Для всех последующих строк условие не будет выполняться, и они будут присоединяться к указанному списку.

Шаг 3. Запись CSV-файла без первой строки

Теперь, когда в списке `csvRows` содержатся все строки, кроме первой, его нужно записать в CSV-файл, который будет находиться в подпапке *headerRemoved*. Добавьте в файл *removeCsvHeader.py* код, выделенный полужирным шрифтом.

```
  #! python3
  # removeCsvHeader.py - удаляет заголовки
  # из всех CSV-файлов в текущем каталоге
  -- Опущено --

  # Цикл по всем файлам в текущем каталоге
❶ for csvFilename in os.listdir('.'):
      if not csvFilename.endswith('.csv'):
          continue    # оставляем только CSV-файлы

      -- Опущено --

      # Запись CSV-файла
      csvFileObj = open(os.path.join('headerRemoved', csvFilename),
                        'w', newline='')
      csvWriter = csv.writer(csvFileObj)
      for row in csvRows:
          csvWriter.writerow(row)
      csvFileObj.close()
```

Объект `writer` записывает список `csvRows` в CSV-файл, находящийся в подпапке *headerRemoved*, используя в качестве имени файла переменную `csvFilename` (которую мы также использовали при чтении). В результате исходный файл будет перезаписан.

Создав объект `writer`, мы организуем цикл по всем спискам, хранящимся в переменной `csvRows`, и записываем каждый из них в файл.

После выполнения данного блока кода внешний цикл `for` ❶ перейдет к следующему файлу из списка `os.listdir('.')`. По окончании внешнего цикла программа завершается.

Чтобы протестировать программу, загрузите файл *removeCsvHeader.zip* из архива примеров книги (см. введение), распакуйте его содержимое в папку и запустите программу *removeCsvHeader.py* в этой папке. Вы должны получить следующие результаты.

```
Удаление заголовка из файла NAICS_data_1048.csv...
Удаление заголовка из файла NAICS_data_1218.csv...
-- Опущено --
Удаление заголовка из файла NAICS_data_9834.csv...
Удаление заголовка из файла NAICS_data_9986.csv...
```

Программа должна выводить имя файла всякий раз, когда из CSV-файла исключается первая строка.

Идеи для создания похожих программ

Аналогичные программы можно создать не только для CSV-файлов, но и для файлов Excel, поскольку в обоих случаях вы имеете дело с файлами электронных таблиц. Поэтому для вас не составит большого труда написать программы для решения следующих задач:

- сравнение данных, находящихся в разных строках CSV-файла или в разных CSV-файлах;
- копирование конкретных данных из CSV-файла в файл Excel и обратно;
- контроль допустимости данных или ошибок форматирования в CSV-файлах и вывод предупреждений об обнаруженных ошибках;
- чтение данных из CSV-файла с целью их использования в качестве входных данных для программ Python.

JSON и программные интерфейсы

JSON (JavaScript Object Notation) – популярный способ форматирования данных в виде одной строки текста. Этот формат применяется JavaScript-программами для записи структур данных и напоминает вывод, получаемый с помощью функции `pprint()` в Python. Для работы с данными в формате JSON знать JavaScript не нужно.

Вот пример представления данных в формате JSON.

```
{"name": "Zophie", "isCat": true, "miceCaught": 0,
 "napsTaken": 37.5, "felineIQ": null}
```

Знакомство с JSON будет полезным, поскольку этот формат используется многими веб-сайтами для обмена данными с программами в рамках *интерфейса прикладного программирования* (Application Programming Interface – API). Доступ через API аналогичен получению доступа к веб-странице посредством URL-адреса. Разница состоит в том, что данные, возвращаемые API-вызовами, форматируются (например, с использованием JSON) для чтения программами; человеку читать такие данные трудно.

Доступ к данным в формате JSON обеспечивают многие популярные сайты: Facebook, Твиттер, Yahoo, Google, Tumblr, Википедия, Flickr, Data.gov, Reddit, IMDb, Rotten Tomatoes, LinkedIn и др. Все они предлагают собственные программные интерфейсы (API). На некоторых из этих сайтов

необходимо предварительно зарегистрироваться (в подавляющем большинстве случаев бесплатно). Вам нужно найти документацию с описанием того, по каким URL-адресам программа должна направлять API-запросы для получения требуемых данных и в каком формате эти данные будут возвращаться. Такая документация предоставляется любым сайтом, у которого есть собственный API. Если на сайте имеется страница "Developers" ("Для разработчиков"), то начните поиск документации оттуда.

С помощью API можно писать программы, способные, в частности, решать следующие задачи.

- Автоматический сбор "сырых" (необработанных) данных с веб-сайтов. (Доступ через API — более удобный способ, чем загрузка веб-страниц и парсинг HTML-разметки с помощью модуля Beautiful Soup.)
- Автоматическая загрузка новых сообщений из учетной записи в социальной сети и их публикация в другой учетной записи. Например, можно получать публикации из Tumblr и пересылать их на Facebook.
- Создание "киноэнциклопедии" для своей личной коллекции фильмов путем сбора данных на сайтах IMDb, Rotten Tomatoes и в Википедии с последующим объединением этих данных в один текстовый файл, хранящийся на вашем компьютере.

С некоторыми примерами программных интерфейсов, реализующих поддержку JSON, можно познакомиться на сайте `https://automatetheboringstuff.com/list-of-json-apis.html`.

Отформатировать данные в виде наглядных строк можно не только с помощью JSON. Существует множество других похожих форматов, таких как XML (eXtensible Markup Language), TOML (Tom's Obvious, Minimal Language), YML (Yet another Markup Language), INI (Initialization) и даже устаревший ASN.1 (Abstract Syntax Notation One), которые позволяют представлять данные в виде удобочитаемого текста. В книге они не рассматриваются, потому JSON стал стандартом де-факто. Достаточно знать, что существуют сторонние модули Python, предназначенные для работы с этими форматами.

Модуль `json`

Модуль `json` выполняет всю работу по преобразованию данных из формата JSON в значения Python (и наоборот), предоставляя для этого функции `json.loads()` и `json.dumps()`. Формат JSON не обеспечивает хранение *всех* типов значений Python. Он позволяет хранить лишь следующие типы данных: строки, целые и вещественные числа, булевы значения, списки, словари и значение `NoneType`. Специфические объекты Python, такие как `File`, `reader` и `writer`, `Regex` или `WebElement`, не поддерживаются

Чтение данных JSON с помощью функции loads()

Чтобы транслировать строку, содержащую данные JSON, в формат Python, передайте ее функции `json.loads()`. Например, введите в интерактивной оболочке следующие инструкции.

```
>>> stringOfJsonData = '{"name": "Zophie", "isCat": true,
                         "miceCaught": 0, "felineIQ": null}'
>>> import json
>>> jsonDataAsPythonValue = json.loads(stringOfJsonData)
>>> jsonDataAsPythonValue
{'isCat': True, 'miceCaught': 0, 'name': 'Zophie', 'felineIQ': None}
```

После импорта модуля `json` можно вызвать функцию `loads()` и передать ей строку JSON-данных. Обратите внимание на то, что в JSON всегда используются двойные кавычки. Эта функция возвращает данные в виде словаря Python. Поскольку словари в Python не упорядочены, при выводе значения переменной `jsonDataAsPythonValue` пары "ключ – значение" могут появляться в произвольном порядке.

Запись данных JSON с помощью функции dumps()

Функция `json.dumps()` транслирует значение Python в строку формата JSON. Введите в интерактивной оболочке следующие инструкции.

```
>>> pythonValue={'isCat': True, 'miceCaught': 0, 'name': 'Zophie',
                 'felineIQ': None}
>>> import json
>>> stringOfJsonData = json.dumps(pythonValue)
>>> stringOfJsonData
'{"isCat": true, "felineIQ": null, "miceCaught": 0,
  "name": "Zophie" }'
```

Значением может быть один из следующих базовых типов данных Python: словарь, список, целое или вещественное число, строка, булево значение и `None`.

Проект: получение текущего прогноза погоды

Получить информацию о погоде не так уж сложно: открываете браузер, вводите в адресной строке URL-адрес сайта погоды (либо выполняете поиск таких сайтов и щелкаете на одной из ссылок), дожидаетесь, пока загрузится страница, пропускаете рекламу и т.д.

При этом вам приходится выполнять множество лишних действий, от которых можно избавиться, написав программу, загружающую прогноз погоды на несколько дней и выводящую его в виде простого текста.

Для загрузки данных из Интернета программа будет использовать модуль `requests`, рассмотренный в главе 12.

Программа должна делать следующее:

1) читать название населенного пункта, заданное в командной строке;
2) загружать погодные данные в формате JSON с сайта *OpenWeatherMap.org*;
3) преобразовывать строку данных JSON в структуру Python;
4) выводить прогноз погоды на сегодня и следующие два дня.

Это означает, что программа будет выполнять следующие операции:

1) объединять строки, хранящиеся в списке `sys.argv`, для определения местоположения;
2) вызывать функцию `requests.get()` для загрузки погодных данных;
3) вызывать функцию `json.loads()` для преобразования данных JSON в структуру Python;
4) выводить прогноз погоды.

Откройте в файловом редакторе новое окно и сохраните программу в файле *quickWeather.py*. Затем посетите сайт `https://openweathermap.org/api/` и создайте бесплатную учетную запись для получения *ключа API*, также называемого идентификатором приложения. Он представляет собой строковый код, который выглядит примерно так: `'30144aba38018987d84710d0e319281e'`. Служба OpenWeatherMap бесплатна, если только вы не планируете совершать более 60 API-вызовов в минуту. Держите ключ API в секрете; любой, кто его знает, может написать сценарий, использующий квоту вашей учетной записи.

Шаг 1. Определение местоположения с помощью аргумента командной строки

Входные данные для этой программы поступают из командной строки. Введите в файл *getOpenWeather.py* следующий код.

```
#! python3
# getOpenWeather.py - вывод прогноза погоды
# для местоположения из командной строки

APPID = 'ВАШ_КЛЮЧ'

import json, requests, sys

# Определение местоположения из аргументов командной строки
if len(sys.argv) < 2:
    print('Применение: getOpenWeather.py город код_страны')
```

```
    sys.exit()
location = ' '.join(sys.argv[1:])

# СДЕЛАТЬ: загрузить данные JSON с сайта OpenWeatherMap.org

# СДЕЛАТЬ: записать данные JSON в переменную Python
```

В Python аргументы командной строки хранятся в списке `sys.argv`. Переменной `APPID` присваивается ключ API вашей учетной записи. Без этого ключа ваши запросы к службе погоды не будут выполнены. После импорта модулей программа проверяет, содержит ли командная строка более одного аргумента. (Помните: в списке `sys.argv` всегда будет как минимум один элемент, `sys.argv[0]`, содержащий имя файла сценария.) Если в списке имеется только один элемент, значит, пользователь не предоставил в командной строке название населенного пункта. В этом случае программа, прежде чем завершить работу, выводит сообщение с описанием синтаксиса вызова.

Служба OpenWeatherMap требует, чтобы запрос был отформатирован как название города, после которого идет запятая и двухбуквенный код страны (например, `'US'` в случае США). Список этих кодов доступен по адресу `https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2`. Наш сценарий отображает прогноз погоды для первого города, указанного в полученном тексте JSON. К сожалению, имеются города с одинаковыми названиями, такие как Портленд, штат Орегон, и Портленд, штат Мэн. Чтобы можно было их различать, текст JSON включает информацию о широте и долготе.

Аргументы командной строки разделяются пробелами. Следовательно, если пользователь введет название населенного пункта в виде `San Francisco, US`, то в переменной `sys.argv` будет содержаться следующий список: `['quickWeather.py', 'San', 'Francisco,', 'US']`. Поэтому мы должны объединить все хранящиеся в списке `sys.argv` строки, за исключением первой, вызвав метод `join()`. Объединенная строка сохраняется в переменной `location`.

Шаг 2. Загрузка данных JSON

Сайт OpenWeatherMap.org предоставляет оперативную информацию о погоде в формате JSON. Сначала нужно зарегистрироваться на сайте для получения бесплатного ключа API. (Этот ключ регулирует частоту запросов к серверу, чтобы не допустить снижения его пропускной способности.) Программе требуется лишь загрузить страницу с URL-адресом

```
https://api.openweathermap.org/data/2.5/forecast/daily?q=
<Местоположение>&cnt=3&APPID=<ключ API>,
```

где *<Местоположение>* — название населенного пункта, для которого нужно получить прогноз погоды, а *<ключ API>* — ваш персональный ключ. Добавьте следующий код в файл *getOpenWeather.py*.

```
#! python3
# getOpenWeather.py - вывод прогноза погоды
# для местоположения из командной строки

-- Опущено --

# Загрузка данных JSON с сайта OpenWeatherMap.org
url ='https://api.openweathermap.org/data/2.5/forecast/
daily?q=%s&cnt=3&APPID=%s ' % (location, APPID)
response = requests.get(url)
response.raise_for_status()

# Раскомментируйте, чтобы увидеть исходный текст JSON
# print(response.text)

# СДЕЛАТЬ: записать данные JSON в переменную Python
```

Мы определяем название населенного пункта из аргументов командной строки. Для создания URL-адреса, к которому необходимо получить доступ, мы используем заместитель `%s` и вставляем в эту позицию строку, хранящуюся в переменной `location`. Результат сохраняется в переменной `url`, которая передается функции `requests.get()`. Эта функция возвращает объект `Response`, корректность которого проверяется с помощью вызова `raise_for_status()`. В случае отсутствия исключений загруженный текст будет находиться в переменной `response.text`.

Шаг 3. Запись данных JSON и вывод прогноза погоды

В переменной `response.text` хранится длинная строка, содержащая данные в формате JSON. Для преобразования этой строки в значение Python необходимо вызвать функцию `json.loads()`. Полученные данные JSON будут выглядеть примерно так.

```
{'city': {'coord': {'lat': 37.7771, 'lon': -122.42},
          'country': 'United States of America',
          'id': '5391959',
          'name': 'San Francisco',
          'population': 0},
 'cnt': 3,
 'cod': '200',
 'list': [{'clouds': 0,
           'deg': 233,
           'dt': 1402344000,
           'humidity': 58,
```

```
        'pressure': 1012.23,
        'speed': 1.96,
        'temp': {'day': 302.29,
                 'eve': 296.46,
                 'max': 302.29,
                 'min': 289.77,
                 'morn': 294.59,
                 'night': 289.77},
        'weather': [{'description': 'sky is clear',
                     'icon': '01d',
-- Опущено --
```

Вы сможете увидеть эти данные, передав переменную `weatherData` (см. ниже) в функцию `pprint.pprint()`. Описание всех полей доступно на сайте `https://openweathermap.org/`. Например, из онлайн-документации можно узнать, что значение `302.29` ключа `'day'` — это дневная температура, выраженная в градусах Кельвина, а не в градусах Цельсия или Фаренгейта.

Интересующие нас погодные данные соответствуют ключам `'main'` и `'description'`. Чтобы организовать их аккуратный вывод, добавьте в файл *grtOpenWeather.py* код, выделенный полужирным шрифтом.

```
! python3
# getOpenWeather.py - вывод прогноза погоды
# для местоположения из командной строки

-- Опущено --

# Запись данных JSON в переменную Python
weatherData = json.loads(response.text)

# Вывод прогноза погоды
❶ w = weatherData['list']
print('Текущая погода в %s:' % (location))
print(w[0]['weather'][0]['main'], '-',
      w[0]['weather'][0]['description'])
print()
print('Завтра:')
print(w[1]['weather'][0]['main'], '-',
      w[1]['weather'][0]['description'])
print()
print('Послезавтра:')
print(w[2]['weather'][0]['main'], '-',
      w[2]['weather'][0]['description'])
```

Обратите внимание на то, как программа сохраняет данные о погоде `weatherData['list']` в переменной `w`, избавляя вас от ручного ввода ❶. Словари с данными о погоде на сегодня, завтра и послезавтра находятся в элементах `w[0]`, `w[1]` и `w[2]` соответственно. В каждом из этих словарей имеется ключ `'weather'`, содержащий список. Нас интересует первый

элемент списка (с индексом `0`), представляющий собой вложенный словарь, который содержит несколько дополнительных ключей. В данном случае мы выводим значения ключей `'main'` и `'description'`, разделяя их дефисами.

Запустив эту программу с аргументом командной строки `getOpenWeather.py San Francisco, US`, вы получите примерно такие результаты.

```
Текущая погода в San Francisco, US:
Clear - sky is clear

Завтра:
Clouds - few clouds

Послезавтра:
Clear - sky is clear
```

Идеи для создания похожих программ

Написанный нами код, получающий доступ к погодным данным, может быть положен в основу многих программ. Например, можно создать программы для решения следующих задач.

- Сбор прогнозов погоды для нескольких популярных туристических мест, чтобы узнать, где в данный момент наилучшая погода.
- Регулярная проверка прогноза погоды и заблаговременное предупреждение о заморозках, чтобы в случае необходимости вы успели занести горшки с растениями в комнату. (Выполнение задач по расписанию рассматривается в главе 17, а отправка сообщений электронной почты — в главе 18.)
- Сбор прогнозов погоды с нескольких веб-сайтов для одновременного отображения или же для вычисления усредненных показателей по совокупности прогнозов.

Резюме

CSV и JSON — популярные текстовые форматы хранения данных. С ними удобно работать в программах, и в то же время они легко читаются людьми, благодаря чему они широко применяются для представления простых электронных таблиц и веб-контента. Модули `csv` и `json` упрощают процесс чтения и записи файлов CSV и JSON.

Из предыдущих глав вы узнали о том, как использовать Python для синтаксического анализа информации, хранящейся в файлах различных форматов. Одна из распространенных задач — получение данных, подготовленных в различных форматах, и извлечение только той информации,

которая представляет для вас интерес. Часто подобные задачи настолько специфичны, что использование коммерческих программ не является оптимальным решением. Написав собственные сценарии, вы сможете заставить компьютер обрабатывать большие массивы данных, представленных в этих форматах.

Контрольные вопросы

1. Какие возможности электронных таблиц Excel не могут быть реализованы в CSV-таблицах?
2. Какие аргументы необходимо передавать функциям `csv.reader()` и `csv.writer()` для создания объектов `reader` и `writer`?
3. В каких режимах должны открываться объекты `File` для объектов `reader` и `writer`?
4. Какой метод получает список и записывает его в CSV-файл?
5. Каково назначение именованных аргументов `delimiter` и `line terminator`?
6. Какая функция получает строку данных JSON и возвращает структуру Python?
7. Какая функция получает структуру Python и возвращает строку данных JSON?

Учебный проект

Чтобы закрепить полученные знания на практике, напишите программу для предложенной ниже задачи.

Программа для преобразования данных из формата Excel в формат CSV

Excel позволяет сохранить электронную таблицу в CSV-файле несколькими щелчками мышью, но если нужно выполнить преобразование сотен файлов, то придется щелкать мышью на протяжении многих часов. Используя модуль `openpyxl` из главы 12, напишите программу, которая считывает все файлы Excel, находящиеся в текущем каталоге, и преобразует их в CSV-файлы.

В файле Excel может содержаться множество листов, поэтому необходимо создавать по одному CSV-файлу на лист. Файлы в формате CSV должны называться следующим образом: *<имя_файла_excel>_<название_листа>*.`csv`, где *<имя_файла_excel>* — имя файла Excel без расширения (например,

'spam_data', а не 'spam_data.xlsx'), а *<название_листа>* — строка из переменной title объекта Worksheet.

Программа будет содержать ряд вложенных циклов for. Каркас программы будет примерно таким.

```
for excelFile in os.listdir('.'):
    # Оставляем только файлы XLSX и загружаем объект Workbook
    for sheetName in wb.sheetnames:
        # Цикл по листам рабочей книги
        sheet = wb.get_sheet_by_name(sheetName)

        # Создание имени CSV-файла из имени файла Excel
        # и названия листа рабочей книги.
        # Создание объекта csv.writer для этого CSV-файла.

        # Цикл по строкам листа
        for rowNum in range(1, sheet.max_row + 1):
            rowData = []    # добавление каждой ячейки в этот список
            # Цикл по всем ячейкам строки
            for colNum in range(1, sheet.max_column + 1):
                # Добавление каждой ячейки в список rowData

            # Запись списка rowData в CSV-файл

    csvFile.close()
```

Загрузите ZIP-файл *excelSpreadsheets.zip* из архива примеров книги (см. введение) и разархивируйте электронные таблицы в тот же каталог, в котором находится программа. Вы сможете использовать эти файлы для тестирования программы.

17

РАБОТА С ДАТОЙ И ВРЕМЕНЕМ, ПЛАНИРОВАНИЕ ЗАДАНИЙ И ЗАПУСК ПРОГРАММ

Запускать программы, сидя за компьютером, — несложное занятие, но было бы удобно, чтобы программы могли выполняться и без нашего вмешательства. Такая возможность действительно есть. Благодаря наличию собственных часов компьютер способен запускать программы в конкретное время или через регулярные промежутки времени. Например, программа может ежечасно выполнять автоматический сбор данных на каком-либо сайте, отслеживая их обновление, или запускать ресурсоемкое задание в 4 часа утра, когда за компьютером никто не работает. Такого рода возможности реализуются модулями `time` и `datetime`.

Кроме того, можно создавать программы, которые будут запускать другие программы по расписанию, используя модули `subprocess` и `threading`. Всегда проще воспользоваться готовым кодом, чем писать его самому.

Модуль `time`

Системные часы компьютера настроены на конкретные дату, время и часовой пояс. Встроенный модуль `time` позволяет программам Python запрашивать показания системных часов для определения текущего времени. В этом модуле чаще всего используются функции `time.time()` и `time.sleep()`.

Функция `time.time()`

В программировании принято вести отсчет времени от так называемой *эпохи Unix*, началом которой считается полночь 1 января 1970 года по шкале UTC (всемирное координированное время). Функция `time.time()` возвращает количество секунд, истекших с этого момента времени, представленное вещественным числом. (Вспомните, что вещественными называют числа, содержащие десятичную точку.) Это количество секунд называется *меткой времени Unix*. Введите в интерактивной оболочке следующие инструкции.

```
>>> import time
>>> time.time()
1543813875.3518236
```

Функция `time.time()` была вызвана 2 декабря 2018 года в 21:11 по тихоокеанскому времени. Возвращаемое значение показывает, сколько секунд прошло от начала эпохи Unix.

Метки времени Unix можно использовать для *профилирования* кода, т.е. для оценки того, как долго выполняются определенные фрагменты программы. Если вызвать функцию `time.time()` в начале профилируемого блока кода, а затем в конце этого блока и вычислить разницу, то вы узнаете, сколько времени прошло между двумя вызовами. Например, откройте в файловом редакторе новую вкладку и введите в ней следующий код.

```
  import time
❶ def calcProd():
      # Вычисление произведения первых 100000 чисел
      product = 1
      for i in range(1, 100000):
          product = product * i
      return product
```

```
❷ startTime = time.time()
  prod = calcProd()
❸ endTime = time.time()
❹ print('Длина результата: %s цифр.' % (len(str(prod))))
❺ print('Расчет занял %s секунд.' % (endTime - startTime))
```

В строке ❶ мы определяем функцию `calcProd()`, которая перемножает в цикле все целые числа от `1` до `100000` и возвращает результат. В строке ❷ мы вызываем функцию `time.time()` и сохраняем ее результат в переменной `startTime`. Сразу после вызова функции `calcProd()` мы вновь вызываем функцию `time.time()` и сохраняем возвращенное ею значение в переменной `endTime` ❸. Далее программа сообщает количество цифр в числе, возвращаемом функцией `calcProd()` ❹, и длительность времени, в течение которого выполнялась эта функция ❺.

Сохраните программу в файле *calcProd.py* и запустите ее. Вы получите примерно следующие результаты.

```
Длина результата: 456569 цифр.
Расчет занял 2.844162940979004 секунд.
```

Примечание

Существует и другая возможность профилирования кода – с помощью функции `cProfile.run()`, *которая предоставляет намного больше информации, чем простая функция* `time.time()`. *Подробности можно узнать по адресу* `https://docs.python.org/3/library/profile.html`.

Значение, возвращаемое функцией `time.time()`, само по себе мало что говорит. Функция `time.ctime()` возвращает строковое описание текущего времени. Ей также можно передать результат функции `time.time()` (количество секунд, прошедших с начала эпохи Unix), чтобы получить строковое значение этого времени. Введите в интерактивной оболочке следующие инструкции.

```
>>> import time
>>> time.ctime()
'Mon Jun 15 14:00:38 2020'
>>> thisMoment = time.time()
>>> time.ctime(thisMoment)
'Mon Jun 15 14:00:45 2020'
```

Функция time.sleep()

Если требуется приостановить работу программы на некоторое время, вызовите функцию `time.sleep()`, передав ей аргумент, задающий длительность паузы в секундах. Введите в интерактивной оболочке следующие инструкции.

```
>>> import time
>>> for i in range(3):
❶       print('Тик')
❷       time.sleep(1)
❸       print('Так')
❹       time.sleep(1)
Тик
Так
Тик
Так
Тик
Так
❺ >>> time.sleep(5)
```

В цикле `for` программа выводит на экран слово `'Тик'` ❶, делает паузу длительностью в одну секунду ❷, выводит слово `'Так'` ❸, снова делает паузу длительностью в одну секунду ❹, и так до тех пор, пока пара слов `'Тик'` и `'Так'` не будет выведена на экран три раза.

Функция `time.sleep()` *блокирует работу программы*, т.е. она не возвращает управление до тех пор, пока не истечет требуемое количество секунд. В частности, если вы введете инструкцию `time.sleep(5)` ❺, то приглашение интерпретатора (`>>>`) появится только через пять секунд.

Округление чисел

При работе со значениями времени вы будете часто сталкиваться с вещественными числами, содержащими очень много цифр после десятичной точки. Такие значения можно укоротить с помощью встроенной функции `round()`, которая округляет числа до заданной точности. Для этого ей нужно передать число, подлежащее округлению, и необязательный второй аргумент, определяющий, сколько цифр после десятичной точки следует оставить. При отсутствии второго аргумента функция `round()` округляет первый аргумент до ближайшего целого числа. Введите в интерактивной оболочке следующие инструкции.

```
>>> import time
>>> now = time.time()
>>> now
1543814036.6147408
```

```
>>> round(now, 2)
1543814036.61
>>> round(now, 4)
1543814036.6147
>>> round(now)
1543814037
```

Мы импортируем модуль `time` и сохраняем значение, возвращаемое функцией `time.time()`, в переменной `now`. Вызов `round(now, 2)` округлит значение `now` до двух цифр после десятичной точки, вызов `round(now, 4)` — до четырех цифр, а вызов `round(now)` — до ближайшего целого числа.

Проект: суперсекундомер

Предположим, вы хотите наладить учет времени, непродуктивно расходуемого на выполнение трудоемких рутинных задач, которые вы еще не успели автоматизировать. Физического секундомера у вас нет, а найти аналогичное по функциональным возможностям бесплатное приложение для смартфона или ноутбука, которое не содержало бы навязчивую рекламу, не так-то просто. Давайте попробуем самостоятельно написать на Python программу-секундомер.

Вот что должна делать программа:

1) замерять промежутки времени между нажатиями клавиши <Enter>, причем каждое очередное нажатие этой клавиши запускает новый отсчет;
2) выводить номер замера, суммарное время и длительность замера.

Это означает, что программа будет выполнять следующие операции:

1) определять текущее время с помощью функции `time.time()` и сохранять его в виде метки времени в начале работы программы, а также перед началом каждого замера;
2) поддерживать счетчик замеров и инкрементировать его всякий раз, когда пользователь нажимает клавишу <Enter>;
3) рассчитывать истекшее время путем вычитания временны́х меток;
4) обрабатывать исключения `KeyboardInterrupt`, чтобы пользователь имел возможность прервать работу программы, нажав комбинацию клавиш <Ctrl+C>.

Откройте в файловом редакторе новое окно и сохраните программу в файле *stopwatch.py*.

Шаг 1. Создание программы для отслеживания времени

Программе-секундомеру потребуется знать текущее время, поэтому необходимо импортировать модуль `time`. Программа должна вывести краткие инструкции для пользователя еще до вызова функции `input()`, чтобы таймер запустился сразу же после нажатия пользователем клавиши <Enter>. После этого программа начнет отсчет времени.

Введите в файловом редакторе следующий код. Комментарий `'СДЕЛАТЬ'` обозначает фрагмент, который предстоит написать.

```
#! python3
# stopwatch.py - простая программа-секундомер

import time

# Вывод инструкций по использованию программы
print('Чтобы начать отсчет, нажмите <ENTER>. Для срабатывания ')
print('секундомера нажимайте клавишу <Enter> повторно. ')
print('Для выхода из программы нажмите <Ctrl+C>.')

input()                     # нажатие клавиши <Enter> запускает отсчет
print('Отсчет начат.')
startTime = time.time()     # получение времени начала первого замера
lastTime = startTime
lapNum = 1

# СДЕЛАТЬ: начать отслеживание замеров
```

Мы выводим инструкции для пользователя, начинаем первый замер, фиксируем время и устанавливаем счетчик замеров равным 1.

Шаг 2. Отслеживание и вывод длительности замеров

Теперь мы напишем код, который начинает новые замеры. Мы отображаем длительность текущего замера и суммарное время всех предыдущих замеров, а также увеличиваем значение счетчика замеров. Добавьте в программу код, выделенный полужирным шрифтом.

```
  #! python3
  # stopwatch.py - простая программа-секундомер

  import time

-- Опущено --

  # Начало отслеживания замеров
❶ try:
❷     while True:
```

```
            input()
❸           lapTime = round(time.time() - lastTime, 2)
❹           totalTime = round(time.time() - startTime, 2)
❺           print('Замер #%s: %s (%s)' % (lapNum, totalTime,
                  lapTime), end='')
            lapNum += 1
            lastTime = time.time()     # переустановить время
                                       # последнего замера
❻ except KeyboardInterrupt:
      # Обработка исключения, возникающего при нажатии
      # комбинации клавиш <Ctrl+C>
      print('\nГотово.')
```

В случае, если пользователь останавливает секундомер нажатием комбинации клавиш <Ctrl+C>, генерируется исключение `KeyboardInterrupt`, которое приводит к аварийному завершению программы. Чтобы избежать этого, мы помещаем код в блок `try` ❶. В результате при нажатии комбинации клавиш <Ctrl+C> управление передается инструкции `except` ❻, и на экран выводится сообщение `'Готово'`, а не сообщение об ошибке `KeyboardInterrupt`. Пока этого не произойдет, программа будет выполнять бесконечный цикл ❷, вызывая функцию `input()` и ожидая нажатия пользователем клавиши <Enter> для завершения замера. Мы вычисляем длительность замера путем вычитания времени его начала, `lastTime`, из текущего времени, `time.time()` ❸. Суммарное истекшее время вычисляется путем вычитания времени запуска секундомера, `startTime`, из текущего времени ❹.

Поскольку все результаты хронометража содержат чересчур большое количество цифр после десятичной точки (например, `4.766272783279419`), мы округляем их до двух цифр с помощью функции `round()` ❸❹.

В строке ❺ выводятся номер замера, суммарное истекшее время и длительность замера. Поскольку при нажатии пользователем клавиши <Enter> в ответ на вызов функции `input()` на экран выводится символ новой строки, функции `print()` необходимо передать именованный аргумент `end=''`, устраняющий появление двойного междустрочного интервала. После вывода информации о замере мы выполняем подготовительные операции для следующего замера, инкрементируя счетчик `lapNum` и записывая в переменную `lastTime` текущее время, которое будет служить началом отсчета для следующего замера.

Идеи для создания похожих программ

Возможность отслеживать время будет полезной в самых разных ситуациях. Для решения подобных задач иногда доступны готовые приложения, но преимущество написания собственных программ заключается в том, что

за них не придется платить и они не будут перегружены назойливой рекламой и бесполезными функциями. Вот несколько идей такого рода.

- Создайте приложение, реализующее простой табель. При вводе имени сотрудника программа должна записывать время прихода или ухода с работы, используя текущие показания часов.
- Добавьте в программу возможность отображения времени, прошедшего с момента запуска какого-либо процесса, например загрузки файла с помощью модуля `requests` (см. главу 12).
- Периодически проверяйте, как долго выполняется программа, и предоставьте пользователю возможность отменять задания, которые выполняются слишком долго.

Модуль `datetime`

Модуль `time` полезен для непосредственной работы с метками времени Unix. Но если необходимо отображать дату в более удобном формате или выполнять арифметические действия над датами (допустим, требуется выяснить, какая дата была 205 дней назад или будет через 123 дня), используйте модуль `datetime`.

В модуле `datetime` имеется собственный тип данных `datetime`, представляющий определенные моменты времени. Введите в интерактивной оболочке следующие инструкции.

```
  >>> import datetime
❶ >>> datetime.datetime.now()
❷ datetime.datetime(2019, 2, 27, 11, 10, 49, 595553)
❸ >>> dt = datetime.datetime(2019, 10, 21, 16, 29, 0)
❹ >>> dt.year, dt.month, dt.day
  (2019, 10, 21)
❺ >>> dt.hour, dt.minute, dt.second
  (16, 29, 0)
```

Функция `datetime.datetime.now()` ❶ возвращает объект `datetime` ❷ для текущих даты и времени в соответствии с показаниями системных часов. Этот объект содержит информацию о годе, месяце, дне, часе, минуте, секунде и микросекунде текущего момента времени. Кроме того, можно получить объект `datetime` для любого заданного момента времени с помощью функции `datetime.datetime()` ❸, передав ей целые числа, представляющие год, месяц, день, час, минуту и секунду. Эти целые числа будут храниться в атрибутах `year`, `month`, `day` ❹, а также `hour`, `minute` и `second` ❺ объекта `datetime`.

Метку времени Unix можно преобразовать в объект `datetime` с помощью функции `datetime.datetime.fromtimestamp()`. При этом дата и время объекта `datetime` будут соответствовать местному часовому поясу. Введите в интерактивной оболочке следующие инструкции.

```
>>> import datetime, time
>>> datetime.datetime.fromtimestamp(1000000)
datetime.datetime(1970, 1, 12, 5, 46, 40)
>>> datetime.datetime.fromtimestamp(time.time())
datetime.datetime(2019, 2, 27, 11, 13, 0, 604980)
```

Получив аргумент `1000000`, функция `datetime.datetime.fromtimestamp()` возвращает объект `datetime` для момента времени спустя 1 000 000 секунд после наступления эпохи Unix. Можно также передать этой функции метку времени, соответствующую текущему моменту (`time.time()`). Таким образом, выражения `datetime.datetime.now()` и `datetime.datetime.fromtimestamp(time.time())` равнозначны: в обоих случаях возвращается объект `datetime`, соответствующий текущему моменту.

Чтобы выяснить, какой из объектов `datetime` предшествует другому, используйте операторы сравнения. Более поздний объект окажется "*бо*льшим". Введите в интерактивной оболочке следующие инструкции.

```
❶ >>> halloween2019 = datetime.datetime(2019, 10, 31, 0, 0, 0)
❷ >>> newyears2020 = datetime.datetime(2020, 1, 1, 0, 0, 0)
  >>> oct31_2019 = datetime.datetime(2019, 10, 31, 0, 0, 0)
❸ >>> halloween2019 == oct31_2019
  True
❹ >>> halloween2019 > newyears2020
  False
❺ >>> newyears2020 > halloween2019
  True
  >>> newyears2020 != oct31_2019
  True
```

Сначала мы создаем объект `datetime` для полуночи 31 октября 2019 года, сохраняя его в переменной `halloween2019` ❶. Затем создается объект `datetime` для полуночи 1 января 2020 года, который сохраняется в переменной `newyears2020` ❷, и еще один объект для полуночи 31 октября 2019 года. Сравнение объектов `halloween2019` и `oct31_2019` показывает, что они равны ❸. Сравнение объектов `newyears2020` и `halloween2019` показывает, что объект `newyears2020` больше, т.е. соответствует более позднему моменту времени ❹ ❺.

Тип данных timedelta

В модуле datetime также имеется тип данных timedelta, который представляет *длительности* промежутков времени, а не *моменты* времени. Введите в интерактивной оболочке следующие инструкции.

```
❶ >>> delta = datetime.timedelta(days=11, hours=10, minutes=9,
                                  seconds=8)
❷ >>> delta.days, delta.seconds, delta.microseconds
(11, 36548, 0)
>>> delta.total_seconds()
986948.0
>>> str(delta)
'11 days, 10:09:08'
```

Для создания объектов timedelta используется функция datetime.timedelta(), у которой есть именованные аргументы weeks, days, hours, minutes, seconds, milliseconds и microseconds. Аргументы month и year не предусмотрены, т.к. это переменные отрезки времени, зависящие от конкретного месяца и года. Для объекта timedelta вычисляется полная длительность, выражаемая в днях, секундах и микросекундах. Соответствующие числовые значения хранятся в атрибутах days, seconds и microseconds соответственно. Метод total_seconds() возвращает длительность, выраженную в секундах. Если передать объект timedelta методу str(), будет получено аккуратно отформатированное строковое представление объекта.

В данном примере мы передаем методу datetime.delta() именованные аргументы, определяющие длительность, равную 11 дням, 10 часам, 9 минутам и 8 секундам, и сохраняем полученный объект timedelta в переменной delta ❶. В атрибуте days объекта timedelta содержится значение 11, а в атрибуте seconds — значение 36548 (длительность 10 часов 9 минут и 8 секунд, пересчитанная в секунды) ❷. Функция total_seconds() сообщает нам, что данный отрезок времени равен 986 948 секунд. Наконец, функция str() возвращает строку с описанием интервала.

Над значениями datetime можно выполнять арифметические операции. Например, чтобы определить дату, которая наступит через тысячу дней от текущей даты, введите в интерактивной оболочке следующие инструкции.

```
>>> dt = datetime.datetime.now()
>>> dt
datetime.datetime(2018, 12, 2, 18, 38, 50, 636181)
>>> thousandDays = datetime.timedelta(days=1000)
>>> dt + thousandDays
datetime.datetime(2021, 8, 28, 18, 38, 50, 636181)
```

Прежде всего мы создаем объект datetime для текущего момента времени и сохраняем его в переменной dt. Затем мы создаем объект timedelta для длительности 1000 дней и сохраняем его в переменной thousandDays. Далее значения переменных dt и thousandDays суммируются, и мы получаем объект datetime для будущей даты, которая наступит через 1000 дней после текущей даты. Python выполняет все необходимые арифметические операции и определяет, что датой, которая наступит через 1000 дней после 2 декабря 2018 года, будет 28 августа 2021 года. Это очень удобно, поскольку при выполнении самостоятельных расчетов вам пришлось бы учитывать количество дней в каждом месяце, не забывая о високосных годах, т.е. помнить о множестве мелких деталей. Модуль datetime выполняет всю работу за вас.

Объекты timedelta могут участвовать в операциях сложения и вычитания с объектами datetime или другими объектами timedelta с использованием операторов + и -. Также объекты timedelta можно умножать или делить на целые или вещественные значения с помощью операторов * и /. Введите в интерактивной оболочке следующие инструкции.

```
❶ >>> oct21st = datetime.datetime(2019, 10, 21, 16, 29, 0)
❷ >>> aboutThirtyYears = datetime.timedelta(days=365 * 30)
  >>> oct21st
  datetime.datetime(2019, 10, 21, 16, 29)
  >>> oct21st - aboutThirtyYears
  datetime.datetime(1989, 10, 28, 16, 29)
  >>> oct21st - (2 * aboutThirtyYears)
  datetime.datetime(1959, 11, 5, 16, 29)
```

Здесь мы создаем объект datetime для даты 21 октября 2019 года ❶ и объект timedelta для длительности, равной примерно 30 лет (мы не учитываем високосные годы) ❷. Вычитание переменной aboutThirtyYears из переменной oct21st дает объект datetime, который соответствует дате за 30 лет до 21 октября 2019 года. Вычитая выражение 2 * aboutThirtyYears из переменной oct21st, мы получаем объект datetime, который соответствует дате за 60 лет до 21 октября 2019 года.

Пауза до наступления заданной даты

Функция time.sleep() позволяет приостанавливать работу программы на заданное количество секунд. Используя цикл while, можно приостановить работу программы до наступления нужной даты. Например, следующая программа будет непрерывно выполнять цикл до наступления Хэллоуина в 2021 году.

```
import datetime
import time

halloween2021 = datetime.datetime(2021, 10, 31, 0, 0, 0)
while datetime.datetime.now() < halloween2021:
    time.sleep(1)
```

Функция `time.sleep(1)` приостанавливает программу, чтобы компьютер не тратил ресурсы процессора на непрерывную проверку времени. Вместо этого условие проверяется раз в секунду, и цикл завершится, как только наступит Хэллоуин 2021 года.

Преобразование объектов `datetime` в строки

Метки времени Unix и объекты `datetime` плохо приспособлены для чтения людьми. Чтобы отобразить объект `datetime` в виде строки, используйте метод `strftime()`. (Буква `'f'` в имени метода — от слова "format".)

В методе `strftime()` используются директивы, аналогичные тем, которые используются при выводе форматированных строк Python (табл. 17.1).

Таблица 17.1. Директивы метода `strftime()`

Директива	Описание
`%Y`	Год с указанием столетия: `'2014'`
`%y`	Год без указания столетия: от `'00'` до `'99'` (в диапазоне от 1970 до 2069)
`%m`	Месяц в виде десятичного числа: от `'01'` до `'12'`
`%B`	Полное название месяца, например `'November'`
`%b`	Сокращенное название месяца, например `'Nov'`
`%d`	День месяца: от `'01'` до `'31'`
`%j`	День года: от `'001'` до `'366'`
`%w`	День недели: от `'0'` (воскресенье) до `'6'` (суббота)
`%A`	Полное название дня недели, например `'Monday'`
`%a`	Сокращенное название дня недели, например `'Mon'`
`%H`	Часы (24-часовая шкала): от `'00'` до `'23'`
`%I`	Часы (12-часовая шкала): от `'01'` до `'12'`
`%M`	Минуты: от `'00'` до `'59'`
`%S`	Секунды: от `'00'` до `'59'`
`%p`	`'AM'` (до полудня) или `'PM'` (после полудня)
`%%`	Экранированный литерал `'%'`

Метод `strftime()` возвращает информацию об объекте `datetime` в виде отформатированной строки. Введите в интерактивной оболочке следующие инструкции.

```
>>> oct21st = datetime.datetime(2019, 10, 21, 16, 29, 0)
>>> oct21st.strftime('%Y/%m/%d %H:%M:%S')
'2019/10/21 16:29:00'
>>> oct21st.strftime('%I:%M %p')
'04:29 PM'
>>> oct21st.strftime("%B of '%y")
"October of '19"
```

Объект `datetime`, соответствующий дате 21 октября 2019 года и времени 16:29, сохраняется в переменной `oct21st`. Получив строку форматирования `'%Y/%m/%d %H:%M:%S'`, метод `strftime()` возвращает числа `2019`, `10` и `21`, разделенные символами обратной косой черты, а также числа `16`, `29` и `00`, разделенные двоеточиями. В случае строки форматирования `'%I:%M% p'` метод возвращает значение `'04:29 PM'`, а в случае строки `"%B of '%y"` — значение `"October of '19"`. Обратите внимание на то, что имя `strftime()` указывается без префикса `datetime.datetime`.

Преобразование строк в объекты `datetime`

Если у вас имеется строка, содержащая информацию о дате, например `'2019/10/21 16:29:00'` или `'October 21, 2019'`, которую необходимо преобразовать в объект `datetime`, используйте для этого функцию `datetime.datetime.strptime()`. Она противоположна методу `strftime()`. Функции `strptime()` необходимо передать такую же строку форматирования, как и в методе `strftime()`, чтобы она понимала синтаксис даты. (Буква `'p'` в имени функции — от слова "parse".)

Введите в интерактивной оболочке следующие инструкции.

```
❶ >>> datetime.datetime.strptime('October 21, 2019', '%B %d, %Y')
  datetime.datetime(2019, 10, 21, 0, 0)
  >>> datetime.datetime.strptime('2019/10/21 16:29:00',
  ...                            '%Y/%m/%d %H:%M:%S')
  datetime.datetime(2019, 10, 21, 16, 29)
  >>> datetime.datetime.strptime("October of '19", "%B of '%y")
  datetime.datetime(2019, 10, 1, 0, 0)
  >>> datetime.datetime.strptime("November of '63", "%B of '%y")
  datetime.datetime(2063, 11, 1, 0, 0)
```

Чтобы получить объект `datetime` из строки `'October 21, 2019'`, передайте эту строку в качестве первого аргумента функции `strptime()`, а строку форматирования — в качестве второго аргумента ❶. Строка с информацией

о дате должна в точности соответствовать строке форматирования, иначе будет сгенерировано исключение `ValueError`.

