Зед А. Шоу

3-е издание

Легкий способ выучить Python

Уникальная методика обучения программированию для начинающих


Мировой
компьютерный
бестселлер

Zed A. Shaw

Third Edition

LEARN PYTHON THE HARD WAY

Addison
Wesley

Зед А. Шоу

3-е издание

Легкий способ выучить Python

Москва
2017

УДК 004.43
ББК 32.973.26-018.1
Ш81

Шоу, Зед.

Ш81 Легкий способ выучить Python / Зед Шоу ; [пер. с англ. М. А. Райтмана]. — Москва : Издательство «Э», 2017. — 352 с. — (Мировой компьютерный бестселлер).

ISBN 978-5-699-98251-6

Никогда не программировали, но мечтаете научиться? Знаменитая на весь мир авторская методика Зеда Шоу поможет вам сделать первые шаги в освоении одного из самых увлекательных и востребованных сегодня языков — Python. Читайте упражнения, копируйте примеры и запускайте свои первые программы легко!

УДК 004.43
ББК 32.973.26-018.1

ISBN 978-5-699-98251-6

Содержание

Предисловие автора

Эта простая книга предназначена для обучения вас программированию с нуля. Хотя ее название* звучит как «трудный способ выучить Python», на самом деле это не так. Слово «трудный» используется потому, что в книге применена техника обучения, называемая инструкцией. Инструкции заключаются в созданных мной упражнениях, следующих одно за другим, они закрепляют навыки программирования благодаря повторению. Этот метод обучения весьма эффективен для новичков, которым необходимо приобрести базовые навыки, прежде чем приступить к освоению более сложных тем. Такой метод обучения, кстати, используется в разных сферах, от боевых искусств и музыки до элементарной математики и обучения чтению.

Эта книга заложит основу и укрепит ваши навыки программирования на Python с использованием техник практики и запоминания, позволяя постепенно продвигаться к решению все более сложных задач. К концу книги вы приобретете знания, необходимые для изучения более сложных тем программирования. Я бы сказал, что моя книга дает вам «черный пояс программиста». Это значит, что прочитав ее, вы приобретете навыки, достаточные для начала программирования.

Если вы будете усердно, не торопясь, работать, то выучите Python.

Благодарности

Я хотел бы поблагодарить Анджелу за помощь, оказанную мне с первыми двумя изданиями этой книги. Без этого я бы, наверное, не смог ее написать. Она редактировала первые черновики книги и поддерживала меня, пока я работал.

Я также хотел бы поблагодарить Грэга Ньюмана за создание обложек первых двух изданий, Брайана Шумейта за дизайн моего веб-сайта и всех читателей предыдущих изданий этой книги, которые нашли время, чтобы отправить мне отзывы и замечания.

Спасибо.

---

* В оригинале книга называется Learn Python the Hard way. — *Прим. ред.*

Трудный путь на самом деле прост

Читая эту книгу и изучая язык программирования, вы будете выполнять невероятно простые действия, через которые прошли все программисты.

1. Изучайте каждое упражнение.
2. С *точностью* вводите код каждого примера.
3. Запускайте программу.

И все. Процесс может быть довольно сложным поначалу, но все равно придерживайтесь курса. Если вы будете читать эту книгу и выполнять упражнения хотя бы пару часов в сутки, то приобретете отличную основу для перехода к изучению других книг. С помощью этого издания вы, возможно, и не получите исчерпывающие навыки программирования, но точно освоите основы, необходимые для дальнейшего изучения языка Python.

Суть книги в том, чтобы привить вам три самых основных навыка, необходимых каждому начинающему программисту: чтение и ввод, внимание к деталям и нахождение различий.

Чтение и ввод

Без сомнения, если у вас есть проблемы с вводом команд, вам будет сложно разобраться с кодом, особенно если вы набираете лишние символы. Без этого простого навыка вы не сможете даже на элементарном уровне разобраться, как работает программное обеспечение.

Набирая примеры кода и выполняя их, вы запоминаете названия элементов языка программирования и учитесь читать код правильно.

Внимание к деталям

Еще один навык, который отличает хороших программистов от плохих, — это внимание к деталям. На самом деле, он применим в любой профессии. Не обращая внимания на мельчайшие детали в вашей работе, вы пропускаете ключевые элементы результата своего труда. В программировании отсутствие

такого навыка чревато ошибками и трудностями, которые возникнут впоследствии при использовании вашей программы.

Читая книгу и в точности воспроизводя каждый пример, вы натренируетесь отслеживать детали каждого своего шага.

Обнаружение различий

Очень важный навык, который у большинства программистов развивается с течением времени, — это способность обращать внимание на различия между двумя фрагментами кода. Опытный программист может сразу определить мельчайшие различия. Существуют программные инструменты, позволяющие упростить и автоматизировать процесс сравнения, но мы не будем их использовать. Сначала вы должны пойти трудным путем — научиться определять различия глазами, а потом уже можно будет пользоваться инструментами.

Во время выполнения упражнений и набора кода вы будете делать ошибки. Это неизбежно; даже опытные программисты совершают их. Ваша задача — сравнить исходный код и написанный вами, а затем устранить все недочеты. Поступая таким образом, вы будете тренироваться находить опечатки, ошибки, баги и другие проблемы кода.

Никакого копирования/вставки

Необходимо вводить код каждого упражнения вручную. Смысла в копировании/вставке нет, иначе можно просто не выполнять эти упражнения. Суть их заключается в получении навыков чтения, написания и понимания кода. Если копировать и вставлять код, эффективность упражнений сойдет на нет.

О практике и настойчивости

Пока вы учитесь программировать, я учусь играть на гитаре. Сначала я тренировался около двух часов каждый день. Я играл звукоряды, аккорды и арпеджио в течение часа, а затем изучал теорию музыки, сольфеджио, песни и все остальное. Спустя несколько дней я стал тренироваться играть на гитаре и изучать музыку уже восемь часов в день, потому что проникся и понял, как это весело. Для меня повторение — естественный и простой способ чему-то

научиться. И я знаю: чтобы достичь хороших результатов, нужно практиковаться каждый день, даже если порой хочется прогулять занятие (что бывает частенько) или же процесс обучения продвигается с трудом. Продолжайте пытаться, и однажды все станет проще и веселее.

Когда вы изучите эту книгу и продолжите программировать, не забывайте, что на первых порах может быть трудно. Возможно, вы боитесь неудачи, поэтому сдаетесь при первой проблеме. А может, вам не хватает самодисциплины, и вы просто ничего не делаете, потому что «это скучно». Или же вы уверены в собственной «одаренности», поэтому все, способное выставить вас глупым или неопытным, не стоит того, чтобы попытаться. Вполне возможно, разумеется, что я несправедливо сравнил вас с собой, человеком, который занимается программированием вот уже 20 лет.

И тем не менее не важно, почему именно вы собираетесь бросить обучение, — *продолжайте его*. Заставьте себя. Если вы безуспешно пытаетесь выполнить практические задания или просто не понимаете тему упражнения, пропустите его и вернитесь к нему позже. Продолжайте обучение, потому что для программирования это обычная ситуация.

В самом начале вы ничего не поймете — это нормально, как и при изучении любого языка. Вы будете сражаться со словами, не понимать предназначения символов, и весь код будет чудовищно запутанным. Но в один прекрасный день — БАХ! — ваш мозг разложит все по полочкам, и на вас снизойдет «озарение». Конечно, вы не превратитесь сразу же в гуру кодинга, но по крайней мере разберетесь, как работают языки программирования.

Если вы бросите обучение, то никогда не достигнете этого результата. Если же вы продолжите пытаться — вводить код, стараясь понять его и прочитать, вы в конце концов получите нужные навыки.

И даже если вы прочитали всю эту книгу, но до сих пор не понимаете, как работает код, — что ж, вы по крайней мере попытались. Можете сказать, что старались изо всех сил и даже больше, но не получилось. Но вы попытались. И вы можете гордиться этим!

Примечание для особенно умных читателей

Некоторые люди, знакомые с программированием, при чтении этой книги могут начать возмущаться. На самом деле, в ней нет ничего, что может оскорбить или принизить читателя. Я просто знаю о программировании больше,

чем *целевая* аудитория моей книги. Поэтому, если вы считаете себя уязвленными, прочитав эту книгу, то я здесь ни при чем — просто вы, по-видимому, не входите в *целевую* аудиторию.

Если вы читаете эту книгу и прерываетесь на каждом третьем предложении, потому что вам кажется, что я принижаю ваши умственные способности, прочитайте следующие три совета.

1. Прекратите читать мою книгу. Я писал ее не для вас. Я написал ее для людей, которые еще не знают все о программировании.

2. Освободитесь от знаний перед обучением. Вам будет сложно учиться под моим руководством, если вам уже известно все.

3. Изучайте Lisp. Я слышал, что люди, которые знают все, интересуются языком Lisp.

Все остальные, кто открыл эту книгу и только что прочел предисловие, заметят мою улыбку и озорной огонек в глазах.

Настройка

В этом упражнении нет примеров кода. Оно предназначено для подготовки компьютера к установке программного обеспечения. Вы должны в точности следовать приведенным ниже инструкциям. К примеру, в операционной системе macOS предустановлено программное обеспечение Python 2, поэтому не устанавливайте Python 3 (или любой другой версии).

Внимание. Если вы не знаете, как использовать программу Windows PowerShell (Windows), Терминал (Terminal) (macOS) или Bash (Linux), вам нужно изучить приемы работы с ними первым делом. Я написал «Экспресс-курс по оболочке командной строки», который вы найдете в конце книги. Изучите его первым делом, а затем вернитесь к этим упражнениям.

---

macOS

Для успешного завершения этого упражнения выполните следующие действия.

1. Перейдите на сайт **www.barebones.com/products/textwrangler** с помощью веб-браузера, скачайте текстовый редактор TextWrangler и установите его.

2. Поместите значок программы TextWrangler (установленного редактора) на панель **Dock**, чтобы быстро получать к нему доступ.

3. Выполните поиск программы Терминал (Terminal). Она расположена в каталоге *Утилиты (Utilities)*.

4. Поместите значок программы Терминал (Terminal) на панель **Dock**, чтобы быстро получать к нему доступ.

5. Запустите программу Терминал (Terminal), это несложно.

6. В окне программы Терминал (Terminal) выполните команду python. После ввода имени команды нажмите клавишу ↩.

7. Нажмите сочетание клавиш **T+Z**, затем клавишу ↩, чтобы завершить работу среды Python.

Вы вновь должны увидеть приглашение оболочки командной строки, похожее на то, которое видели при вводе команды python. Если приглашение не отобразилось, постарайтесь разобраться, по каким причинам это произошло.

8. Разберитесь, как в оболочке командной строки (программе Терминал (Terminal)) создать каталог.

9. Разберитесь, как в оболочке командной строки перемещаться по каталогам. Перейдите в один из них.

10. С помощью текстового редактора создайте файл в этом каталоге. Выберите команду меню **Сохранить** (Save) или **Сохранить как** (Save As) и выберите этот каталог.

11. Вернитесь в окно программы Терминал (Terminal), применив сочетание клавиш для переключения между окнами.

12. Вернувшись в оболочку командной строки, просмотрите содержимое каталога, чтобы увидеть сохраненный файл.

macOS: результат

Ниже представлен результат работы в оболочке командной строки на моем компьютере. В вашем случае результат будет отличаться, вы сможете проанализировать все различия и понять мои действия и то, что нужно сделать вам.

```
Last login: Sat Apr 2400:56:54 on ttys001
~ $ python
Python 2.5.1 (r251:54863, Feb 62009, 19:02:12)
[GCC4.0.1 (Apple Inc. build 5465)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> ^D
~ $ mkdir mystuff
~ $ cd mystuff
mystuff $ ls
# … Используйте TextWrangler для создания файла test.txt…
mystuff $ ls
test.txt
mystuff $
```

Windows

1. Перейдите на сайт **notepad-plus-plus.org** с помощью веб-браузера, скачайте текстовый редактор Notepad++ и установите его. Вам не нужно обладать правами администратора, чтобы сделать это.

2. Для быстрого доступа к Notepad++ поместите ярлык этой программы на рабочий стол и/или панель задач. Обе настройки можно выполнить во время установки.

3. Запустите оболочку PowerShell из меню Пуск (Start). Выполните при необходимости поиск этой программы и нажмите клавишу **Enter**, чтобы запустить ее.

4. Создайте ярлык PowerShell на рабочем столе и/или панели задач для удобства запуска.

5. В оболочке PowerShell запустите среду Python. Для этого просто наберите имя `python` и нажмите клавишу **Enter**.

   А. Если запуск среды Python не происходит (команда `python` не распознается оболочкой командной строки), скачайте ее дистрибутив с сайта **python.org/download**.

   Б. Убедитесь, что вы выбрали для скачивания и установки версию Python 2, а не Python 3.

   В. Как вариант, вам может подойти версия ActivePython, особенно если у вас нет прав администратора на компьютере, на котором вы работаете.

Примечание. Если при запуске инсталлятора Python 2 (msi-файла) установка прерывается ошибкой «There is a problem with this Windows Installer package. A DLL required for this install to complete could not be run. Contact your support personnel or package vendor», необходимо запустить msi-файл с правами администратора. Как это сделать, можно найти в Интернете (например, тут: **goo.gl/T19pfw**).

Г. Если после установки дистрибутива Python команда все еще не распознается, то в PowerShell выполните следующую команду: `[Environment]:: SetEnvironmentVariable("Path", "$env: Path; C:\Python27", "User")`. Завершите работу PowerShell, а затем запустите его снова, чтобы убедиться, что Python теперь работает. Если этого не произойдет, может потребоваться перезагрузка компьютера.

6. Введите команду `quit()` и нажмите клавишу **Enter**, чтобы завершить работу Python.

   Вы должны увидеть приглашение командной строки, похожее на то, что было при вводе команды `python`. Если приглашение не отобразилось, постарайтесь разобраться, по каким причинам это произошло.

7. Разберитесь, как в оболочке командной строки (программе PowerShell) создать каталог.

8. Разберитесь, как в оболочке командной строки перемещаться по каталогам. Перейдите в один из них.

9. С помощью текстового редактора создайте файл в этом каталоге. Выберите команду меню **Сохранить** (Save) или **Сохранить как** (Save As) и выберите этот каталог.

10. Вернитесь в окно программы PowerShell, нажав сочетание клавиш для переключения между окнами.

11. Вернувшись в оболочку командной строки, просмотрите содержимое каталога, чтобы увидеть сохраненный файл.

Внимание! Если после успешной установки Python на компьютере под управлением операционной системы Windows его запуск не происходит, это говорит о неправильной конфигурации переменной окружения. Для решения проблемы в PowerShell выполните команду `[Environment]::SetEnvironmentVariable("Path","$env: Path; C:\Python27", "User")`. Также потребуется перезагрузить оболочку PowerShell, чтобы убедиться, что Python теперь работает. Если этого не произойдет, может потребоваться перезагрузка компьютера.

Windows: результат

```
>python
ActivePython 2.6.5.12 (ActiveState Software Inc.) based on Python 2.6.5
(r265:79063, Mar 20 2010, 14:22:52) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> ^Z
>mkdir mystuff

>cd mystuff

... Используйте Notepad++ для создания файла test.txt в папке mystuff ...

>
   <несколько незначительных ошибок, если вы установили Python без прав
   администратора, — игнорируйте их и нажимайте Enter>
>dir
   Volume in drive C is
   Volume Serial Number is 085C-7E02

Directory of C:\Documents and Settings\you\mystuff

04.05.2010  23:32    <DIR>          .
04.05.2010  23:32    <DIR>          ..
04.05.2010  23:32                 6 test.txt
               1 File(s)              6 bytes
               2 Dir(s)  14 804 623 360 bytes free

>
```

Вы можете увидеть приглашение несколько иного вида, другие сведения о среде Python и прочие данные, но основную идею я продемонстрировал.

Linux

Linux — это операционная система, выпускаемая в различных модификациях, с разными способами установки программного обеспечения. Я полагаю, если вы работаете в Linux, то знаете, как устанавливать пакеты, поэтому приведу лишь краткие инструкции.

1. Запустите менеджер пакетов Linux и с помощью него установите текстовый редактор gedit.

2. Для быстрого доступа к gedit поместите ярлык этой программы на панель быстрого запуска.

   А. Запустите редактор gedit, чтобы изменить некоторые настройки этой программы.

   Б. Откройте окно **Параметры** (Preferences) и перейдите на вкладку **Редактор** (Editor).

   В. Присвойте параметру **Ширина табуляции** (Tab width) значение **4**.

   Г. Установите флажок **Вставлять пробелы вместо табуляций** (Insert spaces instead of tabs).

   Д. Установите флажок **Включить автоматический отступ** (Automatic indentation).

   Е. Перейдите на вкладку **Вид** (View) и установите флажок **Показывать номера строк** (Display line numbers).

3. Запустите оболочку командной строки. Она может называться GNOME Terminal, Konsole или xterm.

4. Поместите также и ярлык оболочки командной строки на панель быстрого запуска.

5. Запустите оболочку командной строки. Это несложно.

6. В окне оболочки командной строки выполните команду `python`. После ввода имени команды нажмите клавишу **Enter**.

   А. Если запуск среды Python не происходит (команда `python` не распознается оболочкой командной строки), установите ее. Убедитесь, что вы выбрали для скачивания и установки версию Python 2, а не Python 3.

7. Введите команду `quit()` и нажмите клавишу **Enter**, чтобы завершить работу Python.

   Вы вновь должны увидеть приглашение оболочки командной строки, похожее на то, которое видели при вводе команды `python`. Если приглашение не отобразилось, постарайтесь разобраться, по каким причинам это произошло.

8. Разберитесь, как в оболочке командной строки создать каталог.

9. Разберитесь, как в оболочке командной строки перемещаться по каталогам. Перейдите в один из них.

10. С помощью текстового редактора gedit создайте файл в этом каталоге. Выберите команду меню **Сохранить** (Save) или **Сохранить как** (Save As) и выберите этот каталог.

11. Вернитесь в окно оболочки командной строки, применив сочетание клавиш для переключения между окнами.

12. Вернувшись в оболочку командной строки, просмотрите содержимое каталога, чтобы увидеть сохраненный файл.

Linux: результат

```
$ python
Python 2.6.5 (r265:79063, Apr 1 2010, 05:28:39)
[GCC 4.4.3 20100316 (prerelease)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
$ mkdir mystuff
$ cd mystuff
# ... Используйте gedit для создания файла test.txt ...
$ ls
test.txt
$
```

Вы можете увидеть приглашение несколько иного вида, другие сведения о среде Python и прочие данные, но основную идею я продемонстрировал.

Начинающим

Вот вы и закончили это упражнение. Оно может показаться сложным в зависимости от вашего уровня знания компьютера. Если вы сталкиваетесь с трудностями при выполнении этого упражнения, потратьте время на тренировку и выполнение описанных действий. Ведь если вы не умеете делать эти

базовые операции, вам будет очень сложно при настоящем программировании в будущих упражнениях.

Если программисты советуют использовать программу vim или emacs, откажитесь. Эти редакторы будут уместны, когда вы достигнете более высокого уровня программирования. Все, что вам нужно сейчас, — программа, которая позволяет помещать текст в файл. Я рекомендую использовать текстовый редактор gedit, TextWrangler или Notepad ++, потому они просты и работают схожим образом на всех компьютерах. Профессиональные программисты тоже используют эти текстовые редакторы.

Программисты могут посоветовать вам изучать Python версии 3. Скажите им на это: «Когда весь код Python на компьютере будет написан в версии Python 3, я обязательно изучу ее». Так вы отделаетесь от подобных советов на ближайшее десятилетие.

Некоторые программисты также могут сказать вам, что следует использовать операционную систему macOS или Linux. В целом это личный выбор каждого. Существует ряд преимуществ при работе в Linux и macOS по сравнению с Windows, но они не являются критичными, и работать можно в той системе, в которой это удобно делать вам. Все, что вам нужно, это текстовый редактор, оболочка командной строки и Python.

И наконец, цель этого упражнения заключается в том, чтобы вы научились четырем навыкам, обязательным для выполнения дальнейших упражнений в книге.

1. Переписывать упражнения, используя ваш текстовый редактор (Gedit в среде Linux, TextWrangler на Mac или Notepad ++ в операционной системе Windows).

2. Выполнять упражнения, которые вы записали.

3. Решать проблемы с ними в случае появления ошибок.

4. Повторять все операции, чтобы запомнить их.

Все остальное будет лишь отвлекать, поэтому придерживайтесь этого плана.

Первая программа

Надеюсь, вы с пользой потратили время, выполняя предыдущее упражнение, и узнали, как установить и запустить текстовый редактор, запустить оболочку командной строки, и научились работать с ними. Если вы еще не сделали этого, то настойчиво рекомендую вернуться к упражнению 0. Первый и последний раз я начинаю упражнение с предупреждения о том, что вы не должны пропускать задания.

Как справиться с «кракозябрами» вместо кириллицы

Ввиду особенностей поддержки кириллицы оболочкой командной строки для правильного отображения русских букв следует учитывать несколько дополнительных настроек. Во-первых, файлы с расширением *.py* должны сохраняться в программе Notepad++ в формате UTF-8 без BOM (BOM — специальная метка порядка следования байтов в файле, с которой файлы с расширением *.py* не будут обработаны оболочкой командной строки). Для этого для каждого *.py* файла в программе Notepad++ следует выбрать команду меню **Кодировки → Кодировка в UTF-8 без BOM** (Encoding → Encode in UTF-8 without BOM), а затем сохранить файл. Во-вторых, в начале каждого *.py* файла необходимо прописать кодировку следующим образом: `# -*- coding: utf- 8 -*-`. И, в-третьих, каждую строку текста с кириллицей нужно предварять символом `u`, вот так: `print u"Привет, мир!"` (вместо `print "Привет, мир!"`). Текст на латинице и с прописанной кодировкой и символами `u` отображается нормально.

Введите следующий текст в файл с именем *ex1.py*. Важно соблюдать расширение, так как Python лучше всего работает с файлами, имена которых заканчиваются на .py.

ex1.py

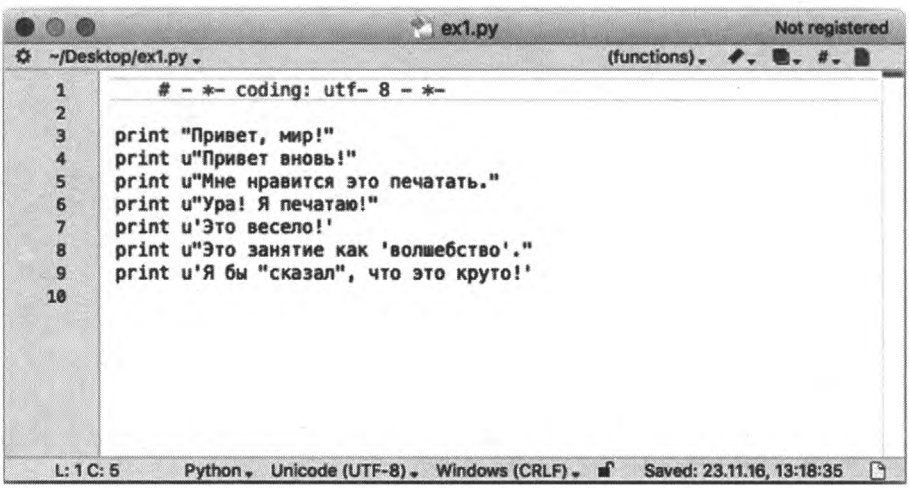
```
1   # -*- coding: utf- 8 -*-
2
3   print u"Привет, мир!"
4   print u"Привет вновь!"
5   print u"Мне нравится это печатать."
6   print u"Ура! Я печатаю!"
```

```
7   print u'Это весело!'
8   print u"Это занятие как 'волшебство'."
9   print u'Я бы "сказал", что это круто!'
```

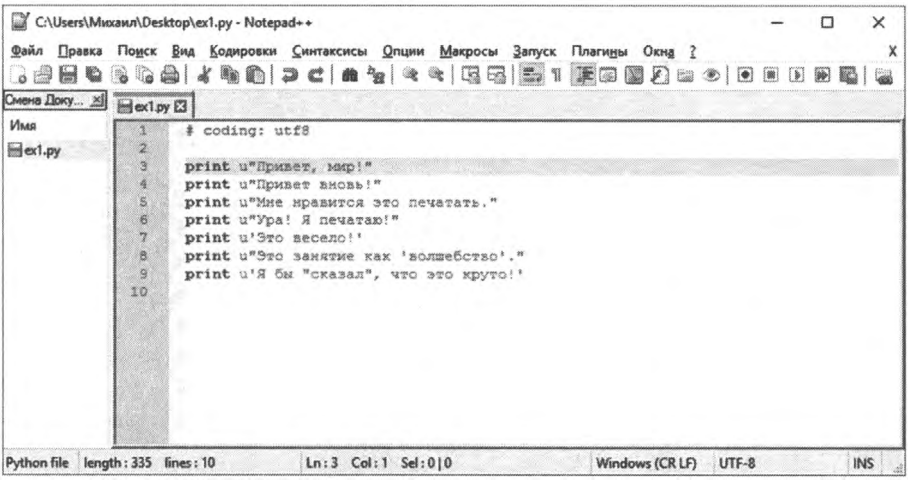
В следующих примерах кода в книге мы не будем указывать строку под номером 1 с кодировкой в начале файла, но вы должны будете вводить ее во всех файлах, содержащих кириллический текст!

Внимание! Если вам все же лень вручную набирать код примеров из этой книги, все файлы с кодом вы можете скачать по адресу **https://eksmo.ru/files/Shaw_Python.zip**.

Если вы работаете на компьютере под управлением операционной системы macOS, код, набранный в программе TextWrangler, будет выглядеть примерно так.



Если вы приверженец операционной системы Windows и программы Notepad++, вы увидите следующее.



Не беспокойтесь, если интерфейс редактора на вашем компьютере выглядит несколько иначе; сейчас главное — уяснить основные моменты.

1. Обратите внимание, что номера строк слева вводить не нужно. Они напечатаны в книге, чтобы вам было проще находить нужные строки, когда я пишу ссылки типа «См. строку 5...». В сценариях Python их вводить не нужно!

2. Оформление кода цветом может различаться в разных редакторах. Не обращайте на это внимания — важны набираемые символы.