Обзор функций Python для работы с датой и временем

Для работы с датами и временем в Python предусмотрено большое количество различных типов данных и функций. Ниже кратко описаны три типа данных, используемых для представления времени.

- Метка времени Unix (используется модулем `time`) — это целое или вещественное число, представляющее количество секунд, прошедших с полуночи 1 января 1970 года по шкале UTC.
- Объект `datetime` (из модуля `datetime`) содержит целочисленные значения, хранящиеся в атрибутах `year`, `month`, `day`, `hour`, `minute` и `second`.
- Объект `timedelta` (из модуля `datetime`) представляет длительность промежутка времени, а не конкретный момент времени.

Ниже приведено краткое описание функций для работы со временем, их параметров и возвращаемых значений.

- Функция `time.time()` возвращает метку времени Unix в виде вещественного значения, которое соответствует текущему моменту.
- Функция `time.sleep(`*`секунды`*`)` приостанавливает выполнение программы на заданное количество секунд.
- Функция `datetime.datetime(`*`год`*`, `*`месяц`*`, `*`день`*`, `*`час`*`, `*`минута`*`, `*`секунда`*`)` возвращает объект `datetime`, который соответствует заданному моменту времени. Если аргументы *`час`*, *`минута`* или *`секунда`* не заданы, то по умолчанию им присваивается значение `0`.
- Функция `datetime.datetime.now()` возвращает объект `datetime`, соответствующий текущему моменту времени.
- Функция `datetime.datetime.fromtimestamp(`*`метка_времени`*`)` возвращает объект `datetime`, который соответствует моменту времени, представленному аргументом *`метка_времени`*.
- Функция `datetime.timedelta(`*`недели`*`, `*`дни`*`, `*`часы`*`, `*`минуты`*`, `*`секунды`*`, `*`миллисекунды`*`, `*`микросекунды`*`)` возвращает объект `timedelta`, представляющий длительность промежутка времени. Все именованные аргументы этой функции необязательны; аргументы *`месяц`* и *`год`* не предусмотрены.
- Метод `total_seconds()` объекта `timedelta` возвращает количество секунд в данном временном отрезке.

- Метод `strftime(`*`формат`*`)` объекта `datetime` возвращает строку времени, представленную в заданном формате (см. табл. 17.1).
- Функция `datetime.datetime.strptime(`*`строка_времени, формат`*`)` возвращает объект `datetime`, который соответствует моменту времени, заданному с помощью аргумента *`строка_времени`* и отформатированному с использованием строкового аргумента *`формат`* (см. табл. 17.1).

Многопоточность

Чтобы разобраться с концепцией многопоточности, лучше всего рассмотреть конкретный пример. Предположим, вы хотите запустить программу в заданное время. Для этого можно добавить в начало программы примерно такой код.

```
import time, datetime

startTime = datetime.datetime(2029, 10, 31, 0, 0, 0)
while datetime.datetime.now() < startTime:
    time.sleep(1)

print('Программа запустится на Хэллоуин 2029 года')
-- Опущено --
```

В данном случае запуск программы запланирован на полночь 31 октября 2029 года. До наступления этого момента в цикле непрерывно вызывается функция `time.sleep(1)`. Программа не сможет делать ничего другого, пока цикл не завершится. Фактически она будет в неактивном состоянии до тех пор, пока не наступит Хэллоуин 2029 года. Это связано с тем, что программы Python по умолчанию имеют только один *поток выполнения*.

Чтобы понять, что такое поток выполнения, вспомните, как в главе 2 мы уподобили порядок выполнения программы перемещению пальца по строкам кода. Палец может перемещаться либо последовательно, от строки к строке, либо скачками, в соответствии с логикой управляющих инструкций. В *однопоточной* программе имеется только один "палец", а вот у *многопоточных* программ таких "пальцев" может быть несколько. Каждый из них по-прежнему следует логике программы, но при этом "пальцы" могут находиться в разных местах программы, одновременно выполняя разные инструкции. (Все программы, которые мы рассматривали до сих пор, были однопоточными.)

Вместо того чтобы заставлять всю программу находиться в состоянии ожидания до тех пор, пока не завершится цикл вызова функции `time.sleep()`, можно выделить для выполнения отложенной или

запланированной задачи отдельный поток, используя модуль threading. Этот отдельный поток будет находиться в состоянии ожидания все то время, пока вызывается функция time.sleep(). А тем временем программа сможет выполнять другую полезную работу в исходном потоке.

Для создания потока необходимо получить объект Thread, вызвав функцию threading.Thread(). Введите в новом окне файлового редактора следующий код и сохраните его в файле *threadDemo.py*.

```
import threading, time
print('Начало программы.')

❶ def takeANap():
    time.sleep(5)
    print('Проснись!')

❷ threadObj = threading.Thread(target=takeANap)
❸ threadObj.start()

print('Конец программы.')
```

В строке ❶ мы определяем функцию, которая будет выполняться в отдельном потоке. Чтобы создать объект Thread, мы вызываем функцию threading.Thread() и передаем ей именованный аргумент target=takeANap ❷. Это означает, что целевой функцией, которую мы хотим вызвать в новом потоке, будет функция takeANap(). Обратите внимание на то, что именованный аргумент записывается как target=takeANap, а не target=takeANap(). В качестве аргумента мы передаем саму функцию takeANap(), а не результат ее вызова.

Сохранив объект Thread, созданный функцией threading.Thread(), в переменной threadObj, мы вызываем метод threadObj.start() ❸ для создания нового потока и запуска в нем целевой функции. Запустив программу, вы получите следующее.

```
Начало программы.
Конец программы.
Проснись!
```

Странно, не так ли? Если инструкция print('Конец программы.') — последняя в программе, то почему текст ее сообщения не был выведен последним? Причина, почему последним выводится текст 'Проснись!', заключается в том, что после вызова threadObj.start() целевая функция объекта threadObj начинает выполняться в отдельном потоке, как если бы в программе появился второй управляющий "палец". Основной поток переходит к инструкции print('Конец программы.'), а тем временем новый поток, выполняющий вызов time.sleep(5), выжидает в течение 5 секунд.

Затем он выходит из своей 5-секундной спячки, выводит на экран строку 'Проснись!' и завершает функцию takeANap(). Вот почему последнее, что выводит программа, — это строка 'Проснись!'.

Обычно выполнение программы завершается на последней строке кода (или в результате вызова функции sys.exit()). Но в программе *threadDemo.py* существуют два потока. Один из них, первоначальный, в котором начала выполняться программа, завершается после инструкции print('Конец программы.'). Второй поток создается вызовом threadObj.start(), запускает функцию takeANap() и завершается вместе с ней.

В Python программа прекращает работу тогда, когда завершаются все ее потоки. В случае программы *threadDemo.py* даже после того, как первоначальный поток завершился, второй поток все еще продолжает выполнять вызов time.sleep(5).

Передача аргументов целевой функции потока

Если целевая функция, которую требуется запустить в отдельном потоке, имеет аргументы, то их можно передать методу threading.Thread(). Предположим, например, что целевой функцией потока должен быть следующий вызов print().

```
>>> print('Коты', 'Собаки', 'Лягушки', sep=' & ')
Коты & Собаки & Лягушки
```

В данном случае у функции print() три обычных аргумента, 'Коты', 'Собаки' и 'Лягушки', и один именованный: sep=' & '. Обычные аргументы передаются в функцию threading.Thread() в виде списка args, а именованные аргументы — в виде словаря kwargs.

Введите в интерактивной оболочке следующие инструкции.

```
>>> import threading
>>> threadObj = threading.Thread(target=print,
                                 args=['Коты', 'Собаки', 'Лягушки'],
                                 kwargs={'sep': ' & '})
>>> threadObj.start()
Коты & Собаки & Лягушки
```

Чтобы передать аргументы 'Коты', 'Собаки' и 'Лягушки' функции print() в новом потоке, мы передаем именованный аргумент args=['Коты', 'Собаки', 'Лягушки'] функции threading.Thread(). Аналогичным образом мы поступаем в отношении именованного аргумента sep=' & ', передавая функции threading.Thread() именованный аргумент kwargs={'sep': ' & '}.

Метод `threadObj.start()` создает новый поток выполнения, в котором вызывается целевая функция `print()`, и он же передает ей строки `'Коты'`, `'Собаки'` и `'Лягушки'` в качестве обычных аргументов, а также строку `' & '` — в качестве значения именованного аргумента `sep`.

Ниже продемонстрирован неправильный способ создания нового потока для вызова функции `print()`.

```
threadObj = threading.Thread(target=print('Коты', 'Собаки',
                                          'Лягушки', sep=' & '))
```

В этом коде сначала вызывается функция `print()`, и в качестве именованного аргумента `target` функции `threading.Thread()` будет передана не сама функция `print()`, а возвращаемое ею значение (т.е. `None`). Для передачи аргументов функции, выполняющейся в новом потоке, следует использовать именованные аргументы `args` и `kwargs` функции `threading.Thread()`.

Проблемы параллелизма

В программе можно легко создать несколько новых потоков, и все они будут выполняться одновременно. Однако запуск нескольких потоков чреват так называемыми *проблемами параллелизма*. Эти проблемы возникают в тех случаях, когда разные потоки одновременно пытаются читать и записывать одни и те же переменные, конкурируя друг с другом. Проблемы параллелизма трудно воспроизводить, что затрудняет отладку программы.

Многопоточное программирование — обширная тема, рассмотрение которой выходит за рамки книги. Главное, запомнить следующее: чтобы избежать проблем параллелизма, никогда не позволяйте нескольким потокам читать и записывать одни и те же переменные. Создавая новый объект `Thread`, убедитесь в том, что его целевая функция использует только локальные переменные.

Примечание

Руководство по многопоточному программированию для начинающих доступно по следующему адресу:

`https://inventwithpython.com/blog/2013/04/22/multithreaded-python-tutorial-with-the-threadworms-demo/`

Проект: многопоточный загрузчик файлов с сайта XKCD

В главе 12 мы написали программу, загружающую все комиксы с сайта XKCD. Это была однопоточная программа: она загружала по одному

комиксу за раз. Значительная часть времени расходовалась на установление сетевого соединения перед началом загрузки изображений. При наличии широкополосного подключения к Интернету однопоточная программа будет использовать не всю доступную полосу пропускания.

Многопоточная программа, в которой загрузка комиксов, установление соединения и запись файлов на диск осуществляются в разных потоках, будет более эффективно использовать интернет-канал, что позволит быстрее загружать коллекцию комиксов. Откройте в файловом редакторе новое окно и сохраните программу в файле *threadedDownloadXkcd.py*. Мы модифицируем предыдущую версию программы, сделав ее многопоточной. Полный код программы доступен в архиве примеров книги (см. введение).

Шаг 1. Модификация программы путем вынесения ее кода в функцию

В этой программе для загрузки файлов будет использоваться в основном тот же код, что и в главе 12, поэтому опустим описание кода, связанного с модулями `requests` и Beautiful Soup. Основные изменения связаны с импортом модуля `threading` и созданием функции `downloadXkcd()`, аргументами которой будут номера начального и конечного комиксов.

Например, функция `downloadXkcd(140, 280)` выполнит цикл для загрузки комиксов, хранящихся на страницах `https://xkcd.com/140`, `https://xkcd.com/141`, `https://xkcd.com/142` и т.д. вплоть до комикса `https://xkcd.com/279`. Каждый создаваемый поток выполнения будет вызывать функцию `downloadXkcd()`, передавая ей разные диапазоны номеров комиксов, подлежащих загрузке.

Добавьте в программу *threadedDownloadXkcd.py* следующий код.

```
#! python3
# threadedDownloadXkcd.py - загружает комиксы XKCD
# с использованием нескольких потоков выполнения

import requests, os, bs4, threading
❶ os.makedirs('xkcd', exist_ok=True)    # сохраняем комиксы
                                        # в папке ./xkcd

❷ def downloadXkcd(startComic, endComic):
❸     for urlNumber in range(startComic, endComic):
          # Загрузка страницы
          print('Загрузка https://xkcd.com/%s...' % (urlNumber))
❹         res = requests.get('https://xkcd.com/%s' % (urlNumber))
          res.raise_for_status()

❺         soup = bs4.BeautifulSoup(res.text, 'html.parser')

          # Поиск URL-адреса комикса
❻         comicElem = soup.select('#comic img')
```

```
        if comicElem == []:
            print('Не удалось найти изображение комикса.')
        else:
❼           comicUrl = comicElem[0].get('src')
            # Загрузка изображения
            print('Загрузка %s...' % (comicUrl))
❽           res = requests.get(comicUrl)
            res.raise_for_status()

            # Сохранение изображения в папке ./xkcd
            imageFile = open(os.path.join('xkcd',
                             os.path.basename(comicUrl)), 'wb')
            for chunk in res.iter_content(100000):
                imageFile.write(chunk)
            imageFile.close()

# СДЕЛАТЬ: создать и запустить объекты Thread
# СДЕЛАТЬ: дождаться завершения всех потоков
```

Импортировав необходимые модули, мы создаем папку для хранения комиксов ❶ и определяем функцию `downloadxkcd()` ❷. В этой функции мы организуем цикл для обработки комиксов в заданном диапазоне номеров ❸ и загружаем каждую страницу ❹. Далее мы используем модуль Beautiful Soup для просмотра HTML-кода каждой страницы ❺ и поиска изображения комикса ❻. Если изображение не найдено, выводится соответствующее сообщение. В противном случае мы получаем URL-адрес изображения ❼ и загружаем само изображение ❽. Наконец, мы сохраняем изображение в созданной папке.

Шаг 2. Создание и запуск потоков выполнения

Теперь, когда у нас есть функция `downloadXkcd()`, мы создадим несколько потоков, каждый из которых вызывает функцию `downloadXkcd()` для загрузки комиксов с номерами, лежащими в различных диапазонах. Добавьте в файл *threadedDownloadXkcd.py* следующий код (он располагается после определения функции `downloadXkcd()`).

```
#! python3
# threadedDownloadXkcd.py - загружает комиксы XKCD
# с использованием нескольких потоков выполнения

-- Опущено --

# Создание и запуск объектов Thread
downloadThreads = []    # список объектов Thread
for i in range(0, 140, 10):    # 14 итераций для создания 14 потоков
    start = i
    end = i + 9
```

```
    if start == 0:
        start = 1      # комикса 0 нет, начинаем с 1
    downloadThread = threading.Thread(target=downloadXkcd, \
                                      args=(start, end))
    downloadThreads.append(downloadThread)
    downloadThread.start()
```

Прежде всего мы создаем пустой список `downloadThreads`, который поможет нам отслеживать имеющиеся объекты `Thread`. Далее запускается цикл `for`. На каждой итерации цикла мы создаем объект `Thread` с помощью функции `threading.Thread()`, добавляем его в список и вызываем метод `start()` для запуска функции `downloadXkcd()` в новом потоке. Так как счетчик цикла принимает значения от `0` до `140` с шагом `10`, на первой итерации он будет иметь значение `0`, на второй — `10`, на третьей — `20` и т.д. Поскольку мы передаем функции `threading.Thread()` аргумент `args=(start, end)`, на первой итерации цикла функция `downloadXkcd()` получит аргументы `0` и `9`, на второй — `10` и `19`, на третьей — `20` и `29` и т.д.

После вызова метода `start()` объекта `Thread` и запуска нового потока основной поток перейдет к следующей итерации цикла `for` и создаст очередной поток.

Шаг 3. Ожидание завершения всех потоков

Основной поток продолжает выполняться как обычно, в то время как другие потоки, которые мы создали, загружают комиксы. Но предположим, в программе есть другой код, который не должен быть запущен до завершения всех остальных потоков. Метод `join()` объекта `Thread` заблокирует программу до тех пор, пока не завершится данный поток. Мы используем цикл `for` для итерации по всем объектам `Thread` в списке `downloadThreads` и вызываем метод `join()` для каждого потока. Добавьте в конце программы следующий код.

```
#! python3
# threadedDownloadXkcd.py - загружает комиксы XKCD
# с использованием нескольких потоков выполнения

-- Опущено --

# Ожидание завершения всех потоков
for downloadThread in downloadThreads:
    downloadThread.join()
print('Готово.')
```

Строка `'Готово.'` не будет выведена до тех пор, пока не завершатся все вызовы метода `join()`. Если окажется, что к моменту вызова метода `join()` объект `Thread` уже закончил загрузку изображений, метод `join()` сразу же

завершится. Таким образом, если в программе требуется выполнить какой-то код после загрузки всех комиксов, вставьте его вместо инструкции `print('Готово.')`.

Запуск других программ из Python

Программа, написанная на Python, может запускать другие программы с помощью функции `Popen()` встроенного модуля `subprocess`. (Буква `'P'` в имени функции означает "process".) Когда открыто несколько экземпляров какого-либо приложения, каждый из них представляет собой отдельный процесс одной и той же программы. Например, если одновременно открыть несколько окон браузера, то все они будут разными процессами браузерной программы. Пример нескольких одновременно выполняющихся процессов программы-калькулятора показан на рис. 17.1.

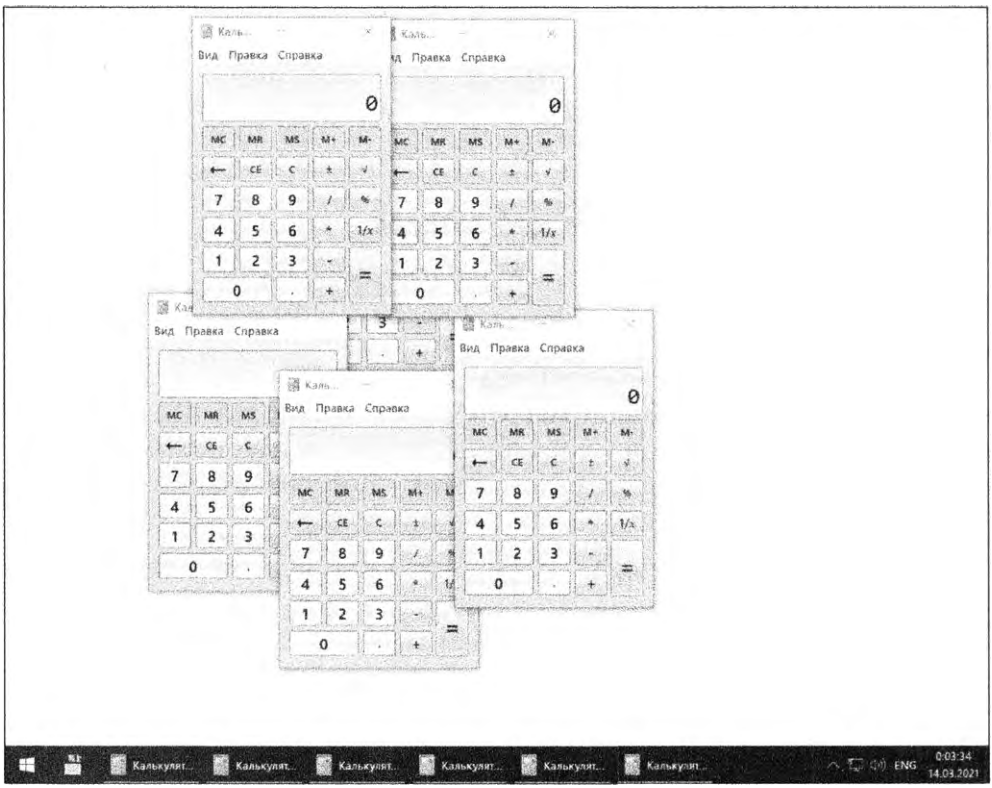


Рис. 17.1. Шесть одновременно выполняющихся процессов одного и того же приложения

У каждого процесса может быть несколько потоков выполнения. В отличие от потоков, процесс не может непосредственно читать и записывать переменные другого процесса. Если о потоках мы говорили как о пальцах,

скользящих по строкам исходного кода, то запуск нескольких процессов одной и той же программы можно уподобить одновременному выполнению отдельных экземпляров программы, которые вы раздали своим друзьям. Каждый из вас будет независимо выполнять одну и ту же программу.

Если необходимо запустить внешнюю программу из сценария Python, передайте имя этой программы функции `subprocess.Popen()`. (Чтобы узнать имя приложения в Windows, щелкните на значке приложения в меню Пуск правой кнопкой мыши и выберите в контекстном меню пункт Свойства. В macOS, чтобы узнать путь к исполняемому файлу, щелкните на значке приложения при нажатой клавише <Ctrl> и выберите пункт Show Package Contents.) Функция `Popen()` сразу же завершится. Имейте в виду, что запущенная таким способом программа будет выполняться в другом потоке.

В Windows введите в интерактивной оболочке следующие инструкции.

```
>>> import subprocess
>>> subprocess.Popen('C:\\Windows\\System32\\calc.exe')
<subprocess.Popen object at 0x0000000003055A58>
```

В Ubuntu Linux инструкции будут такими.

```
>>> import subprocess
>>> subprocess.Popen('/usr/bin/gnome-calculator')
<subprocess.Popen object at 0x7f2bcf93b20>
```

В macOS это делается немного иначе (подробности описаны в разделе "Открытие файлов приложениями, заданными по умолчанию").

Функция `Popen()` возвращает объект `Popen`, имеющий два полезных метода: `poll()` и `wait()`.

Вызывая метод `poll()`, вы словно спрашиваете у водителя: "Мы уже приехали?" Метод возвращает значение `None`, если в данный момент процесс еще выполняется. Если же программа завершила работу, то возвращается целочисленный *код завершения* процесса. Этот код используется в качестве индикатора того, завершился ли процесс без ошибок (код `0`) или же его завершение было обусловлено ошибкой (ненулевой код, обычно — `1`, но могут быть и другие значения, в зависимости от программы).

Вызывая метод `wait()`, вы словно ждете, пока водитель не привезет вас по месту назначения. Метод блокируется до завершения запущенного процесса. Это может оказаться полезным, если необходимо, чтобы программа выжидала, пока пользователь не закончит работать с другой программой. Метод `wait()` возвращает целочисленный код завершения процесса.

Если вы работаете в Windows, введите в интерактивной оболочке приведенные ниже инструкции. Обратите внимание на то, что метод `wait()` блокируется до тех пор, пока вы не завершите работу с программой.

```
  >>> import subprocess
❶ >>> paintProc =
       subprocess.Popen('c:\\Windows\\System32\\mspaint.exe')
❷ >>> paintProc.poll() == None
  True
❸ >>> paintProc.wait()     # не завершается, пока программа открыта
  0
  >>> paintProc.poll()
  0
```

Сначала мы запускаем процесс Paint ❶, после чего проверяем, возвращает ли метод `poll()` значение `None` ❷. Соблюдение этого условия означает, что процесс все еще выполняется. Далее мы закрываем программу Paint и вызываем метод `wait()` для уже завершившегося процесса ❸. Теперь оба метода, `wait()` и `poll()`, возвращают значение `0`, указывающее на то, что выполнение процесса завершилось без ошибок.

Примечание

В отличие от программы mspaint.exe, если вы запустите программу calc.exe в Windows 10 с помощью метода `subprocess.Popen()`*, то заметите, что метод* `wait()` *сразу же завершается, даже несмотря на то что приложение калькулятора все еще работает. Это связано с тем, что программа calc.exe запускает отдельный процесс калькулятора и тут же закрывается. Такова специфика работы "доверенных приложений Microsoft Store", рассмотрение которых выходит за рамки книги.*

Передача аргументов командной строки в функцию `Popen()`

Процессам, создаваемым с помощью функции `Popen()`, можно передавать аргументы командной строки. Самая функция `Popen()` поддерживает единственный аргумент в виде списка, первой строкой в котором должно быть имя исполняемого файла запускаемой программы, а все последующие строки представляют собой аргументы командной строки, передаваемые программе. В конечном итоге этот список будет значением `sys.argv` для запущенной программы.

Приложения с графическим интерфейсом пользователя (GUI) реже нуждаются в аргументах командной строки, в отличие от терминальных программ. Тем не менее большинство GUI-приложений поддерживает одиночный аргумент в виде имени файла, который должен быть открыт приложением сразу после запуска. Например, если вы работаете в Windows, создайте текстовый файл *C:\hello.txt* и введите в интерактивной оболочке следующую инструкцию.

```
>>> subprocess.Popen(['C:\\Windows\\notepad.exe', 'C:\\hello.txt'])
<subprocess.Popen object at 0x00000000032DCEB8>
```

Эта команда не только запустит программу Блокнот, но и немедленно откроет в ней файл *C:\hello.txt*.

Планировщик заданий Windows, демон `launchd` и планировщик `cron`

Опытные пользователи должны быть знакомы с планировщиком заданий в Windows (его аналог в macOS — `launchd`, а в Linux — `cron`). Эти утилиты позволяют планировать запуск приложений по расписанию. Более подробную информацию о них можно получить, воспользовавшись ссылками на соответствующие руководства по адресу `https://automatetheboringstuff.com/schedulers.html`.

Встроенный планировщик заданий операционной системы избавляет вас от необходимости писать собственный код проверки системных часов. Тем не менее, если нужно всего лишь приостановить выполнение программы на короткое время, используйте функцию `time.sleep()`. Кроме того, можно вызывать функцию `time.sleep(1)` на каждой итерации цикла, пока не истечет определенное время.

Открытие веб-сайтов с помощью Python

Вместо того чтобы запускать браузер путем вызова функции `subprocess.Popen()`, можно использовать функцию `webbrowser.open()` для непосредственного открытия нужного сайта в браузере (см. главу 12).

Запуск других сценариев Python

Сценарии Python можно запускать в отдельном процессе точно так же, как любое другое приложение. Для этого достаточно передать функции `Popen()` имя исполняемого файла Python (*python.exe*) вместе с именем файла сценария. Например, следующая инструкция выполнит сценарий *hello.py*, рассмотренный в главе 1.

```
>>> subprocess.Popen(['C:\\Users\\<имя_пользователя>\\AppData\\
Local\\Programs\\python38\\python.exe', 'hello.py'])
<subprocess.Popen object at 0x000000000331CF28>
```

В функцию `Popen()` необходимо передать список, содержащий строку с путем доступа к исполняемому файлу Python и строку с именем файла сценария. Если запускаемый сценарий сам нуждается в аргументах

командной строки, добавьте их в список после имени файла сценария. Расположение исполняемого файла интерпретатора Python зависит от платформы. В Windows это *C:\Users\<имя_пользователя>\AppData\Local\Programs\Python\Python38\python.exe*, в macOS — */Library/Frameworks/Python.framework/Versions/3.8/bin/python3*, в Linux — */usr/bin/python3.8*.

В отличие от программ Python, импортируемых в виде модулей, программа, запущенная из другой программы, будет выполняться в отдельном процессе, и они не смогут получать доступ к переменным друг друга.

Открытие файлов приложениями, заданными по умолчанию

Двойной щелчок на значке файла с расширением *.txt* позволяет автоматически запустить приложение, ассоциированное с этим расширением. В системе заранее настроено множество подобных ассоциаций. В Python тоже можно открывать файлы подобным образом с помощью функции `Popen()`.

В любой операционной системе имеется программа, запускаемая по двойному щелчку на значке документа. В Windows это программа `start`, в OS — программа `open`, в Ubuntu Linux — программа `see`. Введите в интерактивной оболочке следующие инструкции, передавая функции `Popen()` одну из строк `'start'`, `'open'` или `'see'`, в зависимости от операционной системы.

```
>>> fileObj = open('hello.txt', 'w')
>>> fileObj.write('Здравствуй, мир!')
16
>>> fileObj.close()
>>> import subprocess
>>> subprocess.Popen(['start', 'hello.txt'], shell=True)
```

Сначала мы записываем строку `'Здравствуй, мир!'` в новый файл *hello.txt*, а затем вызываем функцию `Popen()`, передавая ей список, содержащий имя программы (в случае Windows — `'start'`) и имя файла. Кроме того, мы передаем именованный аргумент `shell=True`, который требуется лишь в случае Windows. Операционной системе известны все ассоциации программ с расширениями файлов, и она может самостоятельно определить, какую программу следует выполнить. Например, для обработки файла *hello.txt* будет запущена программа Notepad.exe.

В macOS программа `open` используется для открытия как документов, так и других программ. Введите в интерактивной оболочке следующую инструкцию.

```
>>> subprocess.Popen(['open', '/Applications/Calculator.app/'])
<subprocess.Popen object at 0x10202ff98>
```

В результате должно открыться приложение Calculator.

Проект: простая программа обратного отсчета времени

Найти простую программу для обратного отсчета времени не менее трудно, чем приложение, выполняющее функции секундомера. Давайте напишем программу, которая ведет обратный отсчет времени и сообщает о его завершении звуковым сигналом.

Вот что должна делать такая программа:

1) вести обратный отсчет, начиная с 60;
2) воспроизводить звуковой файл (*alarm.wav*), когда счетчик достигает нулевого значения.

Это означает, что программа должна выполнять следующие операции:

1) вызывать функцию `time.sleep()`, делая секундную паузу перед выводом очередного значения счетчика;
2) вызывать функцию `subprocess.Popen()` для открытия звукового файла с помощью программы по умолчанию.

Откройте в файловом редакторе новое окно и сохраните программу в файле *countdown.py*.

Шаг 1. Обратный отсчет

Этой программе потребуется модуль `time` для вызова функции `time.sleep()` и модуль `subprocess` для вызова функции `subprocess.Popen()`. Введите следующий код.

```
#! python3
# countdown.py - простой сценарий обратного отсчета

import time, subprocess

❶ timeLeft = 60
while timeLeft > 0:
❷     print(timeLeft, end='')
❸     time.sleep(1)
❹     timeLeft = timeLeft - 1

# СДЕЛАТЬ: воспроизвести звуковой файл в конце
```

Импортировав модули `time` и `subprocess`, мы создаем переменную `timeLeft`, в которой будет храниться количество секунд, оставшихся до окончания отсчета ❶. В данной программе обратный отсчет начинается от значения 60. Можете изменить это значение или же сделать так, чтобы программа получала его в виде аргумента командной строки.

На каждой итерации цикла `while` отображается текущее значение счетчика ❷, делается секундная пауза ❸ и декрементируется значение переменной `timeLeft` ❹. Цикл продолжается до тех пор, пока значение переменной `timeLeft` больше 0. Как только это условие перестает выполняться, обратный отсчет прекращается.

Шаг 2. Воспроизведение звукового файла

Несмотря на то что существуют сторонние модули, позволяющие воспроизводить звуковые файлы различных форматов, самый простой и быстрый способ заключается в запуске приложения, уже настроенного в системе для этих целей. Операционная система сама определит по расширению *.wav*, какое приложение следует запустить для воспроизведения файла. Разумеется, вместо файла формата *.wav* можно использовать и файлы других аналогичных форматов, таких как *.mp3* или *.ogg*.

В качестве файла, воспроизводимого по завершении обратного отсчета, используйте любой имеющийся у вас звуковой файл либо загрузите файл *alarm.wav* из архива примеров книги (см. введение).

Добавьте в программу следующий код.

```
#! python3
# countdown.py - простой сценарий обратного отсчета

import time, subprocess

-- Опущено --

# Воспроизведение звукового файла по завершении
# обратного отсчета
subprocess.Popen(['start', 'alarm.wav'], shell=True)
```

По завершении цикла `while` пользователь оповещается об окончании работы программы воспроизведением файла *alarm.wav* (или другого выбранного вами файла). В Windows не забудьте включить строку `'start'` в список, передаваемый функции `Popen()`, и одновременно передать именованный аргумент `shell=True`. В macOS передайте строку `'open'` вместо `'start'` и удалите аргумент `shell=True`.

Вместо звукового файла можно подготовить текстовый файл с сообщением наподобие "Перерыв закончился!" и использовать функцию `Popen()`

для его открытия по завершении обратного отсчета. Другой вариант — вызывать функцию `webbrowser.open()`, которая по завершении обратного отсчета будет открывать заданный веб-сайт. В отличие от бесплатных программ, доступных в Интернете, ваш вариант оповещения пользователя может быть любым!

Идеи для создания похожих программ

Обратный отсчет — это простейший способ организовать паузу, по окончании которой программа сможет продолжить выполнение. Вот несколько идей такого рода.

- Используйте функцию `time.sleep()`, чтобы дать пользователю возможность отменить какое-либо действие, например удаление файла, путем нажатия комбинации клавиш <Ctrl+C>. Программа может выводить сообщение "Для отмены нажмите <Ctrl+C>", а затем обрабатывать исключения `KeyboardInterrupt` с помощью инструкций `try` и `except`.
- Для организации обратного отсчета в течение длительных промежутков времени можно использовать объекты `timedelta`, чтобы отмерять количество дней, часов, минут и секунд, оставшихся до наступления определенного события (например, дня рождения или юбилея).

Резюме

Эпоха Unix (полночь 1 января 1970 года по шкале UTC) — это стандартная точка отсчета времени во многих языках программирования, включая Python. Функция `time.time()` в Python возвращает метку времени, т.е. вещественное число, которое соответствует количеству секунд, прошедших с начала эпохи Unix. Кроме того, модуль `datetime` позволяет выполнять арифметические операции с датами, а также форматировать и анализировать строки, содержащие информацию о дате.

Функция `time.sleep()` блокируется (т.е. не завершается) в течение заданного количества секунд. Это можно использовать для добавления пауз в программу. Но если необходимо запланировать запуск программы на определенный момент времени, воспользуйтесь инструкциями, предоставленными на сайте `https://automatetheboringstuff.com/schedulers.html`, чтобы узнать, как сделать это с помощью планировщика заданий операционной системы.

Модуль `threading` используется для создания нескольких потоков выполнения. Это может пригодиться для пакетной загрузки файлов или одновременного выполнения других задач. Но вы должны убедиться в том, что

потоки читают и записывают только свои локальные переменные, иначе вы рискуете столкнуться с проблемами параллелизма.

Наконец, сценарии Python могут запускать другие приложения с помощью функции `subprocess.Popen()`. Ей можно передавать аргументы командной строки, задавая документы, которые должны быть открыты в запускаемом приложении. Другой вариант заключается в том, чтобы запускать с помощью функции `Popen()` одну из программ `start`, `open` или `see`, позволяя операционной системе автоматически определять, в каком приложении должен быть открыт документ на основе имеющихся файловых ассоциаций. Взаимодействуя с другими приложениями, установленными в системе, сценарии Python могут задействовать их возможности для автоматизации решаемых задач.

Контрольные вопросы

1. Что такое эпоха Unix?
2. Какая функция возвращает количество секунд, прошедших с начала эпохи Unix?
3. Как сделать в программе паузу длительностью ровно 5 секунд?
4. Что возвращает функция `round()`?
5. В чем разница между объектами `datetime` и `timedelta`?
6. Как с помощью модуля `datetime` определить, на какой день недели выпало 7 января 2019 года?
7. Предположим, имеется функция `spam()`. Как запустить ее в отдельном потоке?
8. Что нужно сделать для того, чтобы избежать проблем параллелизма при работе с несколькими потоками?

Учебные проекты

Чтобы закрепить полученные знания на практике, напишите программы для предложенных ниже задач.

Наглядный секундомер

Расширьте программу-секундомер, рассмотренную в начале главы, "украсив" вывод за счет использования методов `rjust()` и `ljust()` (см. главу 6). Вместо

```
Замер #1: 3.56 (3.56)
Замер #2: 8.63 (5.07)
Замер #3: 17.68 (9.05)
Замер #4: 19.11 (1.43)
```

результаты должны выглядеть так:

```
Замер # 1:   3.56 (  3.56)
Замер # 2:   8.63 (  5.07)
Замер # 3:  17.68 (  9.05)
Замер # 4:  19.11 (  1.43)
```

Учтите, что для вызова указанных строковых методов вам понадобятся строковые версии целочисленных и вещественных переменных `lapNum`, `lapTime` и `totalTime`.

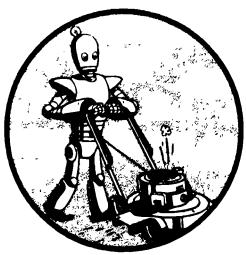
Далее воспользуйтесь модулем `pyperclip`, рассмотренным в главе 6, для копирования результатов работы программы в буфер обмена, благодаря чему пользователь сможет быстро вставить их в текстовый файл или в сообщение электронной почты.

Загрузка веб-комиксов по расписанию

Напишите программу, которая проверяет несколько сайтов веб-комиксов и автоматически загружает изображения в случае обновления комикса с момента последнего посещения сайта. Системный планировщик может запускать эту программу раз в день. Программа будет загружать комикс и копировать его на рабочий стол. Это избавит вас от необходимости самостоятельно посещать сайты и проверять наличие обновлений. (Список сайтов с веб-комиксами доступен по адресу `https://automatetheboringstuff.com/list-of-web-comics.html`.)

18

ОТПРАВКА ЭЛЕКТРОННОЙ ПОЧТЫ И ТЕКСТОВЫХ СООБЩЕНИЙ



Просмотр сообщений электронной почты и ответы на них отнимают много времени. Конечно, невозможно написать программу, которая обрабатывала бы всю электронную почту вместо вас, поскольку на каждое сообщение приходится отвечать по-разному. И все же имеется множество задач, связанных с обработкой электронной почты, которые поддаются автоматизации, при условии, что вам известно, как написать программу, способную отправлять и получать электронные письма.

Предположим, например, что у вас есть электронная таблица с данными о клиентах, и вы хотите отправить каждому из них письмо, форма которого зависит от возраста и места жительства клиента. Коммерческие программы тут не помогут. К счастью, можно написать для этих целей собственную программу, которая позволит сэкономить массу времени, избавив вас от многократного копирования и вставки формы письма.

Кроме того, можно написать программу, которая будет отправлять вам уведомления о важных событиях, даже когда вы находитесь вдали от компьютера. Если вы автоматизировали задачу, которая может выполняться несколько часов, то вряд ли захотите контролировать программу каждые пять минут, проверяя, не завершилась ли она. Вместо этого программа будет слать вам SMS сразу же после завершения, что даст вам возможность заняться другими делами.

В этой главе рассматривается модуль EZGmail, позволяющий отправлять и читать сообщения электронной почты через учетные записи Gmail, а также модули `smtplib` и `imapclient`, поддерживающие применение стандартных протоколов электронной почты SMTP и IMAP.

Предупреждение

Настоятельно рекомендуется создать отдельную учетную запись электронной почты для любых сценариев, которые отправляют или получают письма. Это защитит вашу личную учетную запись от последствий ошибок в программах (например, от случайного удаления электронных писем или непреднамеренного получения спама). Желательно сначала выполнить пробный прогон рискованной программы, закомментировав код отправки или удаления писем и заменив его временными вызовами функции `print()`*. Тем самым вы протестируете программу, прежде чем запускать ее по-настоящему.*

Отправка и получение электронной почты с помощью Gmail API

Gmail охватывает почти треть рынка почтовых клиентов, и наверняка у вас есть хотя бы один адрес Gmail. Благодаря наличию дополнительных мер безопасности и защиты от спама проще управлять учетной записью Gmail с помощью модуля EZGmail, а не модулей `smtplib` и `imapclient`, о которых пойдет речь далее. EZGmail — это модуль, написанный автором книги. Он работает поверх официального Gmail API и предоставляет функции, облегчающие взаимодействие с Gmail из Python. Полная информация о EZGmail доступна на сайте `https://github.com/asweigart/ezgmail/`. Этот модуль никак не связан с Google и не имеет отношения к веб-службам

данной компании. Официальная документация по Gmail API предоставляется на сайте `https://developers.google.com/gmail/api/v1/reference/`.

Чтобы установить модуль EZGmail в Windows, выполните команду `pip install --user --upgrade ezgmail` (в macOS и Linux используйте утилиту `pip3`). Флаг `--upgrade` обеспечит установку последней версии пакета, которая необходима для взаимодействия с постоянно обновляющейся веб-службой, такой как Gmail.

Подключение Gmail API

Прежде чем приступать к написанию программ, создайте учетную запись электронной почты Gmail на сайте `https://gmail.com/`. Затем перейдите на сайт `https://developers.google.com/gmail/api/quickstart/python/`, щелкните на кнопке Enable the Gmail API и заполните появившуюся форму.

После заполнения формы на странице появится ссылка на файл *credentials.json*, который нужно загрузить и поместить в ту же папку, где находится файл *.py*. Файл *credentials.json* содержит информацию об идентификаторе клиента (Client ID) и секретном коде (Client Secret), к которой следует относиться так же, как к паролю Gmail, и никому не передавать.

Теперь введите в интерактивной оболочке следующие инструкции.

```
>>> import ezgmail, os
>>> os.chdir(r'C:\path\to\credentials_json_file')
>>> ezgmail.init()
```

Убедитесь в том, что текущий каталог соответствует папке, в которой находится файл *credentials.json*, и что имеется подключение к Интернету. Функция `ezgmail.init()` откроет в браузере страницу входа в Google. Введите свой адрес Gmail и пароль. Может появиться предупреждение "This app isn't verified" (Это приложение не верифицировано), но не обращайте на это внимание. Просто щелкните на кнопке Advanced, а затем выберите ссылку Go to Quickstart (unsafe). (Если вы пишете сценарии Python для других пользователей и не хотите, чтобы они видели это предупреждение, ознакомьтесь в Интернете с информацией о процессе верификации приложений Google.) Когда на следующей странице появится запрос "Quickstart wants to access your Google Account", щелкните на кнопке Allow и закройте браузер.

В результате будет сгенерирован файл *token.json*, который позволит сценариям Python получать доступ к вашей учетной записи Gmail. Браузер откроет страницу входа только в том случае, если не сможет найти существующий файл *token.json*. С помощью файлов *credentials.json* и *token.json* сценарии Python смогут отправлять и читать электронные письма из вашей учетной записи Gmail, не требуя указания пароля Gmail в исходном коде.

Отправка электронной почты через учетную запись Gmail

Как только в вашем распоряжении появится файл *token.json*, модуль EZGmail сможет отправлять электронную почту с помощью единственного вызова функции.

```
>>> import ezgmail
>>> ezgmail.send('recipient@example.com', 'Тема', 'Тело письма')
```

Если необходимо прикрепить к письму файлы, предоставьте функции `send()` дополнительный аргумент-список:

```
>>> ezgmail.send('recipient@example.com', 'Тема', 'Тело письма',
                 ['вложение1.jpg', 'вложение2.mp3'])
```

Имейте в виду, что в рамках стратегии безопасности и защиты от спама Gmail может блокировать отправку повторных писем с одинаковым текстом (он распознаются как спам) или писем, содержащих файлы с расширениями *.exe* или *.zip* (они распознаются как вирусы).

Можно также передавать необязательные именованные аргументы `cc` и `bcc` для отправки копий и скрытых копий.

```
>>> import
>>> ezgmail.send('recipirnt@example.com', 'Тема', 'Тело письма',
                 cc='friend@example.com',
                 bcc='otherfriend@example.com, someone@example.com')
```

Если хотите узнать, для какого адреса Gmail настроен файл *token.json*, просмотрите значение переменной `ezgmail.EMAIL_ADDRESS`. Учтите, что эта переменная получает значение только после вызова метода `ezgmail.init()` или любой другой функции модуля EZGmail.

```
>>> import ezgmail
>>> ezgmail.init()
>>> ezgmail.EMAIL_ADDRESS
'example@gmail.com'
```

Относитесь к файлу *token.json* как к паролю. Если кто-то получит этот файл, то получит и доступ к вашей учетной записи Gmail (хоть и не сможет изменить ваш пароль Gmail). Чтобы отозвать ранее созданные файлы *token.json*, перейдите по адресу `https://security.google.com/settings/security/permissions?pli=1/` и отмените доступ к Quickstart. После этого потребуется вызвать метод `ezgmail.init()` и заново пройти процедуру входа в систему, чтобы получить новый файл *token.json*.

Чтение электронной почты с помощью учетной записи Gmail

Gmail организует ответные письма в цепочки сообщений. Когда вы входите в Gmail через браузер или через приложение, вы фактически видите цепочки писем, а не отдельные письма (даже если в цепочке всего одно письмо).