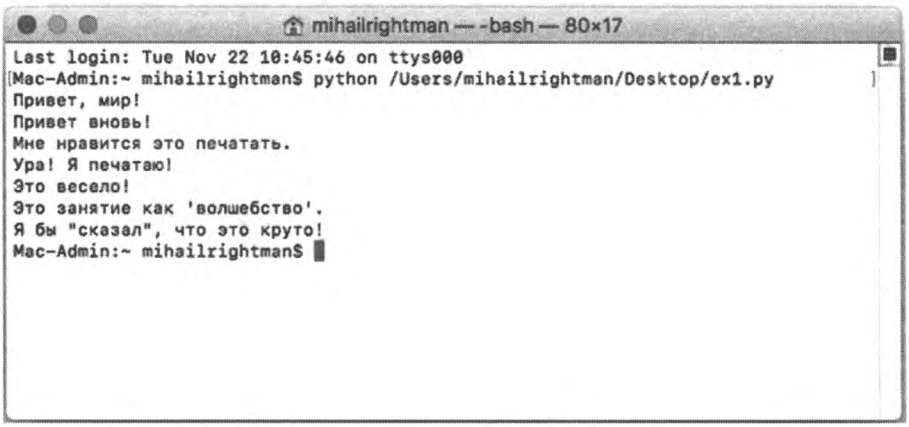
Сохраните файл. Затем в терминале запустите файл, выполнив следующую команду:

```
python ex1.py
```

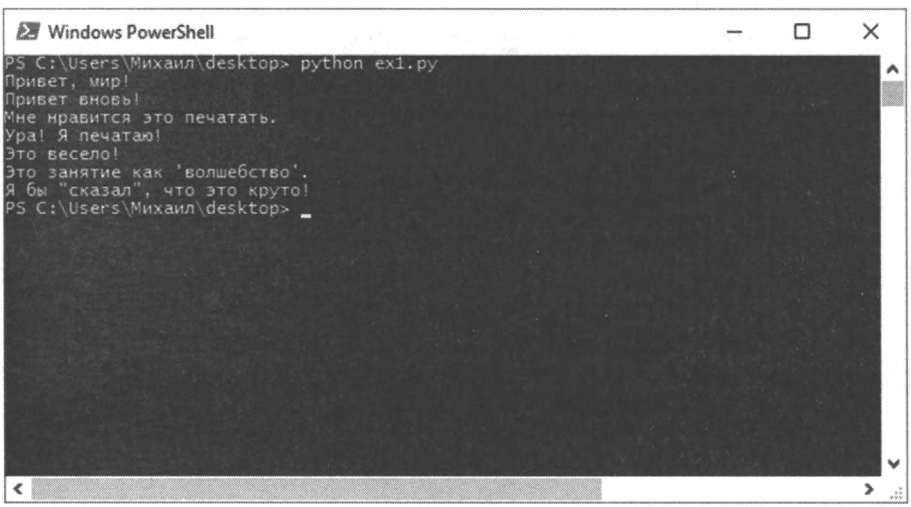
Если вы все сделали правильно, то увидите тот же результат, что представлен на рисунке. Если нет — вы где-то ошиблись. И не надо думать, что ошибся компьютер.

Результат выполнения

В операционной системе macOS в окне программы Терминал (Terminal) вы увидите следующее.



На компьютере под управлением операционной системы Windows в оболочке PowerShell результат будет таким.



Вы можете увидеть другие имена папок, компьютера или других элементов перед командой `python ex1.py`, это не важно. Важно то, что вы вводите команду и должны увидеть результат ее выполнения — такой же, как и у меня.

Если в код закралась ошибка (опечатка), результат будет выглядеть примерно так:

```
$ python ex1.py
File "ex1.py", line 5
    print u"Мне нравится это печатать.
                                      ^
SyntaxError: EOL while scanning string literal
```

Важно, чтобы вы понимали, в чем причина тех или иных ошибок, так как вы будете делать многие из них. Даже я совершаю ошибки. Давайте проанализируем приведенный выше вывод по строкам.

1. В первой строке мы выполнили в оболочке команду `python`, сославшись на файл сценария *ex1.py*.

2. Python сообщил, что в строку 5 файла *ex1.py* закралась ошибка.

3. Далее отображена строка с ошибкой.

4. Символ ^ (вставки) указывает на точную позицию проблемного участка кода. Обратите внимание, что в строке не хватает символа закрывающей кавычки ".

5. И, наконец, он вывел строку `SyntaxError` (ошибка синтаксиса) и сообщил о типе ошибки. Как правило, это сообщение выглядит очень таинственно, но, если скопировать его в поисковую систему и погуглить, вы найдете единомышленников, столкнувшихся с такой же ошибкой, и, вероятно, узнаете, как ее исправить.

Внимание! Напоминаю: если вы набираете текст в коде на другом языке, кроме английского, и получаете сообщения об ошибках или непонятные «кракозябры», в верхней части ваших сценариев Python помещайте строку `# -*- coding: utf-8 -*-`, а текстовые элементы предваряйте символом `u` (см. начало упражнения). Так вы справитесь с проблемой, применив кодировку Юникод UTF-8 в своих сценариях.

Практические задания

Каждое упражнение содержит практические задания. Задания необходимо выполнять для закрепления пройденного материала. Если вы не можете их выполнить сейчас, пропустите и вернитесь к ним позже.

В этом упражнении попробуйте выполнить следующие действия.

1. Добавьте в ваш сценарий еще одну строку текста, которую Python должен вывести с помощью команды `print`.

2. Измените код своего сценария так, чтобы Python выводил на экран только одну из строк.

3. Укажите символ # (решетку) в начале строки. Что произойдет? Постарайтесь выяснить, для чего предназначен этот символ.

Далее в упражнениях я не буду объяснять, как их следует выполнять, если нет никаких исключений.

Примечание. Символ решетки также называют «октоторпом», «хешем», «знаком номера» и другими именами. Выберите то, которое вам по душе.

Распространенные вопросы

Эти вопросы действительно задавались людьми при чтении этой книги в Интернете. Вы можете столкнуться с некоторыми из них, поэтому я собрал вопросы и ответил на них для вас.

Могу ли я использовать IDLE*?

Нет, вы должны использовать программу Терминал (Terminal) в операционной системе macOS и PowerShell в Windows, как и я. Если вы не знаете, как пользоваться ими, перейдите в конец книги и прочитайте приложение «Экспресс-курс по оболочке командной строки».

---

* Integrated DeveLopment Environment — интегрированная среда разработки на языке Python. *Прим. пер.*

Почему в ваших примерах в редакторе код выделен разными цветами?

Сохраните файл с расширением *.py*, например *ex1.py*. После этого весь вводимый код будет отображаться в цвете.

Я получаю ошибку синтаксиса SyntaxError, когда запускаю файл *ex1.py*. Что делать?

Вероятно, вы запустили Python, а затем пытаетесь вновь запустить его с помощью команды `python`. Завершите работу оболочки, запустите ее снова и сразу введите только одну строку — `python ex1.py`.

Я не могу открыть файл *ex1.py*. Отображается ошибка [Errno 2] No such file or directory. Что делать?

Эта ошибка означает, что файл не обнаружен. Вы должны находиться том же каталоге, что и созданный вами файл. Убедитесь, что перешли в этот каталог, с помощью команды `cd`. Например, если вы сохранили файл в каталог *folder/ex1.py*, то нужно первым делом выполнить команду `cd folder`, прежде чем пытаться запустить команду `python ex1.py`. Если вы не в состоянии перемещаться по каталогам, изучите приложение «Экспресс-курс по оболочке командной строки», упоминаемое в первом вопросе.

Как в моем файле отобразить символы на языке моей страны?

Убедитесь, что в верхней части файла указана строка `# -*- coding: utf-8 -*-`, а текстовые элементы предваряйте символом `u` (см. начало упражнения).

Мой файл не запускается. Отображается только приглашение без какого-либо вывода. Что делать?

Вы скорее всего попробовали ввести только текст `"Привет, мир!"` без команды `print`. Код должен быть введен в точности, как показано в листингах и на снимках экрана. Сверьте свой код с моим; он должен работать.

УПРАЖНЕНИЕ 2

Комментарии и символы

Комментарии в коде программ очень важны. Они используются, чтобы на человеческом языке сообщить какую-нибудь информацию, а также отключить (закомментировать) части кода вашей программы, если нужно, чтобы он временно не работал. Ниже показано, как комментарии используются в языке Python.

ex2.py

```
1  # Комментарии позволяют разобраться в предназначении кода.
2  # Строки после символа # игнорируются python.
3
4  print u"Это моя программа." # этот комментарий игнорируется
5
6  # С помощью комментариев можно "отключать" части кода:
7  # print "Код не работает."
8
9  print u"Код работает."
```

Начиная с этого упражнения, код будет приводиться так, как показано в этом примере. Важно, чтобы вы воспринимали код правильно, разделяя сам код и комментарии. Ваш текстовый редактор или оболочка командной строки могут выглядеть по-разному, но важно лишь то, что вы набираете в текстовом редакторе. Я могу работать в любом текстовом редакторе, и результаты выполнения сценариев будут идентичными.

Результат выполнения

Сеанс упражнения 2

```
$ python ex2.py
Это моя программа.
Код работает.
```

Повторюсь, я не буду приводить результаты выполнения кода во всех возможных оболочках командной строки. Вы должны понимать, что вывод из примера выше на вашем компьютере может выглядеть несколько иначе. Важно то, что отображается после строки `$ python ex2.py`.

Практические задания

1. Удостоверьтесь, что понимаете, как используется символ # и как он называется (решетка или октоторп).

2. Изучите код из примера *ex2.py*, просматривая каждую строку в обратном порядке. Начните с последней строки и, проверяя каждое слово, разберитесь, что вы должны ввести.

3. Обнаружили ошибки? Устраните их.

4. Прочитайте вслух код, введенный ранее, проговаривая название каждого символа. Опять обнаружили ошибки? Устраните их.

Распространенные вопросы

Почему символ # называется решеткой?

Я называю этот символ октоторпом, и это единственное его название, которое не используется ни в одной стране. Программисты из разных стран используют и разные названия символа #. Вы можете выбрать любое название для этого символа — «решетка», «октоторп», «хеш» или «знак номера». Это сейчас не так важно, как необходимость научиться читать и понимать код.

Если символ # используется для комментариев, то почему код `# -*- coding: utf-8 -*-` работает?

Python также игнорирует эту строку как код, но она используется в качестве так называемого «хака», или обходного пути для решения проблем, связанных с настройкой и распознаванием формата файла. Подобные комментарии также используются для настройки редактора.

Почему символ # в строке `print "Привет # всем."` не игнорируется?

В этом коде символ # находится внутри строки, перед заключительной кавычкой ". Этот символ решетки отображается обычным текстом и не считается началом комментария.

Как создать комментарий из нескольких строк?

Укажите символ # в начале каждой строки.

Не могу найти символ # на клавиатуре.

На клавиатурах в некоторых странах используется сочетание с клавишей-модификатором **Alt** для ввода специальных символов. С помощью поиска в Интернете выясните, как ввести символ решетки на вашей клавиатуре.

Почему нужно читать код в обратном направлении?

Этот прием позволяет вашему мозгу воспринимать код в виде несвязанных частей и обрабатывать их более точно. Удобная техника для отлова ошибок.

Числа и математика

Каждый язык программирования определенным образом обрабатывает числа и математические действия. Не переживайте: программисты часто говорят о том, что они гениальные математики, хотя таковыми на самом деле не являются. Если бы они были математиками, они бы и занимались математикой, а не писали игры для соцсетей и рекламные сервисы.

Это упражнение содержит несколько математических символов. Я назову их сразу, чтобы вы запомнили их названия. Введите каждый из них, произнося вслух название. Когда вы их запомните, можете перестать называть. Итак, запомните следующие математические знаки:

+	плюс
-	минус
/	слеш
*	звездочка
%	процент
<	меньше
>	больше
<=	меньше или равно
>=	больше или равно

Обратите внимание, что в списке не указаны действия, которые выполняет каждый из операторов. Изучив код этого упражнения, вернитесь к списку и разберитесь, для выполнения какого действия предназначен каждый арифметический оператор. Например, оператор + выполняет сложение

ex3.py

```
print u"Я подсчитаю свою живность:"

print u"Куры", 25 + 30 / 6
print u"Петухи", 100-25 * 3 % 4

print u"А теперь подсчитаю яйца:"

print 3 + 2 + 1-5 + 4 % 2-1 / 4 + 6

```

```
10  print u"Верно ли, что 3 + 2 < 5-7?"
11
12  print 3 + 2 < 5-7
13
14  print u"Сколько будет 3 + 2?", 3 + 2
15  print u"А сколько будет 5-7?", 5-7
16
17  print u"Ой, я, кажется, запутался… Почему False?"
18
19  print u"Еще раз посчитаю…"
20
21  print u"Это больше?", 5 > - 2
22  print u"А это больше или равно?", 5 >= - 2
23  print u"А это меньше или равно?", 5 <= - 2
```

Результат выполнения

Сеанс упражнения 3

```
$ python ex3.py
Я подсчитаю свою живность:
Куры 30
Петухи 97
А теперь подсчитаю яйца:
7
Верно ли, что 3 + 2 < 5-7?
False
Сколько будет 3 + 2? 5
А сколько будет 5-7? -2
Ой, я, кажется, запутался… Почему False?
Еще раз посчитаю…
Это больше? True
А это больше или равно? True
А это меньше или равно? False
```

Практические задания

1. Выше каждой строки кода напишите комментарий (предваряя текст комментария символом #) для себя, указывая, что делает данный код.

2. Помните, вы в упражнении 0 запускали Python? Вновь запустите Python, как там описано, и при помощи вышеуказанных символов используйте оболочку командной строки в качестве калькулятора.

3. Придумайте вычисление посложнее и создайте новый файл *.py*, который выполнит его.

4. Чего не хватает в математических вычислениях? В них не используются дроби, только целые числа. Изучите этот вопрос, проанализировав понятие «плавающей точки».

5. Перепишите код из примера *ex3.py*, используя числа с плавающей точкой, чтобы произвести более точные вычисления (подсказка: `20.0` — число с плавающей точкой).

Распространенные вопросы

Почему символ % обозначается как «деление по модулю», а не «процент»?

Если кратко, то разработчики просто решили использовать именно этот символ для данного действия. Обычно вы читаете и используете его как «процент», и вы правы. В программировании вычисление процентов, как правило, выполняется с помощью оператора деления (`/`). Деление по модулю — другая операция, в качестве оператора в которой используется символ %.

Как выполняется деление по модулю?

Эта операция вычисляет остаток от деления. Например, *X* делится на *Y* с остатком *J*. Например, 100 делится на 16 с остатком 4. Результатом деления по модулю является значение *J*, или остаток.

Каков порядок вычислений?

Общепринятый порядок вычислений выглядит так: Скобки, Экспоненты (степени), Умножение, Деление, Сложение и Вычитание. Чтобы проще было

запомнить, можно сократить до *С.Э.У.Д.П.В.* и придумать маленькое предложение: «*Смотрите, Эти Умельцы Делают Порхающий Велосипед*». В Python используется этот же порядок вычислений.

Почему при использовании оператора деления (/) результат округляется?

На самом деле округления не происходит; дробная часть после точки отбрасывается. Попробуйте выполнить операцию `7.0 / 4.0`, а затем `7 / 4`, и вы увидите разницу.

Переменные и имена

Вы уже можете выполнять вычисления и выводить текст с помощью команды `print`. Теперь разберемся с переменными. В программировании переменная является всего лишь именем для чего-нибудь. Программисты используют переменные, чтобы код был похож на человеческий язык и его проще было читать. Если бы удобных имен переменных не существовало, программисты бы заблудились в дебрях собственного кода.

Если вы испытываете сложности при выполнении этого упражнения, вспомните советы по поиску различий и сосредоточения на деталях.

1. Выше каждой строки кода напишите комментарий для себя (предваряя текст комментария символом #), указывая, что делает данный код.

2. Прочитайте код файла *.py* в обратном направлении.

3. Прочитайте код файла *.py* вслух, произнося названия специальных символов.

ex4.py

```
cars = 100
space_in_a_car = 4.0
drivers = 30
passengers = 90
cars_not_driven = cars - drivers
cars_driven = drivers
carpool_capacity = cars_driven * space_in_a_car
average_passengers_per_car = passengers / cars_driven


print u"В наличии", cars, u"автомобилей."
print u"И только", drivers, u"водителей вышли на работу."
print u"Получается, что", cars_not_driven, u"машин пустуют."
print u"Мы можем перевезти сегодня", carpool_capacity, u"человек."
print u"Сегодня нужно отвезти", passengers, u"человек."
print u"В каждой машине будет примерно",
average_passengers_per_car, u"человека."
```

Примечание. В имени переменной `space_in_a_car` используется символ нижнего подчеркивания. Разберитесь, как ввести его. Этот символ используется вместо пробела между словами в именах переменных.

Примечание. В именах переменных во избежание ошибок не следует использовать русские буквы.

Результат выполнения

Сеанс упражнения 4

```
$ python ex4.py
В наличии 100 автомобилей.
И только 30 водителей вышли на работу.
Получается, что 70 машин пустуют.
Мы можем перевезти сегодня 120.0 человек.
Сегодня нужно отвезти 90 человек.
В каждой машине будет примерно 3 человека.
```

Практические задания

Когда я написал эту программу впервые, то допустил ошибку, и Python сообщил мне об этом следующим образом:

```
Traceback (most recent call last):
    File "ex4.py", line 8, in <module>
      average_passengers_per_car = car_pool_capacity / passenger
    NameError: name 'car_pool_capacity' is not defined
```

Опишите ошибку своими словами. Убедитесь, что используете номера строк, и поясните, зачем вы это делаете.

Дополнительные практические задания.

1. Я присвоил значение `4.0` переменной `space_in_a_car`, почему? Что произойдет, если указать значение `4`?

2. Как вы уже знаете, значение 4.0 является «числом с плавающей точкой». Объясните, что это значит.

3. Укажите комментарии выше каждой строки с присвоением значения переменным.

4. Знаете ли вы, что символ = (равно) называется оператором присваивания? Как он функционирует?

5. Запомните, что _ — это символ подчеркивания.

6. Используя Python в качестве калькулятора, как вы делали это ранее, примените в вычислениях имена переменных. Популярные имена переменных включают в себя `i`, `x` и `j`.

Распространенные вопросы

В чем разница между оператором = (равно) и == (двойное равно)?

Оператор присваивания = (одиночный символ равенства) присваивает значение, указанное справа, переменной, указанной слева. Оператор сравнения == (двойной символ равенства) проверяет, имеют ли оба операнда одинаковое значение. Подробнее вы узнаете об этом в упражнении 27.

Можно ли написать `x=100` вместо `x = 100`?

Можно, но это не очень хорошо. Пробелы по обеим сторонам оператора добавляются для облегчения чтения кода.

Как получить вывод без пробелов между словами?

Это можно сделать так: `print u"С %s утром!" % u"добрым"`. Скоро вы узнаете об этом.

Что вы имеете в виду под фразой «читать код файла в обратном порядке»?

Все просто. Допустим, у вас есть файл с 16 строками кода. Начав со строки 16, сравните ее со строкой 16 в моем файле. Затем переходите к строке 15, и так далее, пока не прочитаете весь файл в обратном направлении.

Почему вы используете значение `4.0` в переменной `space_in_a_car`, обозначающей количество мест в машине?

Прочитайте, что такое число с плавающей точкой, и вновь задайте себе этот вопрос. См. практические задания.

Дополнительно о переменных и выводе

В этом упражнении мы углубимся в тему переменных и их вывода. На этот раз мы разберем форматирование строк. Каждый раз, когда вы окружаете символами " (двойными кавычками) некоторый текст, вы создаете строку. Строки позволяют получать желаемые результаты. Вы выводите их на экран, сохраняете в файлы, отсылаете на веб-серверы и т. д. и т. п.

Строки очень удобны, и в этом упражнении вы узнаете, как формировать строки, в которые встроены переменные. Вы будете встраивать переменные внутрь строк с использованием специального формата, а затем помещать переменные в конец с помощью специального синтаксиса, сообщая Python: «Эй, это форматированные строки, помести туда эти переменные».

Как всегда, просто в точности наберите приведенный ниже код, даже если вы не понимаете, как он работает.

ex5.py

```
1   my_name = u'Зед Шоу'
2   my_age = 35 # это правда!
3   my_height = 188 # см
4   my_weight = 80 # кг
5   my_eyes = u'Голубые'
6   my_teeth = u'Белые'
7   my_hair = u'Каштановые'
8
9   print u"Давайте поговорим о человеке по имени %s." % my_name
10  print u"Его рост составляет %d см." % my_height
11  print u"Он весит %d кг." % my_weight
12  print u"На самом деле это не так и много."
13  print u"У него %s глаза и %s волосы." % (my_eyes, my_hair)
14  print u"Его зубы обычно %s, хотя он и любит пить кофе." % my_teeth
15
16  # эта строка кода довольно сложная, не ошибитесь!
```

```
17  print u"Если я сложу %d, %d и %d, то получу %d." % (
18          my_age, my_height, my_weight, my_age + my_height + my_weight)
```

Внимание! Поместите строку `# -*- coding: utf-8 -*-` в верхней части сценария, чтобы кириллические буквы отображались правильно.

Результат выполнения

Сеанс упражнения 5

```
$ python ex5.py
Давайте поговорим о человеке по имени Зед Шоу.
Его рост составляет 188 см.
Он весит 80 кг.
На самом деле это не так и много.
У него Голубые глаза и Каштановые волосы.
Его зубы обычно Белые, хотя он и любит пить кофе.
Если я сложу 35, 188 и 80, то получу 303.
```

Практические задания

1. Измените имена всех переменных, удалив из них значение `my_`. Убедитесь, что вы изменили имена переменных во всем коде, а не только в строках, где присваивали им значения.

2. Попробуйте другие символы форматирования. К примеру, очень полезен символ `%r`. Его суть аналогична высказыванию «вывести это, несмотря ни на что».

3. Найдите в Интернете все символы форматирования языка Python.

4. Попробуйте написать код, который преобразует сантиметры и килограммы в дюймы и фунты. Следует не просто ввести значения в этих единицах измерения, а выполнить математические расчеты с помощью Python.

Распространенные вопросы

Могу ли я создать переменную следующим образом: `1 = u'Зед Шоу'`?

Нет, `1` — недопустимое имя переменной. Имена переменных должны начинаться с буквы (латинской), поэтому переменная `d1` будет работать, а `1` — нет.

Какую функцию выполняют символы `%s`, `%r` и `%d`?

Как вы узнаете в следующих упражнениях, это оператор `%` с разными символами форматирования. Они инструктируют Python взять значение переменной, указанной справа, и поместить его вместо символа `%s`.

Я не понимаю, что такое «оператор форматирования строк».

Проблема в том, что при обучении программированию многие понятия вы сможете освоить, лишь практикуясь в написании кода. Поэтому вы сначала выполняете упражнения, а затем я сообщаю вам необходимые сведения. Когда возникают подобные вопросы, записывайте их и возвращайтесь к ним, когда я буду объяснять соответствующие понятия.

Как округлить число с плавающей точкой?

Вы можете использовать функцию `round()` следующим образом: `round(1.7333)`.

У меня возникает следующая ошибка: `TypeError: 'str' object is not callable`. Что происходит?

Судя по всему, вы забыли указать символ `%` между текстовой строкой и списком переменных.

Можно ли изменить код примера под мои данные?

Конечно, попробуйте изменить в этом сценарии значения переменных, указав собственные. Пример с данными о себе покажется вам более понятным.

УПРАЖНЕНИЕ 6

Строки и текст

Хотя вы уже писали строки, но до сих пор не знаете, как они работают. В этом упражнении мы создадим несколько переменных со сложными значениями, чтобы вы поняли, как они используются.

Для начала разберемся со строками.

Строка, как правило, это некоторый текст, который вы хотите вывести на что-либо или «экспортировать» из программы, которую пишете. Python определяет строку по двойным или одиночным кавычкам вокруг текста. Вы уже несколько раз сталкивались со строками, используя команду `print` с текстом, заключенным в кавычки `"` или `'`. После выполнения команды Python выводит ее.

Строки могут содержать символы форматирования. Вы помещаете символ форматирования в строку, указывая оператор `%` (процент), а затем перечисляя переменные. Единственное, о чем следует упомянуть: если вы хотите использовать несколько символов форматирования в строке для вывода и несколько переменных, вам нужно поместить их имена в круглые скобки `()`, разделив имена запятыми. Это как если бы вы попросили меня что-нибудь купить в магазине: «Купи, пожалуйста, молоко, яйца, хлеб и макароны». Только на языке программирования мы скажем: `(молоко, яйца, хлеб, макароны)`.

Теперь мы наберем несколько строк, переменных и символов форматирования, а также выведем их на экран с помощью команды `print`. Кроме того, вы попрактикуетесь использовать сокращенные имена переменных. Программисты любят экономить время, используя раздражающие загадочные сокращения, поэтому давайте тоже научимся их понимать.

ex6.py

```
1   x = u"Существует %d типов людей." % 10
2   binary = "Python"
3   do_not = u"нет"
4   y = u"Те, кто понимает %r, и те, кто — %s." % (binary, do_not)
5
6   print x
7   print y
8
```

```
9   print u"Я сказал: %s." % x
10  print u"А еще я сказал: ' %s'." % y
11
12  hilarious = False
13  joke_evaluation = u"Разве это не смешно?! %r"
14
15  print joke_evaluation % hilarious
16
17  w = u"Это часть строки слева…"
18  e = u" а это справа."
19
20  print w + e
```

Результат выполнения

Сеанс упражнения 6

```
$ python ex6.py
Существует 10 типов людей.
Те, кто понимает 'Python', и те, кто — нет.
Я сказал: Существует 10 типов людей…
А еще я сказал: 'Те, кто понимает 'Python', и те, кто — нет.'.
Разве это не смешно?! False
Это часть строки слева… а это справа.
```

Практические задания

1. Изучите код этой программы и выше каждой строки кода напишите комментарий для себя, указывая, что делает данный код.

2. Определите все позиции, где строка помещается внутрь другой строки. Всего их должно быть четыре.

3. Вы уверены, что строка помещается внутрь другой строки только в четырех позициях? Откуда вы знаете? Может быть, я вру.

4. Объясните, почему при сложении двух строк из переменных `w` и `e` с символом + получается одна длинная строка.

Распространенные вопросы

В чем разница между `%r` и `%s`?

Оператор форматирования `%` с символом `r` используется для отладки, так как отображает «сырое» значение переменной, а оператор `%s` выводит значение так, как оно воспринимается пользователем. В данном упражнении операторы `%r` и `%s` используются для демонстрации разницы между ними.

Зачем нужны операторы `%s` и `%d`, когда вы можете использовать `%r`?

Оператор `%r` обычно используется для отладки, а другие форматы — для отображения значений переменных в подходящем для пользователей виде.

Если вы считаете шутку смешной, почему просто не написать `hilarious = True`?

Согласен с вами, можно так поступить. О подобных логических значениях вы узнаете в упражнении 27.

Почему вы заключили некоторые строки в одиночные кавычки, а не двойные?

В основном для соблюдения форматирования. Но я также использую одиночные кавычки внутри строки, заключенной в двойные кавычки. Взгляните на строку 10, чтобы понять, как я это делаю.

У меня появляется следующее сообщение об ошибке: `TypeError: not all arguments converted during string formatting`. Что происходит?

Нужно убедиться, что вы набрали строку кода в точности как у меня. Суть ошибки в том, что в строке указано больше символов `%` оператора форматирования, чем переменных, значения которых должны быть подставлены. Проанализируйте свой код и выясните, где ошиблись.

Еще о выводе

Теперь мы выполним несколько упражнений, в которых вы просто напечатаете код и выполните его. Будет минимум теории, так как эти упражнения продолжают уже рассмотренную тему. Цель состоит в том, чтобы приобрести опыт набора кода. Выполняйте эти несколько упражнений и не ленитесь! Набирайте вручную!

ex7.py

```
print u"У Мэри был маленький барашек."
print u"Его шерсть была белой как %s." % u'снег'
print u"И всюду, куда Мэри шла,"
print u"Маленький барашек всегда следовал за ней."
print u"." * 10 # чтобы это могло значить?

end1 = u"Б"
end2 = u"а"
end3 = u"д"
end4 = u"д"
end5 = u"и"
end6 = u"Г"
end7 = u"а"
end8 = u"й"

# обратите внимание на запятую в конце строки.
# уберите ее и посмотрите, что произойдет.
print end1 + end2 + end3 + end4 + end5,
print end6 + end7 + end8
```

Результат выполнения

Сеанс упражнения 7

```
$ python ex7.py
У Мэри был маленький барашек.
Его шерсть была белой как снег.
```

```
И всюду, куда Мэри шла,
Маленький барашек всегда следовал за ней.
.....
Бадди Гай
```

Практические задания

В нескольких следующих упражнениях выполняйте эти же практические задания.

1. Изучите код этой программы и выше каждой строки кода напишите комментарий для себя, указывая, что делает данный код.

2. Прочитайте код в обратном порядке или вслух, чтобы обнаружить ошибки.

3. Если ошибки обнаружены, запишите их в блокнот.

4. При переходе к следующему упражнению изучите обнаруженные ошибки и постарайтесь не повторять их.

5. Помните, что все совершают ошибки. Многие программисты считают, что их действия идеальны и они никогда не ошибаются, но это неправда. Они постоянно совершают ошибки.

Распространенные вопросы

Как работает оператор `end`?

На самом деле это не «оператор `end`», а обычные имена переменных, в которых присутствует слово «end».

Почему вы используете переменную с именем `'снег'`?

На самом деле это не переменная, а обычная строка со словом «снег» в ней. Имена переменных не окружаются одиночными кавычками.

Правильно ли я понимаю, что комментарии нужно писать для каждой строки кода, как вы говорили в первом практическом задании?

Нет, как правило, комментируются только сложные части кода, или же в комментариях объясняется, почему в коде вы поступили именно так. Тем не менее иногда для выполнения нужных действий пишется настолько сложный код, что требуется комментировать каждую его строку. В этом случае вам просто необходимо расшифровать код и перевести на человеческий язык.

Для создания строк можно равнозначно использовать как одиночные, так и двойные кавычки, или они предназначены для разных целей?

В коде на языке Python для создания строк приемлем любой вариант, хотя обычно одиночные кавычки используются для любых коротких строк, таких как `'a'` или `'снег'`.

Можно ли опустить запятую и превратить две последние строки кода в одну?

Да, можно и так поступить. Самое главное — соблюдать ограничение рекомендованной длины строки в 80 символов.

Вывод, вывод

ex8.py

```
1   formatter = "%s %s %s %s"
2
3   print formatter % (1, 2, 3, 4)
4   print formatter % (u"раз", u"два", u"три", u"четыре")
5   print formatter % (True, False, False, True)
6   print formatter % (formatter, formatter, formatter, formatter)
7   print formatter % (
8           u"Спят в конюшне пони,",
9           u"начал пес дремать,",
10          u"только мальчик Джонни",
11          u"не ложится спать!"
12  )
```

Результат выполнения

Сеанс упражнения 8

```
$ python ex8.py
1 2 3 4
раз два три четыре
True False False True
%s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s
Спят в конюшне пони, начал пес дремать, только мальчик Джонни
не ложится спать!
```

Практические задания

1. Проверьте свой код, запишите ошибки и постарайтесь не совершать их в следующем упражнении.

2. Обратите внимание, что во второй строке используется последовательность символов `%s`. Почему она отображается именно в таком виде?

Распространенные вопросы

Для форматирования я должен использовать оператор `%s` или `%r`?

Нужно использовать оператор форматирования `%s`. Оператор `%r` используется только для получения отладочных данных и выводит «сырую» версию переменных, также известную как «представление».

Почему значение `"pas"` нужно заключать в кавычки, а значение `True` — нет?

Потому, что Python распознает значения `True` и `False` как ключевые слова, представляющие понятия истинного и ложного. Если вы заключите эти слова в кавычки, то они превратятся в обычные строки и не будут работать должным образом. Вы узнаете больше об этом в упражнении 27.

Я попытался использовать русские, китайские (или какие-либо другие не-ASCII символы) в строках, но оператор `%r` выводит вместо текста «кракозябры».

Используйте оператор `%s` для вывода текста так, как воспринимает его человек.

Почему оператор `%r` иногда выводит значения, заключенные в одиночные кавычки, хотя я их писал в двойных кавычках?

Оператор `%r` в Python обрабатывает строки как «сырые» данные, помещая в одиночные кавычки. Все в порядке, так как `%r` используется для отладки и анализа кода.

Почему этот пример не работает в Python 3?

Не следует использовать Python версии 3. Используйте Python 2.7 или более поздней минорной версии, хотя и Python 2.6 будет работать нормально.

Могу ли я использовать IDLE для выполнения упражнений?

Нет, вы должны научиться использовать оболочку командной строки. Это очень важно для обучения программированию и прекрасно подойдет в качестве отправной точки изучения программирования. С IDLE вы потерпите неудачу, выполняя следующие упражнения в книге.

Вывод, вывод, вывод

ex9.py

```
1   # Несколько непривычный код — набирайте его в точности.
2
3   days = u"Пн Вт Ср Чт Пт Сб Вс"
4   months = u"Янв\nФевр\nМарт\nАпр\nМай\nИюнь\nИюль\nАвг"
5
6   print u"Это дни недели: ", days
7   print u"А это месяцы: ", months
8
9   print u"""
10  Что же тут творится?
11  Используются три двойные кавычки.
12  Мы можем набрать текста сколько угодно.
13  Даже 4 строки, 5 или 6.
14  """
```

Результат выполнения

Сеанс упражнения 9

```
$ python ex9.py
Это дни недели: Пн Вт Ср Чт Пт Сб Вс
А это месяцы: Янв
Февр
Март
Апр
Май
Июнь
Июль
Авг

Что же тут творится?
Используются три двойные кавычки.
Мы можем набрать текста сколько угодно.
Даже 4 строки, 5 или 6.
```

Практические задания

1. Проверьте свой код, запишите ошибки и постарайтесь не совершать их в следующем упражнении.

Распространенные вопросы

Как сделать так, чтобы названия месяцев начинали отображаться с новой строки?

Нужно начать список названий месяцев с символов `\n`, как показано ниже:

```
months = u"\nЯнв\nФевр\nМарт\nАпр\nМай\nИюнь\nИюль\nАвг"
```

Почему символы перевода строки, `\n`, не срабатывают, если я использую оператор `%r`?

Оператор форматирования `%r` работает следующим образом; значение переменной выводится именно так, как вы указали его (или близко к нему). Это «сырой» формат для отладки.

Почему отображается сообщение об ошибке, если я ввожу пробелы между тремя двойными кавычками?

Вы должны ввести `"""`, но не `" " "`, то есть не указывать пробелы между ними.

Так ли плохо, если все мои ошибки исключительно орфографические?

Большинство программных ошибок в начале обучения (и даже с опытом) носят орфографический характер или являются опечатками, но и они могут сделать код неработоспособным.

Управляющие последовательности

В упражнении 9 я продемонстрировал некоторые новые возможности форматирования вывода. Вы увидели два способа, позволяющие разделить одну строку на несколько. В первом случае я указываю символы \n (обратный слеш и буква n) между названиями месяцев. Все, что делают эти два символа, — помещают символ перехода на новую строку в соответствующей позиции. Применение символа \ (обратный слеш) — второй способ помещения специальных символов в строку. Существует много так называемых управляющих (или escape-) последовательностей, используемых для вставки различных специальных символов. Среди них есть особенная управляющая последовательность в виде двойного обратного слеша, которая выводит одиночный обратный слеш. Мы рассмотрим некоторые из этих последовательностей, чтобы вы поняли принцип их работы. Две другие важные управляющие последовательности позволяют вставлять одиночные и двойные кавычки.

Допустим, у вас есть строка, заключенная в двойные кавычки, а вы хотите указать двойные кавычки внутри самой строки. Если вы напишете нечто вроде "я "понимаю" Володю", Python запутается, так как посчитает, что кавычка перед словом «понимаю» заканчивает строку. Тут нужен способ сообщить Python, что кавычки внутри «ненастоящие».

Чтобы решить эту проблему, нужно экранировать двойные или одинарные кавычки, включенные в строку. Ниже представлен пример.

```
u"Nec'овский монитор с диагональю 19\"."      # экранирование двойной
                                              кавычки в строке
u'Nec\'овский монитор с диагональю 19".'      # экранирование одиночной
                                              кавычки в строке
```

Второй способ заключается в использовании тройных кавычек, """, содержимое в которых обрабатывается как отдельные строки по достижении завершающих символов """. Этот вариант мы также рассмотрим в примере ниже.

ex10.py

```
1   d_artagnan = u"\tМеня зовут д'Артаньян."
2   athos = u"Эта строка\nразделена на две."
3   porthos = u"Я \\ – \\ мушкетер!"
4
5   aramis = u"""
6   Подсказки:
7   \t* Граф из Гаскони
8   \t* Связан с Миледи
9   \t* Барон\n\t* Генерал Ордена
10  """
11
12  print d_artagnan
13  print athos
14  print porthos
15  print aramis
```

Внимание! Если вам все же лень вручную набирать код примеров из этой книги, все файлы с кодом вы можете скачать по адресу **https://eksmo.ru/files/Shaw_Python.zip**.

Результат выполнения

Посмотрите на отступ второй строки вывода. В этом упражнении используется управляющая последовательность для вставки символа отступа по горизонтали.

Сеанс упражнения 10

```
$ python ex10.py
Меня зовут д'Артаньян.
Эта строка
разделена на две.
Я \ – \ мушкетер!

Подсказки:
  * Граф из Гаскони
  * Связан с Миледи
  * Барон
  * Генерал Ордена
```

Управляющие последовательности

Это список всех управляющих последовательностей, которые поддерживаются языком Python. Вам вряд ли понадобятся многие из них, но запомните их код и представление. Кроме того, опробуйте их в оболочке командной строки, чтобы понять, как они работают.

Управляющая последовательность	Представление
`\\`	Обратный слеш (\)
`\'`	Одиночная кавычка (')
`\"`	Двойная кавычка (")
`\a`	ASCII-символ звонка (BEL)
`\b`	ASCII-символ возврата (BS)
`\f`	ASCII-символ перевода страницы (FF)
`\n`	ASCII-символ новой строки (LF)
`\N{`*имя*`}`	Символ с именем из базы данных Юникод (только Юникод)
`\r`	ASCII-символ возврата каретки (CR)
`\t`	ASCII-символ отступа по горизонтали (TAB)
`\u`*xxxx*	16-битный символ Юникод со значением *xxxx* (только Юникод)
`\U`*xxxxxxxx*	32-битный символ Юникод со значением *xxxxxxxx* (только Юникод)
`\v`	ASCII-символ отступа по вертикали (VT)
`\ooo`	Символ ASCII в восьмеричной нотации *ooo*
`\xhh`	Символ ASCII в шестнадцатеричной нотации *hh*

Еще один пример кода для практики:

```
while True:
    for i in ["/","- ","|","\\","|"]:
      print " %s\r" % i,
```

Практические задания

Постарайтесь запомнить все управляющие последовательности, записав их код и представления на карточки.

Попробуйте использовать символы `''' '''` (тройные одиночные кавычки). Можно ли их использовать вместо символов `"""`?

Скомбинируйте управляющие последовательности и операторы форматирования, чтобы создать более сложный вывод.

Помните оператор `%r`? Объедините его в строках с управляющими последовательностями `\'` и `\"`, и выведите результат с помощью команды `print`. Сравните между собой операторы `%r` и `%s`. Обратите внимание на то, как оператор `%r` в точности выводит значения переменных, а `%s` — так, как воспринимает их человек.

Распространенные вопросы

Я до сих пор не до конца разобрался в последнем упражнении. Мне переходить к следующему?

Да, переходите. Вместо остановки пометьте листинги каждого упражнения, которое вы не понимаете. Периодически возвращайтесь к пометкам и смотрите, можете ли вы разобраться в ранее непонятном коде после выполнения большего числа упражнений. Иногда потребуется возвращаться на несколько упражнений назад и выполнять их снова.

Так для чего служит управляющая последовательность `\\`, выделенная среди других?

Только для того, чтобы вывести один символ обратного слеша (\). Задумайтесь о том, в каких случаях эта последовательность может понадобиться.

Я ввел символы // или /n, а код не работает. Почему?

Потому, что вы используете обычный слеш, /, а не обратный — \. Это разные символы, которые служат для выполнения различных задач.

Когда я использую оператор %r, ни одна из управляющих последовательностей не работает. Почему?

Оператор %r выводит «сырое», необработанное представление набранного вами кода, который будет включать в себя исходные управляющие последовательности. Используйте вместо него оператор %s. Запомните: %r — для отладки, %s — для вывода обработанных данных.

Я не понял практическое задание № 3. Что вы имеете в виду под «объединением управляющих последовательностей и операторов форматирования»?

Я стараюсь, чтобы вы поняли: примеры из каждого продемонстрированного упражнения могут быть объединены для достижения нужного результата. Используя полученные знания об управляющих последовательностях, напишите новый код, который использует и их и операторы форматирования, изученные ранее.

Что лучше использовать, ''' или """?

Все зависит от стиля. Используйте сейчас символы '''(тройные одиночные кавычки), если хотите, но будьте готовы использовать другое форматирование в зависимости от ситуации или в случае, если это необходимо для полноценной совместной работы.

УПРАЖНЕНИЕ 11

Получение ответов на вопросы

Настало время ускорять темпы программирования. Вы должны выводить много данных, чтобы набрать опыт во вводе простого кода, но печатать простой код довольно скучно. Поэтому сейчас мы попробуем запросить у пользователей сведения о них. Это несколько сложнее, поэтому нужно научиться делать две вещи, которые, возможно, на первый взгляд покажутся бессмысленными, но вам стоит довериться мне. Вы убедитесь в этом через несколько упражнений.

В основном приложения обрабатывают ввод следующим образом.

1. Берутся некие данные, введенные пользователем.
2. Данные изменяются.
3. Выводятся на экран с изменениями.

До сих пор вы использовали команду `print` только для вывода на экран введенного вами текста и не могли получить от пользователя какие-либо сведения или изменить их. Даже если вы не знаете, что такое «входные» данные, давайте сейчас обойдемся без теории, а просто попробуем выполнить пример. В следующем упражнении мы проработаем тему, чтобы разобраться в ней.

ex11.py

```
print u"Сколько тебе лет?",
age = raw_input()
print u"Каков твой рост?",
height = raw_input()
print u"Сколько ты весишь?",
weight = raw_input()

print u"Итак, тебе %r лет, в тебе %r см роста и %r кг веса." % (
        age, height, weight)
```

Примечание. Обратите внимание на то, что в конце каждой строки со значением команды `print` указан символ , (запятая). Это необходимо, чтобы добавить символ перехода на новую строку и перейти к ней.

Результат выполнения

Сеанс упражнения 11

```
$ python ex11.py
Сколько тебе лет? 37
Каков твой рост? 187
Сколько ты весишь? 75
Итак, тебе '37' лет, в тебе '187' см роста и '75' кг веса.
```

Практические задания

1. Выполните поиск во Всемирной паутине и выясните, для чего предназначена функция `raw_input()` языка Python.

2. Обнаружили ли вы другие способы ее использования? Поэкспериментируйте с некоторыми из найденных способов.

3. Измените «форму», задав пользователям другие вопросы.

4. Вспомните про управляющие последовательности и ответьте на вопрос, почему, если в ответе пользователя используется одновременно и одиночная и двойная кавычка, в выводе будет присутствовать обратный слеш. Изучите, как одиночные кавычки должны быть экранированы, чтобы не означать конец строки.

Распространенные вопросы

Как я могу получить ответ в виде числа, выполнить математические действия?

Это несколько сложнее. Попробуйте использовать код `x = int(raw_input())`, который получает числовое значение в виде строки от функции `raw_input()`, а затем преобразует его в целое с помощью функции `int()`.

Я указал значение с одиночной кавычкой, `raw_input("6'2")`, но код не работает. Что я сделал неправильно?

Не следует присваивать это значение непосредственно функции; его нужно вводить в оболочке командной строки. Сначала вернитесь и введите код в точности так, как у меня. Затем запустите сценарий и, когда возникнет пауза после вывода запроса, введите значение с кавычкой с клавиатуры. И все будет работать.

Почему вы разделили код на строки 8 и 9 вместо того, чтобы уместить его на одной строке?

Я сделал так, потому что, согласно рекомендациям для программистов, на языке Python при наборе кода не рекомендуется использовать строки длиной свыше 80 символов для соблюдения стиля. Вы можете уместить код в одну строку, если вам так нравится.

В чем разница между функциями `input()` и `raw_input()`?

Функция `input()` преобразует ввод так, словно он является кодом на языке Python. Это небезопасно, поэтому вы должны избегать использования этой функции.

Вместо текста в ответном выводе отображаются символы типа `u'35'`.

Так Python сообщает вам, что пользователем были введены символы Юникода. Используйте оператор форматирования `%s` вместо `%r`, чтобы получить нормальный текст в выводе. Либо к *каждой* функции `raw_input()` нужно добавить код `.decode(sys.stdin.encoding or locale.getpreferredencoding(True))`. Например, так:

```
2   age = raw_input().decode(sys.stdin.encoding or locale.
    getpreferredencoding(True))
```

Помимо этого, в начале файла нужно указать следующий код:

```
1   # -*- coding: utf- 8 -*-
2
3   import codecs, sys
4   outf = codecs.getwriter('cp866')(sys.stdout, errors='replace')
5   sys.stdout = outf
```

УПРАЖНЕНИЕ 12

Осведомление пользователей

При вводе функции `raw_input()` вы набираете символы скобок — `(`и`)`. Это подобно форматированию с дополнительными переменными, например `"%s %s" % (x, y)`. Функцию `raw_input` вы можете поместить в приглашение, чтобы показать пользователю, что следует ввести. Строку текста запроса следует поместить внутрь скобок `()`, как показано в примере ниже:

```
y = raw_input(u"Имя?").decode(sys.stdin.encoding or locale.
getpreferredencoding(True))
```

Обратите внимание, что код `.decode(sys.stdin.encoding or locale.getpreferredencoding(True))` нужен для того, чтобы пользователь смог ввести ответ на русском языке. В примере ниже он не используется, так как ответ вводится в виде чисел.

Пользователь увидит запрос «Имя?» и сможет ввести ответ, присвоив его в качестве значения переменной `y`. Это как если задать кому-то вопрос и получить ответ.

Это означает, что мы можем полностью переписать код из предыдущего упражнения, используя только функцию `raw_input` для формирования запросов.

Внимание! Обратите внимание, что в начале сценария приводится незнакомый вам код с командой `import`. В данном сценарии он необходим для управления кодировками символов и правильного отображения значений функции `raw_input`, содержащих русские буквы.

---

ex12.py

```
1   # -*- coding: utf- 8 -*-
2
3   import codecs, sys
4   outf = codecs.getwriter('cp866')(sys.stdout, errors='replace')
```

```
5   sys.stdout = outf
6
7   age = raw_input(u"Сколько тебе лет? ")
8   height = raw_input(u"Каков твой рост? ")
9   weight = raw_input(u"Сколько ты весишь? ")
10
11  print u"Итак, тебе %r лет, в тебе %r см роста и %r кг веса." % (
12  age, height, weight)
```

Результат выполнения

Сеанс упражнения 12

```
$ python ex12.py
Сколько тебе лет? 37
Каков твой рост? 187
Сколько ты весишь? 75
Итак, тебе '37' лет, в тебе '187' см роста и '75' кг веса.
```

Практические задания

1. В оболочке командной строки, в которой вы запускаете сценарии Python, введите команду `pydoc raw_input`. Прочитайте показанное сообщение. Если вы работаете в операционной системе Windows, выполните команду `python -m pydoc raw_input`.

2. Завершите работу `pydoc`, введя `Q`.

3. Найдите во Всемирной паутине сведения о том, для чего предназначена команда `pydoc`.

4. Используя `pydoc`, прочитайте о модулях `open`, `file`, `os` и `sys`. Ничего страшного, если вы не понимаете того, что там написано: просто прочитайте информацию и запишите интересные/непонятные сведения.

Распространенные вопросы

Почему возникает ошибка синтаксиса (`SyntaxError: invalid syntax`) каждый раз, когда я выполняю команду `pydoc`?

Значит, вы выполняете `pydoc` не из оболочки командной строки, а из Python. Завершите работу Python.

Почему в моем случае результат выполнения команды `pydoc` не разбивается на страницы с паузой, как у вас?

Иногда, если справочный документ достаточно короткий и помещается на одном экране, он не разбивается на страницы при выводе.

Когда я выполняю команду `pydoc`, выводится сообщение, что она не распознана.

Некоторые версии Windows не поддерживают эту команду, и ввод `pydoc` будет выводить ошибку. Вы можете пропустить эти практические задания и выполнить поиск документации по Python в Интернете.

Почему вместо `%s` используется оператор `%r`?

Как вы уже знаете, оператор `%r` используется для отладки и выводит «сырое» представление переменной, в то время как `%s` выводит данные так, как их воспринимает человек. Я не буду отвечать на этот вопрос в дальнейшем, поэтому вы должны запомнить эту информацию. Это самый распространенный вопрос, который люди задают снова и снова, а значит, они не нашли время, чтобы запоминать эту простую истину. Хватит задавать вопросы, и наконец запомните этот факт.

Почему я не могу выполнить код `print "Сколько тебе лет?", raw_input()`?

Можно предположить, что код будет работать, но Python так не считает. Единственный ответ, который я могу дать, — код работать не будет.

Параметры, распаковка, переменные

В этом упражнении мы рассмотрим еще один метод ввода, который можно использовать для передачи переменных сценарию (сценарий или скрипт — другое имя ваших файлов с расширением *.py*). Как вы знаете, набрав код `python ex13.py`, вы запустите файл *ex13.py*? Значение `ex13.py` в этой команде называется «аргументом». Далее мы напишем сценарий, который будет принимать аргументы.

Введите следующий код, а затем я объясню его.

ex13.py

```
1   from sys import argv
2
3   script, first, second, third = argv
4
5   print u"Этот сценарий называется:", script
6   print u"Моя первая переменная называется:", first
7   print u"Моя вторая переменная называется:", second
8   print u"Моя третья переменная называется:", third
```

В строке 1 мы делаем то, что называется «импортом». Это добавление в сценарий новых функций из числа доступных в Python. Вместо того чтобы предоставлять сразу все возможности, Python запрашивает, какие из них вы собираетесь использовать. Так код ваших программ сохраняет небольшой размер и сообщает другим программистам, читающим ваш код впоследствии, какие функции языка использованы.

`argv` — стандартное имя переменной во многих языках программирования. Эта переменная содержит аргументы, которые вы передаете сценарию Python при запуске. В упражнениях далее вы разберетесь, как она работает.

Строка 3 «распаковывает» `argv`, чтобы вместо хранения всех аргументов назначить четыре переменные, с которыми вы можете работать: `script`, `first`, `second` и `third`. Это может показаться странным, но «распаковать»,

вероятно, наиболее подходящее слово, описывающее то, что происходит. Получается инструкция: «Возьми все содержимое `argv`, распакуй и назначь его в качестве переменных по порядку слева».

После этого мы сможем работать с ними в обычном порядке.

Внимание!

У «возможностей» другое название

Я называю их в книге «возможностями» (эти маленькие компоненты, которые вы импортируете, чтобы ваша программа на Python выполняла больше действий), но никто другой их так не называет. Я использовал это слово, потому что мне нужно, чтобы вы захотели узнать, как они называются на профессиональном языке. Перед тем как продолжить, вам нужно узнать их реальное имя — модули.

С этого момента мы будем называть эти «возможности», которые импортируем, модулями. Дальше я могу сказать что-нибудь наподобие: «Вам нужно импортировать модуль `sys`». Другие программисты также называют модули «библиотеками», но мы с вами будем придерживаться названия «модули».

Результат выполнения

Запустите программу, как показано ниже (не забудьте передать три аргумента!).

Сеанс упражнения 13

```
$ python ex13.py first 2nd 3rd
Этот сценарий называется: ex13.py
Моя первая переменная называется: first
Моя вторая переменная называется: 2nd
Моя третья переменная называется: 3rd
```

Результаты выполнения сценария с другими аргументами.