Модуль EZGmail содержит объекты `GmailThread` и `GmailMessage` для представления цепочек сообщений и отдельных писем соответственно. У объекта `GmailThread` есть атрибут `messages`, содержащий список объектов `GmailMessage`. Функция `unread()` возвращает список объектов `GmailThread` для всех непрочитанных писем, который затем можно передать функции `ezgmail.summary()`, чтобы вывести сводку цепочек сообщений в этом списке.

```
>>> import ezgmail
>>> unreadThreads = ezgmail.unread()  # список объектов GmailThread
>>> ezgmail.summary(unreadThreads)
Al, Jon - Do you want to watch RoboCop this weekend? - Dec 09
Jon - Thanks for stopping me from buying Bitcoin. - Dec 09
```

Функция `summary()` удобна для отображения быстрой сводки по темам писем, но для доступа к конкретным сообщениям (и их фрагментам) необходимо исследовать атрибут `messages` объекта `GmailThread`. Этот атрибут содержит список объектов `GmailMessage`, составляющих цепочку. У каждого такого объекта есть атрибуты `subject`, `body`, `timestamp`, `sender` и `recipient`, описывающие содержимое письма.

```
>>> len(unreadThreads)
2
>>> str(unreadThreads[0])
"<GmailThread len=2 snippet='Do you want to watch RoboCop this
weekend?'>"
>>> len(unreadThreads[0].messages)
2
>>> str(unreadThreads[0].messages[0])
"<GmailMessage from='Al Sweigart <al@inventwithpython.com>'
to='Jon Doe <example@gmail.com>' timestamp=datetime.datetime(2018,
12, 9, 13, 28, 48) subject='RoboCop' snippet='Do you want to watch
RoboCop this weekend?'>"
>>> unreadThreads[0].messages[0].subject
'RoboCop'
>>> unreadThreads[0].messages[0].body
'Do you want to watch RoboCop this weekend?\r\n'
>>> unreadThreads[0].messages[0].timestamp
datetime.datetime(2018, 12, 9, 13, 28, 48)
>>> unreadThreads[0].messages[0].sender
'Al Sweigart <al@inventwithpython.com>'
```

```
>>> unreadThreads[0].messages[0].recipient
'Jon Doe <example@gmail.com>'
```

Функция `ezgmail.recent()` возвращает 25 самых последних цепочек писем в вашей учетной записи Gmail. Ей можно передать необязательный именованный аргумент `maxResults`, чтобы изменить глубину поиска.

```
>>> recentThreads = ezgmail.recent()
>>> len(recentThreads)
25
>>> recentThreads = ezgmail.recent(maxResults=100)
>>> len(recentThreads)
46
```

Поиск почты в учетной записи Gmail

Функция `ezgmail.search()` позволяет искать электронные письма так же, как если бы вы вводили запросы в поле поиска на сайте `https://gmail.com/`.

```
>>> resultThreads = ezgmail.search('RoboCop')
>>> len(resultThreads)
1
>>> ezgmail.summary(resultThreads)
Al, Jon - Do you want to watch RoboCop this weekend? - Dec 09
```

Функция `search()` выдаст те же результаты, что и поиск фразы "Robo Cop" в поле поиска (рис. 18.1).

Рис. 18.1. Поиск писем, содержащих фразу "RoboCop", на сайте Gmail

Подобно функциям `unread()` и `recent()`, функция `search()` возвращает список объектов `GmailThread`. Ей также можно передать любой из специальных операторов, доступных в поле поиска, например:

- `'label:UNREAD'` — поиск непрочитанных сообщений;
- `'from:al@inventwithpython.com'` — поиск сообщений, полученных от адреса `al@inventwithpython.com`;
- `'subject:hello'` — поиск сообщений со словом "hello" в теме;
- `'has:attachment'` — поиск сообщений с файловыми вложениями;

Полный список поисковых операторов доступен по адресу `https://support.google.com/mail/answer/7190?hl=en/`.

Загрузка вложений из писем Gmail

У объектов `GmailMessage` есть атрибут `attachments`, который представляет собой список имен файлов, вложенных в сообщение. Любое из этих имен можно передать методу `downloadAttachment()` объекта `GmailMessage` для загрузки соответствующего файла. Можно также загрузить все сообщения сразу с помощью метода `downloadAllAttachments()`. По умолчанию модуль EZGmail сохраняет вложения в текущем каталоге. Если требуется задать другой каталог, используйте необязательный именованный аргумент `downloadFolder` методов `downloadAttachment()` и `downloadAllAttachments()`.

```
>>> import ezgmail
>>> threads = ezgmail.search('vacation photos')
>>> threads[0].messages[0].attachments
['tulips.jpg', 'canal.jpg', 'bicycles.jpg']
>>> threads[0].messages[0].downloadAttachment('tulips.jpg')
>>> threads[0].messages[0].downloadAllAttachments(
        downloadFolder='vacation2019')
['tulips.jpg', 'canal.jpg', 'bicycles.jpg']
```

Если файл с именем вложения уже существует, он будет автоматически перезаписан.

Полная документация к модулю EZGmail доступна по адресу `https://github.com/asweigart/ezgmail/`.

SMTP

Подобно протоколу HTTP, который служит для отправки веб-страниц, протокол *SMTP* (Simple Mail Transfer Protocol) применяется для передачи сообщений электронной почты. SMTP определяет порядок форматирования и шифрования писем, стандартизирует процесс их ретрансляции между почтовыми серверами, а также описывает другие детали обработки почтовых сообщений. Впрочем, знать технические подробности вовсе не обязательно, поскольку благодаря модулю `smtplib` все сводится к использованию нескольких функций.

Протокол SMTP отвечает лишь за отправку почты. Для получения писем применяется другой протокол: IMAP.

В дополнение к SMTP и IMAP, большинство служб электронной почты задействует другие меры безопасности для защиты от спама, фишинга и прочих вредоносных применений электронной почты. Эти меры

предотвращают доступ сценариев Python к учетным записям электронной почты с помощью модулей `smtplib` и `imapclient`. В то же время у многих почтовых служб есть программные интерфейсы и специальные модули Python, которые позволяют сценариям получать к ним доступ. В этой главе рассматривается модуль для работы с Gmail. В остальных случаях обращайтесь к онлайн-документации.

Отправка электронной почты по протоколу SMTP

Вы наверняка привыкли отправлять электронную почту с помощью таких приложений, как Outlook или Thunderbird, либо посредством таких сайтов, как Gmail или Yahoo! Mail. К сожалению, Python не предлагает удобный графический интерфейс для работы с электронной почтой. Вместо этого придется вызывать функции, реализующие взаимодействие по протоколу SMTP, как показано в следующем интерактивном примере.

Примечание

Не пытайтесь выполнять этот пример в интерактивной оболочке. Он не будет работать, поскольку `smtp.example.com`, `bob@example.com`, `МОЙ_ПАРОЛЬ` и `alice@example.com` – строки-заменители. Данные инструкции дают лишь общее представление о том, как реализуется отправка электронной почты в Python.

```
>>> import smtplib
>>> smtpObj = smtplib.SMTP('smtp.example.com', 587)
>>> smtpObj.ehlo()
(250, b'mx.example.com at your service, [216.172.148.131]\nSIZE
35882577\n8BITMIME\nSTARTTLS\nENHANCEDSTATUSCODES\nCHUNKING')
>>> smtpObj.starttls()
(220, b'2.0.0 Ready to start TLS')
>>> smtpObj.login('bob@example.com', 'МОЙ_ПАРОЛЬ')
(235, b'2.7.0 Accepted')
>>> smtpObj.sendmail('bob@example.com', 'alice@example.com',
'Subject: So long.\nDear Alice, so long and thanks for all the
fish. Sincerely, Bob')
{}
>>> smtpObj.quit()
(221, b'2.0.0 closing connection ko10sm23097611pbd.52 - gsmtp')
```

В следующих разделах мы рассмотрим каждый шаг, подставляя вместо строк-заменителей ваши реальные данные. Вы узнаете, как установить соединение с SMTP-сервером, войти в свою учетную запись, отправить сообщение и разорвать соединение с сервером.

Подключение к серверу SMTP

Если вам когда-либо приходилось настраивать Thunderbird, Outlook или любую другую почтовую программу для подключения к своей учетной записи электронной почты, то, вероятно, вы знакомы с процедурой конфигурирования SMTP-сервера и порта. У каждого почтового провайдера эти настройки будут разными. Выполните в Интернете поиск по фразе "*<ваш_провайдер>* настройки SMTP", чтобы узнать, как настроить соединение с сервером через нужный порт.

Как правило, доменное имя SMTP-сервера будет совпадать с доменным именем провайдера, дополненным префиксом `smtp`. Например, в случае Verizon имя SMTP-сервера — `smtp.verizon.com`. В табл. 18.1 приведены имена SMTP-серверов некоторых популярных провайдеров электронной почты. (*Порт* — это целочисленное значение, почти всегда равное 587, которое используется протоколом TLS.)

Таблица 18.1. Провайдеры электронной почты и их SMTP-серверы

Провайдер	Доменное имя SMTP-сервера
Gmail*	`smtp.gmail.com`
Outlook.com/Hotmail.com*	`smtp-mail.outlook.com`
Yahoo Mail*	`smtp.mail.yahoo.com`
AT&T	`smpt.mail.att.net` (порт 465)
Comcast	`smtp.comcast.net`
Verizon	`smtp.verizon.net` (порт 465)

* Дополнительные меры безопасности не позволяют сценариям Python войти на эти серверы с помощью модуля `smtplib`. Модуль EZGmail решает проблему для учетных записей Gmail.

Как только вы определите доменное имя и порт, которые используются для подключения к серверу вашего почтового провайдера, создайте объект `SMTP` с помощью функции `smtplib.SMTP()`, передав ей два аргумента: строку доменного имени и целочисленный номер порта. Объект `SMTP` управляет подключением к почтовому SMTP-серверу и располагает методами для отправки писем. Ниже показано, как создать объект `SMTP` для подключения к вымышленному серверу электронной почты:

```
>>> smtpObj = smtplib.SMTP('smtp.example.com', 587)
>>> type(smtpObj)
<class 'smtplib.SMTP'>
```

Инструкция `type(smtpObj)` позволяет убедиться, что в переменной `smtpObj` хранится объект `SMTP`. Этот объект нужен для вызова методов, которые позволят регистрироваться на почтовом сервере и отправлять

сообщения. Если вызов `smptlib.SMTP()` завершается неудачей, значит, ваш SMTP-сервер не поддерживает подключение по протоколу TLS через порт 587. В таком случае необходимо создать объект `SMTP` с помощью функции `smtplib.SMTP_SSL()`, указав порт 465.

```
>>> smtpObj = smtplib.SMTP_SSL('smtp.example.com', 465)
```

Примечание

При отсутствии подключения к Интернету Python сгенерирует ошибку `'socket.gaierror: [Errno 11004] getaddrinfo failed'` или аналогичную ей.

Для программы разница между протоколами TLS и SSL несущественна. Чтобы подключиться к своему SMTP-серверу, вам достаточно знать, какой стандарт шифрования он использует. В последующих примерах, выполняемых в интерактивной оболочке, переменная `smtpObj` содержит объект `SMTP`, возвращаемый функцией `smtplib.SMTP()` или `smtplib.SMTP_SSL()`.

Отправка строки приветствия серверу SMTP

После создания объекта `SMTP` необходимо вызвать метод со странным именем `ehlo()`, который отправит приветственное сообщение почтовому серверу. Это первый шаг установления соединения с сервером SMTP. Знать детали вовсе необязательно. Нужно лишь помнить, что для созданного объекта `SMTP` в первую очередь должен быть вызван метод `ehlo()`, иначе все последующие вызовы методов будут приводить к ошибке. Вот пример вызова метода `ehlo()`.

```
>>> smtpObj.ehlo()
(250, b'mx.example.com at your service, [216.172.148.131]\nSIZE
35882577\n8BITMIME\nSTARTTLS\nENHANCEDSTATUSCODES\nCHUNKING')
```

Если первый элемент возвращаемого кортежа — целое число `250` (код успешного завершения операции в SMTP), значит, что процедура приветствия прошла успешно.

Начало TLS-шифрования

Если вы подключаетесь к серверу SMTP через порт 587 (т.е. используете TLS-шифрование), то следующим должен быть вызван метод `starttls()`. Тем самым включается шифрование соединения. Если вы подключаетесь к порту 465 (т.е. используете протокол SSL), то шифрование уже настроено, и этот шаг пропускается.

Вот пример вызова метода `starttls()`.

```
>>> smtpObj.starttls()
(220, b'2.0.0 Ready to start TLS')
```

Метод `starttls()` переводит SMTP-соединение в режим TLS. Возвращаемое этим методом значение `220` означает, что сервер готов к работе.

Регистрация на сервере SMTP

После настройки зашифрованного соединения с SMTP-сервером можно выполнить процедуру входа, указав свое имя пользователя (обычно это ваш адрес электронной почты) и пароль при вызове метода `login()`.

```
>>> smtpObj.login('my_email_address@example.com',
                  'МОЙ_ПАРОЛЬ')
(235, b'2.7.0 Accepted')
```

В качестве первого аргумента передается адрес электронной почты, в качестве второго — пароль. Возвращаемое значение `235` говорит о том, что процедура аутентификации успешно пройдена. В случае ввода неверного пароля Python сгенерирует исключение `smtplib.SMTPAuthenticationError`.

Предупреждение

Будьте осторожны, указывая пароли в исходном коде. Если кто-то посторонний скопирует вашу программу, то получит доступ к вашей учетной записи! Лучше вызывать метод `input()`, чтобы пользователь сам ввел пароль. Возможно, необходимость вводить пароль при каждом запуске программы доставит определенные неудобства, зато так вы не будете хранить пароль в незашифрованном файле, до которого легко сможет добраться хакер или вор, похитивший ваш ноутбук.

Отправка письма

После регистрации на SMTP-сервере почтового провайдера можно начать вызывать метод `sendmail()` для отправки сообщений электронной почты. Вот пример вызова метода `sendmail()`.

```
>>> smtpObj.sendmail('my_email_address@example.com',
'recipient@example.com', 'Subject: So long.\nDear Alice, so long
and thanks for all the fish. Sincerely, Bob')
{}
```

У метода `sendmail()` три аргумента:

- адрес электронной почты отправителя в виде строки (для поля "От");
- адрес электронной почты получателя в виде строки или список строк в случае нескольких получателей (для поля "Кому");
- тело сообщения в виде строки.

Строка с телом сообщения *должна* начинаться с текста `'Subject: \n'`, задающего тему сообщения. Символ новой строки `'\n'` отделяет строку темы от основного текста сообщения.

Метод `sendmail()` возвращает словарь, в котором будет содержаться по одной паре "ключ – значение" для каждого из получателей, кому *не удалось* доставить сообщение. Пустой словарь означает, что сообщение было *успешно* доставлено всем получателям.

Разрыв соединения с сервером SMTP

После отправки электронной почты не забудьте вызвать метод `quit()`. Это приведет к разрыву соединения с SMTP-сервером.

```
>>> smtpObj.quit()
(221, b'2.0.0 closing connection ko10sm23097611pbd.52 - gsmtp')
```

Возвращаемое значение `221` означает завершение сеанса связи.

IMAP

Подобно тому как протокол SMTP применяется для отправки электронной почты, протокол *IMAP* (Internet Message Access Protocol) определяет порядок обмена данными с почтовым сервером для получения электронных писем, высланных на ваш адрес. В Python имеется стандартный модуль `imaplib`, но проще использовать сторонний модуль `imapclient`. Документация к нему доступна на сайте `https://imapclient.readthedocs.org/`.

Модуль `imapclient` загружает электронную почту из хранилища на IMAP-сервере в довольно сложном формате. Скорее всего, вы будете конвертировать письма из этого формата в простые строковые значения. Всю трудоемкую работу по синтаксическому анализу писем выполняет модуль `pyzmail`, документация к которому доступна на сайте `https://www.magiksys.net/pyzmail/`.

Установите модули `imapclient` и `pyzmail` с помощью команд `pip install --user -U imapclient==2.1.0` и `pip install --user -U pyzmail36==1.0.4` в Windows (в macOS и Linux используйте утилиту `pip3`). Процедура установки сторонних модулей описана в приложении А.

Получение и удаление сообщений электронной почты по протоколу IMAP

Поиск и получение сообщений электронной почты в Python — многоэтапный процесс, требующий использования сторонних модулей `imapclient` и `pyzmail`. Чтобы вы могли получить общее представление, ниже приведен пример, демонстрирующий прохождение каждого этапа: вход на сервер IMAP, поиск сообщений, получение писем и последующее извлечение из них текста.

```
>>> import imapclient
>>> imapObj = imapclient.IMAPClient('imap.gmail.com', ssl=True)
>>> imapObj.login('my_email_address@example.com', 'МОЙ_ПАРОЛЬ')
'my_email_address@example.com Jane Doe authenticated (Success)'
>>> imapObj.select_folder('INBOX', readonly=True)
>>> UIDs = imapObj.search(['SINCE 05-Jul-2019'])
>>> UIDs
[40032, 40033, 40034, 40035, 40036, 40037, 40038, 40039, 40040,
 40041]
>>> rawMessages = imapObj.fetch([40041], ['BODY[]', 'FLAGS'])
>>> import pyzmail
>>> message =
    pyzmail.PyzMessage.factory(rawMessages[40041][b'BODY[]'])
>>> message.get_subject()
'Hello!'
>>> message.get_addresses('from')
[('Edward Snowden', 'esnowden@nsa.gov')]
>>> message.get_addresses('to')
[(Jane Doe', 'jdoe@example.com')]
>>> message.get_addresses('cc')
[]
>>> message.get_addresses('bcc')
[]
>>> message.text_part != None
True
>>> message.text_part.get_payload().decode(message.text_part.charset)
'Follow the money.\r\n\r\n-Ed\r\n'
>>> message.html_part != None
True
>>> message.html_part.get_payload().decode(message.html_part.charset)
'<div dir="ltr"><div>So long, and thanks for all the fish!
<br><br></div>-Al<br></div>\r\n'
>>> imapObj.logout()
```

Пока что не нужно запоминать эти действия. После того как мы подробно разберем каждый этап, вы сможете вернуться к этому примеру, чтобы освежить материал в памяти.

Подключение к серверу IMAP

Точно так же, как для подключения к SMTP-серверу и отправки сообщений электронной почты требуется объект `SMTP`, для установления соединения с IMAP-сервером и получения писем требуется объект `IMAPClient`. Первое, что вам понадобится, – доменное имя IMAP-сервера вашего почтового провайдера. Это имя будет отличаться от доменного имени SMTP-сервера. В табл. 18.2 приведены имена IMAP-серверов некоторых популярных провайдеров электронной почты.

Таблица 18.2. Провайдеры электронной почты и их IMAP-серверы

Провайдер	Доменное имя IMAP-сервера
Gmail*	`imap.gmail.com`
Outlook.com/Hotmail.com*	`imap-mail.outlook.com`
Yahoo Mail*	`imap.mail.yahoo.com`
AT&T	`imap.mail.att.net`
Comcast	`imap.comcast.net`
Verizon	`incoming.verizon.net`

* Дополнительные меры безопасности не позволяют сценариям Python войти на эти серверы с помощью модуля `imapclient`.

Определив доменное имя IMAP-сервера, вызовите функцию `imapclient.IMAPClient()` для создания объекта `IMAPClient`. Большинство почтовых провайдеров применяют SSL-шифрование, поэтому передайте функции именованный аргумент `ssl=True`. Введите в интерактивной оболочке следующие инструкции (используя доменное имя своего провайдера).

```
>>> import imapclient
>>> imapObj = imapclient.IMAPClient('imap.example.com', ssl=True)
```

Во всех интерактивных примерах, приводимых в последующих разделах, переменная `imapObj` содержит объект `IMAPClient`, возвращаемый функцией `imapclient.IMAPClient()`. В данном контексте *клиент* – это объект, который подключается к серверу.

Регистрация на сервере IMAP

После создания объекта `IMAPClient` вызовите его метод `login()`, передав ему имя пользователя (обычно это ваш адрес электронной почты) и пароль в виде строк.

```
>>> imapObj.login('my_email_address@example.com', 'МОЙ_ПАРОЛЬ')
'my_email_address@example.com Jane Doe authenticated (Success)'
```

Предупреждение

Не забывайте о том, что не рекомендуется хранить пароль непосредственно в коде. Проектируйте программу так, чтобы она запрашивала пароль с помощью функции `input()`.

Если IMAP-сервер отвергнет переданную ему комбинацию *имя_пользователя/пароль*, то Python сгенерирует исключение `imaplib.error`.

Поиск сообщений

Поиск писем, который становится возможным после входа на сервер, осуществляется в два этапа: сначала нужно выбрать папку, в которой будет выполняться поиск, а затем вызвать метод `search()` объекта `IMAPClient`, передав ему строку ключей поиска IMAP.

Выбор папки

По умолчанию почти в каждой учетной записи имеется папка `INBOX`. Можно также получить список папок, вызвав метод `list_folders()` объекта `IMAPClient`, который возвращает список кортежей. Каждый кортеж содержит информацию об одной папке. Введите в интерактивной оболочке следующие инструкции.

```
>>> import pprint
>>> pprint.pprint(imapObj.list_folders())
[(('\\HasNoChildren',), '/', 'Drafts'),
(('\\HasNoChildren',), '/', 'Filler'),
(('\\HasNoChildren',), '/', 'INBOX'),
(('\\HasNoChildren',), '/', 'Sent'),
-- Опущено --
 (('\\HasNoChildren', '\\Flagged'), '/', 'Starred'),
(('\\HasNoChildren', '\\Trash'), '/', 'Trash')]
```

Три значения, входящие в каждый кортеж, например `(('\\HasNoChildren',), '/', 'INBOX')`, имеют следующий смысл:

- кортеж флагов папки (подробное обсуждение этих флагов выходит за рамки книги; можете игнорировать данное поле);
- разделитель, используемый в строке имени для разделения родительских папок и подпапок;
- полное имя папки.

Чтобы выбрать папку, в которой должен осуществляться поиск, передайте ее имя в виде строки методу `select_folder()` объекта `IMAPClient`.

```
>>> imapObj.select_folder('INBOX', readonly=True)
```

Значение, возвращаемое методом `select_folder()`, можно игнорировать. Если выбранной папки не существует, Python сгенерирует исключение `imaplib.error`.

Именованный аргумент `readonly=True` позволяет избежать случайного изменения или удаления любого сообщения в данной папке при последующих вызовах методов. Если вы не планируете удалять сообщения, то имеет смысл всегда устанавливать значение именованного аргумента `readonly` равным `True`.

Процедура поиска

После выбора папки можно приступать к поиску сообщений, используя метод `search()` объекта `IMAPClient`. Аргументом, передаваемым методу `search()`, будет список строк, каждая из которых содержит поисковые ключи IMAP (табл. 18.3).

Таблица 18.3. Поисковые ключи IMAP

Ключ поиска	Описание
`'ALL'`	Поиск всех сообщений, хранящихся в данной папке. Запрашивая все сообщения в крупной папке, вы рискуете столкнуться с ограничениями, которые модуль `imaplib` налагает на допустимый суммарный размер загружаемых сообщений (об этом будет говориться далее)
`'BEFORE` *дата*`'`, `'ON` *дата*`'`, `'SINCE` *дата*`'`	Эти три ключа задают поиск сообщений, помеченных соответственно более ранней датой, чем текущая, текущей датой и более поздней датой, чем текущая. Требуемый формат даты — `'05-Jul-2019'`. Также учитывайте следующий нюанс: строке `'SINCE 05-Jul-2019'` соответствуют сообщения, полученные 5 июля или позже, тогда как строке `'BEFORE 05-Jul-2019'` соответствуют лишь сообщения, предшествующие 5 июля (сама эта дата не включается)
`'SUBJECT` *строка*`'`, `'BODY` *строка*`'`, `'TEXT` *строка*`'`	Поиск сообщений, содержащих заданную строку в поле темы, в теле письма или в любом из этих блоков. Строки с пробелами следует заключать в кавычки, например: `'TEXT "поиск с учетом пробелов"'`
`'FROM` *строка*`'`, `'TO` *строка*`'`, `'CC` *строка*`'`, `'BCC` *строка*`'`	Поиск сообщений, содержащих заданную строку в поле "От", "Кому", "Сс" (Копия) или "Bcc" (Скрытая копия) соответственно. При наличии нескольких адресов их следует разделять пробелами и заключать в кавычки, например: `'CC "firstcc@example.com secondcc@example.com"'`
`'SEEN'`, `'UNSEEN'`	Поиск сообщений, соответственно помеченных или не помеченных флагом `\Seen` (просмотрено). Этим флагом помечаются сообщения, к которым осуществлялся доступ с помощью метода `fetch()` или на которых вы выполнили щелчок во время просмотра почты с помощью клиентской почтовой программы

Окончание табл. 18.3

Ключ поиска	Описание
`'ANSWERED'`, `'UNANSWERED'`	Поиск всех сообщений, соответственно помеченных или не помеченных флагом `\Answered`. Сообщение помечается этим флагом, если на него был дан ответ
`'DELETED'`, `'UNDELETED'`	Поиск сообщений, соответственно помеченных или не помеченных флагом `\Deleted`. Сообщения, удаленные с помощью метода `delete_messages()`, снабжаются флагом `\Deleted`, но не удаляются безвозвратно до тех пор, пока не будет вызван метод `expunge()` (об этом будет рассказано далее). Учите, что некоторые почтовые провайдеры автоматически выполняют безвозвратное удаление сообщений
`'DRAFT'`, `'UNDRAFT'`	Поиск сообщений, соответственно помеченных или не помеченных флагом `\Draft` (черновик). Как правило, черновики сообщений хранятся в отдельной папке `Drafts`, а не в папке `INBOX`
`'FLAGGED'`, `'UNFLAGGED'`	Поиск сообщений, соответственно помеченных или не помеченных флагом `\Flagged`. Обычно этот флаг используется для пометки сообщений как важных или срочных
`'LARGER` *N*`'`, `'SMALLER` *N*`'`	Поиск сообщений, размер которых соответственно больше или меньше *N* байт
`'NOT` *ключ_поиска*`'`	Поиск сообщений, которые не будут найдены по указанному ключу поиска
`'OR` *ключ_поиска1 ключ_поиска2*`'`	Поиск сообщений, которые соответствуют первому или второму ключу поиска

Следует учитывать, что разные IMAP-серверы могут по-разному обрабатывать свои флаги и ключи поиска. Для того чтобы выяснить, как ведет себя конкретный сервер, вам придется немного поэкспериментировать в интерактивной оболочке.

Методу `search()` можно передать список с несколькими строками поисковых ключей IMAP. При этом метод вернет те сообщения, которые соответствуют *всем* ключам. Если вам достаточно соответствия *любому* из ключей, то используйте поисковый ключ `OR`. За ключами `NOT` и `OR` должны следовать один или два полных поисковых ключа соответственно.

Ниже приведены примеры вызовов метода `search()` с краткими пояснениями.

- **`imapObj.search(['ALL'])`**. Возвращает все сообщения, хранящиеся в текущей выбранной папке.
- **`imapObj.search(['ON 05-Jul-2019'])`**. Возвращает все сообщения, отправленные 5 июля 2019 года.
- **`imapObj.search(['SINCE 01-Jan-2019', 'BEFORE 01-Feb-2019', 'UNSEEN'])`**. Возвращает все сообщения, отправленные в январе 2019 года, которые остались непрочитанными. (Заметьте, что указанный период *включает* 1 января и все последующие дни вплоть до, *но не включая*, 1 февраля.)

- **imapObj.search(['SINCE 01-Jan-2019', 'FROM alice@example.com'])**. Возвращает все сообщения от пользователя alice@example.com, отправленные с начала 2019 года.
- **imapObj.search(['SINCE 01-Jan-2019', 'NOT FROM alice@example.com'])**. Возвращает все сообщения от всех пользователей, кроме alice@example.com, отправленные с начала 2019 года.
- **imapObj.search(['OR FROM alice@example.com FROM bob@example.com'])**. Возвращает все сообщения, отправленные когда-либо пользователем alice@example.com или bob@example.com.
- **imapObj.search(['FROM alice@example.com', 'FROM bob@example.com'])**. Пример с подвохом! В результате этого поиска не будет получено ни одно сообщение, поскольку сообщения должны соответствовать *всем* поисковым критериям. Но в поле "От" сообщения может находиться только один адрес, так что не может быть сообщений, в которых в качестве отправителя фигурировали бы одновременно и alice@example.com, и bob@example.com.

Метод search() возвращает не сами сообщения, а их уникальные идентификаторы (UID) в виде целых чисел. Чтобы получить содержимое сообщений, следует передать эти идентификаторы методу fetch().

Введите в интерактивной оболочке следующие инструкции.

```
>>> UIDs = imapObj.search(['SINCE 05-Jul-2019'])
>>> UIDs
[40032, 40033, 40034, 40035, 40036, 40037, 40038, 40039, 40040, 40041]
```

Здесь список идентификаторов сообщений (полученных начиная с 5 июля), возвращаемый методом search(), сохраняется в переменной UIDs. Список UIDs, полученный на вашем компьютере, будет другим: идентификаторы уникальны для конкретной учетной записи электронной почты. Когда впоследствии вы будете использовать идентификаторы в других вызовах функций, указывайте значения, полученные вами, а не те, которые представлены в примерах книги.

Предельный размер сообщений

Если в результате поиска обнаруживается слишком большое количество сообщений, удовлетворяющих заданным критериям, Python может сгенерировать исключение вида 'imaplib.error: got more than 10000 bytes'. Когда такое происходит, необходимо разорвать, а затем восстановить соединение с IMAP-сервером и попытаться вновь выполнить поиск.

Этот предел введен для того, чтобы не позволить программам Python потреблять слишком много памяти. К сожалению, заданный по умолчанию предельный размер сообщений слишком мал. Можете изменить его

с 10 000 на 10 000 000 байт, выполнив следующие инструкции, что позволит вам избавиться от повторного получения таких сообщений.

```
>>> import imaplib
>>> imaplib._MAXLINE = 10000000
```

Имеет смысл включать эти две строки кода во все программы для работы с IMAP, которые вы будете писать.

Получение сообщений электронной почты и пометка их как прочитанных

Имея список идентификаторов сообщений, можно вызвать метод `fetch()` объекта `IMAPClient` для получения фактического содержимого писем.

Список идентификаторов (`UIDs`) — это первый аргумент метода `fetch()`. Вторым аргументом должен быть список `['BODY[]']`, который инструктирует метод `fetch()` загрузить все содержимое тела сообщений, указанных в первом аргументе.

Введите в интерактивной оболочке следующие инструкции.

```
>>> rawMessages = imapObj.fetch(UIDs, ['BODY[]'])
>>> import pprint
>>> pprint.pprint(rawMessages)
{40040: {'BODY[]': 'Delivered-To: my_email_address@gmail.com\r\n'
                   'Received: by 10.76.71.167 with SMTP id '
-- Опущено --
                   '\r\n'
                   '------=_Part_6000970_707736290.1404819487066--
                    \r\n',
        'SEQ': 5430}}
```

Импортируйте модуль `pprint` и передайте значение, полученное от метода `fetch()` и сохраненное в переменной `rawMessages`, функции `pprint.pprint()`, чтобы вывести его на экран в красиво оформленном виде. Вы увидите, что это значение представляет собой словарь сообщений, в котором идентификаторы служат ключами. Каждое сообщение сохраняется в виде словаря с двумя ключами: `'BODY[]'` и `'SEQ'`. Ключ `'BODY[]'` соответствует фактическому телу письма. Ключ `'SEQ'` играет роль *порядкового номера* и выполняет функции, аналогичные UID. Можете смело его игнорировать.

Нетрудно заметить, что содержимое ключа `'BODY[]'` выглядит непонятно. Это обусловлено тем, что оно хранится в формате RFC 822, предназначенном для чтения серверами IMAP. Знать данный формат не нужно.

В следующем разделе мы познакомимся с модулем pyzmail, который позволяет представить тело сообщения в удобочитаемом виде.

Когда мы выбирали папку, в которой следует выполнить поиск, вы вызывали метод select_folder() с именованным аргументом readonly=True. Это делалось для того, чтобы предотвратить случайное удаление почты, но это также означает, что письма не будут помечены как прочитанные, если для их извлечения применяется метод fetch(). Чтобы письма помечались как прочитанные при извлечении, методу select_folder() следует передать именованный аргумент readonly=False. Если же текущая папка уже была выбрана для работы в режиме "только чтение", можно выбрать ее заново с помощью другого вызова метода select_folder(), на этот раз – с именованным аргументом readonly=False:

```
>>> imapObj.select_folder('INBOX', readonly=False)
```

Получение адресов электронной почты из необработанных сообщений

От необработанных сообщений, возвращаемых методом fetch(), мало пользы, если требуется всего лишь прочитать почту. Модуль pyzmail выполняет синтаксический анализ необработанных сообщений и возвращает их в виде объектов PyzMessage, с помощью которых можно легко получить доступ к теме, телу, полям "Кому" и "От", а также другим разделам письма.

Продолжите интерактивный пример, выполнив следующие инструкции (используйте уникальные идентификаторы сообщений из своей учетной записи, а не те, которые здесь представлены).

```
>>> import pyzmail
>>> message =
    pyzmail.PyzMessage.factory(rawMessages[40041][b'BODY[]'])
```

Сначала мы импортируем модуль pyzmail, а затем создаем объект PyzMessage сообщения, вызывая функцию pyzmail.PyzMessage.factory() и передавая ей раздел 'BODY[]' необработанного сообщения. (Наличие префикса b означает работу с байтовым, а не строковым значением. Разница не особо важна. Просто используйте данный префикс в коде.) Результат сохраняется в переменной message, которая теперь содержит объект PyzMessage. С помощью методов этого объекта можно получить строку темы сообщения, а также адреса отправителя и получателя. Метод get_subject() возвращает тему сообщения в виде обычной строки. Метод get_addresses() возвращает список адресов для переданного ему поля. Введите в интерактивной оболочке следующие инструкции.

```
>>> message.get_subject()
'Hello!'
>>> message.get_addresses('from')
[('Edward Snowden', 'esnowden@nsa.gov')]
>>> message.get_addresses('to')
[(Jane Doe', 'my_email_address@gmail.com')]
>>> message.get_addresses('cc')
[]
>>> message.get_addresses('bcc')
[]
```

В данном случае методу `get_addresses()` передаются аргументы `'from'`, `'to'`, `'cc'` и `'bcc'`. Значение, возвращаемое методом, представляет собой список кортежей. Каждый кортеж состоит из двух строк: первая из них — это имя, связанное с адресом электронной почты, вторая — сам адрес. Если запрошенное поле оказывается пустым, метод возвращает пустой список. В данном случае таковыми являются поля `'cc'` (копия) и `'bcc'` (скрытая копия), поэтому для них возвращаются пустые списки.

Получение тела письма из необработанного сообщения

Сообщения электронной почты могут быть отправлены в виде простого текста, в формате HTML или в комбинированном формате. В первом случае сообщения содержат только текст, тогда как в HTML-сообщениях могут использоваться цвета, различные шрифты, изображения и другие графические средства, благодаря которым электронные письма будут выглядеть как небольшие веб-страницы. Если письмо отправлено в текстовом виде, то значением атрибута `html_part` объекта `PyzMessage` будет `None`. Аналогичным образом, если в сообщении используется только формат HTML, то значением атрибута `text_part` объекта `PyzMessage` будет `None`.

Во всех остальных случаях у объекта, хранящегося в атрибуте `text_part` или `html_part`, будет метод `get_payload()`, который возвращает тело сообщения в виде значения байтового типа. (Этот тип данных в книге не рассматривается.) Это еще не то значение, которое нам нужно. Последний шаг заключается в вызове метода `decode()` для значения, возвращаемого методом `get_payload()`. У метода `decode()` один аргумент: кодировка символов сообщения, которая хранится в атрибуте `text_part.charset` или `html_part.charset`. Результат, возвращаемый методом `decode()`, представляет собой строку с телом сообщения.

Введите в интерактивной оболочке следующие инструкции.

```
❶ >>> message.text_part != None
  True
  >>> message.text_part.get_payload().decode(message.text_part.charset)
❷ 'So long, and thanks for all the fish!\r\n\r\n-Al\r\n'
```

```
❸ >>> message.html_part != None
  True
❹ >>> message.html_part.get_payload().decode(message.html_part.charset)
  '<div dir="ltr"><div>So long, and thanks for all the fish!<br><br>
  </div>-Al<br></div>\r\n'
```

Письмо, с которым мы работаем, содержит как простой текст, так и HTML-содержимое, поэтому в объекте `PyzMessage`, сохраненном в переменной `message`, имеются атрибуты `text_part` и `html_part`, значения которых не равны `None` ❶❸. В результате вызова метода `get_payload()` для атрибута `text_part` объекта `message` и последующего вызова метода `decode()` возвращается строка с текстом сообщения ❷. Вызов методов `get_payload()` и `decode()` для атрибута `html_part` объекта `message` позволяет получить HTML-версию сообщения ❹.

Удаление писем

Для удаления писем передайте список идентификаторов сообщений методу `delete_messages()` объекта `IMAPClient`. В результате сообщения помечаются флагом `\Deleted` (удалено). Вызов метода `expunge()` приведет к безвозвратному удалению всех сообщений в текущей папке, помеченных флагом `\Deleted`. Введите в интерактивной оболочке следующие инструкции.

```
❶ >>> imapObj.select_folder('INBOX', readonly=False)
❷ >>> UIDs = imapObj.search(['ON 09-Jul-2019'])
  >>> UIDs
  [40066]
  >>> imapObj.delete_messages(UIDs)
❸ {40066: ('\\Seen', '\\Deleted')}
  >>> imapObj.expunge()
  ('Success', [(5452, 'EXISTS')])
```

В этом примере мы выбираем папку `INBOX` (Входящие), вызывая метод `select_folder()` объекта `IMAPClient` и передавая ему строку `'INBOX'` в качестве первого аргумента. Кроме того, методу `select_folder()` передается именованный аргумент `readonly=False`, чтобы сделать возможным удаление писем ❶. В папке `INBOX` выполняется поиск сообщений с указанной датой получения, и идентификаторы этих сообщений сохраняются в списке `UIDs` ❷. Метод `delete_message()`, которому передается список `UIDs`, возвращает словарь. В этом словаре каждая пара "ключ – значение" представляет собой идентификатор сообщения и кортеж флагов, который теперь должен включать флаг `\Deleted` ❸. Последующий вызов метода `expunge()` безвозвратно удаляет письма, помеченные флагом `\Deleted`, и возвращает сообщение `'Success'`, если все прошло успешно. Имейте в виду, что

некоторые провайдеры автоматически удаляют сообщения безвозвратно, не дожидаясь соответствующей команды от клиента IMAP.

Разрыв соединения с сервером IMAP

Когда программа завершит обработку писем, вызовите метод `logout()` объекта `IMAPClient`, чтобы разорвать соединение с сервером IMAP.

```
>>> imapObj.logout()
```

Если программа выполняется несколько минут или дольше, то сервер IMAP может автоматически разорвать соединение по *тайм-ауту*. В подобных случаях очередная попытка вызова какого-либо метода объекта `IMAPClient` приведет к появлению следующего исключения.

```
imaplib.abort: socket error: [WinError 10054] An existing connection
was forcibly closed by the remote host
```

Если это произойдет, то для повторного установления соединения программа должна будет вновь вызвать функцию `imapclient.IMAPClient()`.

Наконец-то! Мы проделали долгий путь, зато теперь вы знаете, как заставить свои программы входить в учетные записи электронной почты и получать сообщения.

Проект: рассылка напоминаний об уплате членских взносов

Предположим, вы на добровольных началах вызвались вести учет уплаты взносов членами *Клуба обязательного волонтерства*. Это утомительное занятие, требующее составления электронной таблицы, в которой отмечается, кто из членов клуба уплатил ежемесячный членский взнос, а кто — нет, причем последним необходимо рассылать по электронной почте уведомления с напоминанием о необходимости уплаты взноса. Вместо того чтобы вручную просматривать список членов клуба и готовить письма-напоминания, копируя и вставляя в них один и тот же текст, лучше, как вы наверняка уже догадались, написать сценарий, который выполнит эту работу за вас.

Вот что должна делать такая программа:

1) читать данные из электронной таблицы Excel;
2) находить тех членов клуба, которые не уплатили взнос за прошедший месяц;
3) находить их адреса электронной почты и отправлять им персональные напоминания.

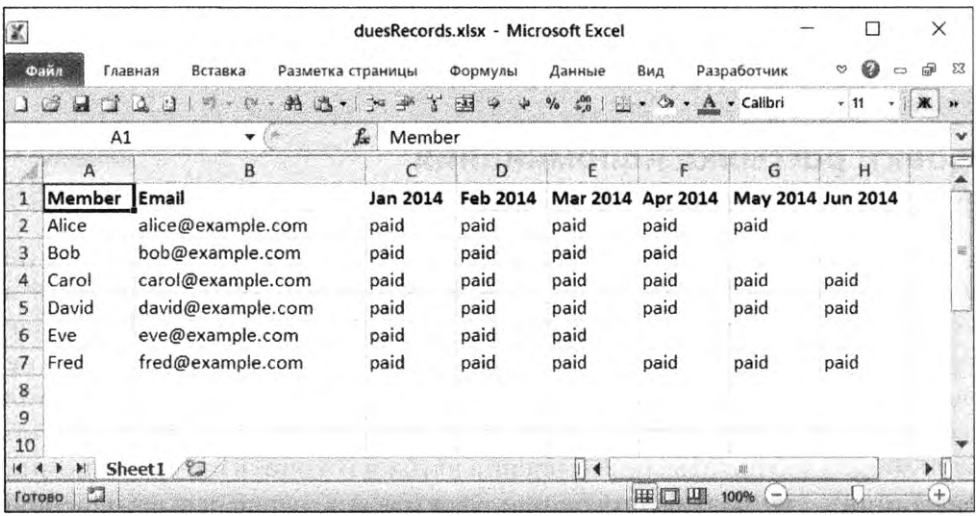
Это означает, что программа должна выполнять следующие операции:

1) открывать документ Excel и считывать содержимое его ячеек с помощью модуля `openpyxl` (о работе с таблицами Excel см. в главе 13);
2) создавать словарь членов клуба, не уплативших членские взносы;
3) входить в учетную запись на SMTP-сервере с помощью методов `SMTP()`, `ehlo()`, `starttls()` и `login()` модуля `smtplib`;
4) отправлять по электронной почте персональные напоминания членам клуба, просрочившим уплату взносов, с помощью метода `sendmail()`.

Откройте в файловом редакторе новое окно и сохраните программу в файле *sendDuesReminders.py*.

Шаг 1. Открытие файла Excel

Предположим, что электронная таблица Excel, которую вы используете для контроля уплаты членских взносов, выглядит примерно так, как показано на рис. 18.2, и хранится в файле *duesRecords.xlsx*. Этот файл доступен в архиве примеров книги (см. введение).



	A	B	C	D	E	F	G	H
1	Member	Email	Jan 2014	Feb 2014	Mar 2014	Apr 2014	May 2014	Jun 2014
2	Alice	alice@example.com	paid	paid	paid	paid	paid	
3	Bob	bob@example.com	paid	paid	paid	paid		
4	Carol	carol@example.com	paid	paid	paid	paid	paid	paid
5	David	david@example.com	paid	paid	paid	paid	paid	paid
6	Eve	eve@example.com	paid	paid	paid			
7	Fred	fred@example.com	paid	paid	paid	paid	paid	paid
8								
9								
10								

Рис. 18.2. Электронная таблица для учета уплаты членских взносов

В этой таблице хранятся имена членов клуба и адреса их электронной почты. Каждому месяцу соответствует отдельный столбец, в котором делаются отметки об уплате взносов. Отметкой об уплате взноса служит текст `'paid'`.

Программа должна открывать файл *duesRecords.xlsx* и находить столбец, соответствующий последнему месяцу, считывая атрибут `sheet.max_column`.

(Для получения более подробной информации о доступе к ячейкам электронных таблиц Excel с помощью модуля openpyxl см. главу 13.) Введите в файловом редакторе следующий код.

```
#! python3
# sendDuesReminders.py - рассылает сообщения на основании
# сведений из электронной таблицы об уплате взносов

import openpyxl, smtplib, sys

# Открытие электронной таблицы и получение последних
# данных об уплате взносов
❶ wb = openpyxl.load_workbook('duesRecords.xlsx')
❷ sheet = wb.get_sheet_by_name('Sheet1')
❸ lastCol = sheet.max_column
❹ latestMonth = sheet.cell(row=1, column=lastCol).value

# СДЕЛАТЬ: проверить статус уплаты взносов
#          для каждого члена клуба

# СДЕЛАТЬ: войти в учетную запись электронной почты

# СДЕЛАТЬ: отправить сообщения с напоминанием об уплате взносов
```

Импортировав модули openpyxl, smtplib и sys, мы открываем файл *duesRecords.xlsx* и сохраняем результирующий объект Workbook в переменной wb ❶. Далее мы получаем лист 'Sheet1' и сохраняем полученный объект Worksheet в переменной sheet ❷. Теперь, когда в нашем распоряжении имеется объект Worksheet, мы можем обращаться к строкам, столбцам и ячейкам электронной таблицы. Мы сохраняем номер последнего столбца в переменной lastCol ❸, а затем извлекаем содержимое ячейки с номером строки 1 и номером столбца lastCol и сохраняем его в переменной latestMonth ❹.