Сеанс упражнения 13

```
$ python ex13.py stuff things that
Этот сценарий называется: ex13.py
Моя первая переменная называется: stuff
Моя вторая переменная называется: things
Моя третья переменная называется: that
$

$ python ex13.py apple orange grapefruit
Этот сценарий называется: ex13.py
Моя первая переменная называется: apple
Моя вторая переменная называется: orange
Моя третья переменная называется: grapefruit
```

Вы можете заменять аргументы `first`, `2nd` и `3rd` любыми значениями. Если код написан с ошибкой, вы увидите сообщение типа этого:

Сеанс упражнения 13

```
$ python ex13.py first 2nd
Traceback (most recent call last):
  File "ex13.py", line 3, in <module>
    script, first, second, third = argv
ValueError: need more than 3 values to unpack
```

Данная ошибка появится, если вы указали недостаточное количество аргументов в команде, которую выполняете (в данном случае указаны только два аргумента — `first` и `2nd`). Сообщение `ValueError` расшифровывается как «для распаковки требуется больше 3 значений» и информирует, что вы не передали нужное количество аргументов.

Практические задания

1. Попробуйте передать меньше трех аргументов в команде вашего сценария. Какая ошибка возникает? Как вы можете ее истолковать?

2. Напишите сценарий с меньшим количеством аргументов и сценарий с бóльшим. Присваивайте переменным подходящие имена.

3. Совместите функцию `raw_input` с `argv` в сценарии, который требует дополнительный пользовательский ввод.

4. Помните, что модули предоставляют вам возможности программирования. Модули! Запомните их, так как мы с ними будем работать в дальнейшем.

Распространенные вопросы

При выполнении сценария возникает ошибка `ValueError: need more than 1 value to unpack`.

Помните, что очень важно обращать внимание на детали. Если вы взглянете на раздел «Результат выполнения», вы увидите, что я запускаю в оболочке командной строки сценарий с параметрами. Вы должны повторить этот код в точности.

Какая разница между `argv` и `raw_input()`?

Различие связано с тем, где пользователю нужно выполнить ввод данных. Если требуется ввод в сценарии в командной строке, используйте переменную `argv`. Если требуется ввод с клавиатуры во время выполнения сценария, используйте функцию `raw_input()`.

Аргументы командной строки — это строки?

Да, они обрабатываются в виде строк, даже если в оболочке командной строки вы ввели число. Используйте функцию `int()`, чтобы преобразовывать их по аналогии с `raw_input()`.

Как использовать оболочку командной строки?

Вы должны уже уметь это делать. В противном случае см. приложение «Экспресс-курс по оболочке командной строки».

У меня не получается использовать `argv` вместе с `raw_input()`.

Это просто. Укажите две строки в конце этого сценария, применив функцию `raw_input()`, чтобы получить нужные данные, а затем выполните команду `print`. Начав с этого, поэкспериментируйте и найдите дополнительные способы использовать оба компонента в одном сценарии.

Почему не получается выполнить `raw_input('? ') = x`?

Потому что у вас все наоборот. Сделайте так, как это делаю я, и код будет работать.

УПРАЖНЕНИЕ 14

Запросы и подтверждения

Давайте выполним упражнение, используя переменную `argv` и функцию `raw_input` вместе, чтобы спросить у пользователя нечто специфическое. Вам понадобится это для следующего упражнения, в котором мы научимся считывать и записывать файлы. В этом упражнении мы применим функцию `raw_input` несколько иначе, выводя простое приглашение >. Это похоже на старенькую игру типа Zork или Adventure.

Внимание! Обратите внимание, что в начале сценария приводится код для управления кодировками символов и правильной обработки значений функции `raw_input`, содержащих русские буквы. Для тех же целей используется код `.decode(sys.stdin.encoding or locale.getpreferredencoding(True))`.

---

ex14.py

```
1   # -*- coding: utf- 8 -*-
2
3   import codecs, sys
4   outf = codecs.getwriter('cp866')(sys.stdout, errors='replace')
5   sys.stdout = outf
6
7   from sys import argv
8
9   script, user_name = argv
10  prompt = '> '
11
12  print u"Привет %s, Я — сценарий %r." % (user_name, script)
13  print u"Я хочу задать тебе несколько вопросов."
14  print u"Я тебе нравлюсь, %s?" % user_name
15  likes = raw_input(prompt).decode(sys.stdin.encoding or
                    locale.getpreferredencoding(True))
16
17  print u"Где ты живешь, %s?" % user_name
18  lives = raw_input(prompt).decode(sys.stdin.encoding or
                    locale.getpreferredencoding(True))
```

```
19
20  print u"На каком компьютере ты работаешь?"
21  computer = raw_input(prompt).decode(sys.stdin.encoding or
                    locale.getpreferredencoding(True))
22
23  print u"""
24  Итак, ты ответил %r на вопрос, нравлюсь ли я тебе.
25  Ты живешь в %r. Не представляю, где это.
26  И у тебя есть компьютер %r. Прекрасно!
27  """ % (likes, lives, computer)
```

Внимание! Для корректной работы сценария необходимо ввести имя латинскими буквами!

Обратите внимание, что мы запрашиваем переменную для подстановки в строку и используем функцию `raw_input()` вместо постоянного ввода. Теперь, если мы хотим запросить другие данные, то можем быстро изменить их и вновь запустить сценарий. Очень удобно.

Результат выполнения

При запуске сценария не забудьте передать ему свое имя в качестве аргументов переменной `argv`. Указывать имя следует латинскими буквами.

Сеанс упражнения 14

```
$ python ex14.py zed
Привет zed, Я — сценарий 'ex14.py'.
Я хочу задать тебе несколько вопросов.
Я тебе нравлюсь, zed?
> Ага
Где ты живешь, zed?
> в Москве
На каком компьютере ты работаешь?
> Tandy 1000
```

```
Итак, ты ответил Ага на вопрос, нравлюсь ли я тебе.
Ты живешь в Москве. Не представляю, где это.
И у тебя есть компьютер Tandy 1000. Прекрасно!
```

Практические задания

1. Найдите информацию об играх Zork и Adventure. Попробуйте скачать их из Интернета и поиграть.
2. Измените приглашение командной строки на другое.
3. Добавьте еще один аргумент и примените его в сценарии.
4. Разберитесь, как я объединил многострочный текст """ с оператором форматирования % в последней команде `print`.

Распространенные вопросы

Я получаю ошибку синтаксиса, `SyntaxError: invalid syntax`, при запуске этого сценария.

Повторюсь, вы должны запустить его прямо в оболочке командной строки, а не из Python. Если вы наберете команду `python`, а затем введете `python ex14.py zed`, то потерпите неудачу, потому что попытаетесь запустить Python из-под уже запущенного Python. Завершите работу Python, а затем введите `python ex14.py zed`.

Я не понимаю, что вы имеете в виду под «изменением приглашения командной строки»?

Взгляните на код `prompt = '> '`. Под изменением я имею в виду присвоение переменной `prompt` другого значения. Вы уже проходили это; это просто строка, и вы выполнили 13 упражнений с ними, но потребуется время, чтобы понять и запомнить это.

Я получаю ошибку `ValueError: need more than 1 value to unpack`.

Помните, в прошлом упражнении я сказал, что вы должны внимательно изучить раздел «Результат выполнения» и в точности повторить то, что я сделал? Вы должны сделать то же самое здесь и сосредоточиться на том, как я набираю команду и какие аргументы командной строки использую.

Могу ли я использовать двойные кавычки при указании значения переменной `prompt`?

Конечно, можете. Попробуйте!

У вас есть компьютер Tandy?

Был, когда я был маленьким.

Я получаю ошибку `NameError: name 'prompt' is not defined`, когда я запускаю сценарий.

Вы либо ошиблись в имени переменной `prompt`, либо вообще не указали эту строку. Смотрите код и сравните каждую строку своего кода с моим. Читайте код в обратном порядке, снизу вверх.

Как запустить этот сценарий в IDLE?

Не используйте IDLE.

УПРАЖНЕНИЕ 15

Чтение файлов

Всего, что вы уже знаете о функции `raw_input()` и переменной `argv`, достаточно для чтения файлов. Возможно, вам понадобится разобраться в этом упражнении, чтобы понять, что происходит: делайте это тщательно и фиксируйте проблемы и вопросы. Работая с файлами, вы легко можете стереть их содержимое, если не будете осторожны.

В этом упражнении используются два файла. Один из них — это привычный сценарий с именем *ex15.py*, который будет запускаться, а другой носит имя *ex15_sample.txt*. Второй файл представляет собой не сценарий, а обычный текстовый документ, содержимое которого мы будем считывать в нашем сценарии. Ниже представлено содержимое этого файла.

```
У Мэри был барашек
Его шерсть была белой как снег
И всюду, куда Мэри шла, барашек всегда следовал за ней
```

Все, что мы хотим сделать, это «открыть» файл с помощью нашего сценария и вывести на экран его содержимое. Тем не менее мы не хотим «жестко программировать» имя файла *ex15_sample.txt* в сценарии. «Жесткое программирование» означает внедрение некоторых данных, которые должны быть получены от пользователя, в виде строки прямо в код программы. Это не очень хорошо, потому что впоследствии мы хотим загружать другие файлы. Решение проблемы состоит в применении переменной `argv` и функции `raw_input`, запрашивающих у пользователя, какой файл он хочет открыть.

ex15.py

```
1   from sys import argv
2
3   script, filename = argv
4
5   txt = open(filename)
6
7   print u"Содержимое файла %r:" % filename
```

```
8   print txt.read()
9
10  print u"Введите имя файла снова:"
11  file_again = raw_input("> ")
12
13  txt_again = open(file_again)
14
15  print txt_again.read()
```

Несколько интересных вещей происходят в этом файле, поэтому давайте его быстренько разберем.

Строки 1–3 должны быть вам знакомы — в них используется переменная `argv`, позволяющая получить имя файла. Далее вы видите строку 5, в которой используется новая команда `open`. Не откладывая, выполните команду `pydoc` и прочитайте справку о команде `open` (выполните `pydoc open`). Обратите внимание на то, как работают ваши сценарии и функция `raw_input`, она принимает параметр и возвращает значение, которое вы можете присвоить собственной переменной. Итак, вы открыли файл.

Строка 7 выводит коротенький текст, а в строке 8 вы видите что-то совсем новое и интересное. Мы вызываем функцию из переменной `txt`. Вы присваиваете функцию `open`, открывающую файл, переменной `txt`, которую в дальнейшем можете использовать. Вы пишете код, используя . (точку), имя команды и параметры, по аналогии с функциями `open` и `raw_input`. Разница заключается в том, что, когда вы выполняете функцию `txt.read()`, вы говорите: «Эй, `txt`! Выполни команду `read` без параметров!»

Остальная часть кода аналогична, мы проведем ее анализ в разделе «Практические задания».

Результат выполнения

Я создал файл с именем *ex15_sample.txt* и запустил сценарий.

Сеанс упражнения 15

```
$ python ex15.py ex15_sample.txt
Содержимое файла 'ex15_sample.txt':
У Мэри был барашек
```

```
Его шерсть была белой как снег
И всюду, куда Мэри шла, барашек всегда следовал за ней

Введите имя файла снова:
> ex15_sample
У Мэри был барашек
Его шерсть была белой как снег
И всюду, куда Мэри шла, барашек всегда следовал за ней
```

Внимание! Если вместо русских букв в выводе отображаются иероглифы, используйте кодировку текстового файла CP866. В Notepad++ это можно сделать, создав текстовый документ и выбрав команду меню **Кодировки → Кодировки → Кириллица → OEM 866** (Encoding → Character Set → Cyrillic → OEM 866). Как вариант, можно сначала выполнить упражнение 16 и использовать файл, который будет получен в результате его выполнения.

Практические задания

Упражнения ощутимо усложнились, поэтому выполните эти практические задания как можно тщательнее, прежде чем переходить к следующему.

1. Выше каждой строки кода напишите комментарий (предваряя текст комментария символом #) для себя, указывая, что делает данный код.

2. Если вы не уверены, что полностью разобрались в коде, попросите кого-нибудь помочь или выполните поиск во Всемирной паутине по запросу «Python открытие файлов».

3. В этом упражнении я использовал слово «команды» для обозначения функций (также называемых «методами»). Выполните поиск во Всемирной паутине, чтобы посмотреть, какие обозначения используют другие программисты. Не волнуйтесь, если полученная информация собьет вас с толку. Это обычное дело для программистов с их обширными знаниями.

4. Удалите строки 10–15, в которых используется функция `raw_input`, и выполните сценарий снова.

5. Используя только функцию `raw_input`, выполните сценарий снова. Задумайтесь, в каких случаях тот или иной способ получения имени файла будет лучше.

6. Выполните команду `pydoc file` и прокрутите справочную информацию, пока не увидите команду `read()` (метод или функцию). Просмотрите остальные команды, которые вы можете использовать. Пропустите те, которые содержат символы `__` (два подчеркивания) в начале имени, потому что их использовать не рекомендуется. Попробуйте выполнить некоторые из команд, о которых вы узнали.

7. Выполните команду `python` снова и попробуйте открыть текстовый файл прямо из командной строки. Изучите, как открывать файлы и читать их содержимое в оболочке командной строки, используя функции `open` и `read`.

8. Выполните функцию `close()` из переменных `txt` и `txt_again`. Весьма важно закрывать файлы после работы с ними.

Распространенные вопросы

Выполнение команды `txt = open(filename)` возвращает содержимое файла?

Нет, это не так. На самом деле происходит работа с так называемым объектом файла. Думайте об этом как о старом ленточном накопителе, который использовался в больших ЭВМ 50-х годов, или как о DVD-проигрывателе. Вы можете перемещаться по их файлам и «считывать» их, но файлы не являются элементами этих устройств.

Я не могу выполнить практическое задание № 7 в своей программе Терминал (Terminal)/PowerShell.

Первым делом в оболочке командной строки просто введите команду `python` и нажмите клавишу **Enter**. Теперь вы запустили Python, как делали это ранее. После этого можете вводить код, и Python будет выполнять его маленькими порциями. Поэкспериментируйте. Чтобы завершить работу, выполните команду `quit()` и нажмите клавишу **Enter**.

Для чего нужен код `from sys import argv`?

На данный момент вы должны понимать, что `sys` представляет собой пакет, и этот код позволяет получить функцию `argv` из этого пакета. Подробнее об этом вы узнаете позже.

Я указал имя файла, но код `script, ex15_sample.txt = argv` не работает.

Вы делаете неправильно. Напишите код в точности так, как показано в моем примере, а затем запустите его из командной строки точно так же, как я. Не нужно подставлять имена файлов; вы дадите команду Python подставить имя.

Почему не возникает ошибка, если мы открываем файл два раза?

Python не ограничивает число открытий файла, иногда это даже необходимо.

Чтение и запись файлов

Если вы выполнили практические задания из прошлого упражнения, то должны были увидеть все виды команд (методов/функций), которые можно применить к файлам. Ниже представлен список команд, которые вы должны запомнить.

- `close` — закрывает файл. Аналогично команде **Файл → Сохранить** (File → Save) в графическом интерфейсе.
- `read` — считывает содержимое файла. Результат можно присвоить в качестве значения переменной.
- `readline` — считывает только одну строку из текстового файла.
- `truncate` — очищает файл. Будьте осторожны, чтобы не потерять важные данные.
- `write(stuff)` — записывает данные в файл.

Сейчас это самые важные команды, которые вам нужно уметь использовать. Некоторые из них принимают параметры, но пока это не важно. Вам только нужно запомнить, что команда `write` принимает параметр в виде строки, которую можно затем записать в файл.

Изучим некоторые команды, создав простой текстовый редактор.

Внимание! Обратите внимание, что в начале сценария приводится незнакомый вам код с командой `import`. В данном сценарии он необходим для управления кодировками символов и правильного отображения значений функции `raw_input`, содержащих русские буквы.

---

ex16.py

```
1   # -*- coding: utf- 8 -*-
2
3   import codecs, sys
4   outf = codecs.getwriter('cp866')(sys.stdout, errors='replace')
```

```
5   sys.stdout = outf
6
7   from sys import argv
8
9   script, filename = argv
10
11  print u"Я собираюсь стереть файл %r." % filename
12  print u"Если вы не хотите стирать его, нажмите сочетание клавиш
    CTRL+C (^C)."
13  print u"Если хотите стереть файл, нажмите клавишу Enter."
14
15  raw_input("?")
16
17  print u"Открытие файла…"
18  target = open(filename, 'w')
19
20  print u"Очистка файла. До свидания!"
21  target.truncate()
22
23  print u"Теперь я запрашиваю у вас три строки."
24
25  line1 = raw_input(u"строка 1: ")
26  line2 = raw_input(u"строка 2: ")
27  line3 = raw_input(u"строка 3: ")
28
29  print u"Это я запишу в файл."
30
31  target.write(line1)
32  target.write("\n")
33  target.write(line2)
34  target.write("\n")
35  target.write(line3)
36  target.write("\n")
37
38  print u"И наконец, я закрою файл."
39  target.close()
```

Это объемный файл, вероятно, самый большой, набранный вами на сей момент. Поэтому набирайте код медленно, делайте пометки и проверьте ошибки, прежде чем его запускать. Посоветую запускать его частями. Сначала строки 1–8, затем следующие пять, потом еще немного, и так далее, до тех пор, пока весь код не будет набран и запущен.

Результат выполнения

Вы должны увидеть две вещи. Во-первых, результат работы вашего сценария.

Сеанс упражнения 16

```
$ python ex16.py test.txt
Я собираюсь стереть файл 'test.txt'.
Если вы не хотите стирать его, нажмите сочетание клавиш CTRL+C (^C).
Если хотите стереть файл, нажмите клавишу Enter.
?
Открытие файла…
Очистка файла. До свидания!
Теперь я запрашиваю у вас три строки.
строка 1: У Мэри был барашек
строка 2: Его шерсть была белой как снег
строка 3: И всюду, куда Мэри шла, барашек всегда следовал за ней
Это я запишу в файл.
И наконец, я закрою файл.
```

Во-вторых, откройте созданный файл (в моем случае, *test.txt*) в вашем редакторе, например Notepad++, и проверьте его содержимое. Все правильно?

Практические задания

1. Если вы запутались в коде, вернитесь к началу сценария и комментируйте каждую строку. Каждый комментарий поможет вам разобраться в коде или по крайней мере понять, что вам нужна дополнительная информация.

2. Напишите сценарий, похожий на этот, в котором используются команды `read` и `argv`, позволяющие прочитать содержимое созданного файла.

3. В коде этого упражнения очень много повторов. Используйте строки, форматирование и управляющие последовательности, чтобы вывести строки 1, 2 и 3 с помощью только одной функции `target.write()` вместо шести.

4. Разберитесь, для чего нужно использовать дополнительный параметр `'w'` в функции `open`. Подсказка: функцию `open` необходимо обезопасить, поскольку вы явно сообщаете, что хотите записать файл.

5. Если открывать файл в режиме `'w'`, нужна ли функция `target.truncate()`? Прочитайте справочную документацию Python в части функции `open` и ответьте, правда это или нет.

Распространенные вопросы

Нужна ли функция `target.truncate()`, если открывать файл в режиме `'w'`?

См. практическое задание № 5.

Что такое `'w'`?

Это обычная строка с символом, обозначающим режим обработки файла. Если вы указали параметр `'w'`, то вы «открываете этот файл в режиме записи» (от англ. слова write (запись) и происходит символ w). Точно так же используются параметры `'r'` — для «чтения (read)», `'a'` — для «присоединения (append)» и их модификаторы.

Какие модификаторы режимов обработки файлов мы можем использовать?

Наиболее важным из них, который следует знать, на данный момент является только один модификатор, +. С помощью него вы можете использовать параметры `'w+'`, `'r+'` и `'a+'`. Так вы сможете открыть файл в обоих режимах, чтения и записи, и в зависимости от используемого символа обрабатывать файл по-разному.

Можно ли просто выполнить функцию `open(filename)`, чтобы открыть файл в режиме чтения, `'r'`?

Да, это режим по умолчанию для функции `open()`.

Еще о файлах

Теперь выполним еще несколько действий с файлами. Мы напишем сценарий Python для копирования содержимого из одного файла в другой. Пример будет очень коротким, но послужит вам источником идей о других операциях, которые вы можете производить с файлами.

ex17.py

```
1   from sys import argv
2   from os.path import exists
3
4   script, from_file, to_file = argv
5
6   print u"Копирование данных из файла %s в файл %s" % (from_file, to_file)
7
8   # как код ниже разместить в одну строку?
9   in_file = open(from_file)
10  indata = in_file.read()
11
12  print u"Исходный файл имеет размер %d байт" % len(indata)
13
14  print u"Файл назначения существует? %r" % exists(to_file)
15  print u"Готов к работе, нажмите клавишу Enter для продолжения или
        CTRL+C для отмены."
16  raw_input()
17
18  out_file = open(to_file, 'w')
19  out_file.write(indata)
20
21  print u"Отлично, все сделано…"
22
23  out_file.close()
24  in_file.close()
```

Вы должны сразу обратить внимание, что мы импортируем функцию с именем `exists`. Она возвращает значение `True`, если файл существует, основываясь на его имени, указываемом в качестве аргумента. Или возвращает значение `False`, если файл не существует. Мы будем использовать эту

функцию во второй половине книги, чтобы выполнить много разных действий, но сейчас вы должны научиться импортировать ее.

Использование оператора `import` позволяет получить доступ к огромному разнообразию бесплатного кода, написанного другими (как правило, более опытными) программистами, а это сильно экономит время.

Результат выполнения

По аналогии с другими сценариями запустите текущий с двумя аргументами: файл, содержимое которого будет копироваться, и файл, в который будут вставлены скопированные данные. Я вновь использую обычный тестовый файл с именем *test.txt*.

Сеанс упражнения 17

```
$ cat test.txt
Это файл test.txt
$
$ python ex17.py test.txt new_file.txt
Копирование данных из файла test.txt в файл new_file.txt
Исходный файл имеет размер 21 байт
Файл назначения существует? False
Готов к работе, нажмите клавишу Enter для продолжения или CTRL+C
для отмены.

Отлично, все сделано…
```

Внимание! Если вместо русских букв в выводе отображаются иероглифы, используйте кодировку текстового файла CP866. В Notepad++ это можно сделать, создав текстовый документ и выбрав команду меню **Кодировки → Кодировки → Кириллица → OEM 866** (Encoding → Character Set → Cyrillic → OEM 866).

Сценарий должен работать с любым файлом. Попробуйте несколько разных файлов и проанализируйте, что происходит. Главное — будьте осторожны, чтобы не очистить файл с нужными данными.

Внимание! Вам понравился мой прием с командой `cat`, позволяющей прочитать содержимое файла? Вы можете узнать, как это сделать, из приложения.

Практические задания

1. Прочитайте справочные сведения об операторе `import` в Python, а затем поэкспериментируйте с ним в оболочке командной строки. Попробуйте импортировать некоторые компоненты и узнайте, имеете ли вы на это право. Если нет, так и должно быть.

2. Этот сценарий не очень хорош. В нем не появляется запрос на подтверждение операции копирования, а на экран выводится слишком много текста. Постарайтесь улучшить сценарий путем удаления части кода.

3. Попробуйте сделать максимально короткий сценарий. Я могу уложиться в одну строку.

4. Обратите внимание на предупреждение в конце раздела «Результат выполнения» насчет команды `cat`? Это старая команда, которая объединяет (или конкатенирует — concatenates) файлы. Ее также можно использовать как простой способ вывести содержимое файла на экран. Выполните команду `man cat`, чтобы прочитать про нее.

5. Пользователи операционной системы Windows, попробуйте найти альтернативу команде `cat`, которую используют в Linux/macOS. Аналогичное задание насчет команды `man`.

6. Разберитесь, для чего используется функция `output.close()`.

Распространенные вопросы

Почему символ `'w'` заключен в кавычки?

Потому что это строка. Вы уже использовали этот параметр и должны знать, что это строка.

Этот сценарий невозможно написать в одну строку!

Это зависит от того, какой вы представляете себе одну строку кода.

Для чего нужна функция `len()`?

Она получает длину строки, которую вы передаете ей, и возвращает значение в виде числа. Поэкспериментируйте с ней.

Когда я пытаюсь укоротить код этого сценария, я получаю сообщение об ошибке при закрытии файлов.

Вы, наверное, сделали что-то вроде этого: `indata = open(from_file).read()`. Тем самым вам не нужна функция `in_file.close()` в конце сценария. Файл должен быть закрыт Python уже после первой строки.

Мне кажется, это упражнение слишком трудное.

Это совершенно нормально. Вы можете не освоить программирование, пока не доберетесь до упражнения 36 или даже до конца книги. Только потом вы сможете самостоятельно что-то написать на Python. Все люди разные, так что просто продолжайте читать дальше и выполнять упражнения, с которыми у вас возникают затруднения. Потерпите.

Я получаю следующую ошибку: `Syntax: EOL while scanning string literal error`.

Вы забыли закончить строку кавычкой. Внимательно просмотрите строку кода с ошибкой.

Имена, переменные, код, функции

Длинное название упражнения, не так ли? Я собираюсь представить вам функции! Каждый программист должен знать о функциях и о том, как они работают и для чего предназначены. Функции выполняют три вещи.

1. Они присваивают имена фрагментам кода так, как переменные именуют строки и числа.
2. Они принимают аргументы, как и сценарии принимают атрибут `argv`.
3. Учитывая пункты 1 и 2, вы можете создавать собственные «мини-сценарии» или «крошечные команды».

Вы можете создать функцию, используя команду `def` в Python. Мы создадим четыре различные функции, которые будут работать как сценарии, и я продемонстрирую вам их взаимосвязь.

ex18.py

```
# похоже на сценарии с argv
def print_two(*args):
arg1, arg2 = args
   print "arg1: %r, arg2: %r" % (arg1, arg2)

# ок, здесь вместо *args мы делаем следующее
def print_two_again(arg1, arg2):
   print "arg1: %r, arg2: %r" % (arg1, arg2)

# принимает только один аргумент
def print_one(arg1):
   print "arg1: %r" % arg1

# не принимает аргументов
def print_none():
```

```
16     print u"А я ничего не получаю."
17
18
19  print_two("Zed","Shaw")
20  print_two_again("Zed","Shaw")
21  print_one("First!")
22  print_none()
```

Разберем первую функцию, `print_two`, которая наиболее близка к тому, что вы уже знаете про создание сценариев.

1. Сначала мы сообщаем Python, что хотим создать (объявить) функцию, используя для этого команду `def`*.

2. На той же строке мы присваиваем функции имя. В этом упражнении мы назвали ее `print_two`, но имя может быть любое. Главное — учитывайте, что функция должна иметь короткое имя, характеризующее, что она делает.

3. Далее мы сообщаем, что нам понадобится передать произвольное число аргументов, `*args` (звездочка и `args`). Этот код похож на параметр `argv`, но используется для функций. Чтобы он работал, данное значение нужно указать в круглых скобках, `()`.

4. Затем мы заканчиваем эту строку кода символом двоеточия, :, а новую строку начинаем с отступа.

5. После двоеточия все строки кода, имеющие отступ в четыре пробела, будут привязаны к указанному имени, `print_two`. Первая строка с отступом распаковывает аргументы так же, как и сценарии.

6. Чтобы продемонстрировать принцип работы, мы выводим эти аргументы.

Проблема `print_two` в том, что это далеко не самый простой способ создать функцию. В Python мы можем пропустить распаковку аргументов и указать желаемые имена прямо внутри скобок `()`. Этот прием демонстрирует функция `print_two_again`.

---

* От англ. *define* — определение.

Затем я привел пример создания функции `print_one`, принимающей один аргумент.

И, наконец, далее следует функция, которая не имеет никаких аргументов. Ее имя — `print_none`.

Внимание! Это очень важно. Не расстраивайтесь, если сейчас совсем ничего не понимаете. Мы выполним несколько упражнений, связывающих функции в сценариях и демонстрирующих дополнительные приемы работы. На данный момент я подразумеваю «мини-сценарии», когда говорю о функциях, и далее продолжу рассказывать вам о них.

Результат выполнения

Если вы запустите сценарий, вы должны увидеть следующее.

Сеанс упражнения 18

```
$ python ex18.py
arg1: 'Zed', arg2: 'Shaw'
arg1: 'Zed', arg2: 'Shaw'
arg1: 'First!'
А я ничего не получаю.
```

Теперь вы можете видеть, как работает функция. Обратите внимание на то, что функции используются так же, как и команды `exists`, `open` и другие. На самом деле, я обманываю вас, потому что в Python эти «команды» — тоже функции. Это означает, что вы можете создавать собственные команды и использовать их в своих сценариях.

Практические задания

Ниже приведен контрольный список вопросов о функциях для следующих упражнений. Запишите их на карточки (с ответами на обороте) и храните,

пока не выполните все упражнения или пока не почувствуете, что запомнили все, что нужно.

1. Можете ли вы объявить (создать) функцию с помощью ключевого слова `def`?

2. Имя объявляемой функции содержит только латинские буквы и символ _ (подчеркивание)?

3. Указали ли вы открывающую круглую скобку, `(`, сразу после имени функции?

4. Указали ли вы нужные аргументы через запятую после скобки, `(`?

5. Имя каждого аргумента уникально (не дублируются ли имена)?

6. Указали ли вы закрывающую скобку и двоеточие, `):`, после аргументов?

7. Отступы строк кода внутри функции состоят из четырех пробелов? Не больше, не меньше!

8. «Завершили» ли вы свою функцию строкой кода без отступа?

При выполнении («использовании» или «вызове») функции проверьте следующее.

1. Вы вызываете/используете/выполняете эту функцию, указывая ее имя?

2. Указали ли вы символ `(` после имени вызываемой функции?

3. Указали ли вы значения аргументов в круглых скобках, через запятую?

4. Завершили ли вы вызов функции символом `)`?

Используйте эти два контрольных списка при выполнении всех последующих упражнений, пока не выучите их наизусть. И, наконец, повторите несколько раз: «выполнить», «вызвать» или «использовать» функцию означает одно и то же.

Распространенные вопросы

Какие имена функций допустимы?

Так же как и имена переменных, имена функций нельзя начинать с цифры. В именах можно использовать латинские буквы, цифры и символ нижнего подчеркивания.

Для чего нужен символ * в `*args`?

Он сообщает Python, что нужно принять все аргументы функции, а затем поместить их в `args` в виде списка. Код работает по аналогии с параметром `argv`, который вы использовали ранее, но и предназначен для функций. Обычно не используется так часто, кроме специальных случаев.

Мне очень скучно вводить однообразный код.

Это хорошо. Значит, вы начинаете понимать вводимый код. Чтобы было не так скучно, набирайте все, что я говорю, а затем разбирайте код.

Функции и переменные

Функции могут казаться очень сложными, но не волнуйтесь. Просто продолжайте выполнять упражнения и учитывайте контрольный список из упражнения 18, и в конце концов вы все поймете.

Сейчас я расскажу о взаимодействии функций и переменных. В предыдущем упражнении переменные в вашей функции не связаны с переменными в сценарии. В новом упражнении мы решим эту проблему.

ex19.py

```
def cheese_and_crackers(cheese_count, boxes_of_crackers):
    print u"У нас есть %d бутылок лимонада!" % cheese_count
    print u"У нас есть %d коробок чипсов!" % boxes_of_crackers
    print u"Этого достаточно для вечеринки!"
    print u"Поехали!\n"


print u"Мы можем непосредственно передать числа функции:"
cheese_and_crackers(20, 30)


print u"ИЛИ, мы можем использовать переменные из нашего сценария:"
amount_of_cheese = 10
amount_of_crackers = 50

cheese_and_crackers(amount_of_cheese, amount_of_crackers)


print u"Мы даже можем выполнять вычисления внутри функции:"
cheese_and_crackers(10 + 20, 5 + 6)


print u"И объединять переменные с вычислениями:"
cheese_and_crackers(amount_of_cheese + 100, amount_of_crackers + 1000)
```

В примере продемонстрированы различные способы передачи нашей функции `cheese_and_crackers` значений, необходимых для ее выполнения. Мы можем непосредственно присвоить функции значения. А можем передать переменные. Или же передать вычисления. И, разумеется, объединить вычисления и переменные.

В некотором смысле аргументы функции похожи на символ =, используемый при создании переменных. По факту, если вы можете использовать символ =, чтобы присвоить имя, то, как правило, его можно передать функции в качестве аргумента.

Результат выполнения

Вы должны изучить результат выполнения этого сценария и сравнить его с предположениями, которые вы строили для каждого из примеров в сценарии.

Сеанс упражнения 19

```
$ python ex19.py
Мы можем непосредственно передать числа функции:
У нас есть 20 бутылок лимонада!
У нас есть 30 коробок чипсов!
Этого достаточно для вечеринки!
Поехали!

ИЛИ, мы можем использовать переменные из нашего сценария:
У нас есть 10 бутылок лимонада!
У нас есть 50 коробок чипсов!
Этого достаточно для вечеринки!
Поехали!

Мы даже можем выполнять вычисления внутри функции:
У нас есть 30 бутылок лимонада!
У нас есть 11 коробок чипсов!
Этого достаточно для вечеринки!
Поехали!

И объединять переменные с вычислениями:
У нас есть 110 бутылок лимонада!
```

```
У нас есть 1050 коробок чипсов!
Этого достаточно для вечеринки!
Поехали!
```

Практические задания

1. Выше каждой строки кода напишите комментарий (предваряя текст комментария символом #) для себя, указывая, что делает данный код.

2. Прочитайте код файла *.py* в обратном направлении, произнося вслух названия всех важных символов.

3. Напишите по крайней мере одну собственную функцию и выполните ее 10 различными способами.

Распространенные вопросы

Разве может быть 10 различных способов выполнения функции?

Верите вы или нет, но теоретически количество способов вызвать любую функцию неограниченно. В этом упражнении сделайте это так, как у меня в строках 8–12, и будьте немного креативны.

Есть ли способ разобраться, что делает функция, а то я не понимаю?

Существует много различных способов, но постарайтесь указать комментарий на русском языке выше каждой строки, описывающий, что происходит в строке кода. Еще одна уловка — читать код вслух. Или можно распечатать код, а рядом писать комментарии и зарисовывать процесс выполнения.

Что нужно сделать, если я хочу спросить у пользователя количество лимонада и чипсов?

Вам нужно использовать функцию `int()`, чтобы преобразовать результат выполнения `raw_input()`.

Переменные в строках 13 и 14 вносят изменения в переменные в функции?

Нет, эти переменные независимы и существуют вне функции. Затем они передаются функции, а временные версии создаются только для выполнения функции. При ее завершении эти временные переменные также завершаются, а остальной код продолжает работать. Продолжайте читать, и все станет ясно.

Можно ли использовать глобальные переменные (например, в строках 13 и 14) с теми же именами, что и переменные функции?

Да, но с этого момента вы не можете быть уверены, какую именно переменную вы имеете в виду. Иногда возникает необходимость использовать те же имена, но лучше избегать таких ситуаций.

Строки кода 12–19 перезаписывают функцию `cheese_and_crackers`?

Нет, совсем нет. Это их вызов, который по сути представляет собой временный переход к первой строке функции, а затем возвращение назад после того, как достигнута последняя строка функции. Функция не заменяется.

Есть ли ограничение на количество аргументов функции?

Зависит от версии Python и компьютера, но количество довольно велико. На практике при достижении количества примерно в пять аргументов функцию становится неудобно использовать.

Можно ли вызвать функцию из функции?

Да, вы сделаете это, когда начнете создавать игру далее в этой книге.

УПРАЖНЕНИЕ 20

Функции и файлы

Вспомните контрольный список из упражнения 18 и выполняйте это упражнение, обращая особое внимание на совместное выполнение функций и обработку файлов.

ex20.py

```
1   from sys import argv
2
3   script, input_file = argv
4
5   def print_all(f):
6       print f.read()
7
8   def rewind(f):
9       f.seek(0)
10
11  def print_a_line(line_count, f):
12      print line_count, f.readline()
13
14  current_file = open(input_file)
15
16  print u"Первым делом выведем этот файл целиком:\n"
17
18  print_all(current_file)
19
20  print u"Теперь отмотаем назад, словно это кассета."
21
22  rewind(current_file)
23
24  print u"Выведем три строки:"
25
26  current_line = 1
27  print_a_line(current_line, current_file)
28
29  current_line = current_line + 1
30  print_a_line(current_line, current_file)
31
```

```
32  current_line = current_line + 1
33  print_a_line(current_line, current_file)
```

Обратите особое внимание на то, как мы передаем номер текущей строки каждый раз, когда выполняем функцию `print_a_line`.

Внимание! Если вам все же лень вручную набирать код примеров из этой книги, все файлы с кодом вы можете скачать по адресу **https://eksmo.ru/files/Shaw_Python.zip**.

Результат выполнения

Сеанс упражнения 20

```
$ python ex20.py test.txt
Первым делом выведем этот файл целиком:

Это строка 1
Это строка 2
Это строка 3
Теперь отмотаем назад, словно это кассета.
Выведем три строки:
1 Это строка 1

2 Это строка 2

3 Это строка 3
```

Практические задания

1. Выше каждой строки кода напишите комментарий (предваряя текст комментария символом #) для себя, указывая, что делает данный код.

2. При каждом выполнении функции `print_a_line` вы передаете номер строки в переменной `current_line`. Напишите код, в котором значение переменной `current_line` не меняется при каждом

вызове функции, и проанализируйте, как переменная `line_count` обрабатывается в функции `print_a_line`.

3. Найдите каждое применение функции и проверьте их объявление (`def`), чтобы убедиться, что ей передаются правильные аргументы.

4. Выполните поиск в Интернете и разберитесь, для чего используется функция `seek` в отношении файлов. Выполните команду `pydoc file` и попробуйте сформулировать суть работы этой функции.

5. Исследуйте краткую нотацию `+=` и перепишите сценарий с ее использованием.

Распространенные вопросы

Что обозначает символ `f` в `print_all` и других функциях?

`f` представляет собой переменную в этой и других функциях в упражнении 18, обозначающую в данном случае файл. Принцип обработки файлов в Python аналогичен записи на старый магнитный накопитель мэйнфрейма или DVD-плеер. Там используется «считывающая головка» (read head), и с помощью нее вы можете выполнять «поиск» (seek) в этом файле для ее позиционирования, а затем работать в нужной позиции. Каждый раз, когда вы выполняете функцию `f.seek(0)`, вы двигаетесь к началу файла. При выполнении функции `f.readline()` считывается одна строка из файла, и считывающая головка перемещается вправо после символа `\n`, которым завершается строка. Подробнее вы узнаете позже.

Почему отображаются пустые строки в конце вывода?

Функция `readline()` обрабатывает символ `\n`, который находится в файле в конце каждой строки. Этот символ добавляется к результату, возвращаемому функцией `readline()`. Чтобы изменить это поведение, нужно добавить символ , (запятую) в конце команды `print`, чтобы не выводить символ `\n`.

Почему `seek(0)` не присваивает переменной `current_line` значение 0?

Во-первых, функция `seek()` выполняет операции в байтах, а не строках. Поэтому результат вычисления — 0 байт (первый байт в файле). Во-вторых, `current_line` — это обычная переменная и не имеет фактической связи с файлом. Мы вручную увеличиваем значение.

Что такое +=?

Это сокращенная запись двух операций, = и +. Выражение `x = x + y` можно кратко записать как `x += y`.

Как функция `readline()` определяет строки?

Код функции `readline()` сканирует каждый байт файла до тех пор, пока не обнаружит символ `\n`, после чего прекращает чтение файла, чтобы вернуть найденное в качестве результата. Символ `f` отвечает за сохранение текущей позиции в файле после каждого вызова функции `readline()`, поэтому функция определяет каждую строку.

УПРАЖНЕНИЕ 21

Что возвращают функции

Ранее вы использовали символ =, чтобы именовать переменные и присваивать им значения в виде чисел или строк. Теперь мы продолжим обучение использованию оператора = и научимся присваивать переменным значения, получаемые от функции. Кое на что потребуется обратить пристальное внимание, но об этом позже. Сейчас первым делом наберите следующий код.

ex21.py

```
1  def add(a, b):
2      print u"СЛОЖЕНИЕ %d + %d" % (a, b)
3      return a + b
4
5  def subtract(a, b):
6      print u"ВЫЧИТАНИЕ %d - %d" % (a, b)
7      return a - b
8
9  def multiply(a, b):
10     print u"УМНОЖЕНИЕ %d * %d" % (a, b)
11     return a * b
12
13 def divide(a, b):
14     print u"ДЕЛЕНИЕ %d / %d" % (a, b)
15     return a / b
16
17
18 print u"Давайте выполним несколько вычислений с помощью функций!"
19
20 age = add(30, 7)
21 height = subtract(190, 4)
22 weight = multiply(35, 2)
23 iq = divide(220, 2)
24
25 print u"Возраст: %d, Рост: %d, Вес: %d, IQ: %d" % (age, height,
   weight, iq)
26
27
28 # Головоломка в качестве дополнительного задания, введите код
```

```
    в любом случае.
29  print u"Это головоломка."
30
31  what = add(age, subtract(height, multiply(weight, divide(iq, 2))))
32
33  print u"Получается: ", what, u"Вы можете это вычислить вручную?"
```

Мы создали математические функции для решения примеров сложения, вычитания, умножения и деления. Обратите внимание на последнюю строку кода функции `add`, в которой указано `return a + b`. Вот что там происходит.

1. Наша функция вызывается с двумя аргументами: `a` и `b`.

2. С помощью команды `print` мы выводим результат выполнения функции, в данном случае операции сложения.

3. Затем мы сообщаем Python, что нужно вернуть: мы возвращаем результат сложения `a + b`. Своими словами, это «я складываю `a` и `b`, а ты верни результат».

4. Python складывает два числа. Затем, когда функция завершается, результат сложения присваивается в качестве значения переменной и может быть использован в строках далее в коде.

Как и со многими другими понятиями в этой книге, вы должны разбираться в этой теме медленно и досконально и попытаться проследить, что происходит. Чтобы помочь, предлагаю решить дополнительную головоломку и узнать нечто интересное.

Результат выполнения

Сеанс упражнения 21

```
$ python ex21.py
Давайте выполним несколько вычислений с помощью функций!
СЛОЖЕНИЕ 30 + 7
ВЫЧИТАНИЕ 190-4
УМНОЖЕНИЕ 35 * 2
ДЕЛЕНИЕ 220 / 2
Возраст: 37, Рост: 186, Вес: 70, IQ: 110
```

```
Это головоломка.
ДЕЛЕНИЕ 110 / 2
УМНОЖЕНИЕ 70 * 55
ВЫЧИТАНИЕ 186-3850
СЛОЖЕНИЕ 37 + -3664
Получается: -3627 Вы можете это вычислить вручную?
```

Практические задания

1. Если вы не до конца понимаете, как работает команда `return`, попробуйте написать несколько собственных функций, возвращающих некоторые значения. Вы можете вернуть все, что будет помещено справа от символа =.

2. В конце сценария приведена головоломка. Я беру значение, возвращаемое одной функцией, и использую его в качестве аргумента другой функции. Я делаю это в цепочке, создавая своего рода формулу из функций. Выглядит очень необычно, но если вы запустите сценарий, то сможете увидеть результаты. Ваша задача — попытаться написать нормальную формулу, которая бы воссоздавала тот же набор операций.

3. После того как у вас появится формула головоломки, проанализируйте, что произойдет, если вы измените аргументы функций. Попробуйте изменить код с целью получить другое значение.

4. И, наконец, сделайте обратное. Напишите простую формулу и используйте функции таким же образом, чтобы вычислить значение.

Это упражнение может действительно показаться очень трудным, но постарайтесь выполнять его медленно и с расстановкой. Отнеситесь к нему как к игре. Решение головоломки вроде этой превращает программирование в удовольствие, поэтому я и в дальнейшем буду создавать для вас небольшие проблемы.

Распространенные вопросы

Почему Python выводит формулы или функции «в обратном порядке»?

На самом деле, это не обратный порядок, а «изнанка». Когда вы разберете функцию на отдельные формулы и вызовы функций, вы поймете, как это работает. Постарайтесь понять, что я имею в виду «изнанку», а не «обратный порядок».

Как использовать функцию `raw_input()`, чтобы указать собственные значения?

Помните `int(raw_input())`? Проблема в том, что вы не сможете использовать числа с плавающей точкой, поэтому попробуйте использовать код `float(raw_input())`.

Что вы имеете в виду под выражением «написать формулу»?

Для начала возьмите уравнение `24 + 34/100 - 1023`. Преобразуйте его так, чтобы можно было использовать функции. Теперь придумайте собственные аналогичные математические уравнения и используйте переменные, чтобы это было похоже на формулу.

УПРАЖНЕНИЕ 22

Что вы теперь знаете?

В этом и следующем упражнениях нет кода и разделов «Результат выполнения» и «Практические задания». Хотя, по сути, это упражнение — словно один большой раздел практических заданий. Оно посвящено обзору знаний, которые вы получили на данный момент.

Во-первых, просмотрите каждое выполненное упражнение и запишите каждое ключевое слово и символ, которые вы использовали. Убедитесь, что ваш список содержит все изученные символы.

Во-вторых, рядом с каждым словом или символом напишите его название (имя) и что он делает. Если вы не можете найти имя символа в этой книге, выполните поиск в Интернете. Если вы не знаете, для чего нужно то или иное ключевое слово или символ, поищите о нем сведения и попробуйте использовать его в коде.

Вы можете встретить слова или символы, информацию о которых не сможете найти или не понимаете, как они работают. Сохраните такие слова или символы в списке и вернитесь к ним, когда информация будет найдена.

После того как список будет готов, потратьте несколько дней и перепишите список, дважды проверяя, чтобы не было ошибок. Это покажется скучным, но может существенно помочь в освоении программирования.

После того как вы заучите список слов или символов и будете точно знать, что они делают, попробуйте составить новый список, указывая имена и принцип работы *из памяти*. Если некоторые из них вы не можете вспомнить, снова попробуйте заучить их.

Внимание! Самый главный тезис при выполнении этого упражнения: «Не бывает неудач, бывают попытки».

Что вы изучили

Данные упражнения по заучиванию очень важны, хотя и могут показаться скучными и бессмысленными. Они помогут вам сосредоточиться на цели и понять суть всех ваших усилий.

В этом упражнении ваша задача — выучить имена символов, что поможет быстрее и проще читать исходный код. Это похоже на изучение алфавита и заучивание основных слов иностранного языка, за исключением того, что «алфавит» Python содержит дополнительные неизвестные вам символы.

Просто не торопитесь и делайте передышки. Надеюсь, у вас все получится. Лучше всего тренироваться по 15 минут, делая перерывы. Давая мозгу отдохнуть, вы освоите тему быстрее и с меньшим разочарованием.

УПРАЖНЕНИЕ 23

Чтение кода

Я надеюсь, что на прошлой неделе вы выучили свой список символов и ключевых слов. Эту неделю можно посвятить чтению кода сценариев, доступных в Интернете. Это упражнение кажется сложным на первый взгляд. Я собираюсь покинуть вас на некоторое время, а вы должны стараться изо всех сил, читая и пытаясь понять исходный код реальных проектов. Конечная цель не в том, чтобы заставить вас понять код, а чтобы привить вам три следующих навыка.

1. Поиск исходного кода необходимых компонентов Python.
2. Чтение кода и поиск файлов.
3. Попытки понять найденный код.

На этой стадии обучения у вас не хватит навыков, чтобы понять весь найденный код, но выгода от анализа кода реальных проектов очевидна.

Во время выполнения упражнения представляйте себя исследователем на неизвестной земле, пытающимся понять местный диалект, чтобы выжить. За тем исключением, конечно, что из этой передряги вы выйдете живым, так как Всемирная паутина — все-таки не джунгли.

Вот что вам следует сделать.

1. В своем любимом браузере откройте сайт **bitbucket.org**, **github.com** или **gitorious.org** и выполните поиск по запросу **python**.
2. Избегайте проектов, в которых упоминается версия Python 3. Она будет только сбивать вас с толку.
3. Выберите проект наугад и откройте его.
4. Перейдите на вкладку **Source** (Исходники) и просмотрите список файлов и каталогов на предмет наличия файла с расширением *.py* (только не *setup.py* — это не то, что нужно).
5. Начав сверху, прочитайте код, записывая предположения, как он работает.

6. Если какие-либо символы или слова показались вам странными или неизвестными, запишите их, чтобы позже найти соответствующие справочные сведения.

Суть упражнения состоит в том, чтобы, применяя полученные знания, попробовать прочитать код и разобраться, для чего он предназначен. Попробуйте сначала бегло просмотреть его, а затем углубляться в детали. Можно также усложнить задачу и произносить каждый знакомый символ вслух.

Посетите следующие веб-сайты:

- **launchpad.net**
- **sourceforge.net**
- **freecode.com**

УПРАЖНЕНИЕ 24

Дополнительная практика

Вы достигли определенного этапа обучения и уже можете читать и понимать код. Вы уже «чувствуете Python своими пальцами» и готовы перейти к более глубокому уровню обучения программированию. Но сначала давайте еще попрактикуемся. Это и следующее упражнения направлены на развитие полученных навыков. Выполните их, соблюдая точность ввода кода и проверяя код на отсутствие ошибок.

ex24.py

```
print u"Давайте попрактикуемся!"
print u'Вы должны знать об управляющих последовательностях
с символом \\, которые \n управляют переносом строк и \t отступами.'

poem = u"""
\tДля счастья
мне совсем немного надо.
Хочу тебя \n я нежно обнимать,
Хочу всегда
я быть с тобою рядом
\n\t\tИ никогда не отпускать!
"""

print "--------------"
print poem
print "--------------"


five = 10-2 + 3-6
print u"Здесь должна быть пятерка: %s" % five

def secret_formula(started):
    jelly_beans = started * 500
    jars = jelly_beans / 1000
    crates = jars / 100
    return jelly_beans, jars, crates


```

```
28  start_point = 10000
29  beans, jars, crates = secret_formula(start_point)
30
31  print u"Начиная с: %d" % start_point
32  print u"У нас есть %d бобов, %d банок и %d ящиков." % (beans, jars,
    crates)
33
34  start_point = start_point / 10
35
36  print u"Также можно поступить следующим образом:"
37  print u"У нас есть %d бобов, %d банок и %d ящиков." %
    secret_formula(start_point)
```

Результат выполнения

Сеанс упражнения 24

```
$ python ex24.py
Давайте попрактикуемся!
Вы должны знать об управляющих последовательностях с символом \,
которые
  управляют переносом строк и     отступами.
--------------

      Для счастья
мне совсем немного надо.
Хочу тебя
  я нежно обнимать,
Хочу всегда
я быть с тобою рядом

              И никогда не отпускать!

--------------
Здесь должна быть пятерка: 5
Начиная с: 10000
У нас есть 5000000 бобов, 5000 банок и 50 ящиков.
Также можно поступить следующим образом:
У нас есть 500000 бобов, 500 банок и 5 ящиков.
```

Практические задания

1. Проверьте себя: прочитайте код в обратном порядке, затем вслух и укажите комментарии выше строк кода, которые вам были непонятны.

2. Разделите код на отдельные задачи, а затем выполните сценарий. Разберите полученные ошибки и попробуйте справиться с ними.

Распространенные вопросы

Как получилось, что вы вызываете переменную `jelly_beans`, а затем используете имя `beans`?

Это по части принципа работы функции. Если вы помните, внутри функции переменная носит временный характер, и когда вы возвращаете ее, то можете присвоить другой функции для последующего использования. Я просто создал новую переменную `beans` для хранения возвращаемого значения.

Что вы имеете в виду под фразой «читать код в обратном порядке»?

Все очень просто. Начните с последней строки и сравните ее с той же строкой в моем файле. Затем переходите к следующей строке, и так далее, пока не прочитаете весь файл в обратном направлении.

Кто автор этого стихотворения?

Это стихотворение найдено на просторах Интернета.