Шаг 2. Поиск всех членов клуба, не уплативших взнос

После того как мы определили столбец последнего месяца (строка месяца хранится в переменной lastCol), можно организовать цикл по всем его строкам, кроме первой (в ней находится заголовок столбца), чтобы выяснить, напротив фамилий каких членов клуба стоит отметка 'paid'. Если кто-либо из членов клуба не уплатил взнос, его имя и адрес электронной почты можно извлечь из столбцов 1 и 2 соответственно. Эта информация заносится в словарь unpaidMembers, с помощью которого будут отслеживаться члены клуба, пропустившие оплату за последний месяц. Добавьте в файл *sendDuesReminder.py* следующий код.

```
#! python3
# sendDuesReminders.py - рассылает сообщения на основании
# сведений из электронной таблицы об уплате взносов

-- Опущено --

# Проверка статуса уплаты взносов для каждого члена клуба
unpaidMembers = {}
❶ for r in range(2, sheet.max_row + 1):
❷     payment = sheet.cell(row=r, column=lastCol).value
      if payment != 'paid':
❸         name = sheet.cell(row=r, column=1).value
❹         email = sheet.cell(row=r, column=2).value
❺         unpaidMembers[name] = email
```

Мы создаем пустой словарь `unpaidMembers` и выполняем цикл по всем строкам, кроме первой ❶. Для каждой строки значение, находящееся в последнем столбце, сохраняется в переменной `payment` ❷. Если значение `payment` не равно `'paid'`, то строка, находящаяся в первом столбце, заносится в переменную `name` ❸, а строка, находящаяся во втором столбце – в переменную `email` ❹. Кроме того, обе переменные, `name` и `email`, добавляются в словарь `unpaidMembers` ❺.

Шаг 3. Отправка персональных напоминаний по электронной почте

Получив список всех членов клуба, не уплативших взнос, можно отправить им напоминания по электронной почте. Добавьте в программу следующий код, использовав в нем свой реальный адрес электронной почты и информацию о своем почтовом провайдере.

```
#! python3
# sendDuesReminders.py - рассылает сообщения на основании
# сведений из электронной таблицы об уплате взносов

-- Опущено --

# Вход в учетную запись электронной почты
smtpObj = smtplib.SMTP('smtp.example.com', 587)
smtpObj.ehlo()
smtpObj.starttls()
smtpObj.login('my_email_address@example.com', sys.argv[1])
```

Создайте объект SMTP, вызвав функцию `smtplib.SMTP()` и передав ей доменное имя и номер порта своего провайдера. Вызовите сначала методы `ehlo()` и `starttls()`, а затем – метод `login()`. Передайте методу `login()` свой адрес электронной почты и значение переменной `sys.argv[1]`, в которой будет храниться строка с вашим паролем. Пароль придется вводить

в командной строке при каждом запуске программы, чтобы не задавать его в исходном коде.

После того как программа войдет в вашу учетную запись электронной почты, она должна проанализировать содержимое словаря `unpaidMembers` и отправить по электронной почте персональные напоминания всем, кто указан в нем. Добавьте в файл *sendDuesReminders.py* следующий код.

```
#! python3
# sendDuesReminders.py - рассылает сообщения на основании
# сведений из электронной таблицы об уплате взносов

-- Опущено --

# Отправка сообщений с напоминанием об уплате взносов
for name, email in unpaidMembers.items():
❶   body = "Subject: %s: взносы не оплачены.\nУважаемый %s,
        вы до сих пор не уплатили членские взносы в клубе за %s.
        Пожалуйста, сделайте это как можно скорее. Заранее
        спасибо!" % (latestMonth, name, latestMonth)
❷   print('Отправка письма по адресу %s...' % email)
❸   sendmailStatus = smtpObj.sendmail('my_email_address@gmail.com',
                                       email, body)

❹   if sendmailStatus != {}:
        print('Проблемы с отправкой письма для %s: %s' % (email,
              sendmailStatus))
smtpObj.quit()
```

Здесь выполняется цикл по именам и адресам электронной почты, хранящимся в словаре `unpaidMembers`. Для каждого члена клуба, просрочившего уплату взноса, создается персональное сообщение, в котором используется информация о проверяемом месяце и имени члена клуба. Это сообщение сохраняется в переменной `body` ❶. Мы также выводим информационное сообщение об отправке письма в адрес данного члена клуба ❷. Затем мы вызываем метод `sendmail()`, передавая ему адрес отправителя и персонализированное сообщение ❸. Возвращаемое этим методом значение сохраняется в переменной `sendmailStatus`.

Помните о том, что в случае получения от SMTP-сервера сообщения об ошибке при отправке какого-либо сообщения метод `sendmail()` возвращает значение в виде непустого словаря. В конце цикла `for` ❹ проверяется, является ли словарь непустым, и, если это так, выводятся адрес электронной почты получателя и содержимое возвращенного словаря.

Когда программа завершит отправку всех сообщений, следует разорвать соединение с SMTP-сервером, вызвав метод `quit()`.

Запустив программу, вы должны получить примерно такие результаты.

```
Отправка письма по адресу alice@example.com...
Отправка письма по адресу bob@example.com...
Отправка письма по адресу eve@example.com...
```

Полученные членами клуба сообщения электронной почты будут выглядеть так же, как если бы вы отправили их вручную.

Отправка текстовых сообщений с помощью почтового шлюза SMS

Люди чаще пользуются смартфонами, чем сидят за компьютером, поэтому текстовые сообщения — зачастую более быстрый и надежный способ отправки уведомлений, чем электронная почта. Кроме того, текстовые сообщения, как правило, короче, что повышает вероятность их прочтения.

Самый простой, хотя и не самый надежный способ отправки текстовых сообщений — использование почтового шлюза SMS (службы коротких сообщений), т.е. почтового сервера, который был настроен мобильным оператором для получения текста по электронной почте с последующей пересылкой получателю в виде SMS.

Можно написать программу для отправки таких писем с помощью модулей `ezgmail` или `smtplib`. Номер телефона и почтовый сервер мобильного оператора составляют адрес электронной почты получателя. Темой и телом электронного письма будет само текстовое сообщение. Например, чтобы отправить текстовое сообщение на номер телефона 415-555-1234, принадлежащего клиенту Verizon, необходимо отправить электронное письмо по адресу `4155551234@vtext.com`.

Вы сможете узнать почтовый SMS-шлюз оператора мобильной связи, выполнив поиск в Интернете по фразе "почтовый шлюз SMS *<имя_оператора>*". В табл. 18.4 перечислены шлюзы нескольких популярных мобильных операторов. У многих операторов есть отдельные почтовые серверы для SMS с лимитом в 160 символов и MMS (служба обмена мультимедийными сообщениями), где нет ограничений по количеству символов. Например, чтобы отправить фотографию, необходимо использовать шлюз MMS и прикрепить файл к электронному письму.

Таблица 18.4. Почтовые SMS-шлюзы операторов мобильной связи

Мобильный оператор	Шлюз SMS	Шлюз MMS
AT&T	`number@txt.att.net`	`number@mms.att.net`
Boost Mobile	`number@sms.myboostmobile.com`	То же, что и для SMS
Cricket	`number@sms.cricketwireless.net`	`number@mms.cricketwireless.net`

Окончание табл. 18.4

Мобильный оператор	Шлюз SMS	Шлюз MMS
Google Fi	number@msg.fi.google.com	То же, что и для SMS
Metro PCS	number@mymetropcs.com	То же, что и для SMS
Republic Wireless	number@text.republicwireless.com	То же, что и для SMS
Sprint	number@messaging.sprintpcs.com	number@pm.sprint.com
T-Mobile	number@tmomail.net	То же, что и для SMS
U.S. Cellular	number@email.uscc.net	number@mms.uscc.net
Verizon	number@vtext.com	number@vzwpix.com
Virgin Mobile	number@vmobl.com	number@vmpix.com
XFinity Mobile	number@vtext.com	number@mypixmessages.com

Если вы не знаете, какой у получателя оператор мобильной связи, воспользуйтесь одним из сайтов поиска мобильных операторов по номеру телефона. Лучший способ найти такие сайты – ввести поисковую фразу "найти оператора мобильной связи по номеру". Многие сайты позволяют искать номера бесплатно (хотя и будут взимать плату за просмотр сотен или тысяч телефонных номеров через их программный интерфейс).

Несмотря на то что почтовые SMS-шлюзы бесплатны и просты в использовании, у них есть ряд недостатков.

- Нет гарантии, что текст будет доставлен быстро или вообще доставлен.
- Нет возможности узнать, доставлен ли текст.
- Получатель не имеет возможности ответить.
- Почтовые SMS-шлюзы могут блокировать вас, если вы рассылаете слишком много электронных писем, и нет способа узнать, сколько это – "слишком много".
- То, что SMS-шлюз доставляет текстовые сообщения сегодня, не означает, что оно будет доставлено завтра.

Отправка текстовых сообщений через SMS-шлюз – идеальное решение, если вам нужно время от времени рассылать несрочные сообщения. Если же требуется более надежная услуга, воспользуйтесь непочтовым SMS-шлюзом, как описано далее.

Отправка текстовых сообщений с помощью Twilio

В этом разделе вы узнаете о том, как подписаться на бесплатную службу Twilio и использовать ее модуль Python для отправки текстовых сообщений.

Twilio представляет собой SMS-шлюз, с помощью которого можно рассылать текстовые сообщения из программ. Несмотря на то что бесплатная учетная запись имеет месячный лимит сообщений, а их текст будет предваряться фразой "Sent from a Twilio trial account", этого вполне достаточно для многих программ

Twilio – не единственная служба такого рода. Вы сможете найти альтернативные варианты, выполнив в Интернете поиск по ключевым словам "бесплатный sms шлюз", "python sms api" или даже "twilio альтернативы".

Прежде чем создавать учетную запись Twilio, установите модуль `twilio`, выполнив команду `pip install --user --upgrade twilio` (в macOS и Linux используйте утилиту `pip3`). Более подробно об установке сторонних модулей рассказывается в приложении А.

Примечание

Этот раздел специфичен для США. Twilio предлагает услуги по обмену SMS-сообщениями и для других стран, однако их специфика здесь не рассматривается. Тем не менее модуль `twilio` и его функции будут работать одинаково и за пределами США. Более подробная информация по этой теме доступна на сайте `https://twilio.com/`.

Создание учетной записи Twilio

Перейдите на сайт `https://twilio.com/` и заполните регистрационную форму. После создания учетной записи вам нужно будет подтвердить номер мобильного телефона, на который вы хотите отправлять текстовые сообщения. (Верификация этого номера необходима, чтобы исключить возможность использования службы для рассылки спама на случайные номера.)

Перейдите на страницу "Verified Caller IDs" и добавьте номер телефона, к которому у вас есть доступ. Twilio отправит на этот номер код, который нужно будет ввести для верификации номера. Теперь вы сможете отправлять текстовые сообщения на этот номер с помощью модуля `twilio`.

Twilio предоставит вам пробную учетную запись с номером телефона для использования в качестве отправителя текстовых сообщений. Вам потребуются еще два значения: идентификатор вашей учетной записи (SID) и токен аутентификации. Соответствующую информацию вы найдете на странице Dashboard, когда войдете в свою учетную запись Twilio. Эти значения будут играть роль имени пользователя и пароля при входе в Twilio из программ Python.

Отправка текстовых сообщений

Когда вы установите модуль `twilio`, заведете учетную запись Twilio, верифицируете свой номер телефона, зарегистрируете телефонный номер Twilio и получите SID вместе с токеном аутентификации для своей учетной записи, это будет означать, что вы готовы к отправке текстовых сообщений из сценариев Python.

По сравнению с нетривиальной процедурой регистрации сам код Python для работы с Twilio достаточно прост. Введите в интерактивной оболочке приведенные ниже инструкции, подставив для переменных `accountSID`, `authToken`, `myTwilioNumber` и `myCellPhone` реальные значения своего идентификатора безопасности, токена аутентификации, телефонного номера Twilio и собственного номера телефона.

```
❶ >>> from twilio.rest import Client
  >>> accountSID = 'ACxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
  >>> authToken = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
❷ >>> twilioCli = Client(accountSID, authToken)
  >>> myTwilioNumber = '+14955551234'
  >>> myCellPhone = '+14955558888'
❸ >>> message = twilioCli.messages.create(
          body='Mr. Watson - Come here - I want to see you.',
          from_=myTwilioNumber, to=myCellPhone)
```

Спустя какое-то время после ввода последней строки вы должны получить текстовое сообщение следующего содержания: "Sent from your Twilio trial account - Mr. Watson - Come here - I want to see you".

В силу особенностей установки модуля `twilio` его нужно импортировать с помощью команды `from twilio.rest import Client`, а не `import twilio` ❶. Сохраните SID своей учетной записи в переменной `accountSID`, а свой токен аутентификации — в переменной `authToken`, после чего вызовите функцию `Client()`, передав ей переменные `accountSID` и `authToken` в качестве аргументов. Функция `Client()` возвращает объект `Client` ❷, у которого есть атрибут `messages`, в свою очередь имеющий метод `create()`. Именно этот метод инструктирует серверы Twilio о том, что необходимо отправить текстовое сообщение. Сохранив свой номер Twilio и номер мобильного телефона в переменных `myTwilioNumber` и `myCellPhone` соответственно, вызовите метод `create()` и передайте ему именованные аргументы, задающие тело сообщения, номер отправителя (`myTwilioNumber`) и номер получателя (`myCellPhone`) ❸.

Объект `Message`, возвращаемый методом `create()`, будет содержать информацию об отправленном сообщении. Введите в интерактивной оболочке следующие инструкции.

```
>>> message.to
'+14955558888'
>>> message.from_
'+14955551234'
>>> message.body
'Mr. Watson - Come here - I want to see you.'
```

В атрибутах `to`, `from_` и `body` будут храниться соответственно номер вашего мобильного телефона, телефонный номер Twilio и само сообщение. Обратите внимание на символ подчеркивания в названии атрибута `from_`. Это связано с тем, что `from` — ключевое слово в Python (в частности, оно используется в инструкциях импорта `from` *имя_модуля* `import *`) и не может служить именем атрибута.

Теперь введите в интерактивной оболочке следующие инструкции.

```
>>> message.status
'queued'
>>> message.date_created
datetime.datetime(2019, 7, 8, 1, 36, 18)
>>> message.date_sent == None
True
```

Атрибут `status` содержит строку состояния сообщения. Атрибуты `date_created` и `date_sent` будут содержать объект `datetime`, если сообщение было создано и отправлено. Может показаться странным, что атрибут `status` равен `'queued'` (помещено в очередь), а атрибут `date_sent` равен `None`, ведь вы уже получили сообщение. Это объясняется тем, что объект `Message` был сохранен в переменной `message` еще *до* фактической отправки текстового сообщения. Чтобы увидеть текущие значения атрибутов `status` и `date_sent`, следует заново извлечь объект `Message`. Каждому сообщению Twilio присваивается уникальный строковый идентификатор (SID), который можно использовать для извлечения последнего состояния объекта `Message`. Введите в интерактивной оболочке следующие инструкции.

```
>>> message.sid
'SM09520de7639ba3af137c6fcb7c5f4b51'
❶ >>> updatedMessage = twilioCli.messages.get(message.sid)
>>> updatedMessage.status
'delivered'
>>> updatedMessage.date_sent
datetime.datetime(2019, 7, 8, 1, 36, 18)
```

Атрибут `message.sid` содержит длинную строку SID данного сообщения. Передав это значение методу `get()` клиента Twilio ❶, вы получите новый объект `Message`, включающий обновленную информацию о сообщении.

Атрибуты `status` и `date_sent` этого нового объекта `Message` теперь содержат корректные значения.

Атрибут `status` может иметь одно из следующих значений: `'queued'` (помещено в очередь), `'sending'` (отправляется), `'sent'` (отправлено), `'delivered'` (доставлено), `'undelivered'` (не доставлено) или `'failed'` (не отправлено). Названия этих состояний говорят сами за себя, но если вам нужны более подробные сведения, то обратитесь по следующему адресу:

```
https://support.twilio.com/hc/en-us/articles/223134347-What-are-
the-Possible-SMS-and-MMS-Message-Statuses-and-What-do-They-Mean-
```

Получение текстовых сообщений с помощью Python

К сожалению, процедура получения текстовых сообщений с помощью Python немного сложнее процедуры отправки. Twilio требует, чтобы у вас был веб-сайт, на котором выполняется собственное веб-приложение. Рассмотрение этой темы выходит за рамки книги; за дополнительной информацией обратитесь по следующему адресу:

```
https://www.twilio.com/docs/sms/tutorials/
how-to-receive-and-reply-python
```

Проект: модуль "Черкни мне"

Чаще всего вы будете отправлять текстовые сообщения самому себе. Это отличный способ посылать себе напоминания, которые можно прочитать, находясь вдали от компьютера. Если вы автоматизируете рутинную задачу, выполняющуюся несколько часов, то можете заставить программу извещать вас о ее завершении с помощью SMS. Другой вариант — программа, регулярно проверяющая прогноз погоды и сообщающая вам о необходимости взять зонтик по случаю дождя.

В качестве простого примера рассмотрим небольшую программу, содержащую функцию `textmyself()`, которая отправляет текстовое сообщение, переданное ей в качестве строкового аргумента. Откройте в файловом редакторе новое окно и введите приведенный ниже код, подставив в него собственные данные. Сохраните программу в файле *textMyself.py*.

```
#! python3
# textMyself.py - содержит функцию textmyself(), которая
# отправляет текстовое сообщение, переданное ей в виде строки

# Предустановленные значения
accountSID = 'ACxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
authToken = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
```

```
myNumber = '+15559998888'
twilioNumber = '+15552225678'

from twilio.rest import Client

❶ def textmyself(message):
❷     twilioCli = Client(accountSID, authToken)
❸     twilioCli.messages.create(body=message, from_=twilioNumber,
                                to=myNumber)
```

Прежде всего в программе сохраняются значения SID и токена аутентификации, а также телефонные номера отправителя и получателя. Затем создается функция `textmyself()` с одним аргументом ❶, которая создает объект `Client` ❷ и вызывает метод `create()`, используя переданное ей сообщение ❸.

Если вы захотите сделать функцию `textmyself()` доступной другим своим программам, то поместите файл *textMyself.py* в ту же папку, в которой находится сценарий Python. Чтобы программа смогла отправить вам текстовое сообщение, включите в нее следующий код.

```
import textmyself
textmyself.textmyself('Длинное задание выполнено.')
```

Таким образом, зарегистрироваться на сайте Twilio и написать код, выполняющий отправку текстовых сообщений, нужно всего один раз. После этого достаточно будет добавить в программу всего пару инструкций для отправки текстового сообщения.

Резюме

Мы общаемся друг с другом по Интернету и посредством мобильной связи самыми разными способами, но преимущественно это электронная почта и текстовые сообщения. Данные каналы связи доступны и в программах Python, что открывает широкие возможности по рассылке оповещений. Можно даже написать программы, которые будут выполняться на разных компьютерах, обмениваясь сообщениями посредством электронной почты, когда одна программа отправляет письма по протоколу SMTP, а другая получает их по протоколу IMAP.

Встроенный модуль `smtplib` содержит функции, с помощью которых вы сможете отправлять электронные письма через SMTP-сервер своего провайдера. Аналогичным образом сторонние модули `imapclient` и `pyzmail` позволяют подключаться к серверам IMAP и получать отправленные вам сообщения. Протокол IMAP немного сложнее протокола SMTP, зато он

позволяет искать конкретные сообщения электронной почты, загружать их и извлекать тему и тело письма в виде строковых значений.

В целях безопасности и защиты от спама некоторые популярные почтовые службы, такие как Gmail, не позволяют использовать стандартные протоколы SMTP и IMAP для доступа к их серверам. Модуль EZGmail служит удобной оболочкой для Gmail API, позволяя сценариям Python получать доступ к вашей учетной записи Gmail. Настоятельно рекомендуется настроить отдельную учетную запись для сценариев, чтобы потенциальные ошибки в программе не создавали проблем для вашей личной учетной записи.

Обмен текстовыми сообщениями отличается от механизмов работы электронной почты, поскольку для отправки SMS требуется не только подключение к Интернету. К счастью, имеются службы наподобие Twilio, которые предоставляют модули, позволяющие отправлять текстовые сообщения из программ Python. Как только вы пройдете этап начальной настройки, вы сможете отправлять текстовые сообщения с помощью буквально нескольких строк кода.

Контрольные вопросы

1. Какой протокол используется для отправки электронной почты? Какой протокол используется для проверки и получения электронной почты?
2. Какие функции/методы модуля `smtplib` необходимо вызвать для того, чтобы войти в учетную запись на SMTP-сервере?
3. Какие функции/методы модуля `imapclient` необходимо вызвать для того, чтобы войти в учетную запись на IMAP-сервере?
4. Какой аргумент нужно передать методу `imapObj.search()`?
5. Какие меры следует предпринять в том случае, если программа получает сообщение об ошибке `'got more than 10000 bytes'`?
6. Модуль `imapclient` отвечает за подключение к серверу IMAP и поиск сообщений электронной почты. Какой модуль отвечает за чтение сообщений электронной почты, извлеченных модулем `imapclient`?
7. Что собой представляют файлы *credentials.json* и *token.json* в Gmail API?
8. В чем разница между цепочкой и сообщением в Gmail API?
9. Как с помощью метода `ezgmail.search()` найти письма с файловыми вложениями?
10. Какие три значения нужно получить от Twilio, чтобы иметь возможность отправлять текстовые сообщения?

Учебные проекты

Чтобы закрепить полученные знания на практике, напишите программы для предложенных ниже задач.

Произвольное распределение заданий путем рассылки по электронной почте

Напишите программу, которая получает список адресов электронной почты и список рутинных задач, подлежащих выполнению, после чего случайным образом распределяет задачи между исполнителями. Каждый исполнитель должен получить электронное письмо со списком назначенных ему задач. Как вариант, можно отслеживать ранее назначенные задачи по каждому исполнителю, чтобы пользователи не получали одни и те же задачи дважды подряд. При желании можете организовать автоматический запуск программы раз в неделю.

Подсказка: если передать функции `random.choice()` список, она вернет элемент, выбираемый случайным образом. Соответствующий фрагмент программы может выглядеть примерно так.

```
chores = ['помыть посуду', 'помыть ванную', 'пропылесосить',
          'выгулять собаку']
randomChore = random.choice(chores)
chores.remove(randomChore)     # Это задание уже распределено,
                               # и его можно удалить из списка
```

Напоминание о зонтике

В главе 12 было продемонстрировано, как организовать сбор данных с сайта `https://weather.gov/` с помощью модуля `requests`. Напишите программу, которая запускается непосредственно перед вашим утренним пробуждением и проверяет, не ожидается ли сегодня дождь. В случае такого прогноза программа должна отправить вам текстовое сообщение с напоминанием о том, что необходимо взять зонт.

Автоматический отказ от подписки

Напишите программу, которая просматривает почтовый ящик вашей учетной записи электронной почты, находит все ссылки `Unsubscribe` в сообщениях и автоматически открывает их в браузере. Программа должна входить в вашу учетную запись на IMAP-сервере провайдера и загружать все сообщения. Для обнаружения гиперссылок, содержащих слово `'Unsubscribe'`, можно применить модуль BeautifulSoup (см. главу 12).

Получив список соответствующих URL-адресов, используйте функцию `webbrowser.open()` для открытия в браузере всех ссылок, позволяющих отказаться от подписки.

Все остальные действия, связанные с отказом от подписки, придется проделать вручную. В большинстве случаев понадобится щелкнуть на ссылке для подтверждения. Тем не менее эта программа избавляет вас от необходимости просматривать всю почту в поиске ссылок `'Unsubscribe'`. Кроме того, программу можно передать друзьям, чтобы они могли выполнять ее применительно к собственным учетным записям электронной почты. (Обязательно проследите, чтобы в коде не был жестко задан ваш собственный пароль для доступа к электронной почте!)

Дистанционное управление компьютером по электронной почте

Напишите программу, которая проверяет электронную почту каждые 15 минут в ожидании поступивших от вас инструкций и автоматически их выполняет. В качестве примера рассмотрим файлообменную систему BitTorrent. Существуют бесплатные утилиты, такие как qBittorrent, с помощью которых можно загружать на компьютер большие мультимедийные файлы. Если отправить программе ссылку BitTorrent, то при очередной проверке электронной почты программа обнаружит это сообщение, извлечет ссылку и запустит клиент qBittorrent для загрузки соответствующего файла. Таким образом, программа сможет загружать файлы в ваше отсутствие, и к тому времени, когда вы вернетесь домой, файлы уже будут храниться на диске.

В главе 17 рассказывалось о том, как запускать программы с помощью функции `subprocess.Popen()`. Например, следующая инструкция запустит утилиту qBittorrent для загрузки указанного торрент-файла:

```
qbProcess = subprocess.Popen(['C:\\Program Files (x86)\\qBittorrent\\
qbittorrent.exe', 'shakespeare_complete_works.torrent'])
```

Естественно, программа должна проверять, поступило ли сообщение именно от вас. Для этого, в частности, можно потребовать, чтобы в сообщении содержался пароль, поскольку хакерам не составляет труда подделать адрес в поле "От" письма. Программа должна удалять все прочитанные сообщения, чтобы не выполнять инструкции повторно при каждой проверке электронной почты. Дополнительно можете предусмотреть, чтобы программа отправляла вам электронное письмо или текстовое сообщение, подтверждающие начало загрузки указанного файла. Поскольку во время выполнения программы вы не будете сидеть за компьютером, желательно организовать ведение журнала (см. главу 11), чтобы впоследствии вы могли проверить, не возникали ли ошибки в процессе загрузки файлов.

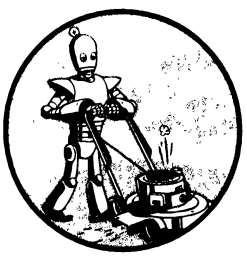
В утилите qBittorrent (как и в других приложениях BitTorrent) предусмотрена возможность автоматического завершения программы по окончании загрузки. В главе 17 объяснялось, как проконтролировать завершение работы запущенной программы с помощью метода `wait()` объекта `Popen`. Вызов метода `wait()` заблокирует дальнейшее выполнение до тех пор, пока утилита qBittorrent не завершится, после чего программа сможет отправить вам электронное письмо или текстовое сообщение, уведомляющее об окончании процесса загрузки.

В подобный проект можно добавить много других интересных функций. Пример реализации такой программы доступен по следующему адресу:

```
https://github.com/asweigart/inventwithpythondotcom/blob/
master/static/torrentStarter.py
```

19

РАБОТА С ИЗОБРАЖЕНИЯМИ



Если у вас есть цифровой фотоаппарат или же вы любите публиковать на Facebook снимки со смартфона, то вам регулярно постоянно приходится иметь дело с файлами цифровых изображений. Возможно, вы умеете пользоваться простыми графическими редакторами наподобие Microsoft Paint или даже такой сложной программой, как Adobe Photoshop. Но когда дело доходит до редактирования большого количества изображений, это быстро превращается в утомительную рутину.

К счастью, на выручку, как всегда, приходит Python. Pillow — это сторонний модуль, предназначенный для обработки файлов изображений. В нем есть ряд функций, упрощающих выполнение таких операций, как обрезка и масштабирование изображений. Располагая средствами редактирования изображений на уровне таких программ, как Microsoft Paint, Python позволяет легко автоматизировать обработку сотен и даже тысяч изображений. Чтобы установить модуль Pillow, выполните команду `pip install --user -U pillow==6.0.0`. (Дополнительные сведения об установке сторонних модулей приведены в приложении А.)

Основы компьютерной обработки изображений

Чтобы заниматься обработкой изображений, нужно хотя бы в общих чертах понимать основы работы с цветом и уметь задавать координаты пикселей в модуле Pillow.

Цвета и значения RGBA

В программировании для представления цвета обычно используют *модель RGBA*. Это группа чисел, задающих долю красной (red), зеленой (green), синей (blue) и альфа-составляющей (прозрачности) в цвете. Каждый из компонентов цвета представляет собой целое число в интервале от `0` (отсутствие данного цвета) до `255` (максимум). Значения RGBA присваиваются отдельным пикселям. *Пиксель* — это наименьший элемент изображения, который может быть представлен на экране компьютера (количество пикселей на экране исчисляется миллионами). RGB-значение пикселя задает его цветовой оттенок. Использование альфа-составляющей приводит к созданию RGBA-значений. Если изображение выводится на экран поверх фонового слоя или обоев рабочего стола, то параметр "альфа" (прозрачность) определяет, насколько интенсивно должен просматриваться фон через пиксели изображения.

В модуле Pillow значения RGBA представляются кортежем из четырех целочисленных значений. Например, красный цвет описывается кортежем `(255, 0, 0, 255)`. В этом цвете содержится максимальное количество красной составляющей, зеленая и синяя составляющие отсутствуют, а параметр "альфа" имеет максимальное значение, которому соответствует полная непрозрачность. Зеленый цвет описывается кортежем `(0, 255, 0, 255)`, а синий — кортежем `(0, 0, 255, 255)`. Белый цвет, представляющий собой сочетание всех цветов, описывается кортежем `(255, 255, 255, 255)`, тогда как черному цвету, в котором нет цветовых составляющих, соответствует кортеж `(0, 0, 0, 255)`.

Если значение альфа-составляющей в цвете равно `0`, то этот пиксель невидимый, и от конкретных значений параметров RGB уже ничего не

зависит. В конце концов, невидимый красный — это все равно что невидимый черный.

В модуле Pillow используются стандартные названия цветов, принятые в HTML. В табл. 19.1 приведены некоторые названия цветов и соответствующие им значения RGBA.

Таблица 19.1. Стандартные названия цветов и их RGBA-значения

Название	Значение RGBA	Название	Значение RGBA
White (Белый)	(255, 255, 255, 255)	Red (Красный)	(255, 0, 0, 255)
Green (Зеленый)	(0, 255, 0, 255)	Blue (Синий)	(0, 0, 255, 255)
Gray (Серый)	(128, 128, 128, 255)	Yellow (Желтый)	(255, 255, 0, 255)
Black (Черный)	(0, 0, 0, 255)	Purple (Пурпурный)	(128, 0, 128, 255)

Модуль Pillow содержит функцию `ImageColor.getcolor()`, которая избавляет от необходимости запоминать RGBA-значения цветов. Эта функция получает название цвета в качестве первого аргумента и строку `'RGBA'` в качестве второго аргумента, возвращая кортеж значений RGBA.

Введите в интерактивной оболочке следующие инструкции.

```
❶ >>> from PIL import ImageColor
❷ >>> ImageColor.getcolor('red', 'RGBA')
  (255, 0, 0, 255)
❸ >>> ImageColor.getcolor('RED', 'RGBA')
  (255, 0, 0, 255)
  >>> ImageColor.getcolor('Black', 'RGBA')
  (0, 0, 0, 255)
  >>> ImageColor.getcolor('chocolate', 'RGBA')
  (210, 105, 30, 255)
  >>> ImageColor.getcolor('CornflowerBlue', 'RGBA')
  (100, 149, 237, 255)
```

В первую очередь необходимо импортировать модуль `ImageColor` из библиотеки `PIL` ❶ (обратите внимание — не Pillow; вскоре вы узнаете, почему). Строка с названием цвета, которая передается функции `ImageColor.getcolor()`, нечувствительна к регистру символов, поэтому как для аргумента `'red'` ❷, так и для аргумента `'RED'` ❸ мы получаем один и тот же кортеж RGBA. Поддерживаются и такие необычные названия цветов, как `'chocolate'` (шоколадный) или `'CornflowerBlue'` (васильковый).

Модуль Pillow распознает огромное количество названий цветов, от `'aliceblue'` до `'whitesmoke'`. Полный список, включающий более 100 стандартных названий цветов, приведен в Википедии:

`https://ru.wikipedia.org/wiki/Цвета_HTML`

Кортежи координат и прямоугольников

Для адресации пикселей изображений используют координаты *x* и *y*, задающие расположение пикселя в изображении соответственно в горизонтальном и вертикальном направлениях. *Началом отсчета* служит пиксель, располагающийся в левом верхнем углу изображения; его координаты — `(0, 0)`. Первый нуль представляет координату *x*, значения которой начинаются с нуля и увеличиваются в направлении слева направо. Второй нуль представляет координату *y*, значения которой начинаются с нуля и увеличиваются в направлении сверху вниз. Еще раз: координаты *y* увеличиваются вниз — это противоположно тому, что принято в математике (рис. 19.1).

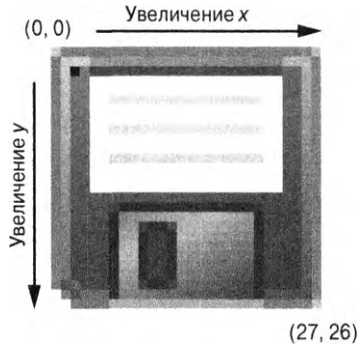


Рис. 19.1. Координаты x и y для области изображения размером 28×27 пикселей

Многим функциям и методам модуля Pillow в качестве аргумента передается *кортеж прямоугольника*. Это означает, что они ожидают кортеж из четырех целочисленных координат, задающих прямоугольную область изображения. Перечислим их в порядке следования.

- **Левая сторона:** координата *x* левой стороны прямоугольника.
- **Верхняя сторона:** координата *y* верхней стороны прямоугольника.
- **Правая сторона:** координата *x* внешнего пикселя, примыкающего к правой стороне прямоугольника. Это число должно быть больше того, которое определяет положение левой стороны.
- **Нижняя сторона:** координата *y* внешнего пикселя, примыкающего к нижней стороне прямоугольника. Это число должно быть больше того, которое определяет положение верхней стороны.

Обратите внимание на то, что прямоугольник включает точки, соответствующие координатам *левой* и *верхней* сторон, и в то же время не включает точки, соответствующие координатам *правой* и *нижней* сторон. Например,

черная прямоугольная область, изображенная на рис. 19.2, описывается кортежем `(3, 1, 9, 6)`.

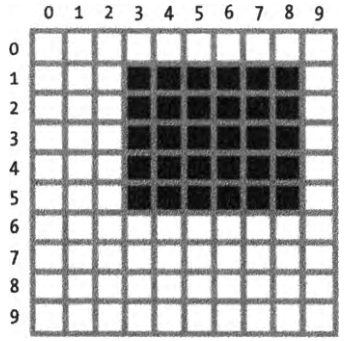


Рис. 19.2. Прямоугольная область, описываемая кортежем `(3, 1, 9, 6)`

Обработка изображений с помощью модуля Pillow

Теперь, когда вам уже известно, как обрабатываются цвета и координаты в модуле Pillow, мы используем его для обработки изображений. На рис. 19.3 показано изображение, с которым мы будем работать в этой главе. Файл *zophie.png* содержится в архиве примеров книги (см. введение).

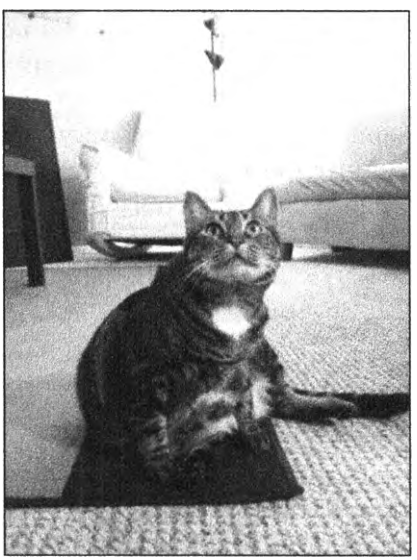


Рис. 19.3. Фотография кошки Зофи

Поместив файл *zophie.png* в текущий каталог, вы сможете загружать фотографию Зофи в Python следующим образом.

```
>>> from PIL import Image
>>> catIm = Image.open('zophie.tif')
```

Чтобы загрузить изображение, импортируйте модуль `Image` из пакета Pillow и вызовите функцию `Image.open()`, передав ей имя файла изображения. Полученное изображение сохраняется в переменной `CatIm`. В качестве имени модуля Pillow используется `PIL`, чтобы обеспечить обратную совместимость со старым модулем, который назывался Python Imaging Library. Именно по этой причине необходимо выполнять инструкцию `from PIL import Image`, а не `from Pillow import Image`. Кроме того, модуль сконфигурирован так, что инструкция импорта записывается в указанном формате, а не как `import PIL`.

Если файл изображения находится не в текущем каталоге, сделайте рабочим каталогом папку, в которой содержится файл изображения, вызвав функцию `os.chdir()`.

```
>>> import os
>>> os.chdir('C:\\папка_с_файлом_изображения')
```

Функция `Image.open()` возвращает объект `Image`. Изображения можно загружать из файлов практически любого формата. Любые изменения, внесенные в объект `Image`, можно сохранить в файле (тоже практически любого формата) с помощью метода `save()`. Любые операции поворота, масштабирования, обрезки, рисования и т.п. выполняются путем вызова соответствующих методов объекта `Image`.

Чтобы сделать примеры главы немного короче, мы будем предполагать, что модуль `Image` уже импортирован, а изображение Зофи сохранено в переменной `catIm`. Проследите за тем, чтобы файл *zophie.png* находился в текущем каталоге, где его сможет найти функция `Image.open()`. В противном случае нужно будет указать полный путь к файлу в строковом аргументе, передаваемом функции `Image.open()`.

Работа с объектами Image

У объекта `Image` есть несколько полезных атрибутов, предоставляющих основную информацию о файле изображения, из которого он был загружен. В частности, можно узнать ширину и высоту изображения, имя файла и графический формат (например, JPEG, GIF или PNG).

Введите в интерактивной оболочке следующие инструкции.

```
>>> from PIL import Image
>>> catIm = Image.open('zophie.png')
>>> catIm.size
```

```
❶ (816, 1088)
❷ >>> width, height = catIm.size
❸ >>> width
  816
❹ >>> height
  1088
  >>> catIm.filename
  'zophie.png'
  >>> catIm.format
  'PNG'
  >>> catIm.format_description
  'Portable network graphics'
❺ >>> catIm.save('zophie.jpg')
```

Создав объект `Image` из файла *Zophie.png* и сохранив его в переменной `catIm`, мы видим, что атрибут `size` объекта содержит кортеж значений ширины и высоты изображения, выраженных в пикселях ❶. Эти значения можно сохранить в переменных `width` и `height` ❷, что позволит работать с ними независимо ❸ ❹. Атрибут `filename` содержит имя исходного файла. Атрибуты `format` и `format_description` — это строки, содержащие описание формата изображения (атрибут `format_description` содержит расшифровку формата).

Наконец, вызвав метод `save()` и передав ему строку `'zophie.jpg'` в качестве аргумента, мы сохраняем изображение в новом файле *zophie.jpg* ❺. Модуль Pillow видит, что расширение нового файла — *.jpg*, и автоматически сохраняет изображение в формате JPEG. Теперь у вас есть два изображения: *zophie.png* и *zophie.jpg*. Несмотря на то что оба этих файла содержат одно и то же изображение, они не идентичны, поскольку имеют разный формат.

Кроме того, в модуле Pillow есть функция `Image.new()`, которая тоже возвращает объект `Image`, как и функция `Image.open()`, только в данном случае изображение будет пустым. Ниже описаны аргументы функции `Image.new()`.

- Строка `'RGBA'`, задающая цветовую модель RGBA. (Также поддерживаются другие цветовые модели, которые в книге не рассматриваются.)
- Кортеж из двух значений, представляющих ширину и высоту нового изображения.
- Цвет фона, определяющий начальный вид изображения. Задается кортежем из четырех целых чисел, образующих значение RGBA. В качестве этого аргумента можно использовать значение, возвращаемое функцией `ImageColor.getcolor()`. Другой вариант — передать функции `Image.new()` строку со стандартным названием цвета.

Введите в интерактивной оболочке следующие инструкции.

```
  >>> from PIL import Image
❶ >>> im = Image.new('RGBA', (100, 200), 'purple')
  >>> im.save('purpleImage.png')
❷ >>> im2 = Image.new('RGBA', (20, 20))
  >>> im2.save('transparentImage.png')
```

В этом примере мы создаем объект `Image` для изображения, ширина и высота которого составляют соответственно 100 и 200 пикселей и которое имеет пурпурный цвет фона ❶. Это изображение сохраняется в файле *purpleImage.png*. Затем мы снова вызываем функцию `Image.new()` для создания другого объекта `Image`, на этот раз с размерами `(20, 20)`, но без указания цвета фона ❷. В тех случаях, когда цвет не задан, по умолчанию используется невидимый черный цвет `(0, 0, 0, 0)`, поэтому второе изображение имеет прозрачный фон. Оно сохраняется в файле *transparentImage.png*.

Обрезка изображений

Под *обрезкой* изображения подразумевается выбор прямоугольной области внутри изображения и удаление всего, что находится вокруг этого прямоугольника. Метод `crop()` объекта `Image` получает кортеж прямоугольника и возвращает объект `Image`, представляющий обрезанное изображение. Обрезка выполняется "неразрушающим" образом, т.е. исходный объект `Image` остается нетронутым, а вместо этого возвращается новый объект `Image`. Учитывайте, что кортеж прямоугольника (в данном случае обрезанная область) включает левый столбец и верхнюю строку пикселей, но не включает правый столбец и нижнюю строку пикселей.

Введите в интерактивной оболочке следующие инструкции.

```
>>> from PIL import Image
>>> catIm = Image.open('zophie.png')
>>> croppedIm = catIm.crop((335, 345, 565, 560))
>>> croppedIm.save('cropped.png')
```

Здесь мы создаем новый объект `Image` для обрезанного изображения, сохраняя его в переменной `croppedIm`, после чего вызываем метод `save()`, чтобы сохранить обрезанное изображение в файле *cropped.png*. В результате на основе исходного изображения создается новый файл (рис. 19.4).

Копирование и вставка изображений в другие изображения

Метод `copy()` возвращает новый объект `Image` с тем же изображением, что и объект `Image`, для которого он был вызван. Это может пригодиться в тех случаях, когда необходимо получить измененную версию изображения,

сохранив нетронутым оригинал. Введите в интерактивной оболочке следующие инструкции.

```
>>> from PIL import Image
>>> catIm = Image.open('zophie.png')
>>> catCopyIm = catIm.copy()
```

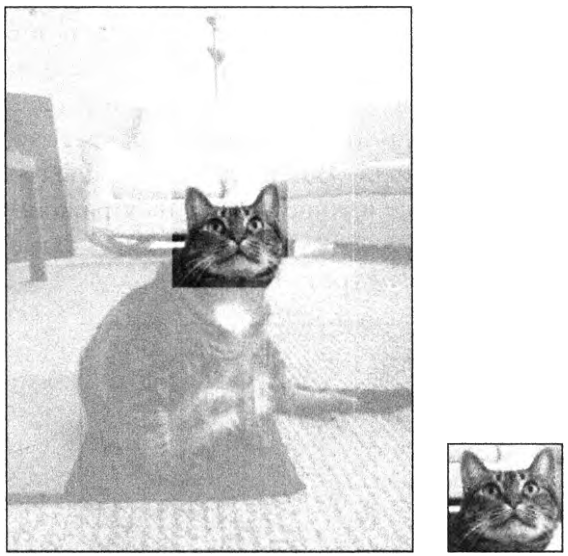


Рис. 19.4. Новое изображение будет содержать лишь фрагмент исходной фотографии

В переменных `catIm` и `catCopyIm` содержатся два независимых объекта `Image`, в каждом из которых хранится одно и то же изображение. Теперь вы сможете изменять копию по своему усмотрению и сохранять ее в другом файле, оставив файл *zophie.png* нетронутым. В качестве примера мы изменим изображение, хранящееся в переменной `catCopyIm`, с помощью метода `paste()`.