И еще практика

Попрактикуемся с применением функций и переменных, чтобы убедиться, что вы их хорошо знаете. Это упражнение должно быть понятным для вас. Тем не менее оно несколько отличается от ранее выполненных. Вы не будете запускать его. Вместо этого вы импортируете сценарий в Python и запустите функции вручную.

ex25.py

```
1   def break_words(stuff):
2       """Эта функция разбирает текст на слова."""
3       words = stuff.split(' ')
4       return words
5
6   def sort_words(words):
7       """Сортирует слова."""
8       return sorted(words)
9
10  def print_first_word(words):
11      """Выводит первое слово после извлечения."""
12      word = words.pop(0)
13      print word
14
15  def print_last_word(words):
16      """Выводит последнее слово после извлечения."""
17      word = words.pop(- 1)
18      print word
19
20  def sort_sentence(sentence):
21      """Принимает целое предложение и возвращает отсортированные
        слова."""
22      words = break_words(sentence)
23      return sort_words(words)
24
25  def print_first_and_last(sentence):
26      """Выводит первое и последнее слова предложения."""
27      words = break_words(sentence)
28      print_first_word(words)
```

```
29      print_last_word(words)
30
31  def print_first_and_last_sorted(sentence):
32      """Сортирует слова, а затем выводит первое и последнее."""
33      words = sort_sentence(sentence)
34      print_first_word(words)
35      print_last_word(words)
```

Сначала, попробуйте запустить сценарий как обычно с помощью команды `python ex25.py`, чтобы обнаружить возможные ошибки, допущенные вами. После этого исправьте их и переходите в раздел «Результат выполнения» для завершения упражнения.

Результат выполнения

В этом упражнении мы будем взаимодействовать с файлом.py внутри интерпретатора Python, который вы время от времени использовали, чтобы делать расчеты. После выполнения команды `python` в оболочке командной строки вы увидите примерно следующее.

```
$ python
Python 2.7.1 (r271:86832, Jun 162011, 16:59:05)
[GCC4.2.1 (Based on Apple Inc. build 5658) (LLVM build 2335.15.00)]
on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

В вашем случае вывод будет выглядеть немного иначе, но не обращайте на это внимание. Сразу после приглашения >>> начинайте вводить код Python, после чего он сразу будет выполнен.

В моем примере это выглядит так.

Сеанс упражнения 25 Python

```
Python 2.7.1 (r271:86832, Jun 162011, 16:59:05) [GCC4.2.1 (Based on
Apple Inc. build 5658) (LLVM build 2335.15.00)] on darwin
```

```
    Type "help", "copyright", "credits" or "license" for more
    information.
3   >>> import ex25
4   >>> sentence = "All good things come to those who wait."
5   >>> words = ex25.break_words(sentence)
6   >>> words
    ['All', 'good', 'things', 'come', 'to', 'those', 'who', 'wait.']
8   >>> sorted_words = ex25.sort_words(words)
9   >>> sorted_words
    ['All', 'come', 'good', 'things', 'those', 'to', 'wait.', 'who']
11  >>> ex25.print_first_word(words)
    All
13  >>> ex25.print_last_word(words)
    wait.
15  >>> wrods
    Traceback (most recent call last):
      File "<stdin>", line 1, in <module>
    NameError: name 'wrods' is not defined
19  >>> words
    ['good', 'things', 'come', 'to', 'those', 'who']
    >>> ex25.print_first_word(sorted_words)
    All
    >>> ex25.print_last_word(sorted_words)
    who
    >>> sorted_words
    ['come', 'good', 'things', 'those', 'to', 'wait.']
    >>> sorted_words = ex25.sort_sentence(sentence)
    >>> sorted_words
    ['All', 'come', 'good', 'things', 'those', 'to', 'wait.', 'who']
    >>> ex25.print_first_and_last(sentence)
    All
    wait.
    >>> ex25.print_first_and_last_sorted(sentence)
    All
    who
```

Давайте разберем код, двигаясь сверху вниз по строкам (важным), чтобы вы поняли, что происходит.

- **Строка 3.** Вы импортируете файл *ex25.py* по аналогии с другими компонентами, импортированными ранее. Обратите внимание, что

не нужно указывать расширение *.py* в конце строки, чтобы импортировать файл. В процессе импорта создается модуль, все функции которого вы можете использовать.

- **Строка 4.** Вы создаете предложение, чтобы впоследствии работать с ним.

- **Строка 5.** Вы используете модуль ex25 и вызываете первую функцию, `ex25.break_words`. Символ . (точка) сообщает Python: «Внутри модуля ex25 есть функция с именем `break_words`, и я хочу, чтобы ты запустил ее».

- **Строка 6.** Введите слово `words`, и Python выведет на экран содержимое переменной с этим именем (строка 7). Выглядит необычно — это объект `list`, с которым вы познакомитесь позже.

- **Строки 8–9.** Сделаем то же самое с функцией `ex25.sort_words`, чтобы получить отсортированное по алфавиту предложение.

- **Строки 11 и 13.** Используются функции `ex25.print_first_word` и `ex25.print_last_word`, чтобы вывести первое и последнее слово предложения.

- **Строка 15.** Это интересно. Я сделал ошибку и напечатал имя переменной `words` как `wrods`, после чего Python выдал ошибку (строки 16–18).

- **Строка 19.** Выводим измененный список слов. Обратите внимание, что, так как мы вывели первое и последнее слова, их нет в этом выводе.

Остальные строки предназначены для того, чтобы вы проанализировали их в практических заданиях.

Практические задания

1. Используя оставшиеся строки вывода из раздела «Результат выполнения», выясните, что они делают. Убедитесь что вы понимаете, как выполняются функции в модуле ex25.

2. Попробуйте выполнить следующее: `help(ex25)`, а также `help(ex25.break_words)`. Уточните, как вы можете получить

справку по вашему модулю и как помогает текст в кавычках """ после каждой функции в коде файла *ex25.py*. Эти специальные строки называются документированием, `documentation comments`, и позднее вы узнаете подробнее о них.

3. Ввод `ex25.` — это сокращение от строки импорта `from ex25 import *`, которую можно расшифровать как «импортируй все из модуля ex25». Программистам нравится такой код. Начните заново и посмотрите, как работают ваши функции.

4. Попробуйте разделить код файла на части и запустить его в Python. Вам придется завершить работу Python нажатием сочетания клавиш **Ctrl+D** (**Ctrl+Z** в операционной системе Windows), чтобы перезагрузить интерпретатор.

Распространенные вопросы

Я получаю ответ `None`, когда вывожу функцию.

Вы, наверное, указали функцию, в коде которой отсутствует команда `return` в конце. Прочитайте код файла в обратном порядке, как вас я учил, и убедитесь, что каждая строка кода верна.

Я получаю ошибку `-bash: import: command not found` при вводе команды `import ex25`.

Обратите внимание на то, что я делаю в разделе «Результат выполнения». Команды вводятся в интерпретаторе Python, а не просто в оболочке командной строки. Это означает, что вы сначала должны запустить Python.

Я получаю ошибку модуля `ImportError: No module named ex25.py` при вводе команды `import ex25.py`.

Не добавляйте расширение файла *.py* в конце команды. Интерпретатор Python знает, что файл заканчивается на *.py*, поэтому вводите просто `import ex25`.

Я получаю ошибку синтаксиса `SyntaxError: invalid syntax` при выполнении упражнения.

Это означает, что вы что-то упустили в коде (" или аналогичная ошибка синтаксиса в указанной строке или ранее). Каждый раз, когда возникает эта ошибка, начните с указанной в тексте ошибки строки кода и проверьте, верна ли она, а затем читайте код в обратном направлении, проверяя каждую строку.

Как функция `words.pop(0)` меняет значение переменной `words`?

Это сложный вопрос, но в данном случае переменная `words` является списком, и, поскольку вы можете передавать команды, в ней будут сохраняться результаты этих команд. Это похоже на то, как работали файлы и многие другие компоненты, когда вы использовали функцию `f.readline()`.

Когда в функции указывается команда `print`, а когда `return`?

Вы должны понимать, что команда `print` используется только для вывода на экран и что и `print,` и `return` возвращают значение. Это следует запомнить. Вы можете использовать команду `print`, когда нужно вывести значение. Или команду `return`, когда нужно вернуть значение.

Внимание, тест!

Практически половина книги позади. Вторая половина содержит самое интересное. Вы узнаете о логике и научитесь делать полезные вещи, например принимать решения.

Прежде чем продолжить, я хочу вас проверить. Этот тест будет очень трудным, потому что вам понадобится изменить чужой код. При программировании вам часто придется иметь дело с кодом других программистов, а также с их высокомерием. Программисты очень часто утверждают, что их код совершенен.

Такие программисты глупы, так как не думают о других разработчиках. Хороший программист, как и хороший ученый, предполагает, что всегда есть некоторая вероятность вторжения ошибок в его код. Хорошие программисты предполагают, что их программное обеспечение недостаточно хорошо, и пытаются исключить все возможные ошибки.

В этом упражнении вы попрактикуетесь с исправлением кода плохого программиста. Я недостаточно внимательно скопировал код из упражнений 24 и 25 и удалил случайные символы, а также добавил дополнительные ошибки. О большинстве ошибок сообщит сам интерпретатор Python. Но некоторые из них являются математическими, и их вы должны найти сами. К другим ошибкам относятся недостатки форматирования и орфографические ошибки в строках.

Все эти ошибки очень распространены, их совершают все программисты, в том числе и опытные.

В этом упражнении ваша задача заключается в исправлении кода этого файла. Используйте все свои навыки, чтобы исправить и улучшить этот сценарий. Во-первых, проанализируйте его, для удобства распечатав на бумаге. Исправьте каждую ошибку и продолжайте улучшать сценарий, пока он не будет работать отлично. Старайтесь не пользоваться помощью. Если вы затрудняетесь с решением, сделайте перерыв и вернитесь к упражнению позже.

Даже если на работу потребуется несколько дней, потратьте их, чтобы исправить сценарий самостоятельно.

Наконец, для этого упражнения не нужно вводить код вручную, вы будете использовать существующий файл. Для этого необходимо перейти на сайт **https://eksmo.ru/files/Shaw_Python.zip** и скачать архив с примерами для этой книги.

Откройте файл *ex26.txt* из папки *ex26*, скопируйте его содержимое и вставьте в новый файл с именем *ex26.py*. Это единственное упражнение, когда я разрешаю вам копирование/вставку.

Распространенные вопросы

Мне нужно импортировать файл *ex25.py* или я могу просто удалить ссылки на него?

Вы можете поступать любым способом. Этот файл содержит функции из файла *ex25*, поэтому во втором случае сначала удалите ссылки на него.

Можно ли выполнять код по мере исправления ошибок?

Безусловно, можно. Компьютер служит для того, чтобы помочь, поэтому используйте его на полную катушку.

Обучение логике

Наступил момент, когда вы начинаете изучать логику. До этого момента вы делали все, что могли: считывали и записывали файлы в оболочке командной строки, узнавали о многих математических возможностях Python.

Теперь вы будете изучать логику. Вы не постигнете сложные теории, которые так любят ученые, — только элементарную логику, которой достаточно для работы обычных программ и которую программисты используют каждый день.

Практиковаться на логических упражнениях мы будем после того, как вы заучите некоторый материал. Я хочу, чтобы вы выполняли это упражнение на протяжении недели. Не сачкуйте. Даже если вам скучно, продолжайте заучивание. Это упражнение содержит набор таблиц истинности, которые вы должны запомнить, чтобы следующие упражнения выполнять было проще.

Я предупреждаю: сначала будет тоскливо. Заучивание таблиц истинности покажется занятием совершенно скучным и утомительным, но оно необходимо, чтобы привить вам как программисту очень важный навык. Все эти понятия необходимы. Большинство из описанных концепций покажутся интересными, как только вы разберетесь в них. Вы будете сражаться с ними и в один прекрасный день победите и поймете. Все затраченные на заучивание усилия с лихвой окупятся позже.

Небольшая подсказка, как запомнить что-то и не сойти с ума: заучивайте материал небольшими порциями в течение всего дня и отмечайте моменты, над которыми нужно поработать дополнительно. Не пытайтесь сесть на два часа подряд и сразу запомнить все эти таблицы. Этот номер не пройдет. В памяти отложится только то, что вы прочитали в первые 15–30 минут занятия. Поэтому лучше создайте нужное количество карточек с одной логической операцией на одной стороне и ответом (Истина или Ложь*) на другой. Затем вы вытаскиваете карточку, смотрите на логическую операцию и должны немедленно сказать: «Истина!» или «Ложь!» Продолжайте практиковаться, пока не выучите все таблицы.

После того как вы заучите их, попробуйте писать в записную книжку собственные таблицы истинности. Не копируйте их. Пишите их по памяти, а в случае затруднения поглядывайте на таблицы, приведенные в этой книге, чтобы

* True или False.

освежить память. Эти упражнения будут тренировать ваш мозг, и вы сможете запомнить всю таблицу.

Не тратьте больше недели на это упражнение, потому что далее вы примените все это на практике.

Терминология

В языке Python используются следующие термины (символы и фразы) для определения истинности (True) или ложности (False) операции. Программная логика пытается выяснить, является ли истинной некоторая комбинация этих символов и переменных в данной точке программы.

- and (И)
- `or` (ИЛИ)
- `not` (НЕ)
- `!=` (не равно)
- == (равно)
- >= (больше или равно)
- <= (меньше или равно)
- `True` (Истина)
- `False` (Ложь)

На самом деле, вы сталкивались с этими символами и раньше, но скорее всего не видели фразы (И, ИЛИ, НЕ).

Таблицы истинности

Теперь мы используем эти символы, чтобы сформировать таблицы истинности, которые следует запомнить.

NOT	Истина?
not False	True
not True	False

OR	Истина?
True or False	True
True or True	True
False or True	True
False or False	False

AND	Истина?
True and False	False
True and True	True
False and True	False
False and False	False

NOT OR	Истина?
not (True or False)	False
not (True or True)	False
not (False or True)	False
not (False or False)	True

NOT AND	Истина?
not (True and False)	True
not (True and True)	False
not (False and True)	True
not (False and False)	True

!=	Истина?
1 != 0	True
1 != 1	False
0 != 1	True
0 != 0	False

==	Истина?
1 == 0	False
1 == 1	True
0 == 1	False
0 == 0	True

Теперь используйте эти таблицы, чтобы создать карточки для запоминания, и потренируйтесь с ними неделю. Помните, что в первое время может быть сложно, но, тренируясь каждый день, со временем вы освоите весь материал.

Распространенные вопросы

Могу ли я просто разобраться в концепциях булевой алгебры, а не запоминать все эти таблицы?

Конечно, можете, но тогда вам придется постоянно возвращаться к теории булевой алгебры по мере выполнения упражнений. Если же вы сначала выучите таблицы, это не только повысит ваши навыки заучивания, но и позволит разобраться в этих операциях. После этого понять булеву алгебру будет проще. Но поступайте так, как удобнее вам.

Логические выражения

Логические комбинации, которые вы выучили в прошлом упражнении, называются логическими (или булевыми) выражениями. Такие выражения повсеместно используются в программировании. Они являются основными фундаментальными компонентами вычислений, и их понимание сродни знанию гаммы в музыке.

В этом упражнении вы примените логические выражения, которые вы запомнили, и затем попробуете использовать их в сценариях Python. Составьте список из логических задач, приведенных ниже, и напишите рядом ответ, который, как вы считаете, верен. В каждом случае это будет или Истина, или Ложь. После того как вы напишете все ответы, запустите Python в оболочке командной строки и поочередно введите все выражения, чтобы проверить свои ответы.

```
 1.True and True
 2.False and True
 3.1 == 1 and 2 == 1
 4."test" == "test"
 5.1 == 1 or 2!= 1
 6.True and 1 == 1
 7.False and 0!= 0
 8.True or 1 == 1
 9."test" == "testing"
10.1!= 0 and 2 == 1
11."test"!= "testing"
12."test" == 1
13.not (True and False)
14.not (1 == 1 and 0!= 1)
15.not (10 == 1 or 1000 == 1000)
16.not (1!= 10 or 3 == 4)
17.not ("testing" == "testing" and "Zed" == "Cool Guy")
18.1 == 1 and not ("testing" == 1 or 1 == 0)
19."chunky" == "bacon" and not (3 == 4 or 3 == 3)
20.3 == 3 and not ("testing" == "testing" or "Python" == "Fun")
```

Я расскажу вам один секрет, как проще понять сложные выражения в конце списка. Всякий раз, когда вы видите эти логические операторы, вы можете с легкостью решить их, соблюдая простую последовательность.

1. Найдите операторы сравнения на равенство (== или !=) и проверьте их истинность.

2. Найдите операции `and/or` внутри скобок и решите их в первую очередь.

3. Найдите все операции `not and` и обратите их.

4. Найдите все оставшиеся операции `and/or` и решите их.

5. Когда вы завершите упражнение, в итоге вы должны получить ответы Истина или Ложь. Я продемонстрирую процесс решения на немного измененном примере № 20:

```
3!= 4 and not ("testing"!= "test" or "Python" == "Python")
```

Далее я пошагово разберу и решу пример, пока не получу один ответ.

1. Найдите операторы сравнения на равенство и проверьте их истинность:

   а) `3 != 4` истинно: `True and not ("testing"!= "test" or "Python" == "Python")`

   б) `"testing"!= "test"` истинно: `True and not (True or "Python" == "Python")`

   в) `"Python" == "Python"`: `True and not (True or True)`

2. Найдите операции `and/or` внутри скобок и решите их:

   а) `(True or True)` истинно: `True and not (True)`

3. Найдите все операции `not and` и обратите их:

   а) `not (True)` ложно: `True and False`

4. Найдите все оставшиеся операции `and`/`or` и решите их:

   а) `True and False` ложно

Решив все это, в итоге получим результат — `False` (Ложь).

Внимание! Длинные выражения из этого упражнения могут показаться очень сложными на первый взгляд. Вы должны уметь решать их, но не расстраивайтесь, если не получается с первого раза. Я занимаю вас этой «логической» гимнастикой, чтобы позднее упражнения решались вами гораздо проще. Просто придерживайтесь показанной выше техники решения и проверяйте, верны ли были ваши предположения. Но не волнуйтесь, если у вас еще нет понимания логики. Со временем вы все поймете.

Результат выполнения

После проверки пары выражений оболочка командной строки с запущенным Python будет выглядеть следующим образом.

```
$ python
Python 2.5.1 (r251:54863, Feb 62009, 19:02:12) [GCC4.0.1 (Apple Inc.
build 5465)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> True and True
True
>>> 1 == 1 and 2 == 2
True
```

Практические задания

1. В Python есть много операторов, подобных != и ==. Постарайтесь найти как можно больше «операторов сравнения». Это могут быть, к примеру, < или <=.

2. Запишите названия каждого из этих операторов сравнения. Например, оператор != я называю «не равно».

3. Поэкспериментируйте с Python, выполняя операции с новыми логическими операторами, и прежде, чем вы нажмете клавишу **Enter**, произносите вслух, что произойдет. Не задумывайтесь, называйте первое, что приходит на ум. Запишите произнесенное значение, а затем нажмите клавишу **Enter** и отметьте, правильно вы ответили или нет.

4. Выбросьте лист бумаги с записями из задания № 3, чтобы по ошибке не использовать его в будущем.

Распространенные вопросы

Почему выражение `"test" and "test"` возвращает `'test'`, а `1 and 1` возвращает `1` вместо `True`?

Python, как и многие другие языки, возвращает один из операндов логических выражений, а не только Истина или Ложь. Это означает, что, если выполнить `False and 1`, вы получите первый операнд (`False`), а если выполнить `True and 1`, то вы получите второй операнд (`1`). Поэкспериментируйте с этим.

Есть ли разница между операторами != и <>?

Программистами на Python не приветствуется использование оператора <> вместо !=, поэтому используйте оператор !=. В остальном между ними нет никакой разницы.

А можно покороче?

Можно. Любое выражение `and` (И), которое содержит `False`, в результате ложно. Любое выражение `or` (ИЛИ), которое содержит `True`, в результате истинно. Но сначала убедитесь, что вы можете разобрать и решить все выражение целиком, поскольку позже этот навык очень пригодится.

Что, если...

Следующий сценарий Python, который вы будете набирать, познакомит вас с условной конструкцией `if` (если). Наберите приведенный ниже код, выполните его в оболочке командной строки и убедитесь, что обучение пошло на пользу.

ex29.py

```
people = 20
cats = 30
dogs = 15


if people < cats:
    print u"Слишком много кошек! Мир обречен!"

if people > cats:
    print u"Не так много кошек! Мир спасен!"

if people < dogs:
    print u"Мир утоп в слюнях!"

if people > dogs:
    print u"Мир сухой!"


dogs += 5

if people >= dogs:
    print u"Людей больше или столько же, сколько собак."

if people <= dogs:
    print u"Людей меньше или столько же, сколько собак."


if people == dogs:
    print u"Людей столько же, сколько собак."
```

Результат выполнения

Сеанс упражнения 29

```
$ python ex29.py
Слишком много кошек! Мир обречен!
Мир сухой!
Людей больше или столько же, сколько собак.
Людей меньше или столько же, сколько собак.
Людей столько же, сколько собак.
```

Практические задания

1. Теперь ваша задача состоит в том, чтобы определить в коде конструкцию `if` и рассказать, что она делает. Попробуйте ответить своими словами на эти вопросы, прежде чем перейти к следующему заданию.

2. Как влияет конструкция `if` на код, расположенный далее?

3. Почему строка кода ниже конструкции `if` имеет отступ в четыре пробела?

4. Что произойдет, если удалить этот отступ?

5. Можете ли вы вставить другие логические выражения из упражнения 27 в конструкцию `if`? Попробуйте.

6. Что произойдет, если изменить исходные переменные `people`, `cats` и `dogs`?

Распространенные вопросы

Что такое +=?

Это сокращенная запись двух операций, = и +. Выражение `x = x + 1` можно кратко записать как `x += 1`. Об операторе присваивания += и других операторах вы узнаете позже.

А если иначе...

В предыдущем упражнении вы работали с некоторыми конструкциями `if`, а затем пытались угадать, как они работают. Прежде чем двигаться дальше, я отвечу на все вопросы, которые были приведены в разделе «Практические задания». Вы выполнили эти задания, верно?

1. Как влияет конструкция `if` на код, расположенный далее?
   Конструкция (оператор) `if` создает так называемую ветвь в коде. Это своего рода квест, когда вы попадаете в одну локацию, если пойдете налево, и в другую, если пойдете направо. Конструкция `if` сообщает сценарию: «Если это логическое выражение истинно, выполни нижеследующий код, в противном случае пропусти его».

2. Почему строка кода ниже конструкции `if` имеет отступ в четыре пробела?
   Двоеточие в конце строки сообщает Python, что вы собираетесь создать новый «блок» кода, а отступ в четыре пробела определяет строки кода в этом блоке. Это работает точно так же, как и при использовании функций, рассмотренных нами ранее.

3. Что произойдет, если удалить этот отступ?
   Если отступ удалить, скорее всего, интерпретатор Python выведет ошибку. Python ожидает *некий код* с отступом после окончания строки с символом : (двоеточие).

4. Можете ли вы вставить другие логические выражения из упражнения 27 в конструкцию `if`? Попробуйте.
   Да, это возможно, и они могут быть настолько сложными, насколько вам по силам, хотя на самом деле сложный код, как правило, считается плохим тоном.

5. Что произойдет, если изменить исходные переменные `people`, `cats` и `dogs`?
   Поскольку вы сравниваете числа, то, если вы измените значения переменных, конструкции `if` по-прежнему будут проверяться на истинность, а блоки кода продолжать работать. Попробуйте подставить разные числа и представить, как работают блоки кода.

Сравните мои ответы с собственными и убедитесь, что вы уяснили понятие «блок» кода. Это важно, поскольку в следующем упражнении вы напишете все конструкции `if`, которые можно использовать.

Напечатайте следующий код и попробуйте запустить сценарий.

ex30.py

```
people = 30
cars = 40
buses = 15


if cars > people:
    print u"Поедем на машине."
elif cars < people:
    print u"Не поедем на машине."
else:
    print u"Мы не можем выбрать."

if buses > cars:
    print u"Слишком много автобусов."
elif buses < cars:
    print u"Может, поехать на автобусе?"
else:
    print u"Мы до сих пор не можем выбрать."

if people > buses:
    print u"Хорошо, давайте поедем на автобусе."
else:
    print u"Прекрасно, давайте останемся дома."
```

Внимание! Если вам все же лень вручную набирать код примеров из этой книги, все файлы с кодом вы можете скачать по адресу **https://eksmo.ru/files/Shaw_Python.zip**.

Результат выполнения

Сеанс упражнения 30

```
$ python ex30.py
Поедем на машине.
Может, поехать на автобусе?
Хорошо, давайте поедем на автобусе.
```

Практические задания

1. Расскажите, что делают команды `elif` и `else`.
2. Измените количество автомобилей, людей и автобусов, а затем отследите результат выполнения каждой конструкции `if`.
3. Попробуйте использовать некоторые более сложные логические выражения, наподобие `cars > people` и `buses < cars`.
4. Выше каждой строки кода напишите комментарий (предваряя текст комментария символом #) для себя, указывая, что делает данный код.

Распространенные вопросы

Что произойдет, если несколько блоков `elif` будут истинны?

Интерпретатор Python обрабатывает сценарий сверху вниз, так что запустит первый блок. Этот блок истинен, поэтому будет выполнен только первый блок.

УПРАЖНЕНИЕ 31

Принятие решений

В первой части этой книги вы выводили в основном результаты выполнения функций, причем код выполнялся, как правило, сверху вниз. Создав функцию, вы можете запустить ее позднее, но для такого ветвления вам понадобится код, позволяющий принимать решения. Теперь, когда вы изучили конструкции `if`, `else` и `elif`, можно начать создавать сценарии, содержащие компоненты принятия решений.

Ранее вы написали простой набор тестов, опрашивающих пользователя. В этом сценарии вы также будете задавать пользователю вопросы и принимать решения, основываясь на его ответах. Напишите этот сценарий, а затем поэкспериментируйте с ним, чтобы понять, как он работает.

ex31.py

```
1   print u"Ты находишься в темной комнате с двумя дверями. Ты войдешь
    в дверь № 1 или № 2?"
2
3   door = raw_input("> ")
4
5   if door == "1":
6      print u"В этой комнате гигантский медведь поедает чизкейк. Твои
       действия?"
7      print u"1. Отберешь чизкейк."
8      print u"2. Крикнешь медведю в ухо."
9
10     bear = raw_input("> ")
11
12     if bear == "1":
13             print u"Медведь укусил тебя за голову. Отличная работа!"
14     elif bear == "2":
15             print u"Медведь укусил тебя за ногу. Отличная работа!"
16       else:
17          print u"Прекрасно, это действие %s было единственным
            верным. Медведь убежал прочь." % bear
18
19  elif door == "2":
20     print u"Ты смотришь в бесконечную пропасть глаз Ктулху. Твои
       действия?"
```

```
21      print u"1. Рассказать Ктулху про чернички."
22      print u"2. Потрепать желтые заклепки на куртке."
23      print u"3. С помощью револьвера насвистеть мелодии."
24
25      insanity = raw_input("> ")
26
27      if insanity == "1" or insanity == "2":
28          print u"Твое тело спасено благодаря тому, что Ктулху
            превратился в желе. Отличная работа!"
29        else:
30          print u"Безумие охватило тебя, и ты упал в бассейн
            с грязью. Отличная работа!"
31
32  else:
33      print u"Безумие охватило тебя, и ты выстрелил себе в ухо.
        Отличная работа!"
```

Ключевым моментом здесь является то, что вы поместили одни конструкции `if` внутрь других как код, который может быть выполнен. Это очень мощная возможность, которая может быть использована для создания «вложенных» решений, в которых одна ветвь ведет к другой и т. д.

Убедитесь, что вы понимаете концепцию конструкций `if`, вложенных друг в друга. Практическое задание в этом упражнении поможет разобраться.

Результат выполнения

Ниже представлен вывод оболочки командной строки, где я попробовал сыграть в игру. И проиграл.

Сеанс упражнения 31

```
$ python ex31.py
Ты находишься в темной комнате с двумя дверями. Ты войдешь в дверь
№ 1 или № 2?
> 1
В этой комнате гигантский медведь поедает чизкейк. Твои действия?
1. Отберешь чизкейк.
```

```
2. Крикнешь медведю в ухо.
> 2
Медведь укусил тебя за ногу. Отличная работа!
```

Практические задания

Добавьте в игру новые вопросы и ответы на них. Разверните игровой процесс, насколько сможете.

Распространенные вопросы

Можно ли заменить конструкции `elif` комбинациями `if/else`?

В некоторых случаях; зависит от того, как написана каждая конструкция `if/else`. Кроме того, Python будет проверять каждую из них, а не только первую ложную, как в случае с конструкциями `if/elif/else`. Попробуйте изменить код, чтобы увидеть различия.

Как ограничить число определенным диапазоном?

У вас есть два варианта: классическая нотация (`0 < x < 10` или `1 <= x < 10`) или код `x in range(1, 10)`.

Как добавить дополнительные варианты в блоки `if/elif/else`?

Очень легко, просто добавьте дополнительные блоки `elif` для каждого варианта.

Циклы и списки

Теперь вы умеете писать гораздо более интересные программы. На данный момент вы должны понимать, что можете объединять изученные ранее элементы с конструкциями `if` и логическими выражениями.

Тем не менее программы также должны выполнять повторяющиеся инструкции как можно быстрее. Для решения этой задачи в упражнении мы будем использовать цикл `for`, который применяется для создания и вывода различных списков. По мере выполнения упражнения вы поймете, как это делается. Я не расскажу прямо сейчас. Вы должны сами понять принцип работы цикла.

Прежде чем мы начнем использовать цикл `for`, важно определить способ хранения результатов выполнения циклов. Лучший способ реализовать это — списки. Список представляет собой именно то, что отражает его название, — контейнер элементов, которые организованы надлежащим образом. Это несложно, вам только понадобится выучить новый синтаксис. Во-первых, вот так можно создать список.