Будучи вызванным для объекта `Image`, метод `paste()` помещает поверх него другое изображение. Продолжим выполнение примера в интерактивной оболочке, вставив поверх изображения, хранящегося в переменной `catCopyIm`, обрезанное изображение.

```
>>> faceIm = catIm.crop((335, 345, 565, 560))
>>> faceIm.size
(230, 215)
>>> catCopyIm.paste(faceIm, (0, 0))
>>> catCopyIm.paste(faceIm, (400, 500))
>>> catCopyIm.save('pasted.png')
```

Сначала мы передаем методу `crop()` кортеж прямоугольника, задающий область изображения *zophie.png*, которая соответствует голове Зофи. В результате создается объект `Image`, представляющий обрезанное изображение с размерами 230×215, которое сохраняется в переменной `faceIm`. Теперь мы можем поместить это изображение поверх изображения `catCopyIm`. Метод `paste()` имеет два аргумента: объект `Image` вставляемого изображения и кортеж, определяющий координаты *x* и *y* точки на основном изображении, в которую должен быть помещен левый верхний угол копируемого объекта. В данном примере метод `paste()` вызывается для переменной `catCopyIm` дважды: первый раз для точки `(0, 0)` и второй раз — для точки `(400, 500)`. В результате изображение `faceIm` вставляется поверх изображения `catCopyIm` дважды. В первом случае левый верхний угол изображения `faceIm` помещается в точку `(0, 0)` изображения `catCopyIm`, а во втором случае — в точку `(400, 500)`. Наконец, мы сохраняем измененное изображение в файле *pasted.png* (рис. 19.5).

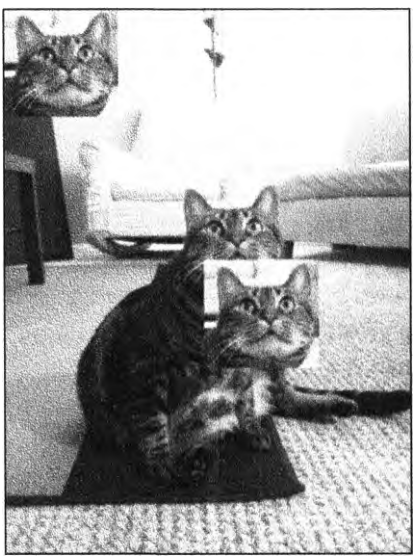


Рис. 19.5. Мордочка Зофи была скопирована дважды

Примечание

Пусть названия методов `copy()` и `paste()` модуля Pillow не вводят вас в заблуждение: их работа никоим образом не связана с буфером обмена.

Учтите, что метод `paste()` изменяет исходный объект `Image`, а не возвращает новый объект `Image` со вставленным изображением. Если требуется вызвать метод `paste()` и при этом сохранить нетронутым исходное

изображение, то сначала создайте его копию, а затем вызовите метод `paste()` для копии.

Предположим, мы хотим покрыть изображениями головы Зофи всю область исходного изображения (рис. 19.6). Для создания этого эффекта достаточно нескольких циклов. Введите в интерактивной оболочке следующие инструкции.

```
  >>> catImWidth, catImHeight = catIm.size
  >>> faceImWidth, faceImHeight = faceIm.size
❶ >>> catCopyTwo = catIm.copy()
❷ >>> for left in range(0, catImWidth, faceImWidth):
❸         for top in range(0, catImHeight, faceImHeight):
              print(left, top)
              catCopyTwo.paste(faceIm, (left, top))
  0 0
  0 215
  0 430
  0 645
  0 860
  0 1075
  230 0
  230 215
-- Опущено --
  690 860
  690 1075
  >>> catCopyTwo.save('tiled.png')
```

Сначала значения ширины и высоты изображения `catIm` сохраняются в переменных `catImWidth` и `catImHeight`. В строке ❶ мы создаем копию объекта `catIm` и сохраняем ее в переменной `catCopyTwo`. Теперь, когда у нас есть копия для экспериментов, мы организуем цикл для вставки изображения `faceIm` поверх изображения `catCopyTwo`. Значение переменной `left` в начале внешнего цикла `for` равно `0`; на каждой итерации цикла оно получает приращение, равное `faceImWidth` (`230`) ❷. Значение переменной `top` в начале внутреннего цикла равно `0`; на каждой итерации цикла оно получает приращение, равное `faceImHeight` (`215`) ❸. Генерируемые в этих вложенных циклах значения переменных `left` и `top` обеспечивают покрытие всего изображения `catCopyTwo` изображениями `faceIm`. Чтобы проследить за тем, как работают оба цикла, мы выводим значения переменных `left` и `top`. По завершении циклов мы сохраняем измененное изображение в файле *tiled.png*.

Изменение размеров изображения

Метод `resize()` объекта `Image` возвращает новый объект `Image` с заданными значениями ширины и высоты. Ему передается аргумент в виде

кортежа из двух целочисленных значений, представляющих новые значения ширины и высоты возвращаемого объекта. Введите в интерактивной оболочке следующие инструкции.

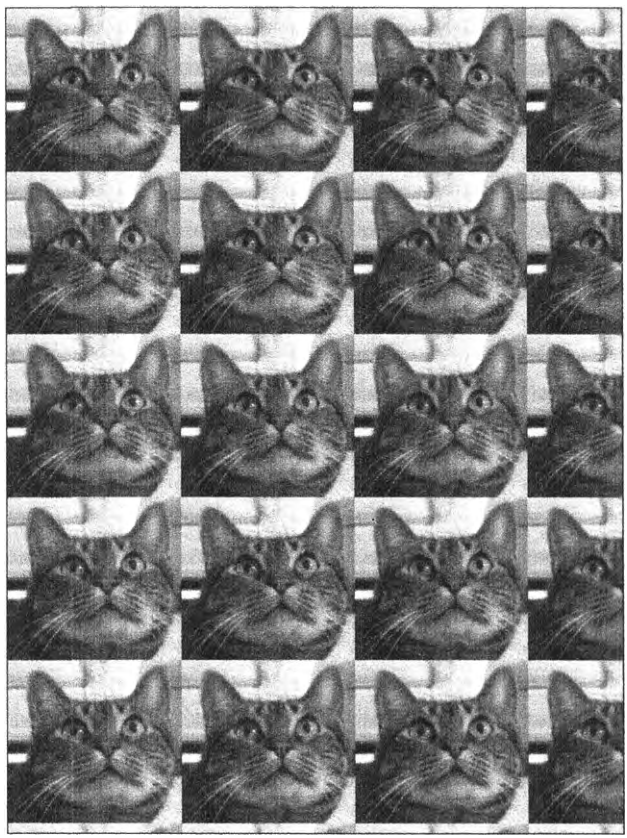


Рис. 19.6. Использование вложенных циклов и метода `paste()` для создания дубликатов изображения

```
  >>> from PIL import Image
  >>> catIm = Image.open('zophie.png')
❶ >>> width, height = catIm.size
❷ >>> quartersizedIm = catIm.resize((int(width / 2),
                                     int(height / 2)))
  >>> quartersizedIm.save('quartersized.png')
❸ >>> svelteIm = catIm.resize((width, height + 300))
  >>> svelteIm.save('svelte.png')
```

Здесь переменным `width` и `height` присваиваются значения, образующие кортеж `catIm.size` ❶. Использование отдельных переменных вместо выражений `catIm.size[0]` и `catIm.size[1]` делает код более компактным и понятным.

В первом вызове метода `resize()` ему передаются значения `int(width / 2)` и `int(height / 2)` в качестве новых значений ширины и высоты ❷, следовательно, объект `Image`, возвращаемый этим методом, будет иметь половинную ширину и высоту, т.е. в целом он будет в четыре раза меньше исходного изображения. В качестве аргумента допустим лишь кортеж целочисленных значений, поэтому обе операции деления на 2 должны быть обернуты в вызовы `int()`.

В данном случае ширина и высота изменяются в одинаковой пропорции. Впрочем, сохранять исходные пропорции изображения вовсе необязательно. Переменная `svelteIm` содержит объект `Image`, ширина которого совпадает с первоначальной, но высота увеличена на 300 пикселей ❸, что делает Зофи более стройной.

Заметьте, что метод `resize()` не изменяет исходный объект `Image`, а возвращает новое изображение.

Поворот и зеркальное отражение изображений

Изображения можно поворачивать с помощью метода `rotate()`, который возвращает новый объект `Image` повернутого изображения, оставляя исходный объект нетронутым. Аргументом метода `rotate()` будет целое или вещественное число, представляющее угол поворота в градусах против часовой стрелки. Введите в интерактивной оболочке следующие инструкции.

```
>>> from PIL import Image
>>> catIm = Image.open('zophie.png')
>>> catIm.rotate(90).save('rotated90.png')
>>> catIm.rotate(180).save('rotated180.png')
>>> catIm.rotate(270).save('rotated270.png')
```

Здесь мы применяем цепочки вызовов методов, вызывая метод `save()` непосредственно для объекта `Image`, возвращаемого методом `rotate()`. В первом случае создается новый объект `Image`, который представляет изображение, повернутое на 90° против часовой стрелки. Это изображение сохраняется в файле *rotated90.png*. В остальных случаях делается то же самое, только изображение поворачивается на 180° и 270° соответственно (рис. 19.7).

Заметьте, что при повороте на 90° или 270° ширина и высота изображения изменяются. При повороте на другие углы поддерживаются исходные размеры изображения. В Windows для заполнения возникающих пустот используется черная заливка, а в macOS — прозрачные пиксели.

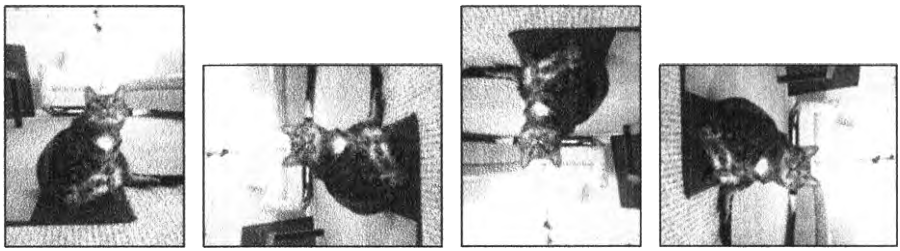


Рис. 19.7. Исходное изображение (слева) и изображения, повернутые на 90°, 180° и 270°

В методе `rotate()` предусмотрен необязательный именованный аргумент `expand`. Если он задан равным `True`, размеры изображения увеличиваются таким образом, чтобы оно вписалось в ограничивающий прямоугольник нового, повернутого изображения. Например, введите в интерактивной оболочке следующие инструкции.

```
>>> catIm.rotate(6).save('rotated6.png')
>>> catIm.rotate(6, expand=True).save('rotated6_expanded.png')
```

В первом случае изображение поворачивается на 6° и сохраняется в файле *rotate6.png* (рис. 19.8, слева). Во втором случае изображение тоже поворачивается на 6° и сохраняется в файле *rotate6_expanded.png*, но на этот раз аргумент `expand` равен `True` (рис. 19.8, справа).

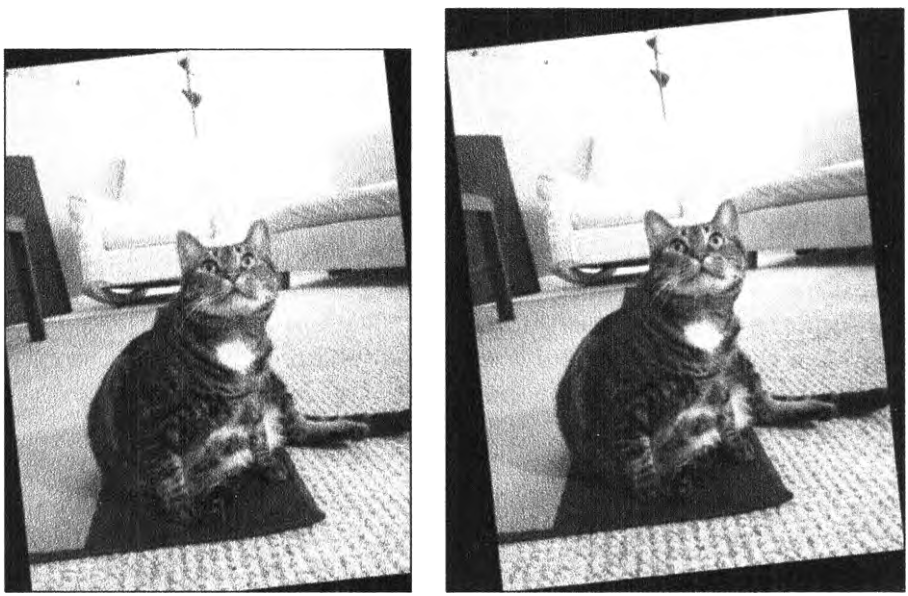


Рис. 19.8. Изображение, повернутое на 6° в обычном режиме (слева) и в режиме `expand=True` (справа)

Метод `transpose()` позволяет получить "зеркальное отображение" изображения. Ему должен передаваться аргумент `Image.FLIP_LEFT_RIGHT` либо `Image.FLIP_TOP_BOTTOM`. Введите в интерактивной оболочке следующие инструкции.

```
>>> catIm.transpose(Image.FLIP_LEFT_RIGHT).save('horizontal_flip.png')
>>> catIm.transpose(Image.FLIP_TOP_BOTTOM).save('vertical_flip.png')
```

Как и метод `rotate()`, метод `transpose()` создает новый объект `Image`. В первом случае методу передается аргумент `Image.FLIP_LEFT_RIGHT`, в результате чего изображение отражается по горизонтали, а затем сохраняется в файле *horizontal_flip.png*. Чтобы получить отражение по вертикали, мы передаем методу `transpose()` аргумент `Image.FLIP_TOP_BOTTOM` и сохраняем полученное изображение в файле *vertical_flip.png*. Результаты показаны на рис. 19.9.

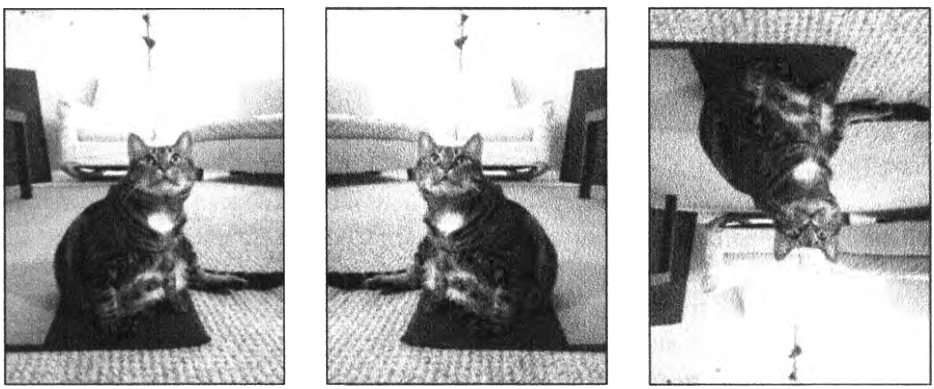


Рис. 19.9. Исходное изображение (слева), результат горизонтального отражения (в центре) и результат вертикального отражения (справа)

Изменение отдельных пикселей

Для получения или изменения цвета отдельного пикселя предназначены методы `getpixel()` и `putpixel()`. Обоим методам в качестве аргумента передается кортеж координат пикселя. Кроме того, у метода `putpixel()` есть дополнительный аргумент в виде кортежа, задающего цвет пикселя. Это может быть либо RGBA-кортеж из четырех целых чисел, либо RGB-кортеж, включающий три целых числа. Введите в интерактивной оболочке следующие инструкции.

```
  >>> from PIL import Image
❶ >>> im = Image.new('RGBA', (100, 100))
❷ >>> im.getpixel((0, 0))
  (0, 0, 0, 0)
```

```
❸ >>> for x in range(100):
          for y in range(50):
❹             im.putpixel((x, y), (210, 210, 210))

  >>> from PIL import ImageColor
❺ >>> for x in range(100):
          for y in range(50, 100):
❻             im.putpixel((x, y), ImageColor.getcolor('darkgray', 'RGBA'))
  >>> im.getpixel((0, 0))
  (210, 210, 210, 255)
  >>> im.getpixel((0, 50))
  (169, 169, 169, 255)
  >>> im.save('putPixel.png')
```

В строке ❶ мы создаем новое изображение в виде прозрачного квадрата с размерами 100×100. Вызов метода `getpixel()` для одной из точек этого изображения возвращает кортеж `(0, 0, 0, 0)`, поскольку изображение прозрачно ❷. Чтобы назначить цвета пикселям этого изображения, мы используем вложенные циклы `for`, перебирая все пиксели в верхней половине изображения ❸ и вызывая для каждого из них метод `putpixel()` ❹. В данном случае методу `putpixel()` передается RGB-кортеж `(210, 210, 210)`, которому соответствует светло-серый цвет.

Предположим, мы хотим закрасить нижнюю половину изображения темно-серым цветом, но не знаем, какой RGB-кортеж ему соответствует. Метод `putpixel()` не поддерживает стандартные названия цветов наподобие `'darkgray'`, поэтому мы генерируем соответствующий кортеж для цвета `'darkgray'` с помощью функции `ImageColor.getcolor()`. Организуя цикл по пикселям нижней половины изображения ❺ и передавая методу `putpixel()` значение, возвращаемое функцией `ImageColor.getcolor()` ❻, мы получаем изображение, верхняя половина которого закрашена светло-серым цветом, а нижняя — темно-серым (рис. 19.10). Для проверки того, что цвет любого заданного пикселя соответствует ожидаемому, можно вызвать метод `getpixel()`. Наконец, мы сохраняем изображение в файле *putPixel.png*.

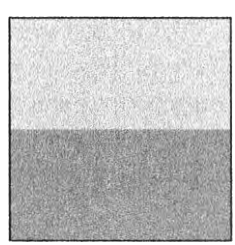


Рис. 19.10. Изображение putPixel.png

Разумеется, попиксельная прорисовка изображения не всегда удобна. Если требуется нарисовать готовые фигуры, используйте функции модуля `ImageDraw`, о котором будет рассказываться далее.

Проект: добавление логотипа

Предположим, вам предстоит утомительная работа по масштабированию тысяч изображений и добавлению на каждое из них небольшого логотипа. Попытка сделать такое с помощью простых графических редакторов наподобие Paintbrush или Paint длилась бы целую вечность. В более сложных графических приложениях, таких как Photoshop, существует возможность пакетной обработки, но не все согласятся заплатить за программу несколько сотен долларов. Поэтому мы напишем сценарий, решающий данную задачу.

На рис. 19.11 показан логотип, который мы будем добавлять в правый нижний угол каждого изображения. Это стилизованный профиль кошки с белым контуром и прозрачным фоном.

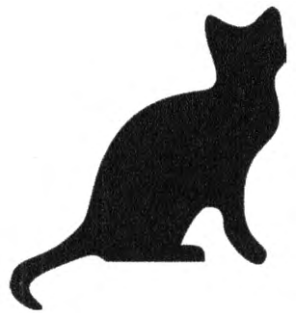


Рис. 19.11. Логотип, который будет добавлен к изображению

Вот что должна делать программа:

1) загружать изображение логотипа;
2) просматривать в цикле все файлы с расширением *.png* или *.jpg* в текущем каталоге;
3) проверять, не превышает ли ширина или высота изображения 300 пикселей;
4) в случае превышения размеров уменьшать ширину или высоту (в зависимости от того, что больше) до 300 пикселей, пропорционально уменьшая другой размер;
5) вставлять логотип в угол изображения;
6) сохранять измененные изображения в другой папке.

Это означает, что программа будет выполнять следующие операции:

1) открывать файл *catlogo.png* в качестве объекта `Image`;
2) проходить в цикле по всем строкам, возвращаемым функцией `os.listdir('.')`;
3) получать ширину и высоту изображения из атрибута `size`;
4) вычислять новые значения ширины и высоты изображения;
5) вызывать метод `resize()` для масштабирования изображения;
6) вызывать метод `paste()` для вставки логотипа;
7) вызывать метод `save()` для сохранения изменений.

Шаг 1. Открытие изображения логотипа

Откройте в файловом редакторе новое окно, введите следующий код и сохраните программу в файле *resizeAndAddLogo.py*.

```
#! python3
# resizeAndAddLogo.py - масштабирование всех изображений
# в текущем каталоге таким образом, чтобы они вписывались
# в квадрат размером 300x300, и добавление логотипа
# catlogo.png в правый нижний угол каждого изображения

import os
from PIL import Image

❶ SQUARE_FIT_SIZE = 300
❷ LOGO_FILENAME = 'catlogo.png'

❸ logoIm = Image.open(LOGO_FILENAME)
❹ logoWidth, logoHeight = logoIm.size

# СДЕЛАТЬ: организовать цикл по всем файлам в текущем каталоге

# СДЕЛАТЬ: проверить, нужно ли масштабировать изображение

# СДЕЛАТЬ: рассчитать новые значения ширины и высоты

# СДЕЛАТЬ: изменить размеры изображения

# СДЕЛАТЬ: добавить логотип

# СДЕЛАТЬ: сохранить изменения
```

Задав в начале программы константы `SQUARE_FIT_SIZE` ❶ и `LOGO_FILENAME` ❷, мы упростили внесение возможных изменений в программу в будущем. Предположим, вы захотите использовать в качестве логотипа другое изображение или ограничить размеры логотипа не 300 пикселями,

а другой величиной. В таком случае вам нужно будет внести изменения только в одном месте программы. (Возможен и другой вариант, когда значения этих констант передаются в качестве аргументов командной строки при вызове программы.) Без использования этих констант вам пришлось бы просматривать весь код в поиске всех вхождений значений `300` и `'catlogo.png'` и заменять их вручную в каждом новом проекте. Другими словами, константы делают программу более универсальной.

Функция `Image.open()` возвращает объект `Image` логотипа ❸. Для наглядности значения, содержащиеся в атрибуте `logoIm.size`, присваиваются отдельным переменным `logoWidth` и `logoHeight` ❹.

Остальная часть программы на данный момент представлена комментариями `'СДЕЛАТЬ'`.

Шаг 2. Цикл по всем файлам и открытие изображений

Теперь нам нужно найти в текущем каталоге все файлы PNG и JPG. При этом необходимо избежать добавления логотипа к самому изображению логотипу, а значит, программа должна пропускать любое изображение с тем же именем файла, которое содержится в константе `LOGO_FILENAME`. Добавьте в программу следующий код.

```
#! python3
# resizeAndAddLogo.py - масштабирование всех изображений
# в текущем каталоге таким образом, чтобы они вписывались
# в квадрат размером 300x300, и добавление логотипа
# catlogo.png в правый нижний угол каждого изображения

import os
from PIL import Image

-- Опущено --

os.makedirs('withLogo', exist_ok=True)
# Цикл по всем файлам в текущем каталоге
❶ for filename in os.listdir('.'):
❷     if not (filename.endswith('.png') or \
          filename.endswith('.jpg')) or filename == LOGO_FILENAME:
❸         continue    # оставить только файлы изображений
                      # и пропустить файл логотипа

❹     im = Image.open(filename)
      width, height = im.size

-- Опущено --
```

Сначала с помощью функции `os.makedirs()` мы создаем отдельную папку *withLogo*, предназначенную для хранения версий изображений

с логотипами, чтобы не затирать исходные файлы. При наличии именованного аргумента `exist_ok=True` функция `os.makedirs()` не будет генерировать исключение в том случае, если папка *withLogo* уже существует. В процессе выполнения цикла по всем файлам текущего каталога с использованием функции `os.listdir('.')` ❶ длинная инструкция `if` ❷ проверяет расширение каждого файла. Если файл не имеет расширение *.png* или *.jpg* или если это файл самого логотипа, то программа должна пропустить его, вызвав инструкцию `continue` ❸ для перехода к следующему файлу. Если же файл имеет расширение *.png* или *.jpg* (и это не файл логотипа), то мы открываем его в виде объекта `Image` ❹ и сохраняем ширину и высоту изображения в переменных `width` и `height`.

Шаг 3. Масштабирование изображений

Программа должна изменять размеры изображения лишь в том случае, если его ширина или высота превышает значение, определяемое константой `SQUARE_FIT_SIZE` (в данном случае — 300 пикселей), поэтому соответствующий код необходимо поместить в инструкцию `if`, проверяющую значения переменных `width` и `height`. Добавьте в программу следующий код.

```
#! python3
# resizeAndAddLogo.py - масштабирование всех изображений
# в текущем каталоге таким образом, чтобы они вписывались
# в квадрат размером 300x300, и добавление логотипа
# catlogo.png в правый нижний угол каждого изображения

import os
from PIL import Image

-- Опущено --

    # Проверяем, необходимо ли изменять размеры изображения
    if width > SQUARE_FIT_SIZE or height > SQUARE_FIT_SIZE:
        # Расчет новых значений ширины и высоты
        if width > height:
❶           height = int((SQUARE_FIT_SIZE / width) * height)
            width = SQUARE_FIT_SIZE
        else:
❷           width = int((SQUARE_FIT_SIZE / height) * width)
            height = SQUARE_FIT_SIZE

        # Изменение размеров изображения
        print('Изменение размеров изображения %s...' % (filename))
❸       im = im.resize((width, height))

-- Опущено --
```

Если размеры изображения требуется изменить, то сначала нужно определить, какой именно из размеров превышает допустимый предел: ширина или высота. Если ширина изображения больше его высоты, то последнюю следует уменьшить в той же пропорции, что и ширину ❶. Пропорция вычисляется делением значения `SQUARE_FIT_SIZE` на текущее значение ширины и умножением результата на текущую высоту. Поскольку результатом операции деления будет вещественное число, а метод `resize()` требует задания целочисленных размеров, не забудьте преобразовать частное в целое число с помощью функции `int()`. Наконец, новое значение ширины устанавливается равным `SQUARE_FIT_SIZE`.

Случай, когда высота изображения больше ширины или равна ей (блок `else`), обрабатывается так же, только теперь переменные `height` и `width` меняются местами ❷.

Установив новые значения переменных `width` и `height`, мы передаем их методу `resize()` и сохраняем полученный объект `Image` в переменной `im` ❸.

Шаг 4. Добавление логотипа и сохранение изменений

Независимо от того, изменялись ли размеры изображения, логотип должен помещаться в правый нижний угол изображения. В какую именно позицию он должен вставляться, определяется размерами как изображения, так и самого логотипа. На рис. 19.12 показано, как рассчитать позицию вставки. Левая координата позиции для вставки логотипа должна быть равна разности между шириной изображения и шириной логотипа, тогда как верхняя координата должна быть равна разности между высотой изображения и высотой логотипа.

Рис. 19.12. Координаты левого и верхнего краев логотипа при его помещении в правый нижний угол изображения определяются разностью значений ширины/высоты изображения и ширины/высоты логотипа

После того как программа вставит логотип в изображение, она должна сохранить измененный объект Image. Добавьте в программу следующий код.

```
#! python3
# resizeAndAddLogo.py - масштабирование всех изображений
# в текущем каталоге таким образом, чтобы они вписывались
# в квадрат размером 300x300, и добавление логотипа
# catlogo.png в правый нижний угол каждого изображения

import os
from PIL import Image

-- Опущено --

    # Проверяем, необходимо ли изменять размеры изображения
     -- Опущено --

    # Добавление логотипа
❶   print('Добавление логотипа в изображение %s...' % (filename))
❷   im.paste(logoIm, (width - logoWidth, height - logoHeight),
                logoIm)

    # Сохранение изменений
❸   im.save(os.path.join('withLogo', filename))
```

Мы выводим сообщение, извещающее пользователя о добавлении логотипа ❶, помещаем изображение logoIm в позицию с рассчитанными координатами ❷ и сохраняем измененный файл в папке *withLogo* ❸. Если запустить программу, когда единственным изображением в текущем каталоге будет *zophie.png*, то получим следующие результаты.

```
Изменение размеров изображения zophie.png...
Добавление логотипа в изображение zophie.png...
```

Изображение *zophie.png* будет масштабировано до размера 225×300 пикселей, и к нему будет добавлен логотип (рис. 19.13). Один нюанс: метод paste() не вставит прозрачные пиксели в результирующее изображение, если не передать ему третий аргумент — маску прозрачности. В данном случае маской служит сам объект logoIm. Теперь программа сможет пометить логотипом сотни изображений и соответствующим образом изменить их размеры буквально за пару минут.

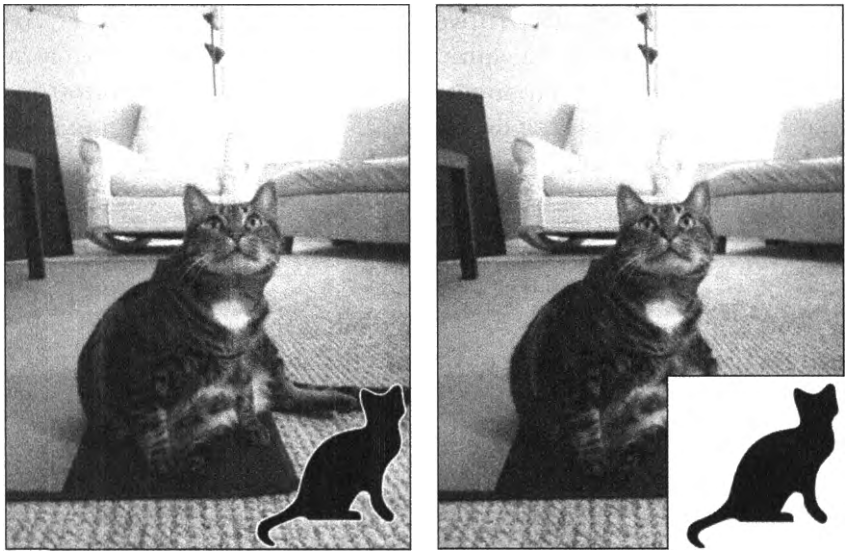


Рис. 19.13. Изображение zophie.png с добавленным логотипом (слева). Если забыть указать третий аргумент метода `paste()`, то прозрачные пиксели логотипа будут скопированы в виде белой заливки (справа).

Идеи для создания похожих программ

Возможность наложения и масштабирования изображений в пакетном режиме будет полезной во многих приложениях. В частности, можно делать следующее:

- добавлять текст или URL-адрес веб-сайта в изображения;
- добавлять метки времени в изображения;
- копировать или перемещать изображения в различные папки, исходя из размеров файлов;
- добавлять водяные знаки в изображения, чтобы защитить их от несанкционированного копирования.

Рисование на изображениях

Модуль `ImageDraw` библиотеки Pillow позволяет нарисовать отрезок, прямоугольник, окружность или другую простую фигуру поверх изображения. Введите в интерактивной оболочке следующие инструкции.

```
>>> from PIL import Image, ImageDraw
>>> im = Image.new('RGBA', (200, 200), 'white')
>>> draw = ImageDraw.Draw(im)
```

В первую очередь мы импортируем модули `Image` и `ImageDraw`. Затем мы создаем новое изображение (в данном случае — квадрат белого цвета размером 200×200 пикселей) и сохраняем объект `Image` в переменной `im`. Далее этот объект `Image` передается функции `ImageDraw.Draw()` для получения объекта `ImageDraw`, у которого есть несколько методов, предназначенных для рисования фигур и текста. Мы сохраняем объект `ImageDraw` в переменной `draw`, чтобы с ним можно было работать в последующих примерах.

Рисование фигур

Описанные ниже методы объекта `ImageDraw` предназначены для рисования различных фигур. Параметры `fill` (заливка) и `outline` (обводка) этих методов необязательные; по умолчанию для них устанавливается белый цвет.

Точки

Метод `point(xy, fill)` рисует отдельные пиксели. Аргумент *xy* задает список точек, которые нужно нарисовать. Это может быть список кортежей координат *x* и *y*, например `[(x, y), (x, y), ...]`, или же список координат *x* и *y* без кортежей, например `[x1, y1, x2, y2, ...]`. Необязательный аргумент *fill* определяет цвет точек и может представлять собой либо RGBA-кортеж, либо строку с названием цвета, такую как `'red'`.

Отрезки

Метод `line(xy, fill, width)` предназначен для рисования одиночных отрезков или серии отрезков. Аргумент *xy* — это либо список кортежей, такой как `[(x, y), (x, y), ...]`, либо список целых чисел, например `[x1, y1, x2, y2, ...]`. Каждая пара координат задает один из концов рисуемого отрезка. Необязательный аргумент *fill* определяет цвет линий и задается в виде RGBA-кортежа или названия цвета. Необязательный аргумент *width* определяет толщину линий и по умолчанию имеет значение `1`.

Прямоугольники

Метод `rectangle(xy, fill, outline)` предназначен для рисования прямоугольников. Аргумент *xy* — это кортеж прямоугольника вида `(left, top, right, bottom)`. Значения *left* и *top* определяют координаты *x* и *y* левого верхнего угла прямоугольника, тогда как значения *right* и *bottom* — координаты правого нижнего угла. Необязательный аргумент *fill* определяет цвет заливки, а необязательный аргумент *outline* — цвет обводки прямоугольника.

Эллипсы

Метод `ellipse(xy, fill, outline)` предназначен для рисования эллипсов. В случае совпадения ширины и высоты эллипса рисуется окружность. Аргумент *xy* — это кортеж прямоугольника `(left, top, right, bottom)`, который задает прямоугольник, описанный вокруг эллипса. Необязательный аргумент *fill* определяет цвет заливки, а необязательный аргумент *outline* — цвет обводки эллипса.

Многоугольники

Метод `polygon(xy, fill, outline)` предназначен для рисования многоугольников с произвольным числом сторон. Аргумент *xy* — это список кортежей, такой как `[(x, y), (x, y), ...]`, либо список целых чисел, например `[x1, y1, x2, y2, ...]`, представляющий углы многоугольника. Последняя пара координат будет автоматически соединяться с первой парой. Необязательный аргумент *fill* определяет цвет заливки, а необязательный параметр *outline* — цвет обводки многоугольника.

Пример рисования фигур

Введите в интерактивной оболочке следующие инструкции.

```
  >>> from PIL import Image, ImageDraw
  >>> im = Image.new('RGBA', (200, 200), 'white')
  >>> draw = ImageDraw.Draw(im)
❶ >>> draw.line([(0, 0), (199, 0), (199, 199), (0, 199), (0, 0)],
                 fill='black')
❷ >>> draw.rectangle((20, 30, 60, 60), fill='blue')
❸ >>> draw.ellipse((120, 30, 160, 60), fill='red')
❹ >>> draw.polygon(((57, 87), (79, 62), (94, 85), (120, 90),
                    (103, 113)), fill='brown')
❺ >>> for i in range(100, 200, 10):
          draw.line([(i, 0), (200, i - 100)], fill='green')

  >>> im.save('drawing.png')
```

Мы создаем объект `Image` для белого квадрата с размерами 200×200 пикселей, передаем его методу `ImageDraw.Draw()` для получения объекта `ImageDraw` и сохраняем этот объект в переменной `draw`. Теперь мы можем вызывать методы объекта `ImageDraw` для рисования фигур. В данном случае мы создаем тонкую черную обводку вдоль краев изображения ❶; голубой прямоугольник с координатами левого верхнего и правого нижнего углов `(20, 30)` и `(60, 60)` соответственно ❷; красный эллипс, вписанный в прямоугольник с координатами `(120, 30)` и `(160, 60)` ❸; и коричневый пятиугольник ❹. В цикле `for` ❺ рисуется сетка из зеленых отрезков. Результирующее изображение сохраняется в файле *drawing.png* (рис. 19.14).

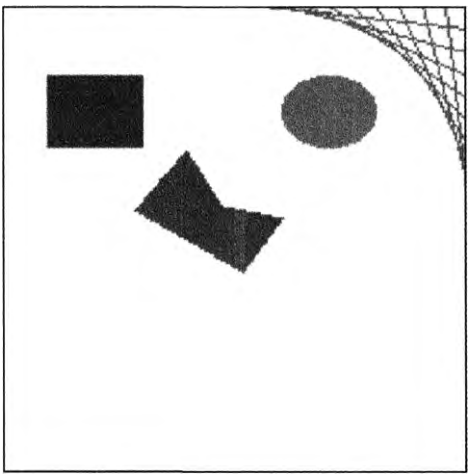


Рис. 19.14. Полученное изображение drawing.png

У объекта `ImageDraw` есть и другие методы, предназначенные для рисования фигур. Полная документация доступна по следующему адресу:

`https://pillow.readthedocs.org/en/latest/reference/ImageDraw.html`

Рисование текста

Кроме описанных методов, у объекта `ImageDraw` есть также метод `text()`, предназначенный для рисования текста поверх изображения. У этого метода четыре аргумента:

- аргумент *xy* — это кортеж из двух целых чисел, определяющий координаты левого верхнего угла текстового поля;
- аргумент *text* — это строка текста, которую нужно отобразить;
- необязательный аргумент *fill* определяет цвет текста;
- необязательный аргумент *font* — это объект `ImageFont`, используемый для задания шрифта и размера текста.

Поскольку во многих случаях трудно заранее определить, каким будет размер текстового поля для конкретного шрифта, в модуле `ImageDraw` предусмотрен метод `textsize()`. Его первый аргумент — это строка текста, которую нужно измерить, а второй аргумент — необязательный объект `ImageFont`. Метод `textsize()` возвращает кортеж из двух целых чисел, представляющих ширину и высоту текстового поля при заданном размере шрифта. Эти значения можно использовать для расчета координат текста.

Первые три аргумента метода `text()` не нуждаются в дополнительных пояснениях. Нам лишь нужно разобраться с необязательным четвертым аргументом — объектом `ImageFont`.

Чтобы создать такой объект, необходимо предварительно выполнить следующую инструкцию для импорта модуля `ImageFont` из пакета Pillow:

```
>>> from PIL import ImageFont
```

Теперь можно вызвать функцию `ImageFont.truetype()`, которая имеет два аргумента. Первый аргумент — это строка с именем файла шрифта *TrueType*, существующего на жестком диске. Такие файлы имеют расширение *.ttf* и обычно располагаются в следующих папках:

- Windows — *C:\Windows\Fonts*;
- macOS — */Library/Fonts* и */System/Library/Fonts*;
- Linux — */usr/share/fonts/truetype*.

Задавать эти пути в качестве части строки, содержащей имя файла шрифта TrueType, не нужно, поскольку Python выполняет автоматический поиск шрифтов. Если же найти указанный шрифт не удается, будет выдано сообщение об ошибке.

Второй аргумент функции `ImageFont.truetype()` — целое число, определяющее размер шрифта в *пунктах* (не в пикселях). По умолчанию модуль Pillow создает изображения в формате PNG с разрешением 72 пикселя на дюйм, где пункт — это 1/72 дюйма.

Введите в интерактивной оболочке следующие инструкции, заменив константу `FONT_FOLDER` именем папки шрифтов вашей операционной системы.

```
  >>> from PIL import Image, ImageDraw, ImageFont
  >>> import os
❶ >>> im = Image.new('RGBA', (200, 200), 'white')
❷ >>> draw = ImageDraw.Draw(im)
❸ >>> draw.text((20, 150), 'Hello', fill='purple')
  >>> fontsFolder = 'FONT_FOLDER'    # например, '/Library/Fonts'
❹ >>> arialFont = ImageFont.truetype(os.path.join(fontsFolder,
      'arial.ttf'), 32)
❺ >>> draw.text((100, 150), 'Привет', fill='gray', font=arialFont)
  >>> im.save('text.png')
```

После импорта модулей `Image`, `ImageDraw`, `ImageFont` и `os` мы создаем сначала объект `Image` для нового изображения в виде квадрата белого цвета с размерами 200×200 пикселей ❶, а затем объект `ImageDraw` на основе объекта `Image` ❷. Далее мы вызываем метод `text()` для отображения текста `'Hello'` пурпурного цвета в позиции с координатами `(20, 150)` ❸. В данном случае мы не передаем методу `text()` необязательный четвертый аргумент, поэтому гарнитура и размер шрифта выбираются по умолчанию.

Чтобы задать гарнитуру и размер шрифта, мы предварительно сохраняем имя папки (например, */Library/Fonts*) в переменной `fontsFolder`. Затем мы вызываем функцию `ImageFont.truetype()`, передавая ей имя *.ttf*-файла шрифта и целочисленный аргумент, определяющий размер шрифта ❹. Объект `Font`, возвращаемый функцией `ImageFont.truetype()`, сохраняется в переменной `arialFont`, которая передается методу `text()` в четвертом аргументе. Метод `text()` выводит текст `'Привет'` серого цвета в позиции с координатами `(100, 150)` и с использованием шрифта Arial размером 32 пункта ❺.

Результирующее изображение сохраняется в файле *text.png* (рис. 19.15).

Рис. 19.15. Результирующее изображение text.png

Резюме

Изображения состоят из пикселей, каждый из которых описывается RGBA-значением, определяющим его цвет и прозрачность, а также координатами *x* и *y*. Два самых распространенных формата изображений — JPEG и PNG. Модуль Pillow способен обрабатывать изображения как этих форматов, так и многих других.

После загрузки изображения в объект `Image` его ширина и высота сохраняются в виде кортежа из двух целых чисел в атрибуте `size`. У объектов `Image` есть методы, позволяющие тем или иным образом манипулировать изображениями: `crop()`, `copy()`, `paste()`, `resize()`, `rotate()` и

`transpose()`. Чтобы сохранить объект `Image` в файле изображения, необходимо вызвать метод `save()`.

Если в программе требуется рисовать фигуры на изображениях, используйте методы объекта `ImageDraw`, позволяющие создавать точки, отрезки, прямоугольники, эллипсы и многоугольники. Кроме того, у этого объекта есть методы для рисования текста с использованием указанной гарнитуры и заданного размера шрифта.

В профессиональных (и недешевых) графических редакторах, таких как Photoshop, имеются возможности пакетной обработки изображений, но многие подобные операции можно выполнять бесплатно с помощью сценариев Python. В предыдущих главах мы создавали программы Python для обработки простых текстовых файлов, электронных таблиц, документов PDF и т.п. Благодаря модулю Pillow у вас появляется возможность писать графические приложения, работающие с изображениями.

Контрольные вопросы

1. Что такое значение RGBA?
2. Как получить RGBA-значение для стандартного цвета `'Cornflower Blue'`?
3. Что такое кортеж прямоугольника?
4. С помощью какой функции можно получить объект `Image` для файла изображения *zophie.png*?
5. Как определить ширину и высоту изображения, хранящегося в объекте `Image`?
6. Какой метод необходимо вызвать, чтобы получить объект `Image` для изображения размером 100×100 пикселей, исключив его левую нижнюю четверть?
7. Как сохранить файл изображения после внесения изменений в объект `Image`?
8. Какой модуль из пакета Pillow содержит методы для рисования фигур?
9. У объекта `Image` нет методов рисования. А у какого объекта они есть? Как получить такой объект?

Учебные проекты

Чтобы закрепить полученные знания на практике, напишите программы для предложенных ниже задач.

Доработка основного проекта главы

Рассмотренная в этой главе программа *resizeAndAddLogo.py* работает с файлами форматов PNG и JPEG, но модуль Pillow поддерживает намного больше форматов. Расширьте программу таким образом, чтобы она могла обрабатывать также изображения в форматах GIF и BMP.

Еще одна проблема заключается в том, что программа может работать с файлами PNG и JPEG лишь в том случае, если их расширения заданы в нижнем регистре. Например, она обработает файл *zophie.png*, но не файл *zophie.PNG*. Измените программу таким образом, чтобы она была нечувствительна к регистру расширения.

Кроме того, изначально предполагается, что логотип, добавляемый в правый нижний угол изображения, покрывает лишь небольшую его часть. Но если размеры изображения и логотипа примерно одинаковы, то результат будет выглядеть примерно так, как на рис. 19.16. Измените программу таким образом, чтобы логотип добавлялся только в том случае, когда размеры изображения по крайней мере в два раза превышают размеры логотипа. Если это условие не выполняется, то логотип не должен добавляться.

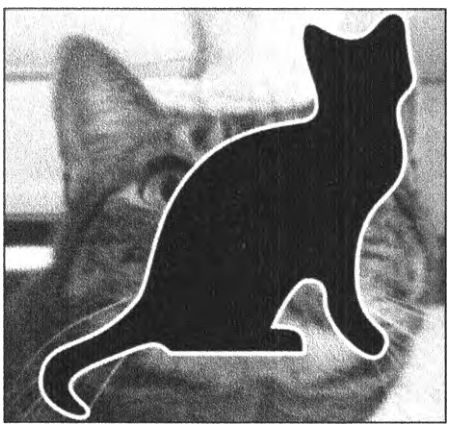


Рис. 19.16. Когда размеры основного изображения лишь ненамного больше размеров логотипа, результат выглядит уродливо

Поиск папок с фотографиями на жестком диске

У меня есть плохая привычка перемещать файлы из своего цифрового фотоаппарата во временные папки на жестком диске, имена которых я, конечно же, впоследствии забываю. Хотелось бы иметь программу, которая сканировала бы весь жесткий диск и находила эти забытые "фотопапки".

Напишите программу, которая просматривает все содержимое жесткого диска и находит потенциальные папки, в которых могут находиться

фотографии. Разумеется, сначала нужно определиться с тем, какие папки следует относить к этой категории. Например, это могут быть папки, более половины файлов в которых — фотографии. Но как определить, какие файлы являются фотографиями?

Прежде всего, файл с фотографией должен иметь расширение *.png* или *.jpg*. Кроме того, фотографии — это большие изображения, ширина и высота которых должны превышать 500 пикселей. Данное ограничение взято с запасом, поскольку ширина и высота фотографий, получаемых с помощью современных цифровых камер, — обычно несколько тысяч пикселей.

Вот общий каркас такой программы.

```
#! python3
# Импорт модулей и комментарий с описанием программы

for foldername, subfolders, filenames in os.walk('C:\\'):
    numPhotoFiles = 0
    numNonPhotoFiles = 0
    for filename in filenames:
        # Проверить, имеют ли файлы расширение .png или .jpg
        if СДЕЛАТЬ:
            numNonPhotoFiles += 1
            continue    # перейти к следующему файлу

        # Открыть файл изображения, используя модуль Pillow

        # Проверить, чтобы ширина и высота изображения
        # превышали 500 пикселей
        if СДЕЛАТЬ:
            # Размеры файла достаточно велики, чтобы
            # его можно было считать фотографией
            numPhotoFiles += 1
        else:
            # Изображение слишком маленькое, чтобы его
            # можно было считать фотографией
            numNonPhotoFiles += 1

    # Если более половины файлов оказались фотографиями,
    # вывести абсолютный путь к папке
    if СДЕЛАТЬ:
        print(СДЕЛАТЬ)
```

Программа должна вывести на экран абсолютные пути доступа ко всем папкам, содержащим фотографии.

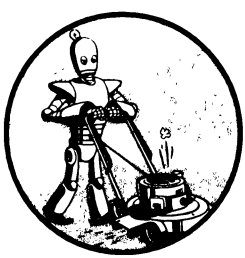
Персональные приглашения

В главе 15 рассматривался учебный проект по созданию персональных приглашений на основе списка гостей, хранящегося в текстовом файле. Расширьте проект, добавляя в каждое приглашение графическое изображение с помощью модуля Pillow. Для каждого из гостей, указанных в файле *guests.txt*, сгенерируйте файл изображения, включающего декоративные элементы и имя гостя.

Чтобы все пригласительные билеты были одного размера, добавьте вокруг изображения черную прямоугольную рамку, которая будет служить ориентиром при разрезании приглашений после их вывода на печать. Модуль Pillow позволяет создавать PNG-файлы с разрешением 72 пикселя на дюйм, так что для приглашения размером 4×5 дюймов потребуется изображение с размерами 288×360 пикселей.

20

УПРАВЛЕНИЕ КЛАВИАТУРОЙ И МЫШЬЮ



На данный момент вы уже изучили довольно много модулей Python, которые могут пригодиться для автоматизации самых разных задач, таких как редактирование электронных таблиц, загрузка файлов или запуск программ по расписанию. Но не для всех приложений есть готовые модули. В этом случае удачным решением может стать написание программ, непосредственно управляющих клавиатурой и мышью. Такие программы способны взаимодействовать с другими приложениями, отправляя им виртуальные нажатия клавиш или щелчки мышью, как если бы вы сами сидели за компьютером.

Подобный режим работы называется *автоматизацией графического интерфейса* (сокращенно — *GUI-автоматизация*). Это можно сравнить с программированием роботизированной руки, которая нажимает за вас клавиши и перемещает указатель мыши. Средства автоматизации будут особенно полезными в задачах, где приходится непрерывно щелкать мышью или заполнять формы.

Некоторые компании продают инновационные (и дорогостоящие) системы, позиционируемые как *роботизированные процессы автоматизации* (Robotic Process Automation — RPA). Эти продукты практически не отличаются от сценариев Python, которые можно создать самостоятельно с помощью модуля PyAutoGUI. В нем есть функции для имитации движений мыши, нажатий кнопок и прокрутки колесика мыши. В данной главе мы рассмотрим только часть функций PyAutoGUI. Полная документация доступна по адресу `https://pyautogui.readthedocs.io/`.

Установка модуля PyAutoGUI

Модуль PyAutoGUI способен генерировать виртуальные нажатия клавиш и щелчки мышью в Windows, macOS и Linux. Пользователи Windows и macOS могут установить модуль с помощью команды `pip`. В Linux нужно сначала установить программное обеспечение, от которого зависит PyAutoGUI. Откройте окно терминала и введите следующие команды:

- **`sudo apt-get install scrot`**
- **`sudo apt-get install python3-tk`**
- **`sudo apt-get install python3-dev`**

Для установки модуля PyAutoGUI выполните команду `pip install --user pyautogui`. Не используйте команду `sudo` совместно с `pip`: есть риск установить модули в дистрибутив Python, используемый операционной системой, что вызовет конфликты со всеми сценариями, которые полагаются на первоначальную конфигурацию среды разработки. В то же время при установке приложений с помощью утилиты `apt-get` необходимо использовать команду `sudo`.

Информация об установке сторонних модулей приведена в приложении А. Чтобы проверить, правильно ли установлен модуль PyAutoGUI, выполните команду **`import pyautogui`** в интерактивной оболочке и убедитесь в отсутствии сообщений об ошибках.

Предупреждение

Не сохраняйте программу в виде файла pyautogui.py. Если выполнить команду `import pyautogui`, Python импортирует программу, а не модуль PyAutoGUI, и вы

увидите сообщение об ошибке типа `AttributeError: module 'pyautogui' has no attribute 'click'`.

Настройка доступности в macOS

В качестве меры безопасности macOS обычно не позволяет программам управлять мышью или клавиатурой. Чтобы заставить модуль PyAutoGUI работать в macOS, необходимо сконфигурировать интерпретатор, выполняющий сценарии Python, как приложение доступности. Без этого вызовы функций PyAutoGUI не будут иметь никакого эффекта.

Независимо от того, запускаете ли вы свои программы из Mu, IDLE или Terminal, это приложение должно быть открыто. Затем откройте окно System Preferences (Системные настройки) и перейдите на вкладку Accessibility (Доступность). Открытые в данный момент приложения будут отображаться под ярлыком Allow the apps below to control your computer (Разрешить этим приложениям управлять вашим компьютером). Отметьте Mu, IDLE, Terminal или любое другое приложение, которое вы используете для запуска сценариев Python. Вам будет предложено ввести пароль для подтверждения изменений.

Контроль над клавиатурой и мышью

Прежде чем приступать к использованию средств GUI-автоматизации, необходимо узнать о том, как избежать потенциальных проблем. Python позволяет перемещать указатель мыши и выполнять виртуальные нажатия клавиш с невероятной скоростью, которая может оказаться непомерно большой для других программ. Если что-то пойдет не так, а программа тем временем будет по-прежнему перемещать указатель мыши, вам будет сложно определить причины неполадок. Подобно волшебным мётлам из фильма "Ученик чародея", которые не прекращали доливать воду в бак, когда он переполнился, программа может выйти из-под контроля, даже если будет идеально выполнять все ваши инструкции. Если указатель мыши начнет хаотично перемещаться по всему экрану, не давая возможности щелкнуть на кнопке закрытия окна Mu, то остановить работу программы будет трудно. К счастью, существует несколько способов избежать проблем, связанных с GUI-автоматизацией.

Паузы и безопасное завершение работы

Если в программе есть ошибка, из-за которой работу программы не удается завершить с помощью клавиатуры или мыши, воспользуйтесь средством безопасного завершения, доступным в модуле PyAutoGUI. Быстро переместите указатель мыши к одному из четырех углов экрана. Каждая

функция PyAutoGUI выдерживает паузу длительностью 1/10 секунды перед завершением, давая пользователю возможность переместить указатель мыши в угол. Если модуль PyAutoGUI обнаруживает, что указатель мыши находится в углу, генерируется исключение `pyautogui.FailSafeException`. У вызовов, не связанных с PyAutoGUI, такой задержки не будет.

Если вы оказались в ситуации, когда нужно остановить работу программы, использующей модуль PyAutoGUI, просто переместите указатель мыши в угол экрана.

Прекращение выполнения всех задач путем выхода из учетной записи

Возможно, самый простой способ остановить вышедшую из-под контроля программу GUI-автоматизации заключается в выходе из учетной записи пользователя, что приведет к прекращению выполнения всех запущенных в ней программ. В Windows и Linux для этого следует нажать комбинацию клавиш <Ctrl+Alt+Del>, а в macOS — комбинацию <⌘+Shift+Option+Q>. После выхода из учетной записи вы потеряете несохраненные результаты работы, зато не придется перезагружать компьютер.

Управление перемещениями мыши

В этом разделе вы узнаете о том, как перемещать указатель мыши и отслеживать его позицию на экране с помощью модуля PyAutoGUI, но сначала необходимо понять, как работать с координатами.

Функции PyAutoGUI проверяют координаты *x* и *y* указателя мыши. Система координат для компьютерного экрана показана на рис. 20.1; она аналогична системе координат, используемой при работе с изображениями, которая обсуждалась в главе 19. Начало координат находится в левом верхнем углу экрана. Координата *x* увеличивается в направлении слева направо, координата *y* — в направлении сверху вниз. Все координаты — неотрицательные целые числа.

Разрешение экрана определяет его ширину и высоту в пикселях. Если разрешение экрана — 1920×1080, то координаты его левого верхнего угла — `(0, 0)`, а правого нижнего — `(1919, 1079)`.

Функция `pyautogui.size()` возвращает кортеж из двух целых чисел, задающих ширину и высоту экрана в пикселях. Введите в интерактивной оболочке следующие команды.

```
>>> import pyautogui
>>> wh = pyautogui.size()     # определяем разрешение экрана
>>> wh
Size(width=1920, height=1080)
```

```
>>> wh[0]
1920
>>> wh.width
1920
```

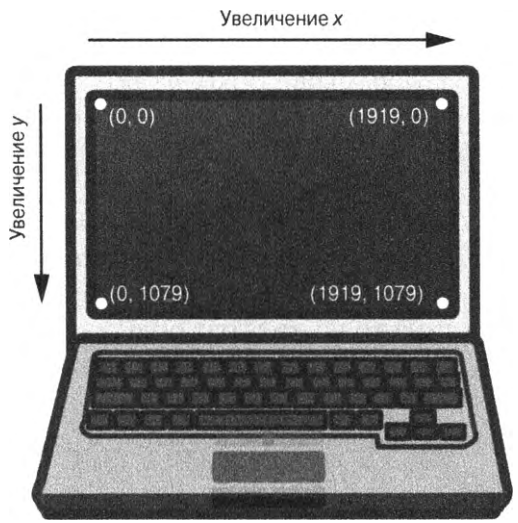


Рис. 20.1. Система координат на компьютерном экране с разрешением 1920×1080

Для экрана с разрешением 1920×1080 функция `pyautogui.size()` возвращает кортеж `(1920, 1080)`; для вашего экрана возвращаемое значение может отличаться. Объект `Size`, возвращаемый этой функцией, представляет собой *именованный кортеж*. У таких кортежей есть и обычные числовые индексы, и имена атрибутов, как у объектов. В частности, оба выражения — `и wh[0]`, и `wh.width` — возвращают ширину экрана. (Описание именованных кортежей выходят за рамки книги. Просто помните, что с ними можно работать как с обычными кортежами.)

Перемещение указателя мыши

Теперь, когда вы понимаете, что собой представляют экранные координаты, можно поэкспериментировать с перемещением указателя мыши. Функция `pyautogui.moveTo()` перемещает указатель в заданную позицию на экране. Первым и вторым аргументами этой функции будут координаты *x* и *y* соответственно. Необязательный именованный аргумент `duration` задаст (в секундах) длительность перемещения указателя в конечную точку. По умолчанию он равен `0`, что соответствует мгновенному перемещению. (Все аргументы `duration` функций PyAutoGUI являются необязательными.) Введите в интерактивной оболочке следующие команды.

```
>>> import pyautogui
>>> for i in range(10):    # перемещаем указатель мыши по квадрату
...     pyautogui.moveTo(100, 100, duration=0.25)
...     pyautogui.moveTo(200, 100, duration=0.25)
...     pyautogui.moveTo(200, 200, duration=0.25)
...     pyautogui.moveTo(100, 200, duration=0.25)
```

В этом примере указатель мыши обходит все стороны квадрата по часовой стрелке десять раз. Каждое перемещение по стороне квадрата осуществляется за четверть секунды (`duration=0.25`). Если опустить третий аргумент в любом из вызовов функции `pyautogui.moveTo()`, то указатель мыши будет мгновенно перемещаться из одной точки в другую.

Функция `pyautogui.move()` перемещает указатель мыши *относительно его текущей позиции*. В следующем примере указатель также перемещается по сторонам квадрата, только на этот раз роль начальной точки квадрата играет та точка экрана, в которой указатель находится на момент выполнения кода.

```
>>> import pyautogui
>>> for i in range(10):
...     pyautogui.move(100, 0, duration=0.25)    # вправо
...     pyautogui.move(0, 100, duration=0.25)    # вниз
...     pyautogui.move(-100, 0, duration=0.25)   # влево
...     pyautogui.move(0, -100, duration=0.25)   # вверх
```

Функция `pyautogui.move()` тоже имеет три аргумента: величина перемещения в пикселях вправо по горизонтали и вниз по вертикали, а также необязательная длительность перемещения (в секундах). Отрицательное значение первого или второго аргумента означает перемещение указателя влево или вверх соответственно.

Получение позиции указателя

Чтобы определить текущую позицию указателя мыши, вызовите функцию `pyautogui.position()`, которая возвращает кортеж координат *x* и *y* указателя мыши на момент вызова функции. Введите в интерактивной оболочке следующие команды, перемещая указатель мыши после каждого вызова.

```
>>> pyautogui.position()    # получение текущей позиции указателя
Point(x=311, y=622)
>>> pyautogui.position()    # повторное получение позиции указателя
Point(x=377, y=481)
>>> p = pyautogui.position()    # и еще раз
>>> p
Point(x=1536, y=637)
```

```
>>> p[0]    # x-координата имеет индекс 0
1536
>>> p.x     # x-координата также представлена одноименным атрибутом
1536
```

Возвращаемые значения будут варьироваться в зависимости от того, где находится указатель мыши.

Управление взаимодействием с мышью

Теперь, когда вы знаете, как перемещать указатель мыши и определять его местоположение на экране, приступим к выполнению таких операций, как щелчки, перетаскивание и прокрутка.

Щелчки мышью

Чтобы отправить компьютеру виртуальный щелчок мышью, вызовите метод `pyautogui.click()`. По умолчанию предполагается, что щелчок выполняется левой кнопкой в месте текущего расположения указателя мыши. Если требуется выполнить щелчок в другом месте, передайте координаты *x* и *y* соответствующей точки в качестве необязательных первого и второго аргументов.

Если нужно задать кнопку, которой выполняется щелчок, то включите в вызов именованный аргумент `button`, имеющий одно из следующих значений: `'left'` (левая), `'middle'` (средняя) или `'right'` (правая). Например, вызову `pyautogui.click(100, 150, button='left')` соответствует щелчок левой кнопкой в точке с координатами `(100, 150)`, а вызову `pyautogui.click(200, 250, button='right')` — щелчок правой кнопкой в точке с координатами `(200, 250)`.

Введите в интерактивной оболочке следующие инструкции.

```
>>> import pyautogui
>>> pyautogui.click(10, 5) # перемещение указателя мыши и щелчок
```

Вы увидите, как указатель мыши переместится в левый верхний угол экрана и выполнит одиночный щелчок. Под "щелчком" понимается нажатие кнопки мыши и последующее ее отпускание без перемещения указателя. Щелчки также можно имитировать с помощью функции `pyautogui.mouseDown()`, которой соответствует нажатие кнопки, и функции `pyautogui.mouseUp()`, которой соответствует отпускание кнопки. Эти функции имеют те ж аргументы, что и функция `click()`, и по сути последняя служит лишь оболочкой для их вызова.

Дополнительно имеется функция `pyautogui.doubleClick()`, выполняющая двойной щелчок левой кнопкой мыши, а также функции `pyautogui.`

`rightClick()` и `pyautogui.middleClick()`, которые выполняют щелчок соответственно правой и средней копками.

Перетаскивание указателя мыши

Термин *перетаскивание* означает перемещение указателя мыши при одновременном удерживании нажатой одной из ее кнопок. Например, можно перемещать файлы между папками, перетаскивая их значки, или перемещать назначенные встречи в приложении календаря.

Модуль PyAutoGUI содержит функции `pyautogui.dragTo()` и `pyautogui.drag()`, позволяющие перетаскивать указатель мыши в точку с заданными координатами и точку, заданную относительно текущего положения. Аргументы этих функций такие же, как и у функций `moveTo()` и `move()`: координата *x* (горизонтальное смещение), координата *y* (вертикальное смещение) и необязательный именованный аргумент, задающий длительность перемещения. (В macOS этот аргумент желательно использовать, поскольку при слишком быстром перемещении указателя мыши операция перетаскивания может выполняться некорректно.)

Чтобы проверить, как работают эти функции, откройте какое-нибудь графическое приложение, например Paint в Windows, Paintbrush в macOS или GNU Paint в Linux. (Можете также воспользоваться онлайн-редактором SumoPaint, доступным на сайте `https://sumopaint.com/`.) Для выполнения операций рисования в этих приложениях мы будем использовать библиотеку PyAutoGUI.

Убедитесь, что указатель мыши находится на холсте графического приложения и в качестве текущего инструмента выбран Pencil (Карандаш) или Brush (Кисть), после чего введите в новом окне файлового редактора следующий код и сохраните программу в файле *spiralDraw.py*.

```
  import pyautogui, time
❶ time.sleep(5)
❷ pyautogui.click()    # щелчок для активизации окна
  distance = 300
  change = 20
  while distance > 0:
❸     pyautogui.drag(distance, 0, duration=0.2)    # вправо
❹     distance = distance - change
❺     pyautogui.drag(0, distance, duration=0.2)    # вниз
❻     pyautogui.drag(-distance, 0, duration=0.2)   # влево
      distance = distance - change
      pyautogui.drag(0, -distance, duration=0.2)    # вверх
```

При запуске программы будет пятисекундная пауза ❶, чтобы вы успели переместить указатель мыши в окно графического редактора и выбрать требуемый инструмент рисования. После этого сценарий *spiralDraw*

py перехватит управление мышью и выполнит щелчок для активизации окна редактора ❷. Окно становится *активным* (получает *фокус ввода*), если реагирует на нажатия клавиш и выполняемые вами действия, такие как перетаскивание указателя мыши, воздействуют на него. Как только окно графического редактора активизируется, сценарий *spiralDraw.py* нарисует квадратную спираль наподобие той, которая показана на рис. 20.2, *слева*. Такие спирали можно рисовать с помощью модуля Pillow, описанного в главе 19, но в редакторе Paint доступны различные стили кисти, как показано на рис. 20.2, *справа*, а также другие расширенные средства рисования, в частности градиенты и заливки. Попробуйте предварительно выбрать настройки кисти, а затем запустить программу для рисования спирали.

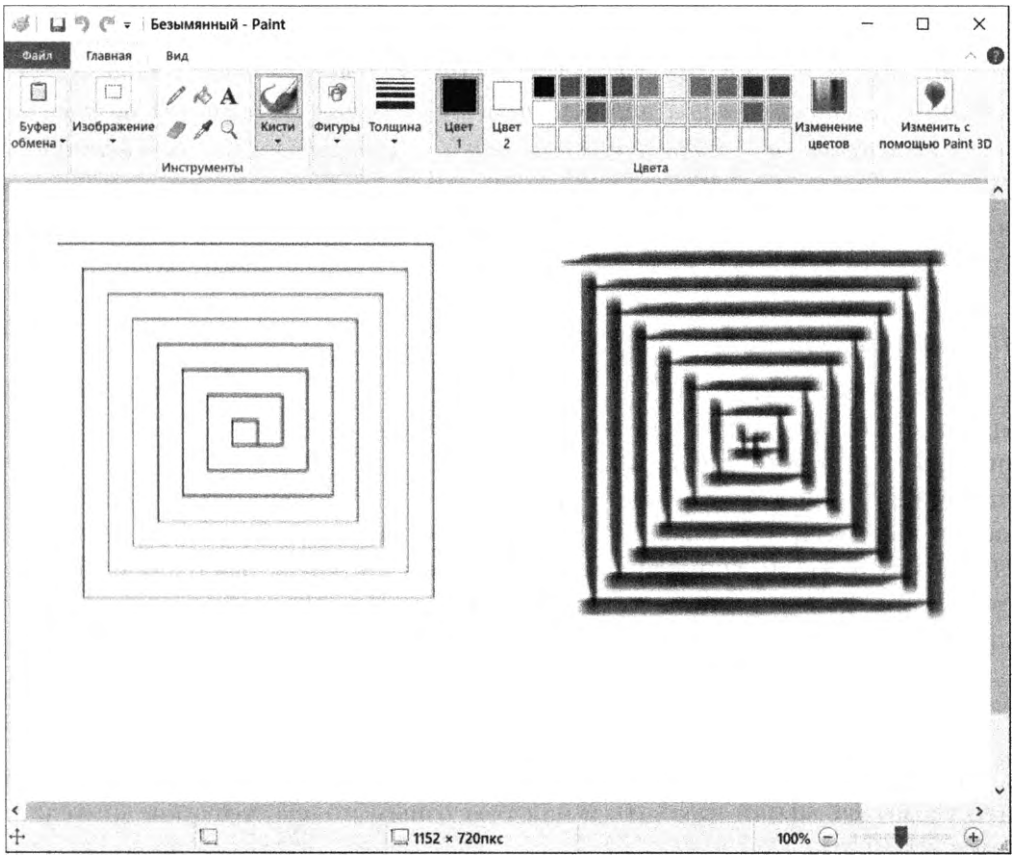


Рис. 20.2. Результат работы программы, в которой используется функция `pyautogui.drag()`

Первоначально переменная `distance` равна `300`, поэтому на первой итерации цикла `while` функция `drag()` перемещает указатель на 300 пикселей вправо за 0,2 секунды ❸. Затем значение переменной `distance` уменьшается до `280` ❹, и функция `drag()` перемещает указатель на 280 пикселей

вниз ❺. В третий раз функция `drag()` перемещает указатель на -280 пикселей по горизонтали (т.е. влево) ❻, после чего значение переменной `distance` уменьшается до `260`, и последний вызов `drag()` перемещает указатель на 260 пикселей вверх. На каждой итерации указатель мыши перемещается вправо, вниз, влево и вверх, и значение переменной `distance` становится меньше, чем на предыдущей итерации. В результате указатель перемещается таким образом, что рисуется квадратная спираль.

Аналогичную спираль можно было бы нарисовать и вручную (вернее, с помощью мыши), но для создания ровных линий вам пришлось бы работать очень медленно. Модуль PyAutoGUI способен выполнить эту работу за считаные секунды!

Примечание

На момент написания книги модуль PyAutoGUI не мог отправлять щелчки мыши и нажатия клавиш определенным программам, таким как антивирусные приложения (в них предусмотрена защита от злонамеренных действий вирусов) или видеоигры в Windows (они по-другому взаимодействуют с клавиатурой и мышью). Обратитесь к онлайн-документации на сайте `https://pyautogui.readthedocs.io/`*, чтобы узнать, были ли добавлены эти возможности.*

Прокрутка

Также в модуле PyAutoGUI имеется функция `scroll()`, целочисленный аргумент которой определяет величину прокрутки вверх или вниз. Единицы измерения будут разными, в зависимости от операционной системы и приложения, поэтому в каждой конкретной ситуации придется экспериментировать. Прокрутка выполняется в текущей позиции указателя мыши. Положительное значение аргумента означает прокрутку вверх, отрицательное – вниз. Выполните в интерактивной оболочке следующую инструкцию, предварительно расположив указатель мыши в окне редактора Mu:

```
>>> pyautogui.scroll(200)
```

Вы увидите, как содержимое окна Mu прокрутится вверх (при условии, что указатель мыши находится над текстовым полем, которое допускает прокрутку).

Планирование перемещений указателя

Одна из трудностей написания программы, которая будет автоматизировать щелчки мышью, – определение координат *x* и *y* области экрана,

в которой требуется выполнить щелчок. В этом случае на помощь придет функция `pyautogui.mouseInfo()`.

Функция `pyautogui.mouseInfo()` вызывается из интерактивной оболочки, а не из программы, и запускает графическую утилиту MouseInfo, которая входит в состав пакета PyAutoGUI (рис. 20.3).

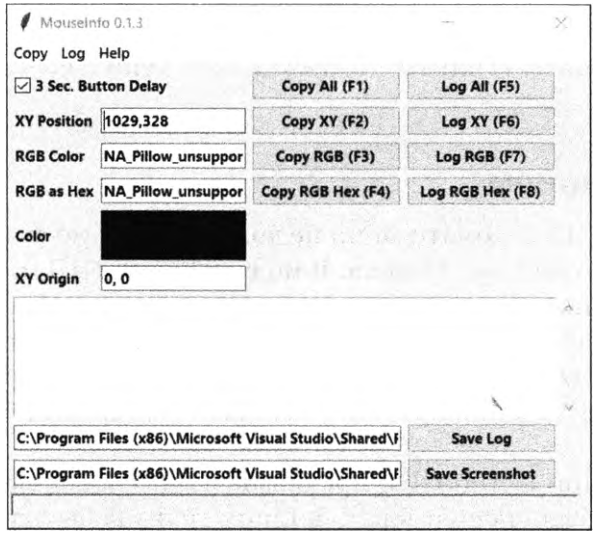


Рис. 20.3. Окно утилиты MouseInfo

Введите в интерактивной оболочке следующие инструкции.

```
>>> import pyautogui
>>> pyautogui.mouseInfo()
```

В результате появится окно MouseInfo, в котором содержится информация о текущем положении указателя мыши, а также цвете нижележащего пикселя в виде кортежа RGB и шестнадцатеричного значения. Образец цвета будет показан в поле Color.

Чтобы записать информацию о координатах или пикселях, щелкните на одной из восьми кнопок Copy или Log. Кнопки Copy All, Copy XY, Copy RGB и Copy RGB Hex копируют соответствующую информацию в буфер обмена. Кнопки Log All, Log XY, Log RGB и Log RGB Hex предназначены для вывода соответствующей информации в большое текстовое поле в окне. Чтобы сохранить текст, содержащийся в этом поле, щелкните на кнопке Save Log.

По умолчанию установлен флажок 3 Sec. Button Delay, что приводит к трехсекундной задержке между щелчком на кнопке Copy или Log и фактическим копированием или выводом данных. Это дает вам время, в течение которого можно щелкнуть на кнопке, а затем переместить указатель мыши

в нужное положение. Проще все же снять флажок, переместить указатель мыши и нажать одну из клавиш <F1>–<F8>. На каждой кнопке указано, какая клавиша ей соответствует.

Снимите флажок 3 Sec. Button Delay и перемещайте указатель мыши по экрану, нажимая кнопку <F6>. Вы увидите, что координаты *x* и *y* указателя отображаются в большом текстовом поле в окне MouseInfo. Эту информацию можно будет использовать в сценариях PyAutoGUI.

Документация к утилите MouseInfo доступна на сайте `https://mouseinfo.readthedocs.io/`.

Работа с экраном

Программы GUI-автоматизации не должны вслепую выполнять щелчки и нажимать виртуальные клавиш. В модуле PyAutoGUI имеются средства создания экранных снимков, позволяющие получить файл изображения на основе текущего содержимого экрана. Эти функции также могут возвращать объект `Image` библиотеки Pillow, соответствующий текущему виду экрана. Установите модуль `pillow` (см. главу 19), прежде чем продолжить чтение раздела.

Чтобы функции PyAutoGUI, предназначенные для получения снимков экрана, можно было использовать в Linux, должна быть установлена утилита `scrot`. Для этого выполните в окне Terminal команду **`sudo apt-get install scrot`**. В Windows или macOS этого не требуется.

Получение снимка экрана

Снимок экрана создается с помощью функции `pyautogui.screenshot()`. Введите в интерактивной оболочке следующие инструкции.

```
>>> import pyautogui
>>> im = pyautogui.screenshot()
```

Переменная `im` будет содержать объект `Image` экранного снимка. Теперь для нее можно вызывать методы объекта `Image`, как и при работе с любыми другими изображениями (см. главу 19).

Анализ снимка экрана

Предположим, в программе GUI-автоматизации требуется выполнить щелчок на серой кнопке. Прежде чем вызвать функцию `click()`, можно получить снимок экрана и проверить цвет текущего пикселя. Если он не серый, значит, что-то пошло не так. Возможно, окно было перемещено или же кнопка перекрыта всплывающим диалоговым окном. В таком случае

щелчок в неподходящем месте имел бы непредсказуемые последствия. Вместо этого программа может предпринять другие действия.

Чтобы получить RGB-значение цвета экранного пикселя, воспользуйтесь функцией `pixel()`. Введите в интерактивной оболочке следующие инструкции.

```
>>> import pyautogui
>>> pyautogui.pixel((0, 0))
(176, 176, 175)
>>> pyautogui.pixel((50, 200))
(130, 135, 144)
```

Функция `pixel()` получает кортеж координат, например `(0, 0)` или `(50, 200)`, и сообщает цвет соответствующего пикселя. Значение, возвращаемое функцией `pixel()`, — это RGB-кортеж из трех целых чисел для красной, зеленой и синей составляющих цвета. (Четвертого значения для альфа-составляющей нет, поскольку снимки экрана полностью непрозрачны.)

Функция `pixelMatchesColor()` возвращает значение `True`, если цвет пикселя с указанными экранными координатами *x* и *y* совпадает с заданным цветом. Ее первый и второй аргументы — это целые числа, соответствующие координатам *x* и *y*; третий аргумент — это кортеж из трех целых чисел, задающих RGB-цвет, которому должен соответствовать экранный пиксель. Введите в интерактивной оболочке следующие инструкции.

```
  >>> import pyautogui
❶ >>> pyautogui.pixel((50, 200))
  (130, 135, 144)
❷ >>> pyautogui.pixelMatchesColor(50, 200, (130, 135, 144))
  True
❸ >>> pyautogui.pixelMatchesColor(50, 200, (255, 135, 144))
  False
```

После того как функция `pixel()` вернет RGB-кортеж, соответствующий цвету пикселя с заданными координатами ❶, передайте эти координаты и RGB-кортеж функции `pixelMatchesColor()` ❷, которая вернет значение `True`. Затем измените RGB-кортеж и вновь вызовите функцию `pixelMatchesColor()` для тех же координат ❸. На этот раз функция вернет `False`. Эту функцию полезно вызывать всякий раз перед тем, как программа GUI-автоматизации собирается вызвать функцию `click()`. Учтите, что совпадение цветов должно быть *точным*. Достаточно минимального отличия, например `(255, 255, 254)` вместо `(255, 255, 255)`, и функция `pixelMatchesColor()` вернет значение `False`.

Распознавание изображений

Но как быть, если вам неизвестно заранее, в каком месте экрана программа должна выполнить виртуальный щелчок? В этом случае можно прибегнуть к распознаванию изображений. Передайте модулю PyAutoGUI изображение, на котором следует выполнить щелчок, и он самостоятельно определит нужные координаты.

Например, если у вас есть изображение кнопки Отправить, сохраненное в файле *submit.png*, то функция `locateOnScreen()` вернет координаты данного изображения на экране. Чтобы увидеть, как это работает, попробуйте создать снимок небольшой области экрана, сохраните его, а затем введите в интерактивной оболочке следующие инструкции, заменив `'submit.png'` именем своего файла.

```
>>> import pyautogui
>>> b = pyautogui.locateOnScreen('submit.png')
>>> b
Box(left=643, top=745, width=70, height=29)
>>> b[0]
643
>>> b.left
643
```

Объект `Box` — это именованный кортеж, возвращаемый функцией `locateOnScreen()`. Он содержит координату *x* левого края, координату *y* верхнего края, а также ширину и высоту первого найденного изображения. В вашем случае возвращаемые значения будут другими.

Если соответствующее изображение не найдено, функция `locateOnScreen()` возвращает значение `None`. Учтите, что изображение на экране должно в точности соответствовать заданному файлу. Если изображения отличаются хоть на пиксель, функция `locateOnScreen()` сгенерирует исключение `ImageNotFoundException`. Кроме того, если вы поменяете разрешение или масштаб экрана, то изображения с предыдущих экранных снимков уже не будут соответствовать изображениям на текущем экране (рис. 20.4).

Если изображение встречается в нескольких местах экрана, то функция `locateAllOnScreen()` вернет объект `Generator`. Описание объектов такого типа выходит за рамки книги, но их можно передать функции `list()`, которая вернет список кортежей, состоящих из четырех целых чисел. Каждому найденному изображению соответствует один кортеж. Введите в интерактивной оболочке следующую инструкцию (только замените `'submit.png'` именем своего файла).

```
>>> list(pyautogui.locateAllOnScreen('submit.png'))
[(643, 745, 70, 29), (1007, 801, 70, 29)]
```

Рис. 20.4. Настройки экрана в Windows (слева) и macOS (справа)

Каждый из кортежей с четырьмя числами представляет прямоугольную область экрана. В данном случае изображение найдено в двух местах. В противном случае вы получите список, содержащий всего один кортеж.

Как только в вашем распоряжении окажется кортеж требуемого изображения, щелкните в центре этой области, передав функции `click()` весь кортеж. Введите в интерактивной оболочке следующую инструкцию:

```
>>> pyautogui.click((643, 745, 70, 29))
```

Можно также передать функции `click()` непосредственно имя файла изображения:

```
>>> pyautogui.click('submit.png')
```

Функции `moveTo()` и `dragTo()` также допускают в качестве аргумента имя файла изображения. Помните: если функция `locateOnScreen()` не может найти изображение на экране, она генерирует исключение, поэтому вызов должен делаться из блока `try`.

```
try:
    location = pyautogui.locateOnScreen('submit.png')
except:
    print('Изображение не найдено.')
```

В отсутствие инструкций `try` и `except` неперехваченное исключение приведет к аварийному завершению программы.

Получение информации об окне

Распознавание изображений — чрезвычайно ненадежный способ находить объекты на экране. Если хотя бы один пиксель будет иметь другой

цвет, функция `pyautogui.locateOnScreen()` не найдет изображение. Чтобы узнать, где находится конкретное окно на экране, проще использовать функции PyAutoGUI для работы с окнами.

Примечание

На момент написания книги в версии 0.9.46 оконные функции PyAutoGUI работают только в Windows, но не в macOS или Linux. Эти функции получены благодаря включению в пакет PyAutoGUI модуля PyGetWindow.

Определение активного окна

Активным считается окно, которое находится на переднем плане и реагирует на ввод данных с клавиатуры. Если в настоящий момент вы вводите код в редакторе Mu, то активным будет окно редактора Mu. Среди всех открытых окон в конкретный момент времени активным будет только одно окно.

Вызовите в интерактивной оболочке функцию `pyautogui.getActiveWindow()`, чтобы получить объект `Window` (в Windows он имеет тип данных `Win32Window`). Атрибуты этого объекта `Window` описывают размер, положение и заголовок окна.

- **`left`**, **`right`**, **`top`**, **`bottom`**. По одному целому числу для координаты *x* или *y* соответствующей стороны окна.
- **`topleft`**, **`topright`**, **`bottomleft`**, **`bottomright`**. Именованный кортеж из двух целых чисел для координат (*x*, *y*) соответствующего угла окна.
- **`midleft`**, **`midright`**, **`midleft`**, **`midright`**. Именованный кортеж из двух целых чисел для координаты (*x*, *y*) середины соответствующей стороны окна.
- **`width`**, **`height`**. Целочисленные размеры окна в пикселях.
- **`size`**. Именованный кортеж из двух целых чисел, определяющих ширину и высоту окна.
- **`area`**. Целое число, определяющее площадь окна в пикселях.
- **`center`**. Именованный кортеж из двух целых чисел для координаты (*x*, *y*) центра окна.
- **`centerx`**, **`centery`**. По одному целому числу для координаты *x* или *y* центра окна.
- **`box`**. Именованный кортеж из четырех целых чисел для четырех измерений окна (левая сторона, верхняя сторона, ширина и высота).
- **`title`**. Текст в строке заголовка окна.

Например, чтобы получить информацию о положении, размере и заголовке окна из объекта `window`, введите в интерактивной оболочке следующие инструкции.

```
>>> import pyautogui
>>> fw = pyautogui.getActiveWindow()
>>> fw
Win32Window(hWnd=2034368)
>>> str(fw)
'<Win32Window left="500", top="300", width="2070", height="1208",
title="Mu 1.0.1 - test1.py">'
>>> fw.title
'Mu 1.0.1 - test1.py'
>>> fw.size
Size(width=2070, height=1208)
>>> fw.left, fw.top, fw.right, fw.bottom
(500, 300, 2570, 1508)
>>> fw.topleft
Point(x=500, y=300)
>>> fw.area
2500560
>>> pyautogui.click(fw.left + 10, fw.top + 20)
```

Эти атрибуты можно использовать для вычисления точных координат в окне. Если вы знаете, что кнопка, на которой вы хотите щелкнуть, всегда находится в 10 пикселях справа и в 20 пикселях ниже от левого верхнего угла окна, а левый верхний угол окна имеет координаты `(300, 500)`, то функция `pyautogui.click(310, 520)` (или `pyautogui.click(fw.left + 10, fw.top + 20)`, если переменная `fw` содержит объект `Window` для данного окна) выполнит щелчок на этой кнопке. Таким образом, вам больше не придется полагаться на более медленную и менее надежную функцию `locateOnScreen()` для поиска нужной кнопки.

Другие способы получения информации об окнах

Функция `getActiveWindow()` возвращает информацию об окне, которое было активным на момент вызова функции. Для работы с другими окнами вам понадобятся другие функции.

Следующие четыре функции возвращают список объектов `Window`. Если никаких окон найти не удается, то возвращается пустой список.

- **`pyautogui.getAllWindows()`**. Возвращает список объектов `Window` для каждого видимого окна на экране.
- **`pyautogui.getWindowsAt(x, y)`**. Возвращает список объектов `Window` для каждого видимого окна, которое включает точку `(x, y)`.

- **pyautogui.getWindowsWithTitle(title)**. Возвращает список объектов Window для каждого видимого окна, которое содержит строку title в строке заголовка.
- **pyautogui.getActiveWindow()**. Возвращает объект Window для активного окна, которое в данный момент имеет фокус ввода.

В модуле PyAutoGUI есть также функция pyautogui.getAllTitles(), которая возвращает список заголовков всех видимых окон.

Манипулирование окнами

Атрибуты окон позволяют не только узнать размер и положение окна. С их помощью можно также масштабировать и перемещать окна. Введите в интерактивной оболочке следующие инструкции.

```
>>> import pyautogui
>>> fw = pyautogui.getActiveWindow()
❶ >>> fw.width    # определение текущей ширины окна
851
❷ >>> fw.topleft    # определение текущей позиции окна
Point(x=94, y=113)
❸ >>> fw.width = 600    # изменение ширины окна
❹ >>> fw.topleft = (300, 300)    # перемещение окна
```

Сначала мы используем атрибуты объекта Window, чтобы получить информацию о размере ❶ и положении окна ❷. После выполнения следующих двух инструкций в интерактивной оболочке окно должно переместиться ❹ и стать уже ❸, как показано на рис. 20.5.

Существует также возможность определять и менять состояние окна (свернутое, развернутое, активное). Введите в интерактивной оболочке следующие инструкции.

```
>>> import pyautogui
>>> fw = pyautogui.getActiveWindow()
❶ >>> fw.isMaximized    # возвращает True, если окно развернуто
False
❷ >>> fw.isMinimized    # возвращает True, если окно свернуто
False
❸ >>> fw.isActive    # возвращает True, если окно активно
True
❹ >>> fw.maximize()    # разворачивает окно на весь экран
>>> fw.isMaximized
True
❺ >>> fw.restore()    # отмена сворачивания/разворачивания
❻ >>> fw.minimize()    # сворачивает окно
>>> import time
>>> # Подождать 5 секунд для активизации другого окна
❼ >>> time.sleep(5); fw.activate()
❽ >>> fw.close()    # закрывает активное окно
```

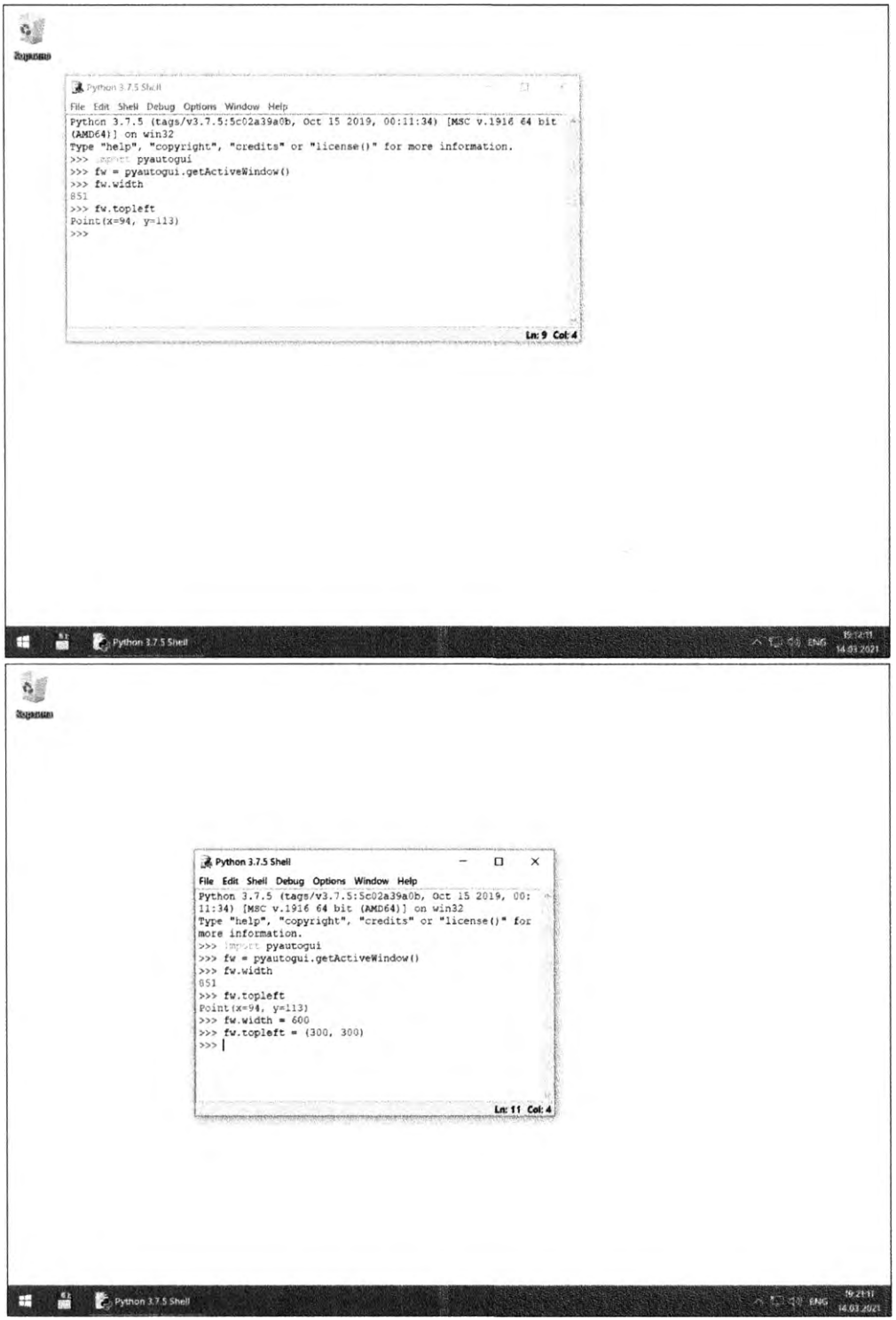


Рис. 20.5. Окно интерактивной оболочки до (сверху) и после (снизу) изменения атрибутов объекта `Window`

Атрибуты `isMaximized` ❶, `isMinimized` ❷ и `isActive` ❸ содержат булевы значения, определяющие, находится ли окно в данный момент в соответствующем состоянии. Методы `maximize()` ❹, `minimize()` ❻, `activate()` ❼ и `restore()` ❺ изменяют состояние окна. После того как вы развернете или свернете окно с помощью метода `maximize()` или `minimize()`, метод `restore()` восстановит окно в его прежнем размере и положении.