```
hairs = [u'шатенка', u'блондинка', u'рыжая']
cars = ['volvo', 'mercedes', 'kia']
weights = [1, 2, 3, 4]
```

Внимание! В коде используйте оператор форматирования `%s` вместо `%r`, чтобы получить нормальный текст списков в выводе.

Список начинается с символа `[` (левая скобка), который «открывает» его. Затем вы перечисляете элементы списка через запятую, как вы делали с аргументами функции. И, наконец, список завершается символом `]` (правая скобка). Юникод-текст, например русский, предваряется символом `u` (сравните первые две строки кода в примере выше). Интерпретатор Python принимает этот список и все его содержимое и присваивает его переменной.

Внимание! И вот теперь все усложняется для людей, которые умеют программировать. Помните, как в предыдущем упражнении вы помещали одни конструкции `if` внутрь других? Вероятно, вы поняли это не сразу, но в программировании это обычная практика. Вы можете создавать функции, которые вызывают другие функции, имеющие конструкции `if` с многоуровневыми вложенными списками. Если вы видите в коде нечто подобное, непонятное на первый взгляд, достаньте карандаш с бумагой и разберите код на элементы, пока не поймете его.

Теперь мы создадим несколько списков, используя циклы, и выведем их.

ex32.py

```
1   the_count = [1, 2, 3, 4, 5]
2   fruits = [u'яблоко', u'апельсин', u'персик', u'абрикос']
3   change = [1, '25', 2, '50', 3, '75']
4
5   # цикл for первого типа обрабатывает список
6   for number in the_count:
7     print u"Счетчик %d" % number
8
9   # то же, что и выше
10  for fruit in fruits:
11    print u"Фрукт: %s" % fruit
12
13  # также можно обрабатывать смешанные списки
14  # обратите внимание, что используется оператор %r, так как
    неизвестен тип значения
15  for i in change:
16    print u"Я получил %r" % i
17
18  # также мы можем создавать списки, начнем с пустого
19  elements = []
20
21  # затем используется функция range для ограничения диапазона
22  for i in range(0, 6):
23    print u"Добавление %d в список." % i
```

```
24      # append — функция для добавления элементов и списков
25      elements.append(i)
26
27  # теперь мы их выводим
28  for i in elements:
29      print u"Номер элемента: %d" % i
```

Результат выполнения

Сеанс упражнения 32

```
$ python ex32.py
Счетчик 1
Счетчик 2
Счетчик 3
Счетчик 4
Счетчик 5
Фрукт: яблоко
Фрукт: апельсин
Фрукт: персик
Фрукт: абрикос
Я получил 1
Я получил '25'
Я получил 2
Я получил '50'
Я получил 3
Я получил '75'
Добавление 0 в список.
Добавление 1 в список.
Добавление 2 в список.
Добавление 3 в список.
Добавление 4 в список.
Добавление 5 в список.
Номер элемента: 0
Номер элемента: 1
Номер элемента: 2
Номер элемента: 3
Номер элемента: 4
Номер элемента: 5
```

Практические задания

1. Изучите, как применяется функция для работы с диапазоном значений. Поищите справочные сведения о функции `range`, чтобы понять принцип ее работы.

2. Нельзя ли полностью избавиться от цикла `for` в строке 22 и применить функцию `range(0,6)` непосредственно к переменной `elements`?

3. Найдите в документации к Python информацию о списках и изучите ее. Какие другие операции могут производиться со списками, кроме `append`?

Распространенные вопросы

Как создать двумерный список?

Например, так: `[[1,2,3],[4,5,6]]`.

Списки и массивы — это одно и то же?

Это зависит от языка и реализации. Обычно списки существенно отличаются от массивов из-за различий в реализации. В Ruby списки называются массивами. В Python — списками. Просто называйте их списками, потому что это принято разработчиками на Python.

Почему в циклах `for` можно использовать переменные, которые еще не объявлены?

Переменная объявляется циклом `for` при его выполнении, инициализирующем ее к текущему элементу итерации цикла.

Почему цикл `for i in range(1, 3)` выполняется только два раза, а не три?

Функция `range()` обрабатывает числа только от первого до последнего, *не включая последнее*. Таким образом, в вышеупомянутом примере она останавливается на двух, а не на трех. Это наиболее распространенный цикл.

Для чего предназначена функция `elements.append()`?

Она добавляет элемент в конец списка. Запустите Python в оболочке командной строки и выполните несколько операций со списком. Каждый раз, когда вы сталкиваетесь с подобными затруднениями в понимании принципа работы, экспериментируйте в интерактивном режиме в оболочке командной строки.

УПРАЖНЕНИЕ 33

Циклы while

Теперь вы познакомитесь с новым циклом — `while`. Цикл `while` выполняет блок кода, пока логическое выражение истинно.

Небольшой отступ: напомню вам важное замечание относительно оформления кода. Вы помните, что если строку кода закончить символом : (двоеточие), мы сообщим Python, что начинается новый блок кода? Затем мы предваряем строки кода отступами, обозначая, что это код блока. Код программы структурируется таким образом, чтобы интерпретатор Python «понимал», что вы имеете в виду. Если вам непонятна эта концепция, вернитесь к предыдущим упражнениям и поработайте дополнительно с конструкциями `if`, функциями и циклами `for`.

Дальнейшие упражнения будут обучать вас этой концепции, подобно тому как вы заучивали логические выражения.

Вернемся к циклам `while`. Все, что они делают, это выполняют проверку, подобно конструкции `if`, но вместо однократного запуска блока кода они после его выполнения переходят в начало и повторяют выполнение кода. Код продолжает выполняться до тех пор, пока выражение не станет ложным.

Налицо проблема с циклами `while`: иногда те становятся бесконечными. Это замечательно, если ваше намерение состоит в том, чтобы цикл выполнялся до конца времен. В противном случае требуется, чтобы циклы рано или поздно завершались.

Чтобы избежать этих проблем, существуют правила, которым необходимо следовать.

1. Используйте циклы `while` как можно реже. Обычно лучше использовать циклы `for`.

2. Внимательно изучите код своих инструкций `while`, чтобы убедится, что выражения, проверяемые этими циклами, в определенный момент станут ложными.

3. Если есть сомнения, выведите тестируемую переменную в верхней и нижней части цикла `while`, чтобы увидеть, как он работает.

В этом упражнении вы научитесь управлять циклами `while`, выполнив следующий сценарий.

ex33.py

```
1   i = 0
2   numbers = []
3
4   while i < 6:
5     print u"Вверху значение i равно %d" % i
6     numbers.append(i)
7
8     i = i + 1
9     print u"Текущие значения: ", numbers
10    print u"Внизу значение i равно %d" % i
11
12
13  print u"Значения: "
14
15  for num in numbers:
16    print num
```

Результат выполнения

Сеанс упражнения 33

```
$ python ex33.py
Вверху значение i равно 0
Текущие значения: [0]
Внизу значение i равно 1
Вверху значение i равно 1
Текущие значения: [0, 1]
Внизу значение i равно 2
Вверху значение i равно 2
Текущие значения: [0, 1, 2]
Внизу значение i равно 3
Вверху значение i равно 3
Текущие значения: [0, 1, 2, 3]
Внизу значение i равно 4
Вверху значение i равно 4
Текущие значения: [0, 1, 2, 3, 4]
```

```
Внизу значение i равно 5
Вверху значение i равно 5
Текущие значения: [0, 1, 2, 3, 4, 5]
Внизу значение i равно 6
Значения:
0
1
2
3
4
5
```

Практические задания

1. Преобразуйте цикл `while` в вызываемую функцию и замените `6` в проверяемом выражении (`i < 6`) на переменную.

2. Теперь, применив эту функцию, перепишите сценарий, чтобы использовать разные числа.

3. Добавьте еще одну переменную в качестве аргумента функции, чтобы изменить `+1` в строке 8 на постоянно увеличивающееся значение.

4. Перепишите сценарий, чтобы использовать эту функцию и увидеть результаты ее работы.

5. Теперь включите в сценарий циклы `for` и функцию `range`. В середине все равно нужен инкрементор? Что произойдет, если вы не удалите его?

6. Если программа ведет себя странно и непредсказуемо (что может быть в результате экспериментов), нажмите сочетание клавиш **Ctrl+C**, чтобы завершить ее работу.

Распространенные вопросы

В чем заключается разница между циклами `for` и `while`?

Цикл `for` может выполнить только обход (цикл) группы элементов. Цикл `while` выполняется, пока проверяемое выражение истинно. Однако циклы

`while` сложнее для применения, и обычно возможностей циклов `for` достаточно для решения многих задач.

Циклы — это очень сложно. Как разобраться в них?

Основная причина, почему люди не понимают циклы, потому что они не могут следовать «прыжку», который делает этот код. При выполнении цикл обходит блок кода, а затем возвращается в начало блока. Чтобы понять принцип работы, укажите инструкции `print` перед циклом, в начале цикла, в середине и в конце. Изучите вывод и попытайтесь понять, что происходит.

УПРАЖНЕНИЕ 34

Доступ к элементам списка

Списки весьма полезны, но только если вы можете получить доступ к элементам внутри них. Вы уже умеете обходить элементы списка по порядку, но как быть, если вы хотите извлечь, скажем, пятый элемент? Вы должны знать, как получить доступ к элементам списка. Ниже показано, как можно получить доступ к первому элементу списка.

```
animals = [u'медведь', u'тигр', u'пингвин', u'зебра']
bear = animals[0]
```

Вы берете список животных, а затем получаете только первый (1-й) элемент, используя значение `0`?! Почему? Как это работает? Из-за особенностей математических вычислений Python начинает перечисление элементов списков с 0, а не 1. Это кажется странным, но есть много преимуществ такого подхода, даже несмотря на то что обычно элементы извлекаются в произвольном порядке.

Лучший способ объяснить такое поведение — продемонстрировать разницу между использованием чисел вами и программистами.

Представьте, что четыре животных из нашего списка (`[u'медведь', u'тигр', u'пингвин', u'зебра']`) состязаются на скорость. Они выигрывают в том порядке, в котором указаны в этом списке. Соревнование было очень интересным, потому что животные не съели друг друга. Ваш друг наконец показывает на часы и хочет знать, кто выиграл. При этом ваш друг разве говорит «Эй, кто финишировал нулевым»? Нет, он говорит: «Эй, кто финишировал первым?»

Это происходит потому, что порядок животных имеет важное значение. Вы не можете использовать второе животное без первого (`1`) и третье без второго. Также невозможно использовать «нулевое» животное, так как ноль не значит ничего. Как вы можете, имея ничего, выиграть гонку? Это просто не имеет смысла. Мы называем эти числа «порядковыми», так как они определяют порядок элементов (объектов).

Программисты, однако, не могут думать так же, потому что способны выбрать любой элемент из списка в любой момент времени. Для программистов приведенный выше список больше похож на колоду карт. Если они

хотят выбрать тигра, они вытаскивают его. Если зебру, то берут ее. В этом случае необходимо извлекать элементы из списков в случайном порядке, для чего нужно последовательно указать для каждого элемента адрес (индекс). При этом лучше всего начинать индексы с 0. Поверьте мне на слово: математический подход — более простой способ осуществления такого доступа. Это так называемые кардинальные, которые вы можете выбрать в случайном порядке и обязательно использовать нулевой элемент. Как это поможет вам работать со списками? Проще говоря, каждый раз, когда вы говорите: «Я хочу выбрать третье животное», вы должны перевести его «порядковое» число к «кардинальному» путем вычитания единицы. «Третье» животное с индексом 2 — это пингвин. Вы должны запомнить это, потому что потратили всю свою жизнь, используя порядковые номера, и теперь должны научиться думать кардинальными числами. Просто вычтите 1, и все будет хорошо.

Запомните: порядковые == по порядку, с 1; кардинальные == в случайном порядке, с 0.

Давайте попрактикуемся. Возьмите следующий список животных и выполните упражнение, используя порядковое или кардинальное число животного. Помните, что если я говорю «первое», «второе» и так далее, то использую порядковый номер, поэтому вычитаем единицу. Если я указываю кардинальное число (0, 1, 2), используйте его без изменений.

```
animals = [u'медведь', u'питон', u'пингвин', u'кенгуру', u'кит',
u'утконос']
```

1. Животное с индексом 1.
2. Третье животное.
3. Первое животное.
4. Животное с индексом 3.
5. Пятое животное.
6. Животное с индексом 2.
7. Шестое животное.
8. Животное с индексом 4.

Для каждого пункта напишите полное предложение вида: «Первое животное имеет индекс 0, и это медведь». Затем скажите это в обратном порядке: «Животное с индексом 0 является первым, и это медведь». С помощью Python проверьте свои ответы.

Практические задания

1. Найдите и прочитайте в Интернете информацию о порядковых и кардинальных числах.

2. С учетом разницы между этими типами чисел можете ли вы объяснить, почему 2010 год в дате «1 января 2010 года» реально 2010-й, а не 2009-й? (Подсказка: вы не можете выбирать год в случайном порядке.)

3. Напишите еще несколько списков и разработайте аналогичные индексы, пока вы не запомните тему.

4. С помощью интерпретатора Python проверьте свои ответы на списки из задания № 3.

Внимание! Программисты посоветуют вам на эту тему почитать труды парня по фамилии Дейкстра. Я рекомендую избегать их, если вам хочется продолжать учиться программировать, а не терять время.

Ветви и функции

Итак, вы изучили конструкции `if`, функции и списки. Теперь пришло время поработать головой. Введите следующий код и попробуйте разобраться, что он делает.

Внимание! Обратите внимание, что в начале сценария приводится код для управления кодировками символов и правильной обработки значений функции `raw_input`, содержащих русские буквы. Для тех же целей используется код `.decode(sys.stdin.encoding or locale.getpreferredencoding(True))`.

---

ex35.py

```
# -*- coding: utf- 8 -*-

import codecs, sys
outf = codecs.getwriter('cp866')(sys.stdout, errors='replace')
sys.stdout = outf

from sys import exit

def gold_room():
  print u"Здесь полно золота. Сколько кг ты унесешь?"

  next = raw_input("> ").decode(sys.stdin.encoding
          or locale.getpreferredencoding(True))
  if "0" in next or "1" in next:
      how_much = int(next)
    else:
      dead(u"Эй, ввести надо число!")

  if how_much < 50:
      print u"Шикарно! Ты не жадный, поэтому выигрываешь!"
      exit(0)
    else:
      dead(u"Ах ты жадина!")
```

```
23
24
25  def bear_room():
26      print u"Здесь сидит медведь."
27      print u"У медведя бочка с медом."
28      print u"Медведь закрыл собой дверь выхода."
29      print u"Как переместить медведя? Отобрать мед или подразнить
        медведя?"
30      bear_moved = False
31
32      while True:
33          next = raw_input("> ").decode(sys.stdin.encoding
                        or locale.getpreferredencoding(True))
34
35          if next == u"отобрать мед":
36              dead(u"Медведь посмотрел на тебя и ударил лапой
                по лицу.")
37          elif next == u"подразнить медведя" and not bear_moved:
38              print u"Медведь отошел от двери. Вы можете войти
                в нее. Подразнить медведя или войти в дверь?"
39              bear_moved = True
40          elif next == u"подразнить медведя" and bear_moved:
41              dead(u"Медведь разозлился и откусил тебе ногу.")
42          elif next == u"войти в дверь" and bear_moved:
43              gold_room()
44          else:
45              print u"Понятия не имею, что происходит."
46
47
48  def cthulhu_room():
49      print u"На вас смотрит великий и ужасный Ктулху."
50      print u"Он смотрит на тебя, и ты начинаешь сходить с ума."
51      print u"Убежать или съесть свою голову?"
52
53      next = raw_input("> ").decode(sys.stdin.encoding
                or locale.getpreferredencoding(True))
54
55      if u"убежать" in next:
56          start()
57      elif u"съесть свою голову" in next:
58          dead(u"Хм, а это даже и вкусно!")
59        else:
60          cthulhu_room()
61
```

```
62
63  def dead(why):
64      print why, u"Великолепно!"
65       exit(0)
66
67  def start():
68      print u"Ты в темной комнате."
69      print u"Отсюда ведут две двери, налево и направо."
70      print u"Куда ты повернешь?"
71
72      next = raw_input("> ").decode(sys.stdin.encoding
            or locale.getpreferredencoding(True))
73
74      if next == u"налево":
75          bear_room()
76      elif next == u"направо":
77          cthulhu_room()
78       else:
79          dead(u"Ты ходишь из комнаты в комнату, пока не умираешь
            с голоду.")
80
81
82  start()
```

Результат выполнения

Ниже показано, как я попробовал поиграть.

Сеанс упражнения 35

```
$ python ex35.py
Ты в темной комнате.
Отсюда ведут две двери, налево и направо.
Куда ты повернешь?
> налево
Здесь сидит медведь.
У медведя бочка с медом.
Медведь закрыл собой дверь выхода.
Как переместить медведя? Отобрать мед или подразнить медведя?
> подразнить медведя
Медведь отошел от двери. Вы можете войти в нее. Подразнить медведя
или войти в дверь?
> войти в дверь
```