Метод `close()` ❽ закрывает окно. Будьте осторожны с этим методом, так как он может отменить любые окна сообщений с просьбой сохранить работу перед выходом из приложения.

С документацией по оконным функциям PyAutoGUI можно ознакомиться на сайте `https://pyautogui.readthedocs.io/`. Эти функции можно использовать отдельно от PyAutoGUI, совместно с модулем PyGetWindow, документация к которому доступна на сайте `https://pygetwindow.readthedocs.io/`.

Управление клавиатурой

Модуль PyAutoGUI также содержит функции, позволяющие посылать компьютеру виртуальные нажатия клавиш, что дает вам возможность заполнять формы или вводить текст в приложениях.

Отправка строки, набранной на виртуальной клавиатуре

Функция `pyautogui.write()` генерирует виртуальные нажатия клавиш. Какие последствия это будет иметь, зависит от того, какое окно в данный момент активно и в каком поле находится фокус ввода. Чтобы гарантировать нахождение фокуса в нужном поле, можно предварительно выполнить в нем виртуальный щелчок мышью.

В качестве примера используем Python для автоматического ввода текста `'Hello world!'` в программе Блокнот. Прежде всего откройте новое окно и расположите его в левом верхнем углу экрана, чтобы в него можно было переместить фокус ввода, выполнив с помощью модуля PyAutoGUI виртуальный щелчок мышью в подходящем месте. Затем введите в интерактивной оболочке следующие инструкции:

```
>>> pyautogui.click(100, 100); pyautogui.write('Hello, world!')
```

Обратите внимание на размещение двух инструкций, разделенных точкой с запятой, в одной строке, что предотвращает появление дополнительного приглашения интерпретатора. Это позволяет избежать случайного перемещения фокуса ввода в новое окно между вызовами функций `click()` и `write()`, что внесло бы путаницу.

Сначала Python выполняет виртуальный щелчок мышью в точке экрана с координатами `(100, 100)`, что приводит к активизации окна программы Блокнот и перемещению в него фокуса ввода. Функция `write()` посылает текст `'Hello, world!'` в это окно (рис. 20.6). Теперь вы знаете, как заставить компьютер набирать текст вместо вас!

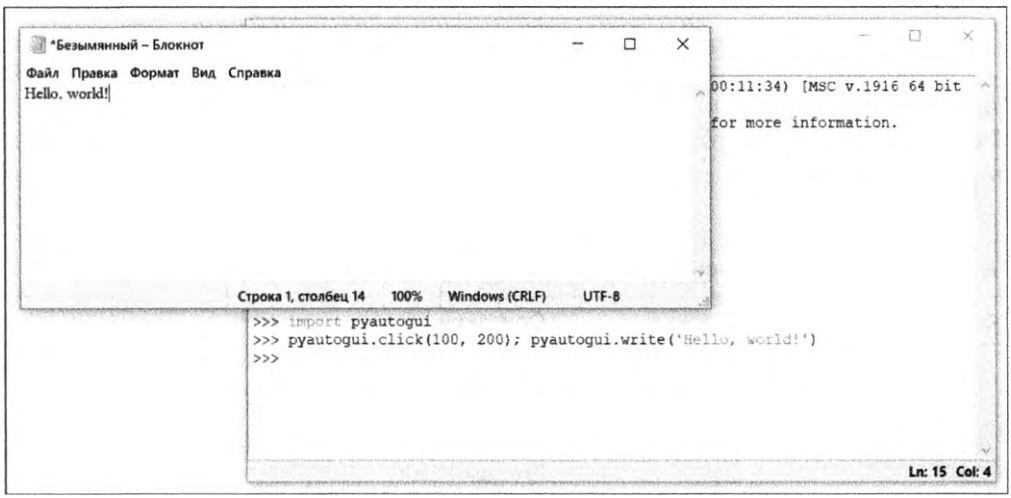


Рис. 20.6. Использование модуля PyAutoGUI для выполнения щелчка в окне программы Блокнот и ввода в нем текста `'Hello, world!'`

По умолчанию функция `write()` мгновенно вводит всю строку. Но можно добавить небольшую задержку между символами, передав функции необязательный второй аргумент в виде целого или вещественного числа, задающего длительность паузы в секундах. Например, функция `pyautogui.write('Hello, world!', 0.25)` будет выжидать четверть секунды после ввода буквы `'H'`, затем еще четверть секунды после ввода буквы `'e'` и т.д. Такой постепенный ввод текста в стиле пишущей машинки может оказаться полезным в случае медленных приложений, которые не в состоянии обрабатывать нажатия клавиш с той же скоростью, с какой их генерирует модуль PyAutoGUI.

Для таких символов, как `A` или `!`, PyAutoGUI автоматически имитирует одновременное нажатие клавиши <Shift>.

Названия клавиш

Не все клавиши могут быть представлены одиночными текстовыми символами. Например, это невозможно сделать для клавиш <Shift> и <↑>. В модуле PyAutoGUI эти клавиши представляются короткими строковыми константами, например `'esc'` для клавиши <Esc> или `'enter'` для клавиши <Enter>.

Вместо одиночной строки функции `write()` можно передать список строковых констант, соответствующих специальным клавишам. Например, следующий вызов приведет к нажатию клавиши <A> и клавиши <B>, затем – двум нажатиям клавиши <←> и наконец – нажатиям клавиш <X> и <Y>:

```
>>> pyautogui.write(['a', 'b', 'left', 'left', 'X', 'Y'])
```

Поскольку нажатию клавиши <←> соответствует перемещение курсора на одну позицию влево, результатом станет вывод строки `'XYab'`. В табл. 20.1 приведены строковые константы PyAutoGUI, которые можно передавать функции `write()` для имитации нажатий любых комбинаций клавиш.

Таблица 20.1. Обозначения клавиш в модуле PyAutoGUI

Строковые обозначения клавиш	Описание
`'a'`, `'b'`, `'c'`, `'A'`, `'B'`, `'C'`, `'1'`, `'2'`, `'3'`, `'!'`, `'@'`, `'#'` и др.	Клавиши одиночных символов
`'enter'` (а также `'return'` или `'\n'`)	Клавиша <Enter>
`'esc'`	Клавиша <Esc>
`'shiftleft'`, `'shiftright'`	Левая и правая клавиши <Shift>
`'altleft'`, `'altright'`	Левая и правая клавиши <Alt>
`'ctrlleft'`, `'ctrlright'`	Левая и правая клавиши <Ctrl>
`'tab'` (или `'\t'`)	Клавиша <Tab>
`'backspace'`, `'delete'`	Клавиши <Backspace> и <Delete>
`'pageup'`, `'pagedown'`	Клавиши <Page Up> и <Page Down>
`'home'`, `'end'`	Клавиши <Home> и <End>
`'up'`, `'down'`, `'left'`, `'right'`	Клавиши <↑>, <↓>, <←> и <→>
`'f1'`, `'f2'`, `'f3'` и т.д.	Клавиши от <F1> до <F12>
`'volumemute'`, `'volumedown'`, `'volumeup'`	Клавиши отключения, уменьшения и увеличения громкости звука (на некоторых клавиатурах их нет, но операционная система все равно будет распознавать их виртуальные нажатия)
`'pause'`	Клавиша <Pause>
`'capslock'`, `'numlock'`, `'scrolllock'`	Клавиши <Caps Lock>, <Num Lock> и <Scroll Lock>
`'insert'`	Клавиша <Ins> или <Insert>
`'printscreen'`	Клавиша <Prtsc> или <Print Screen>
`'winleft'`, `'winright'`	Левая и правая клавиши <Win> (в Windows)
`'command'`	Клавиша <Command (⌘)> (в macOS)
`'option'`	Клавиша <Option> (в macOS)

Все строковые обозначения клавиш, поддерживаемые в PyAutoGUI, содержатся в списке `pyautogui.KEYBOARD_KEYS`. Обозначение `'shift'` соответствует левой клавише <Shift> и эквивалентно `'shiftleft'`. То же самое касается констант `'ctrl'`, `'alt'` и `'win'` — все они ссылаются на одноименные левые клавиши.

Нажатие и отпускание клавиш

Подобно функциям `mouseDown()` и `mouseUp()`, функции `pyautogui.keyDown()` и `pyautogui.keyUp()` посылают компьютеру сигналы виртуальных нажатий и отпусканий клавиш. В качестве аргумента этим функциям передается строковое обозначение клавиши (см. табл. 20.1). Для удобства в модуле PyAutoGUI предусмотрена функция `pyautogui.press()`, которая последовательно вызывает обе указанные функции, имитируя полный цикл нажатия клавиши.

Следующая инструкция выведет знак доллара (нажатие клавиши <4> при одновременном нажатии клавиши <Shift>).

```
>>> pyautogui.keyDown('shift'); pyautogui.press('4');
pyautogui.keyUp('shift')
$
```

Здесь имитируется нажатие клавиши <Shift>, нажатие (и отпускание) клавиши <4> и последующее отпускание клавиши <Shift>. Конечно, если требуется ввести строку в текстовое поле, то лучше воспользоваться функцией `write()`. Но для приложений, поддерживающих простые клавиатурные команды, функция `press()` станет более удачным решением.

Горячие клавиши

Горячие клавиши, или *клавиши быстрого вызова*, — это комбинации клавиш, предназначенные для активизации определенных функций приложения. Например, популярные комбинации для копирования выделенного текста в буфер обмена — <Ctrl+C> (Windows и Linux) и <⌘+C> (macOS). Пользователь должен нажать и удерживать клавишу <Ctrl>, а затем нажать клавишу <C> и после этого отпустить клавиши <C> и <Ctrl>. Чтобы проделать такое с помощью функций `keyDown()` и `keyUp()`, придется ввести следующее.

```
pyautogui.keyDown('ctrl')
pyautogui.keyDown('c')
pyautogui.keyUp('c')
pyautogui.keyUp('ctrl')
```

Получается достаточно громоздко. Вместо этого лучше воспользоваться функцией `pyautogui.hotkey()`, которая получает список клавиатурных констант, выполняет виртуальные нажатия клавиш в указанном порядке, а затем отпускает их в обратном порядке. Для комбинации клавиш <Ctrl+C> вызов будет выглядеть так:

```
pyautogui.hotkey('ctrl', 'c')
```

Эта функция особенно полезна в случае длинных клавиатурных комбинаций. Например, в приложении Word комбинация клавиш <Ctrl+Alt+Shift+S> приводит к отображению панели **Стили**. Вместо того чтобы выполнять восемь разных вызовов функций (четыре вызова `keyDown()` и четыре вызова `keyUp()`), достаточно вызвать функцию `hotkey('ctrl', 'alt', 'shift', 's')`.

Настройка собственных сценариев GUI-автоматизации

Сценарии автоматизации графического интерфейса — отличный способ автоматизировать рутинные задачи, но при их написании следует проявлять осторожность. Если окно находится не в том месте рабочего стола или неожиданно появляется какое-то всплывающее окно, есть риск, что сценарий начнет щелкать на неправильных экранных объектах. Вот несколько советов по настройке сценариев GUI-автоматизации.

- При каждом запуске сценария используйте одинаковое разрешение экрана, чтобы положение окон не менялось.
- Окно приложения, на котором щелкает сценарий, должно быть развернуто, чтобы его кнопки и меню находились в одном и том же месте при каждом запуске сценария.
- Добавьте достаточные паузы для загрузки контента. Сценарий не должен начинать генерировать щелчки мыши еще до того, как приложение будет готово их обработать.
- Используйте функцию `locateOnScreen()`, чтобы находить кнопки и меню для щелчков, а не полагаться на конкретные экранные координаты. Если сценарий не может найти нужный экранный элемент, остановите программу, чтобы она не щелкала где попало.
- Используйте функцию `getWindowsWithTitle()` для проверки существования окна, в котором должен быть выполнен щелчок. Вызовите метод `activate()`, чтобы переместить это окно на передний план.
- Используйте модуль `logging` (см. главу 11), чтобы сохранить журнальный файл, в котором фиксируются действия сценария. Если вдруг

придется остановить сценарий на полпути, вы сможете изменить его так, чтобы он продолжил работу с того места, на котором остановился.

- Добавьте как можно больше проверок в сценарий. Подумайте о том, как он поведет себя, если вдруг появится неожиданное всплывающее окно или компьютер отключится от Интернета.
- Проконтролируйте сценарий при первом запуске, чтобы убедиться в корректности его работы.

Имеет смысл добавить в начало сценария паузу, чтобы пользователь мог настроить окно, с которым будет работать сценарий. В PyAutoGUI есть функция `sleep()`, аналог функции `time.sleep()` (она освобождает вас от необходимости добавлять строку `import time` в сценарии). Имеется также функция `countdown()`, которая выводит числа в обратном порядке в качестве визуальной индикации времени, оставшегося до возобновления работы. Введите в интерактивной оболочке следующие инструкции.

```
>>> import pyautogui
>>> pyautogui.sleep(3)     # приостанавливает программу на 3 секунды
>>> pyautogui.countdown(10)     # обратный отчет в течение 10 секунд
10 9 8 7 6 5 4 3 2 1
>>> print('Начало через ', end=''); pyautogui.countdown(3)
Начало через 3 2 1
```

Эти советы помогут сделать ваши сценарии GUI-автоматизации более удобными в использовании и способными реагировать на возникновение непредвиденных обстоятельств.

Обзор функций PyAutoGUI

Поскольку в главе рассматривалось много разных функций, ниже дан их краткий обзор.

- **`moveTo(x, y)`**. Перемещает указатель мыши в точку экрана с координатами *x* и *y*.
- **`move(xOffset, yOffset)`**. Перемещает указатель мыши на заданное расстояние относительно его текущей позиции.
- **`dragTo(x, y)`**. Перемещает указатель мыши в точку экрана с координатами *x* и *y*, удерживая нажатой ее левую кнопку.
- **`drag(xOffset, yOffset)`**. Перемещает указатель мыши на заданное расстояние относительно его текущей позиции, удерживая нажатой ее левую кнопку.
- **`click(x, y, button)`**. Имитирует щелчок (по умолчанию левой кнопкой мыши) в точке экрана с координатами *x* и *y*.

- **`rightClick()`**. Имитирует щелчок правой кнопкой мыши.
- **`middleClick()`**. Имитирует щелчок средней кнопкой мыши.
- **`doubleClick()`**. Имитирует двойной щелчок левой кнопкой мыши.
- **`mouseDown(x, y, button)`**. Имитирует нажатие указанной кнопки мыши в точке экрана с координатами *x* и *y*.
- **`mouseUp(x, y, button)`**. Имитирует отпускание указанной кнопки мыши в точке экрана с координатами *x* и *y*.
- **`scroll(units)`**. Имитирует прокручивание колесика мыши. Положительному значению аргумента соответствует прокрутка вверх, отрицательному — прокрутка вниз.
- **`write(message)`**. Вводит заданную строку символов.
- **`write([key1, key2, key3])`**. Имитирует нажатие указанных клавиш.
- **`press(key)`**. Имитирует нажатие и отпускание указанной клавиши.
- **`keyDown(key)`**. Имитирует нажатие указанной клавиши.
- **`keyUp(key)`**. Имитирует отпускание указанной клавиши.
- **`hotkey([key1, key2, key3])`**. Имитирует нажатие указанных клавиш в заданном порядке с последующим их отпусканием в обратном порядке.
- **`screenshot()`**. Возвращает снимок экрана в виде объекта `Image`.
- **`getActiveWindow()`**, **`getAllWindows()`**, **`getWindowsAt()`** и **`getWindowsWithTitle()`**. Эти функции возвращают объекты `Window`, с помощью которых можно масштабировать и перемещать окна приложений на экране.
- **`getAllTitles()`**. Возвращает список заголовков всех окон на экране.

Проект: автоматическое заполнение формы

Пожалуй, самая надоедливая из всех рутинных задач — это заполнение форм. И сейчас, в последнем проекте книги, мы наконец займемся ее решением. Предположим, что в электронной таблице хранится огромный массив данных, и вам предстоит кропотливая работа по переносу этих данных в форму другого приложения, а готового интерфейса импорта электронных таблиц у приложения нет. Неужели придется часами щелкать мышью и вводить данные вручную? Конечно же нет! Раз вы дочитали книгу почти до конца, то уже знаете, что *всегда* есть способ автоматизировать рутинную задачу.

В этом проекте будет использоваться форма Google Docs, доступная на сайте `https://autbor.com/form` (рис. 20.7).

Капчи и компьютерная этика

Капча (от CAPTCHA — англ. "Completely Automated Public Turing test to tell Computers and Humans Apart" — полностью автоматизированный публичный тест Тьюринга для различения компьютеров и людей) — это небольшой тест, при выполнении которого нужно ввести буквы, показанные на искаженном изображении, или щелкнуть на фотографиях каких-нибудь пожарных гидрантов. Люди проходят такие тесты легко (хоть и без особого удовольствия), а вот компьютерам сделать это почти невозможно. Прочитав данную главу, вы знаете, что можно легко написать сценарий, который, к примеру, зарегистрируется в миллионе бесплатных учетных записей электронной почты и начнет бомбардировать пользователей спамом. Капчи препятствуют этому, добавляя задание, выполнить которое может только человек.

Но не все сайты используют капчи, что делает их уязвимыми к злоупотреблениям со стороны неэтичных программистов. Умение программировать открывает перед вами огромные перспективы, и возникает соблазн использовать данное умение для извлечения выгоды или даже просто для того, чтобы похвастаться перед друзьями. Помните: открытая дверь не оправдывает вора. Точно так же ответственность за действия программы ложится на вас, программиста. Нет ничего умного в том, чтобы обойти защиту системы для нанесения вреда, вторжения в частную жизнь или получения несправедливого преимущества. Надеюсь, мои усилия по написанию книги помогут вам начать создавать более продуктивные программы, а не программы, используемые для взлома.

Вот что должна сделать данная программа:

1) щелкнуть на первом текстовом поле формы;
2) ввести информацию в каждое поле формы;
3) щелкнуть на кнопке Отправить;
4) повторить весь процесс для следующего набора данных.

Это означает, что программа должна будет выполнять следующие операции:

1) вызывать функцию `pyautogui.click()` для выполнения щелчков на элементах формы и кнопке Отправить;
2) вызывать функцию `pyautogui.write()` для ввода текста в соответствующие поля;
3) обрабатывать исключение `KeyboardInterrupt`, чтобы пользователь мог выйти из программы, нажав комбинацию клавиш <Ctrl+C>.

Откройте в файловом редакторе новое окно и сохраните программу в файле *formFiller.py*.

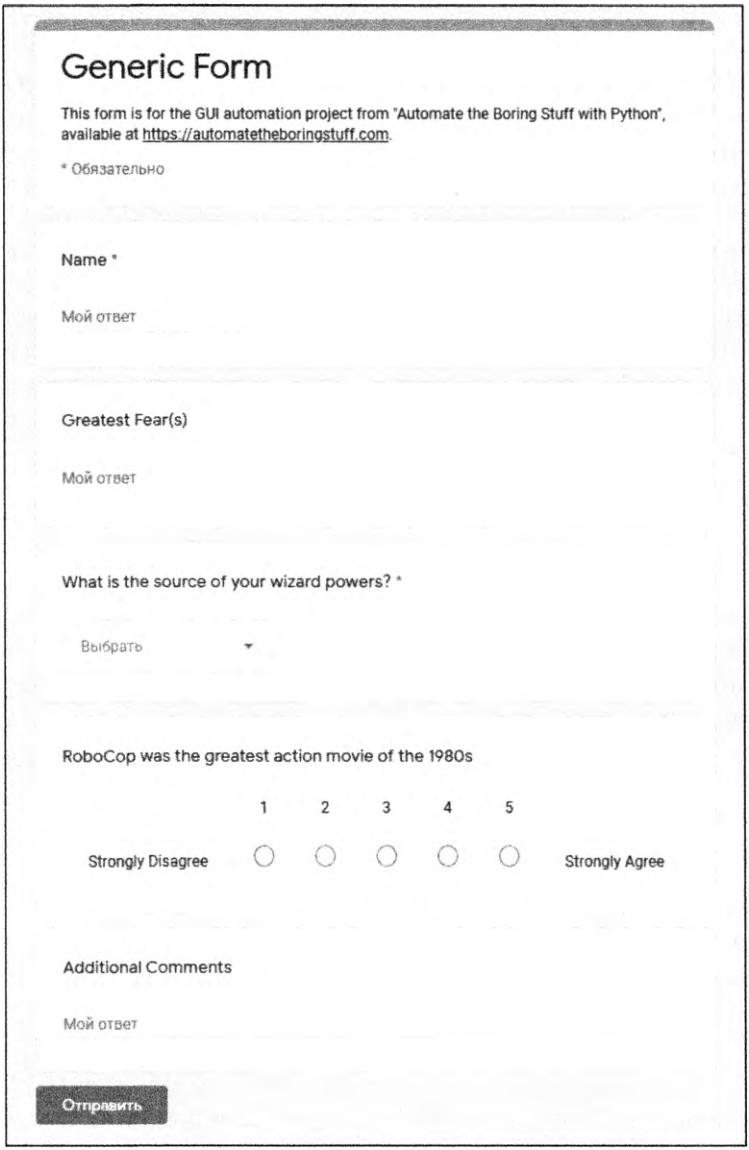


Рис. 20.7. Форма, используемая в данном проекте

Шаг 1. Составление плана действий

Прежде чем приступить к написанию кода, определим точную последовательность нажатий клавиш и щелчков мышью для однократного прохода по форме. Утилита MouseInfo, запускаемая с помощью функции `pyautogui.mouseInfo()`, поможет определить конкретные координаты указателя мыши. Единственное, что нам необходимо узнать, — это координаты

первого текстового поля. После щелчка на нем достаточно нажать клавишу <Tab> для перехода к следующему полю. Таким образом, нам нет необходимости определять координаты *x* и *y* каждого поля формы.

Ниже приведена пошаговая процедура ввода данных в поля формы.

1. Поместить фокус ввода в поле Name (Имя), чтобы можно было ввести в него текст.
2. Ввести имя и нажать клавишу <Tab>.
3. Указать свой самый большой страх (или страхи) в поле Greatest Fear(s) и нажать клавишу <Tab>.
4. Нажать клавишу <↓> необходимое количество раз для выбора источника "магической силы", а затем нажать клавишу <Tab>[1]:
   - 1 – волшебная палочка (Wand),
   - 2 – амулет (Amulet),
   - 3 – хрустальный шар (Crystal Ball)
   - 4 – деньги (Money).
5. Нажать клавишу <→> необходимое количество раз для выбора варианта ответа на вопрос о Робокопе. Ее следует нажать один раз для выбора варианта 2, два раза – для выбора варианта 3, три раза – для выбора варианта 4 и четыре раза – для выбора варианта 5. Для выбора варианта 1 достаточно нажать пробел. Далее следует нажать клавишу <Tab>.
6. Ввести дополнительные комментарии и нажать клавишу <Tab>.
7. Нажать клавишу <Enter> для "щелчка" на кнопке Отправить.
8. После отправки формы браузер перейдет на страницу, на которой вам нужно будет щелкнуть на ссылке для возврата на страницу формы.

Учтите, что в разных браузерах и разных операционных системах порядок действий может оказаться немного другим, поэтому обязательно проверьте, как работают используемые комбинации клавиш в вашем случае, прежде чем запускать программу.

Шаг 2. Настройка координат

Загрузите форму в браузер, перейдя на сайт `https://autbor.com/form` (см. рис. 20.7), и введите в программу следующий код.

[1] Учтите, что в macOS клавишу <↓> необходимо нажимать на один раз больше для каждой опции. Возможно, в некоторых браузерах потребуется также нажать клавишу <Enter>.

```
#! python3
# formFiller.py - автоматическое заполнение формы
import pyautogui, time

# СДЕЛАТЬ: дать пользователю возможность прервать выполнение сценария

# СДЕЛАТЬ: дождаться загрузки страницы с формой

# СДЕЛАТЬ: заполнить поле "Name"

# СДЕЛАТЬ: заполнить поле "Greatest Fear(s)"

# СДЕЛАТЬ: заполнить поле "Source of Wizard Powers"

# СДЕЛАТЬ: заполнить поле "RoboCop"

# СДЕЛАТЬ: заполнить поле "Additional Comments"

# СДЕЛАТЬ: щелкнуть на кнопке "Отправить"

# СДЕЛАТЬ: дождаться загрузки страницы

# СДЕЛАТЬ: щелкнуть на ссылке "Отправить еще один ответ"
```

Теперь нам нужны данные, которые требуется ввести в форму. На практике эти данные будут браться из электронной таблицы, текстового файла или веб-сайта, и для их загрузки понадобится дополнительный код. Но в рассматриваемом проекте мы просто закодируем их в переменной. Добавьте в программу следующий код.

```
#! python3
# formFiller.py - автоматическое заполнение формы

-- Опущено --

formData = [{'name': 'Alice', 'fear': 'eavesdroppers',
             'source': 'wand', 'robocop': 4,
             'comments': 'Tell Bob I said hi.'},
            {'name': 'Bob', 'fear': 'bees', 'source': 'amulet',
             'robocop': 4, 'comments': 'n/a'},
            {'name': 'Carol', 'fear': 'puppets',
             'source': 'crystal ball', 'robocop': 1,
             'comments': 'Please take the puppets out of the
                          break room.'},
            {'name': 'Alex Murphy', 'fear': 'ED-209',
             'source': 'money', 'robocop': 5,
             'comments': 'Protect the innocent. Serve the public
                          trust. Uphold the law.'},
           ]

-- Опущено --
```

Список formData содержит по одному словарю для четырех разных имен. В каждом словаре ключами служат названия текстовых полей, а значениями — ответы на соответствующие вопросы. Последний элемент настройки — конфигурирование переменной PAUSE, с помощью которой задается пауза длительностью полсекунды после каждого вызова функции. Кроме того, нужно напомнить пользователю о необходимости щелкнуть на окне браузера, чтобы сделать его активным. Добавьте в программу следующий код.

```
pyautogui.PAUSE = 0.5
print('Убедитесь, что окно браузера активно и форма загружена!')
```

Шаг 3. Начало ввода данных

Виртуальный ввод данных в текстовые поля будет осуществляться в цикле for, выполняющем итерации по словарям, которые содержатся в списке formData. Значения словаря будут передаваться соответствующим функциям модуля PyAutoGUI.

Добавьте в программу следующий код.

```
  #! python3
  # formFiller.py - автоматическое заполнение формы

-- Опущено --

  for person in formData:
      # Дать пользователю возможность прервать выполнение сценария
      print('>>> 5-СЕКУНДНАЯ ПАУЗА ДЛЯ НАЖАТИЯ <CTRL+C> <<<')
❶     time.sleep(5)

-- Опущено --
```

В качестве дополнительной меры безопасности в сценарии предусмотрена пятисекундная пауза ❶, дающая пользователю возможность нажать комбинацию клавиш <Ctrl+C> (или переместить указатель мыши в левый верхний угол экрана, после чего будет сгенерировано исключение FailSafeException), чтобы прекратить работу программы, если что-то пойдет не так. Добавьте в программу следующий код.

```
  #! python3
  # formFiller.py - автоматическое заполнение формы

-- Опущено --

❶     print('Вводится информация о %s...' % (person['name']))
❷     pyautogui.write(['\t', '\t'])
```

```
    # Заполнение поля "Name"
❸   pyautogui.write(person['name'] + '\t')

    # Заполнение поля "Greatest Fear(s)"
❹   pyautogui.write(person['fear'] + '\t')

-- Опущено --
```

Чтобы проинформировать пользователя о ходе выполнения программы, мы добавили вызов функции `print()`, которая отображает текущий статус в окне терминала ❶.

Поскольку на данный момент форма уже должна быть загружена, мы вызываем функцию `pyautogui.write(['\t', '\t'])`, которая генерирует два нажатия клавиши <Tab>, перемещая фокус ввода в поле Name ❷. Далее мы снова вызываем функцию `write()` для ввода строки `person['name']` ❸. Символ `'\t'`, добавляемый в конец строки, имитирует нажатие клавиши <Tab>, в результате чего фокус ввода перемещается в следующее поле: Greatest Fear(s). После этого мы еще раз вызываем функцию `write()` для ввода строки `person['fear']` и переходим к следующему полю формы ❹.

Шаг 4. Обработка списков выбора и переключателей

Обработка раскрывающегося списка, содержащего варианты источника "магической силы", и кнопок-переключателей, предлагающих варианты ответа на вопрос о фильме *Робокоп*, представляет собой более сложную задачу, чем ввод строки в текстовое поле. Если выбирать варианты с помощью мыши, то придется определять координаты *x* и *y* каждого из элементов управления. Проще использовать для этой цели клавиши управления курсором.

Добавьте в программу следующий код.

```
#! python3
# formFiller.py - автоматическое заполнение формы

-- Опущено --

    # Заполнение поля "Source of Wizard Powers"
❶   if person['source'] == 'wand':
❷       pyautogui.write(['down', '\t'], 0.5)
    elif person['source'] == 'amulet':
        pyautogui.write(['down', 'down', '\t'], 0.5)
    elif person['source'] == 'crystal ball':
        pyautogui.write(['down', 'down', 'down', '\t'], 0.5)
    elif person['source'] == 'money':
        pyautogui.write(['down', 'down', 'down', 'down', '\t'], 0.5)

    # Заполнение поля "RoboCop"
```

```
❸    if person['robocop'] == 1:
❹        pyautogui.write([' ', '\t'], 0.5)
     elif person['robocop'] == 2:
         pyautogui.write(['right', '\t'], 0.5)
     elif person['robocop'] == 3:
         pyautogui.write(['right', 'right', '\t'], 0.5)
     elif person['robocop'] == 4:
         pyautogui.write(['right', 'right', 'right', '\t'], 0.5)
     elif person['robocop'] == 5:
         pyautogui.write(['right', 'right', 'right', 'right', '\t'],
                         0.5)

-- Опущено --
```

Как только раскрывающийся список получает фокус ввода (для этого мы сымитировали нажатие клавиши <Tab> после заполнения поля Greatest Fear(s)), нажатие клавиши <↓> осуществляет переход к следующему элементу списка. Количество нажатий клавиши <↓>, которые должна сгенерировать программа, прежде чем нажать клавишу <Tab> для перехода к следующему полю, определяется значением, хранящимся в элементе `person['source']`. Если значением ключа `'source'` в словаре данного пользователя будет `'wand'` ❶, то мы имитируем однократное нажатие клавиши <↓> (выбор волшебной палочки) и нажатие клавиши <Tab> ❷. Если значением будет `'amulet'`, то мы имитируем два нажатия клавиши <↓> (выбор амулета) и нажатие клавиши <Tab> и т.д. для всех остальных вариантов выбора.

Переключатели, соответствующие различным вариантам ответа на вопрос о фильме *Робокоп*, можно выбирать с помощью клавиши <→> (в случае выбора самого первого варианта ❸ достаточно нажать пробел ❹❶).

Шаг 5. Отправка формы и ожидание

Для заполнения поля Additional Comments (Дополнительные комментарии) нужно вызвать функцию `write()`, передав ей аргумент `person['comments']`. Дополнительно следует ввести символ `'\t'`, чтобы переместить фокус ввода на кнопку Отправить. Как только кнопка Отправить получит фокус, функция `pyautogui.press('enter')` сымитирует нажатие клавиши <Enter>, что приведет к отправке формы. После отправки формы программа должна в течение пяти секунд дожидаться загрузки следующей страницы.

Как только загрузится следующая страница, на ней будет ссылка Отправить еще один ответ, которая возвращает браузер к странице пустой формы. Чтобы сымитировать щелчок на этой ссылке, необходимо последовательно ввести символы `'\t'` и `'\n'` (перемещение фокуса ввода с помощью клавиши <Tab> и нажатие клавиши <Enter>).

Когда снова отобразится пустая форма, цикл `for` перейдет к следующей итерации и начнет ввод данных из очередного словаря.

Завершите программу, добавив в нее следующий код.

```
#! python3
# formFiller.py - автоматическое заполнение формы

-- Опущено --

    # Заполнение поля "Additional Comments"
    pyautogui.write(person['comments'] + '\t')

    # Щелчок на кнопке "Отправить"
    time.sleep(0.5)    # дожидаемся активизации кнопки
    pyautogui.press('enter')

    # Ожидание загрузки страницы
    print('Форма отправлена')
    time.sleep(5)

    # Щелчок на ссылке "Отправить еще один ответ"
    pyautogui.write(['\t', '\n'])
```

Когда цикл `for` завершит свою работу, вся информация по каждому имени будет передана. В данном примере мы имеем дело всего с четырьмя именами. Но если бы речь шла о четырех тысячах человек, то написание подобной программы сэкономило бы вам массу времени!

Отображение окон сообщений

Все программы, которые мы создавали до сих пор, вводили и выводили данные через терминал (с помощью функций `input()` и `print()`). Но в распоряжении функций PyAutoGUI имеется весь экран. В результате окно терминала или редактора Mu может оказаться перекрытым другими окнами, в которых выполнялись щелчки и вводился текст.

Для решения этой проблемы в модуле PyAutoGUI предусмотрены всплывающие окна сообщений, позволяющие выводить уведомления для пользователей и получать от них данные. Окна сообщений создаются с помощью четырех функций.

- **`pyautogui.alert(text)`.** Отображает в окне заданный текст и кнопку OK.
- **`pyautogui.confirm(text)`.** Отображает в окне заданный текст и кнопки OK и Отмена, возвращая `'OK'` или `'Cancel'`, в зависимости от того, на какой кнопке щелкнул пользователь.
- **`pyautogui.prompt(text)`.** Отображает в окне заданный текст и поле ввода, возвращая введенную пользователем строку.

- **pyautogui.password(*text*).** То же, что и `prompt()`, но вместо вводимого текста отображаются звездочки, что позволяет вводить конфиденциальную информацию, такую как пароль.

Эти функции имеют необязательный второй аргумент – строку, используемую в качестве заголовка окна. Функции не возвращают результат до тех пор, пока пользователь не щелкнет на одной из кнопок, поэтому их можно использовать и для добавления пауз в программы, использующие модуль PyAutoGUI. Введите в интерактивной оболочке следующие инструкции.

```
>>> import pyautogui
>>> pyautogui.alert('Это сообщение.', 'Важно')
'OK'
>>> pyautogui.confirm('Будете продолжать?')     # выбираем "Отмена"
'Cancel'
>>> pyautogui.prompt("Как зовут вашего кота?")
'Софи'
>>> pyautogui.password('Введите пароль:')
'охотник2'
```

Соответствующие окна сообщений показаны на рис. 20.8.

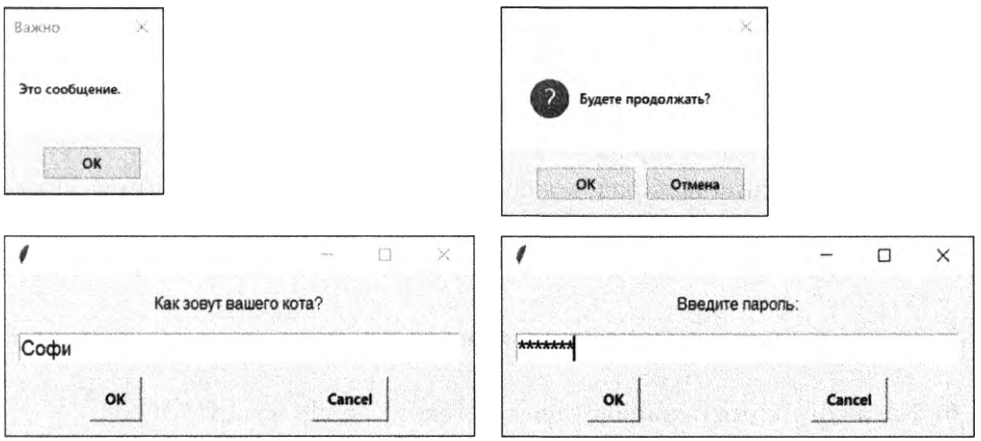


Рис. 20.8. Окна, созданные функциями `alert()`, `verify()`, `prompt()` и `password()`

Онлайн-документация по этим функциям доступна на сайте `https://pymsgbox.readthedocs.io`.

Резюме

Средства GUI-автоматизации, предлагаемые модулем PyAutoGUI, позволяют взаимодействовать с другими приложениями путем управления мышью и клавиатурой. Это достаточно гибкий подход, дающий

возможность делать все то, что обычно делает человек, но есть и недостаток: необходимо правильно определять экранные координаты, чтобы избежать слепого щелканья или ввода. При написании сценариев GUI-автоматизации всегда старайтесь предусмотреть аварийное завершение в случае получения неправильных указаний. Это раздражает, но лучше поступать так, чем выполнять ошибочную программу.

С помощью модуля PyAutoGUI можно перемещать указатель мыши по экрану, а также имитировать щелчки мышью и нажатия клавиш. Кроме того, модуль позволяет проверять цвета экранных пикселей и тем самым анализировать содержимое экрана. Можно даже передать модулю PyAutoGUI снимок экрана, и он самостоятельно определит координаты области, в которой следует выполнить виртуальный щелчок мышью.

Все это позволяет автоматизировать множество рутинных задач. Что может быть лучше, чем наблюдать за тем, как программа самостоятельно перемещает указатель мыши и автоматически вводит текст? Время, потраченное на написание такой программы, окупится сторицей. Ведь так приятно осознавать, что твои навыки программирования помогли тебе избавиться от бремени рутинной работы!

Контрольные вопросы

1. Как включить режим безопасного завершения, предусмотренный в модуле PyAutoGUI для прекращения работы программы?
2. Какая функция возвращает текущее разрешение экрана?
3. Какая функция возвращает координаты текущей позиции указателя мыши?
4. В чем разница между функциями `pyautogui.moveTo()` и `pyautogui.move()`?
5. Какие функции можно использовать для перетаскивания указателя мыши?
6. Какая функция позволяет ввести текст `"Hello world!"`?
7. Как сымитировать нажатия специальных клавиш, таких как <←>?
8. Как сохранить текущее содержимое экрана в файле *screenshot.png*?
9. Как задать паузу длительностью две секунды после вызова любой функции модуля PyAutoGUI?
10. Если требуется автоматизировать щелчки мыши и нажатия клавиш в браузере, то какой модуль следует использовать: PyAutoGUI или Selenium?
11. Из-за чего функции модуля PyAutoGUI бывают подвержены ошибкам?

12. Как найти все окна на экране, содержащие в строке заголовка текст `'Блокнот'`?
13. Как сделать окно браузера Firefox активным, т.е. перевести его на передний план?

Учебные проекты

Чтобы закрепить полученные знания на практике, напишите программы для предложенных ниже задач.

Как притвориться занятым

Многие программы мгновенного обмена сообщениями включают статус "Неактивен", если указатель мыши остается неподвижным в течение определенного периода времени, например 10 минут. Предположим, вы не хотите, чтобы программа сообщала всем о том, что вас нет за компьютером. Напишите сценарий, который будет слегка сдвигать указатель мыши каждые 10 секунд. Эти перемещения должны быть достаточно небольшими, чтобы не мешать вам, если вам потребуется выполнить что-то во время работы сценария.

Использование буфера обмена для чтения текстового поля

Функция `pyautogui.write()` позволяет имитировать нажатия клавиш для ввода текста в поля формы, но в модуле PyAutoGUI нет функций для чтения текста, уже находящегося в текстовом поле. В этой ситуации мог бы пригодиться модуль Pyperclip. Вы используете модуль PyAutoGUI, чтобы активизировать окно текстового редактора, такого как Mu или Блокнот, щелкнуть на нем, затем щелкнуть внутри текстового поля, после чего отправить комбинации клавиш <Ctrl+A>/<⌘+A> для выделения всего текста в поле и <Ctrl+C>/<⌘+C> для копирования текста в буфер обмена. Далее сценарий сможет прочитать содержимое буфера обмена, выполнив инструкции `import pyperclip` и `pyperclip.paste()`.

Напишите программу, которая следует описанной процедуре для копирования текста из полей окна. Используйте функцию `pyautogui.getWindowsWithTitle('Блокнот')`, чтобы получить объект `Window` программы Блокнот (или любого другого текстового редактора по своему усмотрению). Атрибуты `top` и `left` этого объекта помогут определить координаты окна, а метод `activate()` гарантирует, что оно находится поверх остальных окон. Затем можно щелкнуть в основном текстовом поле редактора, чтобы переместить в него фокус ввода. Это можно сделать с помощью функции `pyautogui.click()`, добавив, к примеру, `100` или `200` к значениям атрибутов `top` и `left`. Вызовите функции `pyautogui.hotkey('ctrl', 'a')`

и `pyautogui.hotkey('ctrl', 'c')`, чтобы выделить весь текст и скопировать его в буфер обмена. Далее вызовите функцию `pyperclip.paste()`, чтобы извлечь текст из буфера обмена и скопировать его в сценарий Python. Полученную строку можно просто передать функции `print()`.

Учтите, что в версии 1.0.0 оконные функции PyAutoGUI работают только в Windows, но не в macOS или Linux.

Бот для отправки мгновенных сообщений

Google Talk, Skype, Yahoo Messenger, AIM и другие программы для обмена мгновенными сообщениями часто используют закрытые протоколы, затрудняющие написание сценариев Python, способных взаимодействовать с ними. Но даже закрытость этих протоколов не должна помешать вам писать программы GUI-автоматизации.

В приложении Google Talk имеется поле поиска, в котором можно ввести имя пользователя из списка ваших друзей. При нажатии клавиши <Enter> будет открыто окно чата, куда автоматически переместится фокус ввода. В других программах-мессенджерах предлагаются аналогичные способы открытия окон чата. Напишите программу для автоматической рассылки уведомлений группе людей из списка ваших друзей. В программе придется обрабатывать различные ситуации, такие, например, как отсутствие адресата в сети, появление окна чата в разных местах экрана или появление окон подтверждений, блокирующих обмен сообщениями. Программа должна будет делать снимки экрана, чтобы управлять взаимодействием с пользователем, и распознавать ситуации, в которых виртуальные нажатия клавиш не срабатывают.

Примечание

Желательно создать тестовые учетные записи, чтобы случайно не забросать своих друзей спамом в процессе разработки программы.

Руководство по созданию игрового бота

По указанному ниже адресу доступно обучающее руководство под названием "Programming a Bot to Play the Sushi Go Round Flash Game" (Программирование бота для Flash-игры "Sushi Go Round"):

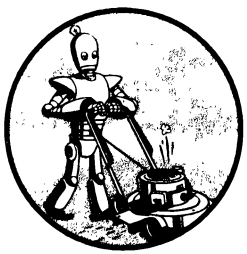
```
http://inventwithpython.com/blog/2014/12/17/
programming-a-bot-to-play-the-sushi-go-round-flash-game/
```

В руководстве объясняется, как написать программу GUI-автоматизации, способную играть в браузерную игру "Sushi Go Round" (Суши-бар). В этой игре нужно щелкать на кнопках, соответствующих различным

ингредиентам, для оформления заказов суши. Чем быстрее вы выполняете заказы, не допуская ошибок, тем больше очков зарабатываете. Такая игра идеально подходит для того, чтобы запрограммировать ее средствами GUI-автоматизации и при этом немного схитрить, добавив себе лишних очков! Руководство охватывает многие из тем, рассмотренных в данной главе, а также включает описание основных функций PyAutoGUI, предназначенных для распознавания изображений. Исходный код бота доступен по адресу `https://github.com/asweigart/sushigoroundbot/`. Видеоролик с примером бота, играющего в игру, можно просмотреть по адресу `https://youtu.be/lfk_T6VKhTE`.

A

УСТАНОВКА СТОРОННИХ МОДУЛЕЙ



Многие разработчики пишут собственные модули, расширяющие возможности стандартной библиотеки Python. Основным средством установки таких модулей служит утилита `pip`. Она обеспечивает безопасную загрузку и инсталляцию модулей Python, доступных на сайте `https://pypi.org/`. PyPI (Python Package Index) — это своего рода магазин бесплатных приложений для Python.

Утилита pip

Утилита pip входит в состав Python 3.4 и более поздних версий на платформах Windows и macOS; в Linux ее нужно устанавливать отдельно. Чтобы узнать, имеется ли она в Linux, выполните команду `which pip3` в окне терминала. Если утилита установлена, вы увидите путь к ее каталогу; в противном случае не отобразится ничего. Чтобы установить утилиту pip3 в Ubuntu или Debian Linux, откройте новое окно терминала и введите **`sudo apt-get install python3-pip`**. Чтобы установить утилиту в Fedora Linux, введите в окне терминала **`sudo yum install python3-pip`**. Установщик потребует ввести пароль администратора.

Утилита pip выполняется в окне *терминала* (также называемого *командной строкой*), а не в интерактивной оболочке Python. Чтобы открыть это окно в Windows, найдите приложение Командная строка в меню Пуск. В macOS запустите приложение Terminal из меню Spotlight. В Ubuntu Linux запустите приложение Terminal из меню Ubuntu Dash или нажмите комбинацию клавиш <Ctrl+Alt+T>.

Если папка утилиты pip отсутствует в списке переменной среды PATH, то перед запуском утилиты придется изменить текущий каталог в окне терминала с помощью команды cd. Для начала узнайте свое имя пользователя с помощью команды `echo %USERNAME%` в Windows или `whoami` в macOS и Linux. Затем выполните команду `cd` *`папка_pip`*, где нужная папка определяется следующим образом:

- Windows

```
C:\Users\<имя_пользователя>\AppData\Local\Programs\Python\
Python37\Scripts
```

- macOS

```
/Library/Frameworks/Python.framework/Versions/3.7/bin/
```

- Linux

```
/home/<имя_пользователя>/.local/bin/
```

После этого можно запускать утилиту pip.

Инсталляция модулей

Исполняемый файл утилиты называется pip в Windows и pip3 в macOS и Linux. В командной строке утилите передается команда install и имя

модуля, который нужно установить. Например, в Windows используется команда `pip install --user` *модуль*.

Поскольку будущие изменения сторонних модулей могут не обеспечивать обратную совместимость, я рекомендую устанавливать конкретные версии модулей, используемые в книге (они будут перечислены далее). Для этого можно воспользоваться аргументом командной строки `-U` *модуль*`==`*версия* (обратите внимание на два знака равенства). Например, команда `pip install --user -U send2trash==1.5.0` устанавливает модуль `send2trash` версии 1.5.0.

Чтобы установить все модули, используемые в книге, загрузите файл требований для своей операционной системы из архива примеров книги (см. введение) и выполните одну из следующих команд:

- Windows

```
pip install --user -r automate-win-requirements.txt --user
```

- macOS

```
pip3 install --user -r automate-mac-requirements.txt --user
```

- Linux

```
pip3 install --user -r automate-linux-requirements.txt --user
```

Ниже перечислены сторонние модули, используемые в книге, с указанием номеров версий. Можете вводить эти команды по отдельности, если нужно установить лишь некоторые из модулей.

- `pip install --user send2trash==1.5.0`
- `pip install --user requests==2.21.0`
- `pip install --user beautifulsoup4==4.7.1`
- `pip install --user selenium==3.141.0`
- `pip install --user openpyxl==2.6.1`
- `pip install --user PyPDF2==1.26.0`
- `pip install --user python-docx==0.8.10` (именно `python-docx`, а не `docx`)
- `pip install --user imapclient==2.1.0`
- `pip install --user pyzmail36==1.0.4`
- `pip install --user twilio`
- `pip install --user ezgmail`

- `pip install --user ezsheets`
- `pip install --user pillow==6.0.0`
- `pip install --user pyobjc-framework-Quartz==5.2` (только в macOS)
- `pip install --user pyobjc-core==5.2` (только в macOS)
- `pip install --user pyobjc==5.2` (только в macOS)
- `pip install --user python3-xlib==0.15` (только в Linux)
- `pip install --user pyautogui`

Примечание

В macOS модуль `pyobjc` устанавливается около 20 минут или даже больше. Предварительно следует установить модуль `pyobjc-core`, что позволит сократить общее время установки.

После установки модуля вы сможете убедиться в его наличии, выполнив команду `import` *модуль* в интерактивной оболочке. Если никаких сообщений об ошибках не появилось, значит, модуль был успешно установлен.

Если в системе уже установлен модуль, но вы хотите обновить его до последней версии, доступной в PyPI, выполните команду `pip install --user -U` *модуль* (или `pip3 install --user -U` *модуль* в macOS и Linux). Опция `--user` задает установку модуля в ваш домашний каталог. Это позволяет избежать потенциальных ошибок, связанных с правами доступа, которые могут возникнуть при установке модуля для всех пользователей системы.

В последних версиях модулей Selenium и OpenPyXL появились новинки, несовместимые с версиями модулей, которые используются в книге. С другой стороны, модули Twilio, EZGmail и EZSheets взаимодействуют с онлайн-службами, поэтому может понадобиться устанавливать их последние версии с помощью команды `pip install --user -U`.

Предупреждение

В первом издании книги предлагалось использовать команду `sudo`, если при запуске утилиты `pip` возникали ошибки, связанные с правами доступа: `sudo pip install` модуль. Это плохая практика, поскольку данная команда устанавливает модули в каталог Python, используемый операционной системой. Операционная система может запускать сценарии Python для выполнения системных задач, и если установить для нее модули, которые конфликтуют с уже имеющимися модулями, можно столкнуться с непонятными ошибками. Никогда не используйте команду `sudo` для установки модулей Python.

Установка модулей для редактора Mu

В редакторе Mu имеется собственная среда Python, отличающаяся от той, которая создается для типичного дистрибутива Python. Чтобы установить модули для сценариев, запускаемых из Mu, откройте панель администратора, щелкнув на значке шестеренки в правом нижнем углу редактора Mu. В появившемся окне перейдите на вкладку Third Party Packages (Сторонние пакеты) и следуйте инструкциям по установке модулей на этой вкладке. Возможность установки модулей в Mu стала доступна относительно недавно, поэтому инструкции могут поменяться.

Если не получается установить модули с помощью панели администратора, откройте окно терминала и запустите утилиту `pip` для редактора Mu. Для этого нужно использовать аргумент командной строки `--target`, после которого указывается папка модуля Mu. В Windows это папка *C:\Users\<имя_пользователя>\AppData\Local\Mu\pkgs*, в macOS – папка */Applications/mu-editor.app/Contents/Resources/app_packages*. В Linux указывать аргумент `--target` не нужно; достаточно запустить утилиту `pip3`.

Например, если вы загрузили файл требований для своей операционной системы из архива примеров книги (см. введение), то выполните следующие команды:

- Windows

```
pip install -r automate-win-requirements.txt
    --target "C:\Users\имя_пользователя\AppData\Local\Mu\pkgs"
```

- macOS

```
pip3 install -r automate-mac-requirements.txt --target
    /Applications/mu-editor.app/Contents/Resources/app_packages
```

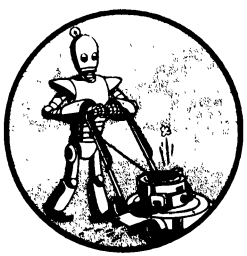
- Linux

```
pip3 install --user -r automate-linux-requirements.txt
```

Если требуется установить лишь некоторые из модулей, выполните обычную команду `pip` (или `pip3`), добавив аргумент `--target`.

Б

ЗАПУСК ПРОГРАММ



Запустить программу, открытую в окне редактора Mu, несложно — для этого достаточно нажать клавишу <F5> или щелкнуть на кнопке Run на панели инструментов. Таким образом можно проверять работу программ в процессе их написания, но открывать Mu для запуска готовых приложений — не лучшее решение. Существуют более удобные способы выполнения сценариев Python, в зависимости от используемой операционной системы.

Запуск программ в окне терминала

Программы можно запускать в окне терминала (также называемого *командной строкой*), хотя для некоторых пользователей это непривычно: у командной строки нет графического интерфейса, и она не предлагает никаких подсказок относительно того, что нужно делать.

Чтобы открыть окно терминала в Windows, щелкните на кнопке Пуск, найдите утилиту Командная строка и запустите ее. В macOS щелкните на значке Spotlight в правом верхнем углу, введите **Terminal** и нажмите клавишу <Enter>. В Ubuntu Linux нажмите клавишу <Win>, чтобы открыть меню Dash, введите **Terminal** и нажмите <Enter>. Можно также нажать комбинацию клавиш <Ctrl+Alt+T>.

Подобно приглашению >>> в интерактивной оболочке, в окне терминала появляется приглашение для ввода команд. В Windows это будет полный путь к текущей папке:

```
C:\Users\Al> здесь вводятся команды
```

В macOS в приглашении отображается имя компьютера, двоеточие, текущий каталог (для краткости домашняя папка обозначается как ~) и ваше имя пользователя, за которым следует знак доллара ($):

```
Als-MacBook-Pro:~ al$ здесь вводятся команды
```

В Ubuntu Linux приглашение начинается с имени пользователя и знака @:

```
al@al-VirtualBox:~$ здесь вводятся команды
```

Строку приглашения можно изменить, но в книге этот вопрос не рассматривается.

При вводе команды, например `python` в Windows или `python3` в macOS и Linux, терминал проверяет наличие программы с таким именем в текущей папке. Если ее там нет, то проверяются папки, перечисленные в переменной среды `PATH`. *Переменные среды* — это глобальные переменные уровня операционной системы, содержащие важные системные настройки. Чтобы увидеть содержимое переменной среды `PATH`, выполните команду **echo %PATH%** в Windows или **echo $PATH** в macOS и Linux. Вот пример для macOS.

```
Als-MacBook-Pro:~ al$ echo $PATH
/Library/Frameworks/Python.framework/Versions/3.7/bin:/usr/local/bin:
/usr/bin:/bin:/usr/sbin:/sbin
```

В macOS файл программы `python3` находится в папке `/Library/Frameworks/Python.framework/Versions/3.7/bin`, поэтому вам не придется вводить команду `/Library/Frameworks/Python.framework/Versions/3.7/bin/python3` или переходить в указанную папку, чтобы запустить программу. Можно просто ввести **`python3`**, находясь в любой папке, и программа будет найдена в одной из папок, перечисленных в переменной среды `PATH`. При установке важной программы всегда имеет смысл добавлять путь к ней в переменную `PATH`.

Если требуется выполнить сценарий Python, необходимо ввести **`python`** (или **`python3`**), а затем указать имя *.py*-файла. Это приведет к запуску интерпретатора Python, а он в свою очередь выполнит код из *.py*-файла. Когда сценарий Python завершится, вы вернетесь в окно терминала. Например, в Windows запуск программы, выводящей сообщение `'Hello, world!'`, будет выглядеть так.

```
Microsoft Windows [Version 10.0.17134.648]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Al>python hello.py
Hello, world!

C:\Users\Al>
```

Если ввести **`python`** (или **`python3`**), не указав имя файла, то будет запущена интерактивная оболочка.

Запуск сценариев Python в Windows

Существуют и другие способы запуска сценариев Python в Windows. Вместо того чтобы открывать окно терминала, можно нажать комбинацию клавиш <Win+R>, чтобы открыть диалоговое окно Выполнить, и ввести **`py C:\путь\Сценарий.py`** (рис. Б.1). Программа *py.exe* установлена в папке *C:\Windows*, которая уже прописана в переменной среды `PATH`, поэтому вводить расширение *.exe* необязательно.

Недостаток такого способа заключается в том, что нужно вводить полный путь к сценарию. Кроме того, при запуске сценария Python из диалогового окна будет открыто новое окно терминала для вывода результатов работы сценария, и это окно автоматически закроется по завершении программы. В итоге можно не успеть увидеть, что же вывела программа.

Для решения проблемы необходимо создать *пакетный сценарий*, представляющий собой небольшой текстовый файл с расширением *.bat*. Из него можно выполнить несколько команд в окне терминала, подобно сценарию командной оболочки в macOS и Linux. Такой файл можно создать в любом текстовом редакторе, таком как Блокнот.

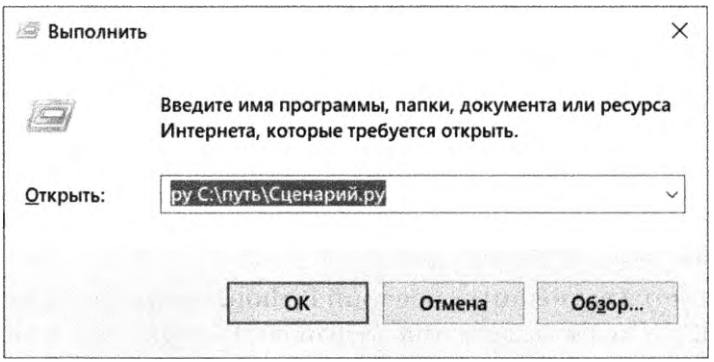


Рис. Б.1. Диалоговое окно **Выполнить** *в Windows*

Например, пакетный сценарий может выглядеть так.

```
@py.exe C:\путь\pythonScript.py %*
@pause
```

Укажите абсолютный путь к своей программе и сохраните этот файл с расширением *.bat* (например, *pythonScript.bat*). Знак `@` в начале каждой команды предотвращает ее отображение в окне терминала, а директива `%*` передает все аргументы командной строки, введенные после имени пакетного файла, сценарию Python. Последний, в свою очередь, считывает аргументы командной строки из списка `sys.argv`. Такой пакетный файл избавит вас от необходимости вводить полный путь к программе Python каждый раз, когда вы захотите ее запустить. Инструкция `@pause` выведет сообщение `'Для продолжения нажмите любую клавишу...'` после завершения сценария, чтобы предотвратить закрытие окна программы. Рекомендую поместить все ваши пакетные файлы и сценарии Python в одну папку, которая прописана в переменной среды `PATH`, например *C:\Users\<имя_пользователя>*.

При наличии пакетного файла вам не нужно открывать окно терминала и вводить полный путь к файлу вместе с именем сценария. Вместо этого просто нажмите комбинацию клавиш <Win+R>, введите **`pythonScript`** (указывать расширение *bat* необязательно) и нажмите клавишу <Enter>, чтобы запустить сценарий.

Запуск сценариев Python в macOS

В macOS можно создать сценарий командной оболочки с расширением *.command*, предназначенный для запуска сценариев Python. Создайте новый файл в текстовом редакторе, таком как TextEdit, и добавьте в него следующий код.

```
#!/usr/bin/env bash
python3 /путь/pythonScript.py
```

Сохраните файл с расширением *.command* в своей домашней папке (например, на моем компьютере это папка */Users/al*). В окне терминала сделайте данный сценарий исполняемым, выполнив команду **chmod u+x pythonScript.command**. Теперь достаточно щелкнуть на значке Spotlight (или нажать комбинацию клавиш <⌘+пробел>) и ввести **pythonScript.command**, чтобы запустить сценарий оболочки, который, в свою очередь, запустит сценарий Python.

Запуск сценариев Python в Ubuntu Linux

Для запуска сценариев Python из меню Dash в Ubuntu Linux потребуется выполнить ряд настроек. Предположим, имеется сценарий */home/al/example.py* (ваш сценарий может называться по-другому и находиться в другой папке), который нужно запускать из меню Dash. Для начала откройте любой текстовый редактор, например `gedit`, и создайте новый файл следующего вида.

```
[Desktop Entry]
Name=example.py
Exec=gnome-terminal -- /home/al/example.sh
Type=Application
Categories=GTK;GNOME;Utility;
```

Сохраните этот файл в папке */home/<al>/.local/share/applications* (замените *al* своим именем пользователя), присвоив ему имя *example.desktop*. Если текстовый редактор не видит папку *.local* (папки, начинающиеся с точки, считаются скрытыми), сохраните файл в своей домашней папке (например, */home/al*), после чего откройте окно терминала и переместите файл с помощью команды `mv /home/al/example.desktop /home/al/.local/share/applications`.

Теперь, когда файл *example.desktop* находится в папке */home/al/.local/share/applications*, вы сможете нажать клавишу <Win>, чтобы открыть меню Dash, и ввести **example.py** (или другое имя, заданное в поле `Name`). Откроется новое окно терминала (в частности, программа `gnome-Terminal`), которая запустит сценарий оболочки */home/al/example.sh*. Его мы сейчас создадим.

В текстовом редакторе создайте новый файл следующего вида.

```
#!/usr/bin/env bash
python3 /home/al/example.py
bash
```

Сохраните его с именем */home/al/example.sh*. Это сценарий командной оболочки, содержащий команды, которые выполняются в окне терминала. Он запустит наш сценарий */home/al/example.py* и оболочку `bash`. Без команды `bash` в последней строке окно терминала закроется, как только сценарий завершит свою работу, и вы не успеете увидеть текст, выводимый функциями `print()`.

Необходимо разрешить выполнение этого сценария, выполнив следующую команду в окне терминала:

```
al@ubuntu:~$ chmod u+x /home/al/example.sh
```

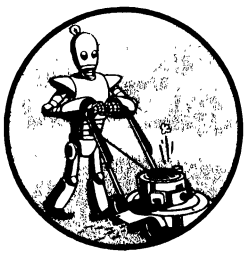
При наличии сконфигурированных файлов *example.desktop* и *example.sh* можно будет запустить сценарий *example.py*, нажав клавишу <Win> и введя **`example.py`** (или любое другое имя, которое вы указали в поле `Name` файла *example.desktop*).

Запуск сценариев Python с отключенными проверками

Скорость работы сценариев Python можно немного увеличить, отключив инструкции `assert`. Если программа запускается из окна терминала, добавьте ключ `-O` после команды `python` или `python3` и перед именем *.py*-файла. Это приведет к запуску оптимизированной версии программы, в которой инструкции `assert` будут игнорироваться.

В

ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ



В этом приложении даны ответы на контрольные вопросы, приведенные в конце каждой главы. Настоятельно рекомендую пытаться самостоятельно находить правильные ответы и обращаться к данному приложению только для контроля. Чтобы научиться программировать, одного лишь знания синтаксиса языка и названий функций недостаточно. Как и при изучении иностранного языка, чем больше вы будете применять свои знания на практике, тем лучше будут результаты.

Существует множество веб-сайтов, на которых можно аналогичным образом проверить свои знания программирования. Небольшая подборка таких сайтов доступна по следующему адресу:

```
https://automatetheboringstuff.com/
list-of-programming-practice-sites.html
```

Что касается учебных проектов, помните: единственно правильной программы не существует. Если ваша программа соответствует требованиям проекта, можете считать ее правильной.

Глава 1

1. Операторы — `*`, `-`, `/` и `+`. Значения — `'привет'`, `-88.8` и `5`.
2. Строка — `'spam'`; переменная — `spam`. Строки всегда берутся в кавычки.
3. Три основных типа данных, рассмотренных в этой главе, — целые числа, вещественные числа и строки.
4. Выражение состоит из значений и операторов. Любое выражение сводится к одиночному значению.
5. Результатом вычисления выражения будет одиночное значение. В случае инструкции это не так.
6. `20`. Выражение `bacon + 1` не изменяет значение переменной `bacon` (для этого нужна операция присваивания: `bacon = bacon + 1`).
7. В обоих случаях будет получена одна и та же строка: `'spamspamspam'`.
8. Имена переменных не могут начинаться с цифры.
9. Функции `int()`, `float()` и `str()` возвращают целочисленное, вещественное и строковое представление числа соответственно.
10. В данном выражении ошибка возникает из-за того, что `99` — целое число, тогда как с помощью оператора `+` можно конкатенировать только строки. Для устранения ошибки следует использовать выражение `'Я съел ' + str(99) + ' котлет.'`.

Глава 2

1. Это значения `True` и `False`. Первые буквы, `'T'` и `'F'`, — прописные, остальные — строчные.
2. `and`, `or` и `not`.
3. `True and True` — `True`.

   `True and False` — `False`.

   `False and True` — `False`.

   `False and False` — `False`.

`True or True` — `True`.

`True or False` — `True`.

`False or True` — `True`.

`False or False` — `False`.

`not True` — `False`.

`not False` — `True`.

4. `False`

   `False`

   `True`

   `False`

   `False`

   `True`

5. ==, !=, <, >, <= и >=.
6. Оператор == сравнивает два значения и возвращает результат сравнения в виде булева значения, тогда как оператор присваивания = сохраняет значение в переменной.
7. Условие — это выражение, вычисление которого дает булево значение (`True` или `False`). Такие выражения применяются в управляющих инструкциях.
8. Первый блок — все тело инструкции `if`, второй блок — строка `print('бекон')`, третий блок — строка `print('ветчина')`.

```
print('яйца')
if spam > 5:
    print('бекон')
else:
    print('ветчина')
print('спам')
```

9.

```
if spam == 1:
    print('Hello')
elif spam == 2:
    print('Howdy')
else:
    print('Greetings!')
```

10. <Ctrl+C>.
11. Инструкция `break` осуществляет досрочный выход из цикла, после чего управление передается инструкции, следующей за циклом.

Инструкция `continue` осуществляет досрочный возврат в начало цикла и переход к следующей итерации.

12. Никакой разницы между ними нет. Вызов `range(10)` задает диапазон счетчика цикла от `0` до (но не включая) `10`. Вызов `range(0, 10)` в явном виде задает начальное значение счетчика цикла равным `0`. Вызов `range(0, 10, 1)` в явном виде задает приращение счетчика цикла равным `1` на каждой итерации.
13.

```
for i in range(1, 11):
    print(i)
```

и

```
i = 1
while i <= 10:
    print(i)
    i = i + 1
```

14. `spam.bacon()`.

Глава 3

1. Функции устраняют потребность в дублировании кода, уменьшая размеры программ, повышая их удобочитаемость и упрощая их обновление.
2. Код функции выполняется тогда, когда она вызывается, а не тогда, когда она определяется.
3. Для определения (т.е. создания) функции предназначена инструкция `def`.
4. Определение функции включает инструкцию `def` и код, образующий ее тело. Вызов функции — это передача управления в тело функции и выполнение ее кода для вычисления возвращаемого значения.
5. Существует только одна глобальная область видимости, тогда как каждый вызов функции создает свою локальную область видимости.
6. Когда функция завершается, ее локальная область видимости уничтожается, и вся информация о ее переменных теряется.
7. Возвращаемое значение — это результат, вычисляемый функцией. Как и любое другое значение, оно может быть частью выражения.
8. `None`.
9. Для этого следует предварить имя переменной ключевым словом `global`.
10. `NoneType`.

11. Эта инструкция импортирует модуль `areallyourpetsnamederic` (кстати, такого модуля в Python не существует).
12. `spam.bacon()`.
13. Поместите строку (или фрагмент) кода, где может возникнуть ошибка, в блок `try`.
14. В блок `try` включается код, в котором может возникнуть ошибка. В блок `except` включается код, который должен быть выполнен в случае возникновения ошибки.

Глава 4

1. Пустой список, т.е. список, не содержащий элементов. Аналогичным образом выражение `''` означает пустую строку.
2. `spam[2] = 'hello'`. (Обратите внимание на то, что третьим элементом списка будет элемент с индексом 2, поскольку индексация начинается с 0.)
3. `'d'`. (Результатом выражения `'3' * 2` будет строка `'33'`, которая передается функции `int()`. Далее полученное значение `33` делится на `11`, и мы получаем целочисленный индекс `3`. Выражения могут использоваться везде, где ожидаются значения.)
4. `'d'`. (Отрицательные индексы отсчитываются с конца.)
5. `['a', 'b']`
6. `1`.
7. `[3.14, 'кот', 11, 'кот', True, 99]`
8. `[3.14, 11, 'кот', True]`
9. Оператор конкатенации списков — `+`, оператор репликации — `*` (те же, что и в случае строк).
10. Функция `append()` добавляет значения только в конец списка, тогда как функция `insert()` может вставлять их в любом месте списка.
11. Для удаления значений из списка можно применить инструкцию `del` и списковый метод `remove()`.
12. Как списки, так и строки могут передаваться функции `len()`, иметь индексы и срезы, использоваться в циклах `for`, конкатенироваться и реплицироваться, а также использоваться совместно с операторами `in` и `not in`.
13. Списки изменяемы; они допускают добавление, удаление и изменение хранящихся в них значений. Кортежи неизменяемы; они не допускают вообще никаких изменений. Кроме того, кортежи записываются с использованием круглых скобок, `(` и `)`, тогда как списки — с использованием квадратных скобок: `[` и `]`.

14. `(42,)` (завершающая запятая обязательна).
15. С помощью функций `tuple()` и `list()` соответственно.
16. Она содержит ссылку на список.
17. Функция `copy.copy()` создает простую копию списка, тогда как функция `copy.deepcopy()` — полную его копию, включающую вложенные списки. Это означает, что только функция `copy.deepcopy()` может дублировать списки, содержащиеся в других списках.

Глава 5

1. `{}`.
2. `{'foo': 42}`
3. Элементы словаря не упорядочены, тогда как элементы списка упорядочены.
4. Возникнет исключение `KeyError`.
5. Никакой разницы нет. Оператор `in` проверяет наличие ключа в словаре.
6. Выражение `'кот' in spam` проверяет, существует ли ключ `'cat'` в словаре, тогда как выражение `'кот' in spam.values()` проверяет, существует ли в словаре `spam` ключ со значением `'кот'`.
7. `spam.setdefault('цвет', 'черный')`
8. `pprint.pprint()`

Глава 6

1. Это символы, которые невозможно сами по себе вставить в строку.
2. `\n` — символ новой строки, `\t` — символ табуляции.
3. С помощью экранированного символа `\\`.
4. Апостроф в слове `"Howl's"` вполне допустим, поскольку строка заключена в двойные кавычки.
5. Многострочные текстовые блоки заключаются в тройные кавычки. В них нет необходимости использовать экранированные символы `\n`.
6.
   - `'д'`
   - `'Здрав'`
   - `'Здрав'`
   - `'авствуй, мир!'`
7.
   - `'ЗДРАВСТВУЙ'`

- `True`
- `'здравствуй'`

8.

- `['Помни', 'о', 'том,', 'что', 'было,', 'не', 'забывай.']`
- `'Должен-остаться-только-один.'`

9. Строковые методы `rjust()`, `ljust()` и `center()` соответственно.
10. С помощью методов `lstrip()` и `rstrip()` соответственно.

Глава 7

1. Функция `re.compile()` возвращает объект `Regex`.
2. Необработанные строки используют для того, чтобы не нужно было экранировать символы обратной косой черты.
3. Объект `Match`.
4. Метод `group()` объекта `Match` возвращает строки, соответствующие шаблону регулярного выражения.
5. Группа 0 соответствует всему совпавшему тексту, группа 1 — тексту, совпавшему с выражением в первой паре круглых скобок, группа 2 — тексту, совпавшему с выражением во второй паре круглых скобок.
6. Точки и круглые скобки можно экранировать символами обратной косой черты: `\.`, `\(` и `\)`.
7. Если в регулярном выражении нет групп, то возвращается список строк, в противном случае возвращается список кортежей строк.
8. Символ `|` означает поиск соответствия одной из двух альтернативных групп.
9. Символ `?` может означать: 1) нулевое или единичное вхождение предшествующей группы; 2) нежадный поиск.
10. Символ `+` означает одно или несколько вхождений искомого выражения. Символ `*` означает нулевое или произвольное количество вхождений искомого выражения.
11. Шаблону `{3}` соответствуют ровно три вхождения группы. Шаблону `{3, 5}` соответствуют от трех до пяти вхождений.
12. Одиночная цифра, буквенно-цифровой символ и пробельный символ соответственно.
13. Одиночный символ, не являющийся цифрой, буквенно-цифровым или пробельным символом соответственно.
14. Шаблон `.*` задает режим жадного поиска, а шаблон `.*?` — нежадного.
15. Либо `[0-9a-z]`, либо `[a-z0-9]`.

16. Передать константу `re.I` или `re.IGNORECASE` в качестве второго аргумента функции `re.compile()`.
17. Обычно символ `.` совпадает с любым символом, за исключением символа новой строки. Если функции `re.compile()` передать аргумент `re.DOTALL`, то точке будет соответствовать также символ новой строки.
18. `'X барабанщиков, X волынщиков, пять колец, X курицы'`
19. Аргумент `re.VERBOSE` делает возможным добавление пробельных символов и комментариев в строку, передаваемую функции `re.compile()`.
20. `re.compile(r'^\d{1, 3}(,\d{3})*$')`, но возможны и другие варианты.
21. `re.compile(r'[A-Z][a-z]*\sWatanabe')`
22. `re.compile(r'(Алиса|Боб|Кэрол)\s(ест|заводит|кидает)\s(яблоки|кошек|мячи)\.', re.IGNORECASE)`

Глава 8

1. Нет, это сторонний модуль.
2. Это позволяет сократить код: можно писать `pyip.inputStr()` вместо `pyinputplus.inputStr()`.
3. Функция `inputInt()` возвращает значение типа `int`, а функция `inputFloat()` — значение типа `float`.
4. Вызовите функцию `pyip.inputint(min=0, max=99)`.
5. Списки регулярных выражений, определяющих, какие значения допустимы, а какие — нет.
6. Функция сгенерирует исключение `RetryLimitException`.
7. Функция вернет строку `'привет'`.

Глава 9

1. Относительно текущего каталога.
2. С корневого каталога, например */* или *C:*.
3. `WindowsPath('C:/Users/Al')`. В других операционных системах создается другой тип объекта `Path` для того же пути.
4. Выражение `'C:/Users' / 'Al'` вызывает ошибку, так как нельзя использовать оператор `/` для конкатенации двух строк.
5. Функция `os.getcwd()` возвращает текущий каталог, а функция `os.chdir()` изменяет текущий каталог.
6. Специальное имя `.` — это текущая папка, а `..` — ее родительская папка.

7. *C:\bacon\eggs* — это имя папки, а *spam.txt* — имя файла.
8. `'r'` — режим чтения, `'w'` — режим записи, `'a'` — режим добавления.
9. Содержимое существующего файла, открытого в режиме записи, уничтожается и полностью перезаписывается.
10. Метод `read()` возвращает все содержимое файла в виде одной большой строки. Метод `readlines()` возвращает список строк, хранящихся в файле.
11. Словарь. В хранилищах, как и в словарях, есть ключи и значения, а также используются методы `keys()` и `values()`, работающие аналогично одноименным методам словарей.

Глава 10

1. Функция `shutil.copy()` копирует одиночный файл, а функция `shutil.copytree()` — всю папку вместе со всем ее содержимым.
2. Функция `shutil.move()` предназначена как для переименования файлов, так и для их перемещения.
3. Функции модуля `send2trash` перемещают файл или папку в корзину, тогда как соответствующие функции модуля `shutil` безвозвратно удаляют файлы и папки.
4. `zipfile.ZipFile()`. Его первым аргументом служит имя файла, а вторым — режим, в котором открывается ZIP-файл (чтение, запись, присоединение).

Глава 11

1.
```
assert(spam >= 10, 'Значение переменной spam меньше 10.')
```

2.
```
assert(eggs.lower() != bacon.lower(), 'Переменные eggs и bacon
       содержат одинаковые строки!')
```
или
```
assert(eggs.upper() != bacon.upper(), 'Переменные eggs и bacon
       содержат одинаковые строки!')
```

3.
```
assert(False, 'Это утверждение всегда генерирует исключение
       AssertionError.')
```

4.
```
import logging
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s -
%(levelname)s - %(message)s')
```

5.
```
import logging
logging.basicConfig(filename='programLog.txt', level=logging.DEBUG,
format='%(asctime)s - %(levelname)s - %(message)s')
```

6. DEBUG, INFO, WARNING, ERROR и CRITICAL.
7. `logging.disable(logging.CRITICAL)`
8. Вывод журнальных сообщений можно отключить, не удаляя вызовы функций протоколирования. Также разрешается селективно отключать вывод журнальных сообщений, соответствующих ошибкам более низких уровней протоколирования. Кроме того, журнальные сообщения содержат метки времени.
9. После щелчка на кнопке Step In отладчик вызывает функцию и останавливается на первой строке ее кода. После щелчка на кнопке Step Over весь код функции выполняется в обычном режиме. После щелчка на кнопке Step Out строки кода выполняются в обычном режиме до тех пор, пока не будет осуществлен возврат из текущей функции.
10. После щелчка на кнопке Continue программа выполняется обычным образом либо до конца, либо до точки останова.
11. Наличие точки останова в строке кода приводит к тому, что при достижении этой строки отладчик приостанавливает выполнение программы.
12. Необходимо щелкнуть на номере строки, чтобы рядом с ним появился красный маркер.

Глава 12

1. В модуле `webbrowser` имеется метод `open()`, который запускает браузер и направляет его по заданному URL-адресу. Модуль `requests` позволяет загружать файлы и веб-страницы из Интернета. Модуль Beautiful Soup предназначен для синтаксического анализа (парсинга) HTML-документов. Наконец, модуль `selenium` позволяет запускать браузер и управлять его работой.
2. `Response`. Атрибут `text` этого объекта содержит загруженное содержимое в виде строки.

3. Метод `raise_for_status()` генерирует исключение, если в процессе загрузки возникли проблемы, и ничего не делает в случае успешной загрузки.
4. Код состояния HTTP содержится в атрибуте `status_code` объекта `Response`.
5. Необходимо открыть новый файл в режиме записи бинарных данных (`'wb'`) и использовать цикл `for` для записи веб-содержимого, возвращаемого методом `iter_content()` объекта `Response`, фрагмент за фрагментом.

```
saveFile = open('filename.html', 'wb')
for chunk in res.iter_content(100000):
    saveFile.write(chunk)
```

6. В браузере Chrome следует нажать клавишу <F12>. В Firefox для этого предназначена комбинация клавиш <Ctrl+Shift+C> (Windows и Linux) или <⌘+Option+C> (macOS).
7. Для этого следует щелкнуть правой кнопкой мыши на элементе веб-страницы и выбрать в контекстном меню пункт Inspect Element (Исследовать элемент).
8. `'#main'`
9. `'.highlight'`
10. `'div div'`
11. `'button[value="favorite"]'`
12. `spam.getText()`
13. `linkElem.attrs`
14. Для импорта модуля `selenium` следует использовать инструкцию `from selenium import webdriver`.
15. Методы `find_element_*` возвращают первый совпавший элемент в виде объекта `WebElement`. Методы `find_elements_*` возвращают список всех совпавших элементов в виде объектов `WebElement`.
16. Методы `click()` и `send_keys()` соответственно.
17. Для отправки формы достаточно вызвать метод `submit()` любого из содержащихся в ней элементов.
18. Щелчки на этих кнопках браузера имитируются методами `forward()`, `back()` и `refresh()` объекта `WebDriver`.

Глава 13

1. Объект `Workbook`.
2. Объект `Worksheet`.

3. `wb['Sheet1']`
4. `wb.active`
5. `sheet['C5'].value` или `sheet.cell(row=5, column=3).value`.
6. `sheet['C5'] = 'Hello'` или `sheet.cell(row=5, column=3).value = 'Hello'`.
7. `cell.row` и `cell.column`.
8. Они содержат целочисленный номер последнего столбца и последней строки листа соответственно.
9. `openpyxl.cell.column_index_from_string('M')`
10. `openpyxl.cell.get_column_letter(14)`
11. `sheet['A1':'F1']`
12. `wb.save('example.xlsx')`
13. Формула для ячейки задается подобно любому другому значению: установите в качестве значения атрибута `value` строку с текстом формулы. Не забывайте о том, что формула начинается со знака =.
14. `sheet.row_dimensions[5].height = 100`
15. `sheet.column_dimensions['C'].hidden = True`
16. Закрепленные области — это строки и столбцы, которые всегда видны на экране. Их полезно использовать в качестве заголовков.
17. `openpyxl.charts.Reference()`, `openpyxl.charts.Series()`, `openpyxl.charts.BarChart()`, `chartObj.append(seriesObj)` и `add_chart()`.

Глава 14

1. Файл учетных данных, файл токена для приложения Google Таблицы и файл токена для хранилища Google Диск.
2. `ezsheets.Spreadsheet` и `ezsheets.Sheet`.
3. Вызовите метод `downloadAsExcel()` объекта `Spreadsheet`.
4. Вызовите функцию `ezsheets.upload()` и передайте ей имя файла Excel.
5. `ss['Студенты']['B2']`
6. `ezsheets.getColumnLetterOf(999)`
7. С помощью свойств `rowCount` и `columnCount` объекта `Sheet`.
8. Вызовите метод `delete()` объекта `Sheet`. Удаление будет необратимым, если передать методу именованный аргумент `permanent=True`.
9. Функция `createSpreadsheet()` и метод `createSheet()` объекта `Spreadsheet` соответственно.
10. Модуль EZSheets будет приостанавливать вызовы методов.

Глава 15

1. Объект `File`, возвращаемый функцией `open()`.
2. В режиме чтения бинарных данных (`'rb'`) в случае функции `PdfFileReader()` и в режиме записи бинарных данных (`'wb'`) в случае функции `PdfFileWriter()`.
3. `getPage(4)` (нумерация страниц ведется с `0`).
4. `numPages`.
5. Вызвать функцию `decrypt('swordfish')`.
6. `rotateClockwise()` и `rotateCounterClockwise()`. Угол поворота в градусах задается в виде целочисленного аргумента.
7. `docx.Document('demo.docx')`
8. Документ содержит множество абзацев, представляемых объектами `Paragraph`. Абзац начинается с новой строки и состоит из нескольких фрагментов с различными стилями форматирования. Каждому такому фрагменту соответствует объект `Run`.
9. `doc.paragraphs`
10. `Run` (у объекта `Paragraph` их нет).
11. Установка значения `True` приводит к тому, что к объекту `Run` всегда будет применяться полужирное начертание. В случае значения `False` полужирное начертание никогда не будет применяться, какой бы ни была настройка стиля. В случае значения `None` к объекту `Run` применяется полужирное начертание в соответствии с установленным для него стилем.
12. `docx.Document()`
13. `doc.add_paragraph('Hello, there!')`
14. Целочисленные значения `0`, `1`, `2`, `3` и `4`.

Глава 16

1. В электронных таблицах Excel могут храниться значения нестроковых типов данных. Ячейки в Excel могут иметь различные настройки шрифтов и цвета, ширину и высоту ячеек можно менять, а смежные ячейки можно объединять. В таблицы Excel можно внедрять изображения и диаграммы.
2. Этим функциям передается объект `File`, возвращаемый функцией `open()`.
3. Объекты `File` необходимо открывать в режиме чтения бинарных данных (`'rb'`) для объектов `reader` и в режиме записи бинарных данных (`'wb'`) для объектов `writer`.

4. `writerow()`.
5. Аргумент `delimiter` заменяет строку, используемую в качестве разделителя ячеек в таблице. Аргумент `lineterminator` заменяет строку, используемую в качестве разделителя строк таблицы.
6. `json.loads()`.
7. `json.dumps()`.

Глава 17

1. Начальная точка отсчета времени, используемая многими программами, которые работают с датой и временем. Ей соответствует полночь 1 января 1970 года по шкале UTC.
2. `time.time()`.
3. `time.sleep(5)`.
4. Она возвращает целое число, ближайшее к переданному аргументу. Например, функция `round(2.4)` вернет значение `2`.
5. Объект `datetime` представляет определенный момент времени. Объект `timedelta` представляет промежуток времени.
6. `datetime.datetime(2019, 1, 7).weekday()`. Эта функция вернет `0`, что соответствует понедельнику.
7.

```
threadObj = threading.Thread(target=spam)
threadObj.start()
```

8. Следует убедиться в том, что код, выполняющийся в одном потоке, не читает и не записывает переменные, доступные для другого потока.

Глава 18

1. SMTP и IMAP соответственно.
2. `smtplib.SMTP()`, `smtpObj.ehlo()`, `smptObj.starttls()` и `smtpObj.login()`.
3. `imapclient.IMAPClient()` и `imapObj.login()`.
4. Список строк, содержащих поисковые ключи IMAP, такие как `'BEFORE` *`дата`*`'`, `'FROM` *`строка`*`'` или `'SEEN'`.
5. Необходимо присвоить переменной `imaplib._MAXLINE` большое целое число, например `10000000`.
6. `pyzmail`.
7. Эти файлы сообщают модулю EZGmail о том, какую учетную запись Google следует использовать при доступе к Gmail.

8. Сообщение представляет собой одно электронное письмо, тогда как цепочка представляет собой серию ответных писем.
9. Включите текст `'has:attachment'` в строку, передаваемую методу `search()`.
10. Идентификатор учетной записи Twilio (SID), токен аутентификации и телефонный номер пользователя Twilio.

Глава 19

1. Это кортеж из четырех целых чисел, каждое из которых может иметь значение в диапазоне от `0` до `255`. Они соответствуют доле красной, зеленой и синей компонент, а также альфа-составляющей (прозрачности) в цвете.
2. Функция `ImageColor.getcolor('CornflowerBlue', 'RGBA')` вернет для этого цвета RGBA-значение `(100, 149, 237, 255)`.
3. Это кортеж из четырех целых чисел: координата *x* левой стороны прямоугольника, координата *y* верхней стороны прямоугольника, а также ширина и высота прямоугольника.
4. `Image.open('zophie.png')`
5. `imageObj.size` — это кортеж из двух целых чисел, задающих ширину и высоту изображения.
6. `imageObj.crop((0, 50, 50, 50))`. Заметьте, что методу `crop()` передается кортеж прямоугольника, а не четыре отдельных целочисленных аргумента.
7. `imageObj.save('new_filename.png')`
8. `ImageDraw`.
9. Методы, предназначенные для рисования фигур, в частности `point()`, `line()` и `rectangle()`, есть у объекта `ImageDraw`. Этот объект можно создать с помощью функции `ImageDraw.Draw()`.

Глава 20

1. Для этого следует переместить указатель мыши в левый верхний угол экрана, т.е. в позицию с координатами `(0, 0)`.
2. `pyautogui.size()`.
3. `pyautogui.position()`.
4. Функция `moveTo()` перемещает указатель мыши в позицию с заданными абсолютными координатами на экране, тогда как функция `move()` перемещает указатель относительно его текущей позиции.
5. `pyautogui.dragTo()` и `pyautogui.drag()`.

6. `pyautogui.write('Hello world!')`
7. Для этого следует либо передать список названий клавиш (таких, как `'left'`) функции `pyautogui.write()`, либо передать обозначение клавиши функции `pyautogui.press()`.
8. `pyautogui.screenshot('screenshot.png')`
9. `pyautogui.PAUSE = 2`
10. Selenium.
11. Функции модуля PyAutoGUI не знают, в правильном ли месте выполняется щелчок или нажимается клавиша. Неожиданные всплывающие окна или смена фокуса ввода могут сбить сценарий с толку и привести к его досрочному завершению.
12. `pyautogui.getWindowsWithTitle('Блокнот')`
13. Выполните инструкцию `w = pyatuogui.getWindowsWithTitle('Firefox')`, а затем `w.activate()`.

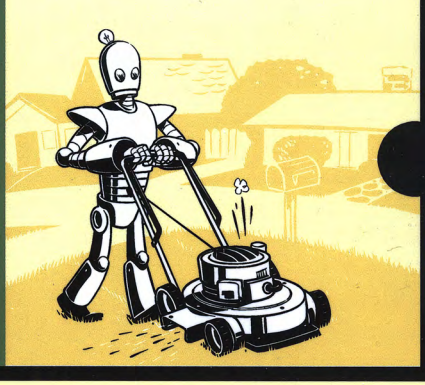
ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

П

Р

С

ИЗУЧАЙТЕ PYTHON И РЕШАЙТЕ ЗАДАЧИ!



ВСЕМИРНЫЙ БЕСТСЕЛЛЕР!

В полностью переработанном втором издании книги вы узнаете, как использовать Python для написания программ, способных за минуту сделать то, на что раньше уходили часы ручного труда, причем никакого опыта программирования не потребуется! Вы освоите основы Python и исследуете стандартную библиотеку модулей, позволяющих решать самые разнообразные задачи, такие как сбор данных с веб-сайтов, чтение документов в формате PDF и Word и автоматизация щелчков мыши.

Во второе издание книги включена новая глава, посвященная проверке вводимых данных. Также рассмотрены вопросы автоматизации работы с приложениями Gmail и Google Таблицы и даны советы по автоматическому обновлению CSV-файлов. Вы узнаете, как писать программы, которые легко выполняют следующие задачи автоматизации:

- поиск текста в файле или в нескольких файлах;
- создание, обновление, перемещение и переименование файлов и папок;
- поиск в Интернете и загрузка веб-контента;
- обработка и шифрование PDF-документов;
- рассылка электронной почты и текстовых уведомлений;
- заполнение веб-форм.

Пошаговые инструкции помогут лучше понять, как работает та или иная программа, а обновленные учебные проекты в конце каждой главы дадут возможность применить полученные навыки для автоматизации аналогичных задач.

Не тратьте время на выполнение рутинной работы. Даже не имея опыта опыта программирования, благодаря данной книге вы вполне сможете заставить компьютер сделать все необходимое!

ОБ АВТОРЕ

Эл Свейгарт — профессиональный разработчик, преподает программирование для детей и взрослых. Автор целого ряда книг по Python для начинающих.

Рассмотрен Python 3.*x*

Категория: компьютерные технологии/Python

ISBN 978-5-907365-55-1

21067

9 785907 365551

ДИАЛЕКТИКА

www.williamspublishing.com

no starch press

www.nostarch.com