```
Здесь полно золота. Сколько кг ты унесешь?
> 1000
Ах ты, жадина! Великолепно!
```

Практические задания

1. Составьте схему игры и с помощью линий и блоков продемонстрируйте игровой процесс.

2. Устраните все свои ошибки, включая орфографические.

3. Напишите комментарии к функциям, работа которых вам неясна. Помните про документирование?

4. Измените игровой процесс. Как вы могли бы упростить и расширить его?

5. Функция `gold_room` реализует довольно странный способ ввода числа. Перечислите все недостатки этого способа. Можете ли вы улучшить код функции, заменив простую проверку 1 или 0 в качестве числа? Для подсказки посмотрите, как работает функция `int()`.

Распространенные вопросы

Ничего не понимаю! Как работает эта программа!?

Каждый раз, когда вам сложно понять код программного обеспечения, просто пишите комментарий выше каждой его строки, объясняя, для чего он используется. После этого, получив информацию о непонятной строке кода, исправьте неправильные комментарии. Затем составьте блок-схему или напишите пару абзацев текста с описанием, как работает код. Если вы сделаете это, то во всем разберетесь.

Для чего используется строка кода `while True:`?

Она создает бесконечный цикл.

Для чего используется код `exit(0)`?

Во многих операционных системах программа может быть прервана с помощью команды `exit(0)`, и в зависимости от переданного в команду числа завершение работы будет считаться ошибочным или нет. Если выполнить `exit(1)`, то это будет считаться выходом с ошибкой, а если `exit(0)` — корректным завершением. Причина, по которой это отличается от нормальной булевой логики (где `0==False`), заключается в том, что вы можете использовать различные значения, которые указывают на различные варианты ошибок. Можно использовать `exit(100)` для ошибки, результат которой отличается от `exit(2)` или `exit(1)`.

Почему функция `raw_input()` в некоторых случаях оформляется как `raw_input('> ')`?

Данный параметр в функции **`raw_input()`** позволяет вывести приглашение к вводу для пользователя.

Разработка и отладка

Вы уже научились работать с конструкциями `if`, и теперь я собираюсь обучить вас некоторым правилам, позволяющим избегать неприятных ситуаций при использовании циклов `for` и `while`. Также вы получите несколько советов по отладке и научитесь разбираться с некоторыми программными проблемами. И, наконец, вы создадите небольшую игру, подобную рассмотренной в предыдущем упражнении, но с некоторыми отличиями.

Правила конструкций if

1. Каждая конструкция `if` должна включать инструкцию `else`.

2. Если инструкция `else` не используется и не должна выполняться, нужно использовать функцию `die` в конструкции `else`, которая выводит сообщение об ошибке и прекращает работу, как мы это делали в предыдущем упражнении. Так вы обнаружите *многие* ошибки.

3. Ни в коем случае не вкладывайте конструкции `if` более чем на два уровня вложенности и всегда используйте один уровень вложенности. Это означает, что если вы помещаете одну конструкцию `if` во вторую, то вторую конструкцию `if` следует помещать в другую функцию.

4. Оформляйте конструкции `if` наподобие текстовых абзацев, в которых каждая инструкция `if`, `elif` и `else` группируется в виде набора предложений. Помещайте пустые строки сверху и снизу.

5. Ваши логические условия должны быть просты. Если простоту соблюсти не удается, поместите вычисления в предстоящие переменные в вашей функции и используйте подходящие имена переменных.

Если вы будете следовать этим простым правилам, ваш код станет лучше, чем у большинства программистов. Вернитесь к предыдущему упражнению и взгляните, следовал ли я всем этим правилам. Если нет, то исправьте его.

Внимание! Никогда не следуйте слепо правилам в реальной жизни. Во время обучения вы должны следовать им, чтобы закрепить полученные знания, но в жизни эти правила иногда использовать попросту глупо. Если вы считаете, что то или иное правило использовать не стоит, то не делайте этого.

Правила циклов

1. Используйте циклы `while`, только если требуется выполнять цикл бесконечно. Это актуально только для Python; другие языки программирования отличаются.

2. Используйте циклы `for` во всех остальных случаях, особенно если количество элементов, используемых в цикле, неизменяемо или лимитировано.

Советы по отладке

1. Не используйте «отладчики». Работа отладчика сравнима с УЗИ всего тела больного человека. Вы не получите какой-либо конкретной полезной информации, только множество специфичных данных, не способных помочь на данном этапе и сбивающих с толку.

2. Лучший способ отладки программы заключается в использовании команды `print` и выводе с помощью нее значений переменных в разных позициях кода программы, чтобы обнаружить ошибки.

3. Выполняйте фрагменты вашей программы по мере набора кода. Не пишите огромный файл кода до конца, чтобы затем выполнить его. Лучше набрать пару инструкций, выполнить, а затем при необходимости исправить.

Домашнее задание

Теперь можно написать игру, подобную той, что я продемонстрировал в предыдущем упражнении. Это может быть любая игра на ваш вкус. Потратьте

неделю, чтобы сделать ее как можно интереснее. Для практики используйте списки, функции и модули (те, что мы использовали в упражнении 13). Используйте как можно больше инструкций Python в своей игре.

Кроме того, прежде чем начать набор кода, зарисуйте дизайн вашей игры. Нарисуйте на бумаге комнаты, монстров, ловушки и задания, которые игрок должен выполнить, запустив вашу игру.

После этого начните набирать код. Если возникнут проблемы с достижением каких-либо целей, нарисованных на бумаге, измените рисунок и код следующим образом.

Совет напоследок: все программисты иррационально паникуют перед началом каждого нового большого проекта. Они откладывают программирование, чтобы избежать этого страха, а в итоге не могут завершить (или даже начать) свою программу. И со мной случается подобное. С каждым. Проще всего справиться с ситуацией поможет дробление большого проекта на несколько небольших отдельных задач, которые следует выполнять по очереди.

Попробуйте сначала сделать небольшую версию игры, затем расширить и т. д.

Знакомство с символами

Пришло время повторить символы и ключевые слова языка Python, которые вы изучили, и рассмотреть новые, для использования в будущих упражнениях. В этом разделе я перечислил все символы и ключевые слова Python, которые просто необходимо знать.

Для максимальной эффективности читайте по очереди ключевые слова и попытайтесь вспомнить, что делает каждое из них. Запишите то, что вспомнили. Затем найдите в Интернете информацию о каждом слове и проверьте, верно ли записали его предназначение. Задание может показаться трудным, потому что некоторые из слов может быть сложно найти, но тем не менее попытайтесь это сделать.

Если вы ошиблись с предназначением того или иного символа/слова, восстанавливая по памяти, возьмите пустую карточку и напишите на ней правильное определение. Затем тренируйтесь, пытаясь запомнить его. Если вы ранее не знали о том или ином символе/слове, также запишите его (с определением) на карточку и сохраните на будущее.

И, наконец, попробуйте поработать с каждым из них, написав небольшую программу на языке Python. Суть в том, чтобы выяснить, для чего предназначен символ/слово, убедиться, что вы используете его правильно, исправить код в случае ошибки, а затем использовать символ/слово, чтобы закрепить полученные знания.

Ключевые слова

- `and`
- `del`
- `from`
- `not`
- `while`
- `as`

- elif
- global
- or
- with
- assert
- else
- if
- pass
- yield
- break
- except
- import
- print
- class
- exec
- in
- raise
- continue
- finally
- is
- return
- def

- `for`
- `lambda`
- `try`

Типы данных

Запишите, для чего предназначен каждый из типов данных и как его использовать. Например, в случае со строками напишите, как вы создаете строку. В случае с числами запишите несколько чисел.

- `True`
- `False`
- `None`
- `строки`
- `числа`
- `числа с плавающей точкой`
- `списки`

Управляющие последовательности

Напишите код с управляющими последовательностями, чтобы проверить свои знания.

- `\\`
- `\'`
- `\"`
- `\a`
- `\b`

- `\f`
- `\n`
- `\r`
- `\t`
- `\v`

Форматирование строк

То же касается и форматирования строк: используйте различные форматы, приведенные ниже, в своем коде, чтобы разобраться в их предназначении.

- `%d`
- `%i`
- `%o`
- `%u`
- `%x`
- `%X`
- `%e`
- `%E`
- `%f`
- `%F`
- `%g`
- `%G`
- `%c`

- `%r`
- `%s`
- `%%`

Операторы

Некоторые из операторов могут быть не знакомы вам, но в любом случае попробуйте найти информацию о каждом из них. Разберитесь, для чего они предназначены, и, если вы до сих пор не пользовались тем или иным оператором, сохраните информацию о них для последующего использования.

- `+`
- `-`
- `*`
- `**`
- `/`
- `//`
- `%`
- `<`
- `>`
- `<=`
- `>=`
- `==`
- `!=`
- `<>`

- `()`
- `[]`
- `{}`
- `@`
- `,`
- `:`
- `.`
- `=`
- `;`
- `+=`
- `-=`
- `*=`
- `/=`
- `//=`
- `%=`
- `**=`

Потратьте несколько дней на данное упражнение. Может быть, у вас уйдет даже меньше времени. Главное — попытаться разобраться со всеми этими символами/словами и запомнить их. Если некоторые из них не поддаются запоминанию, не волнуйтесь — вы освоите их позднее.

Чтение кода

Теперь вы должны попробовать читать код на языке Python. Попробуйте открыть любой фрагмент кода и заимствовать идеи, которые обнаружите.

В реальности вы знаете достаточно, чтобы читать код, но, возможно, какие-то его фрагменты вам будут непонятны. В этом упражнении вы узнаете некоторые приемы, помогающие понять код, написанный другими программистами.

Во-первых, с помощью принтера распечатайте код, в котором вы хотите разобраться. Да, именно распечатайте, потому что ваши глаза и мозг больше привыкли к чтению текста с бумаги, а не с экрана компьютера. Распечатывайте пару-тройку страниц за один раз.

Во-вторых, во время чтения распечатки обратите особое внимание на следующее.

1. Функции и что они делают.
2. Позиции присвоения значений каждой из переменных.
3. Все переменные с одинаковыми именами в разных частях кода программы. Они могут приводить к ошибкам при выполнении.
4. Все конструкции `if` без инструкций `else`. Правильно ли это?
5. Все циклы `while`, которые могут не завершать выполнение.
6. И, наконец, какие-либо части кода, которые вы не можете понять, не важно, по какой причине.

В-третьих, после того как вы пометите все это в распечатке, попытайтесь объяснить предназначение строк кода самому себе, написав соответствующие комментарии по мере чтения кода. Объясните суть работы каждой функции, какие переменные вовлечены в этот процесс и все, что вы можете понять, читая этот код.

И, наконец, во всех сложных местах кода отслеживайте значения каждой переменной построчно, функцию за функцией. Для удобства распечатайте еще одну копию листинга и напишите на полях значение каждой переменной, которую нужно отслеживать.

После того как вы достаточно хорошо разобрались в том, как работает код, вернитесь к компьютеру и прочитайте листинг еще раз. Возможно, вы увидите некоторый новый функционал, не замеченный вами ранее. Продолжайте работать с кодом и делать распечатки, пока не разберетесь в коде программы полностью.

Практические задания

1. Разберитесь, что такое «блок-схема», и составьте несколько штук.

2. Если вы обнаружили ошибки в коде, который читаете, попытайтесь исправить их и отправьте разработчику информацию о внесенных изменениях.

3. В качестве альтернативы распечатанным на принтере листингам вы можете указать в коде комментарии со своими примечаниями. Возможно, такие комментарии в будущем помогут разобраться в коде другому «читателю».

Распространенные вопросы

В чем разница между форматами `%d` и `%i`?

Нет никакой разницы, кроме того что программисты чаще используют оператор `%d`. Так сложилось исторически.

Как в Интернете искать информацию об этих символах/словах?

Просто добавляйте слово **Python** в поисковый запрос. Например, чтобы найти сведения о ключевом слове `yield`, выполните поиск по запросу **yield python**.

Работа со списками

Вы уже знакомы со списками. Когда вы изучали циклы `while`, вы «добавляли» числа в конец списка и выводили их. Были также практические задания, в которых требовалось найти в документации к Python информацию о том, что можно делать со списками. Это было довольно давно, поэтому найдите соответствующий раздел в книге и освежите свои знания, если не понимаете, о чем идет речь.

Нашли, вспомнили? Прекрасно. У вас был список, и вы вызывали функцию `append`. В любом случае, понимаете ли вы, что происходит, или нет, давайте разберемся, что можно делать со списками.

При вводе кода `mystuff.append('hello')` вы фактически инструктируете Python выполнить цепочку событий со списком `mystuff`. Вот что происходит.

1. Интерпретатор Python «видит», что вы упомянули имя `mystuff`, и изучает эту переменную. Возможно, потребуется вернуться назад, чтобы определить, создали ли вы ее с помощью оператора =, а также является ли это имя аргументом функции или глобальной переменной. В любом случае первым делом обнаруживается имя `mystuff`.

2. После обнаружения переменной `mystuff` интерпретатор «видит» оператор `.` (символ точки) и начинает «изучать» переменные, которые являются частью `mystuff`. Поскольку `mystuff` — это список, интерпретатор «знает», что `mystuff` имеет совокупность функций.

3. Затем он добирается до фрагмента `append` и сравнивает имя «append» со всеми именами, которыми (как утверждает `mystuff`) обладает `mystuff`. Если среди них есть `append` (а это так), тогда Python задействует его.

4. Далее Python обнаруживает круглую скобку `(` и «понимает», что это функция. Здесь она вызывается (другими словами — запускается или выполняется) как обычно, за исключением того, что при вызове имеет дополнительный аргумент.

5. Этот дополнительный аргумент — `mystuff`! Странно, не правда ли? Но это принцип работы Python, поэтому лучше просто запомнить такое поведение и сделать вид, что все в порядке. Что

происходит потом, в конце, — это вызов функции, которая выглядит как `append(mystuff, 'hello')`, вместо написанного нами кода `mystuff.append('hello')`.

По большей части вам не нужно так детально погружаться в процесс работы кода, но это может помочь в случаях, когда вы получаете в Python сообщения об ошибках наподобие этой.

```
$ python
Python 2.6.5 (r265:79063, Apr 162010, 13:57:41)
[GCC4.4.3] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>> class Thing(object):
...  def test(hi):
...       print "hi"
...
>>> a = Thing()
>>> a.test("hello")
Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
TypeError: test() takes exactly 1 argument (2 given)
>>>
```

Вам непонятно, что здесь показано? Я набрал в Python некоторый код, включая классы, с которыми вы еще не знакомы, но изучите в дальнейшем. Это сейчас не главное. Как вы видите из результатов вывода, Python сообщает, что функция `test()` ожидала 1 аргумент (а получила 2). Если вы видите подобную ошибку, Python изменил `a.test("hello")` на `test(a, "hello")` a.test ("привет"), и что где-то произошла путаница, приведшая к тому, что к `a` не был добавлен аргумент.

Полученная информация может показаться сложной (и объемной) для запоминания, но в дальнейшем вы выполните несколько упражнений на эту тему. Чтобы начать разбор полетов, ниже я привел упражнение, в котором различными способами смешиваются строки и списки.

ex38.py

```
1   ten_things = u"Apples Oranges Crows Telephone Light Sugar"
2
3   print u"Погодите, тут меньше 10 объектов. Давайте исправим это."
```

```
4
5   stuff = ten_things.split(' ')
6   more_stuff = ["Day", "Night", "Song", "Frisbee", "Corn", "Banana",
    "Girl", "Boy"]
7
8   while len(stuff)!= 10:
9     next_one = more_stuff.pop()
10    print u"Добавляем: ", next_one
11    stuff.append(next_one)
12    print u"Теперь у нас %d объектов." % len(stuff)
13
14  print u"Итак: ", stuff
15
16  print u"Давайте кое-что сделаем с нашими объектами."
17
18  print stuff[1]
19  print stuff[- 1] # хм! интересно
20  print stuff.pop()
21  print ' '.join(stuff) # что? круто!
22  print '#'.join(stuff[3:5]) # просто супер!
```

Результат выполнения

Сеанс упражнения 38

```
$ python ex38.py
Погодите, тут меньше 10 объектов. Давайте исправим это.
Добавляем: Boy
Теперь у нас 7 объектов.
Добавляем: Girl
Теперь у нас 8 объектов.
Добавляем: Banana
Теперь у нас 9 объектов.
Добавляем: Corn
Теперь у нас 10 объектов.
Итак: ['Apples', 'Oranges', 'Crows', 'Telephone', 'Light', 'Sugar',
'Boy', 'Girl', 'Banana', 'Corn']
Давайте кое-что сделаем с нашими объектами.
Oranges
Corn
Corn
Apples Oranges Crows Telephone Light Sugar Boy Girl Banana
Telephone#Light
```

Практические задания

1. Возьмите каждую вызываемую функцию и разложите ее на шаги, как описано выше, чтобы понять, что делает Python. Например, `' '.join(things)` можно представить как `join(' ', things)`.

2. Переведите на человеческий язык оба представления вызова функции. Например, `' '.join(things)` можно прочитать как «с помощью функции `join` объекты `things` выводятся в строку с символом `' '`(пробел) между ними». Код `join(' ', things)` можно прочитать так: «вызывается функция вывода в строку `join` с символом `' '`(пробел) и объектами `things`». Так вы поймете: по сути, они делают одно и то же.

3. Поищите во Всемирной паутине информацию об объектно-ориентированном программировании. Сбиты с толку? Я в свое время тоже. Но не волнуйтесь. Вы уже и так знаете достаточно, чтобы рискнуть, и постепенно будете узнавать все больше и больше.

4. Разузнайте, что такое «классы» в Python. Не вздумайте читать о том, как классы используются в других языках программирования. Вы только запутаетесь.

5. Что общего между функцией `dir(чего-то)` и «классом» `чего-то`?

6. Если вы не имеете ни малейшего представления, о чем я говорю, не волнуйтесь. Программисты любят делать вид, что очень умны, поэтому они придумали объектно-ориентированное программирование, сократили название до ООП, а затем стали использовать везде, где только можно. Если вам все это кажется очень сложным, можно попробовать использовать «функциональное программирование».

Распространенные вопросы

Разве вы не говорили о том, что не следует использовать циклы `while`?

Да, говорил. Но также учитывайте, что иногда вы можете нарушать правила, если на это есть веские основания.

Для чего нужен код `stuff[3:5]`?

Он позволяет получить «кусочек» информации из списка `stuff`, от элемента 3 до элемента 4, не включая элемент 5. Аналогичным образом работает код `range(3,5)`.

Почему не работает код `join(' ', stuff)`?

Тот способ, который приводится в документации для `join`, не является правильным. Функция не работает так, как описано, и вместо этого вам необходимо поместить метод, который вы вызываете для вставляемой строки, перед списком, который следует присоединить.

УПРАЖНЕНИЕ 39

Словари, мои словари...

Сейчас вы познакомитесь с другим контейнером, который можно использовать, чтобы упростить работу по программированию. Это наиболее полезный контейнер, и его название — *словарь*.

Так они называются в языке Python. В других языках программирования словари используются под названием хеш-таблиц. Я предпочитаю использовать оба названия, хотя это и не имеет значения. Рассмотрим процесс работы со словарями в сравнении со списками. Как видно из примера ниже, список позволяет делать следующее:

```
>>> things = ['a', 'b', 'c', 'd']
>>> print things[1]
b
>>> things[1] = 'z'
>>> print things[1]
z
>>> print things
['a', 'z', 'c', 'd']
>>>
```

Вы можете использовать числа для «индексации» элементов в списке, то есть с помощью этих чисел выяснять содержимое списков. На данный момент вам должно быть известно об этой особенности списков. При этом следует учитывать, что вы можете использовать только числа, чтобы извлечь элементы из списка.

Как вы можете догадаться, словари позволяют вам использовать *какие угодно ключи*, а не только числа. Именно так, словари не используют подобные ассоциации, вне зависимости от значений. Взгляните.

```
>>> stuff = {'name': u'Михаил', 'age': 36, 'height': 6*12+2}
>>> print stuff['name']
Михаил
>>> print stuff['age']
36
```

```
>>> print stuff['height']
74
>>> stuff['city'] = u"Москва"
>>> print stuff['city']
Москва
>>>
```

Как вы видите, вместо чисел мы используем строки, чтобы сообщить, что следует извлечь из словаря `stuff`. Мы также можем добавлять новые записи в словарь с помощью строк (и не только строк). Например, так.

```
>>> stuff[1] = u"Ура!"
>>> stuff[2] = u"Ау!"
>>> print stuff[1]
Ура!
>>> print stuff[2]
Ау!
>>> print stuff
   {'city': u'Москва', 2: u'Ау!', 'name': u'Михаил', 1: u'Ура!',
   'age': 36, 'height': 74}
>>>
```

В этом примере кода я сначала использовал числа, а затем числа и строки в качестве ключей в словаре и команду `print`. И я мог бы использовать любой объект, ну, почти любой, но сейчас главное — понять общий принцип.

Разумеется, если в словарь можно только добавить записи, это было бы довольно глупо, поэтому вы можете удалять записи, используя ключевое слово `del`.

```
>>> del stuff['city']
>>> del stuff[1]
>>> del stuff[2]
>>> stuff
{'name': u'Михаил', 'age': 36, 'height': 74}
>>>
```

Теперь приступим к упражнению, которое вы должны изучить очень тщательно. Я хочу, чтобы вы напечатали код этого примера и попытались разобраться,

что происходит. Обратите внимание на то, как я добавляю записи в словарь, извлекаю их, и на все операции, примененные в данном примере.

ex39.py

```
# схема связей аббревиатур с названиями стран
countries = {
    u'Россия': 'RU',
    u'Германия': 'DE',
    u'Узбекистан': 'UZ',
    u'Зимбабве': 'ZW',
    u'Турция': 'TR'
}

# создание базового набора стран и некоторых городов в них
cities = {
    'UZ': u'Газли',
    'TR': u'Сарыгерме',
    'DE': u'Мюнхен'
}

# добавление некоторых городов
cities['ZW'] = u'Гверу'
cities['RU'] = u'Москва'

# вывод некоторых городов
print '- ' * 10
print u"В стране ZW есть город: ", cities['ZW']
print u"В стране RU есть город: ", cities['RU']

# вывод некоторых стран
print '- ' * 10
print u"Аббревиатура Турции: ", countries[u'Турция']
print u"Аббревиатура Германии: ", countries[u'Германия']

# выполняется с учетом страны и словаря городов
print '- ' * 10
print u"В Турции есть город: ", cities[countries[u'Турция']]
print u"В Германии есть город: ", cities[countries[u'Германия']]

# вывод аббревиатур всех стран
print '- ' * 10
for country, abbrev in countries.items():
```

```
39    print u" %s имеет аббревиатуру %s" % (country, abbrev)
40
41  # вывод всех городов в странах
42  print '- ' * 10
43  for abbrev, city in cities.items():
44    print u"В стране %s есть город %s" % (abbrev, city)
45
46  # а теперь сразу оба типа данных
47  print '- ' * 10
48  for country, abbrev in countries.items():
49    print u"В стране %s используется аббревиатура %s и есть город
       %s" % (
50    country, abbrev, cities[abbrev])
51
52  print '- ' * 10
53  # безопасное получение аббревиатуры страны, которая не представлена
54  country = countries.get(u'США', None)
55
56  if not country:
57    print u"Прошу прощения, США не обнаружено."
58
59  # получение города со значением по умолчанию
60  city = cities.get('US', u'не существует')
61  print u"В стране 'US' есть город: %s" % city
```

Результат выполнения

Сеанс упражнения 39

```
$ python ex39.py
--------------
В стране ZW есть город: Гверу
В стране RU есть город: Москва
--------------
Аббревиатура Турции: TR
Аббревиатура Германии: DE
--------------
В Турции есть город: Сарыгерме
В Германии есть город: Мюнхен
--------------
Узбекистан имеет аббревиатуру UZ
Россия имеет аббревиатуру RU
Германия имеет аббревиатуру DE
```

```
Турция имеет аббревиатуру TR
Зимбабве имеет аббревиатуру ZW
--------------
В стране RU есть город Москва
В стране DE есть город Мюнхен
В стране TR есть город Сарыгерме
В стране ZW есть город Гверу
В стране UZ есть город Газли
--------------
В стране Узбекистан используется аббревиатура UZ и есть город Газли
В стране Россия используется аббревиатура RU и есть город Москва
В стране Германия используется аббревиатура DE и есть город Мюнхен
В стране Турция используется аббревиатура TR и есть город Сарыгерме
В стране Зимбабве используется аббревиатура ZW и есть город Гверу
--------------
Прошу прощения, США не обнаружено.
В стране 'US' есть город: не существует
```

Практические задания

1. Сделайте аналогичный вариант сопоставления, только областей и городов вашей или любой другой страны.

2. Поищите документацию Python, касающуюся словарей, и узнайте, что еще можно делать с ними.

3. Узнайте, что нельзя делать со словарями. Основное ограничение заключается в том, что записи в них не имеют порядка, поэтому поэкспериментируйте с этим.

Распространенные вопросы

В чем разница между списком и словарем?

Список — это упорядоченный перечень элементов. Словарь строится на связях (ассоциациях) одних элементов (ключей) с другими (значениями).

Для чего используются словари?

Используйте его в любых ситуациях, когда нужно, используя одно значение, «подсмотреть» другое. По сути, словари можно охарактеризовать как «просмотрщики таблиц».

Для чего тогда используются списки?

Списки предназначены для хранения любой последовательности элементов, следующих по порядку, доступ к которым осуществляется через числовой индекс.

А если мне нужен словарь, но с соблюдением порядка записей в нем?

Найдите информацию о модуле `collections`, который предоставляет в Python специализированный тип данных `OrderedDict`. Воспользуйтесь Интернетом, чтобы найти документацию по этому модулю.

Модули, классы и объекты

Python представляет собой объектно-ориентированный язык программирования. Поэтому в Python есть конструкция под названием *класс*, которая позволяет структурировать ваш код определенным образом. Использование классов позволяет сделать код более согласованным и повысить удобство применения элементов в коде, по крайней мере в теории. Теперь вы будете осваивать азы объектно-ориентированного программирования, применять классы и объекты, используя полученные знания о словарях и модулях. Основная трудность заключается в том, что объектно-ориентированное программирование (сокращенно ООП) может показаться довольно странным. Вы должны просто закрыть на это глаза. Попытайтесь понять, о чем я говорю, набирайте код, а в следующем упражнении мы закрепим материал.

Погнали!

Модули в сравнении со словарями

Вы уже знаете, как создается и используется словарь и что это схема связей одних элементов с другими. Это означает, что, если у вас есть словарь с ключом `'apple'` и вы хотите извлечь соответствующее ему значение, выполняется следующее.

```
mystuff = {'apple': u"Я — ЯБЛОКО!"}
print mystuff['apple']
```

Запомните этот способ «получения X из Y», а затем вспомните о модулях. Вы уже работали с ними ранее и использовали следующим способом.

1. Вы знаете, что модуль представляет собой файл на языке Python с некоторыми функциями или переменными в нем.

2. Вы должны импортировать этот файл.

3. После импорта вы получите доступ к функциям или переменным в этом модуле с помощью оператора . (точка).

Представьте, что у меня есть модуль под названием *mystuff.py*, в который я поместил функцию с именем `apple`. Код модуля *mystuff.py* приведен ниже.

```
# это записано в файле mystuff.py
def apple():
  print u"Я - ЯБЛОКО!"
```

После того как модуль создан, его можно импортировать и получить доступ к функции `apple`.

```
import mystuff
mystuff.apple()
```

Также я мог бы поместить в модуль переменную с именем `tangerine`, вот так.

```
# это записано в файле mystuff.py
def apple():
  print u"Я - ЯБЛОКО!"

# это обычная переменная
tangerine = u"Отражение мечты"
```

Доступ к ней осуществляется похожим образом.

```
import mystuff

mystuff.apple()
print mystuff.tangerine
```

Возвращаясь к словарям, вы должны увидеть, насколько это похоже на использование словаря, хотя синтаксис несколько отличается. Давайте сравним.

```
mystuff['apple'] # получаем apple из словаря
mystuff.apple() # получаем apple из модуля
mystuff.tangerine # аналогично, это обычная переменная
```

Так вы можете сформировать *весьма* общий шаблон действий в Python.

1. Берется контейнер в виде ключ=значение.
2. Из него извлекается значение по имени ключа.

В случае словаря ключ — это строка с синтаксисом [*ключ*]. В случае модуля ключ представляет собой идентификатор и используется синтаксис .*ключ*. За исключением этого, словари и модули практически идентичны.

Классы как мини-модули

Модуль можно представить в виде специализированного словаря, хранящего код Python, к которому вы можете получить доступ с помощью оператора . (точка). Python также поддерживает другую конструкцию, называемую *классом*, которая преследует ту же цель, что и модуль. Класс реализует способ группирования функций и данных и помещения их внутрь контейнера, к которому вы можете получить доступ с помощью оператора . (точка).

Если бы мне понадобилось создать класс, аналогичный модулю *mystuff.py*, я бы поступил примерно так.

```
class MyStuff(object):

def __init__(self):
    self.tangerine = u"Пусть бегут неуклюже…"

def apple(self):
    print u"Я — САМОЕ СЛАДКОЕ ЯБЛОКО!"
```

Код выглядит более сложным по сравнению с модулями, но вы должны научиться создавать «мини-модули», подобные этому. В данном примере используется класс `MyStuff`, содержащий функцию `apple()`. Возможно, вас смутят функция `__init__()` и код `self.tangerine`, используемый для установки переменной `tangerine`.

На самом деле, вот почему классы используются вместо модулей: вы можете использовать упомянутый выше класс и создать еще множество других классов, и все они не будут мешать друг другу. В случае модулей импортировать

в пределах одной программы можно только один модуль, если только вы не хакер со сверхвозможностями.

Прежде чем вы разберетесь во всем этом, вам нужно познакомиться с «объектами» и научиться работать с классом `MyStuff` так же, как ранее с модулем *mystuff.py*.

Объекты как мини-импорты

Если класс можно охарактеризовать как «мини-модуль», то должна существовать аналогичная концепция импорта, но для классов. Эта концепция называется «созданием экземпляров», что по сути можно сократить просто до понятия «создать». Когда вы создаете экземпляр класса, вы получаете то, что называется объектом.

Это производится путем вызова класса по аналогии с функцией, например так.

```
thing = MyStuff()
thing.apple()
print thing.tangerine
```

Первая строка представляет собой операцию по созданию экземпляра класса, очень похожую на вызов функции. Несмотря на схожесть кода, во время выполнения этой операции происходит цепочка событий, выполняемых интерпретатором Python. Ниже я разложу все по полочкам на примере приведенного выше кода для класса `MyStuff`.

Python ищет функцию `MyStuff()` и обнаруживает, что это объявленный вами класс.

Python создает пустой объект со всеми функциями, указанными вами в классе с помощью ключевого слова `def`.

Python затем ищет «волшебную» функцию `__init__` и, если она обнаружена, вызывает ее для инициализации созданного пустого объекта.

В классе `MyStuff` с помощью конструктора `__init__` передается параметр `self`, на место которого будет помещен пустой объект, создаваемый Python, кроме того, я могу установить дополнительные параметры так же, как и при работе с модулем, словарем или другим объектом.

В примере я присвоил переменной `self.tangerine` некоторый текст из песни, а затем инициализировал этот объект.

Теперь Python берет этот созданный объект и присваивает его переменной `thing`, чтобы я смог работать с ней.

Это самые основы того, как Python осуществляет этот «мини-импорт» при вызове класса как функции. Помните, что так вы *не* получаете класс, вместо этого класс используется в качестве основы для создания копий этого типа объектов.

Имейте в виду, что я несколько неточно описываю принцип работы, поэтому вы можете начать разбираться с классами, отталкиваясь от того, что вы знаете о модулях. Правда в том, что классы и объекты затем внезапно перестают быть похожи на модули. Если быть полностью точным, я бы сказал следующее.

- Классы — это как бы чертежи или определения для создания новых мини-модулей.
- Создание экземпляра — это создание одного из этих мини-модулей и одновременное его импортирование.
- В результате создается мини-модуль, называемый объектом, который затем присваивается переменной, чтобы вы могли работать с ним.

После этого классы и объекты существенно отличаются от модулей, и озвученная выше информация должна стать для вас лишь первым шагом к пониманию классов.

Три способа

Теперь я могу использовать любой из трех способов получения элементов из элементов.

```
# способ со словарем
mystuff['apples']

# способ с модулем
mystuff.apples()
print mystuff.tangerine
```

```
# способ с классом
thing = MyStuff()
thing.apples()
print thing.tangerine
```

Первоклассный пример

Теперь вы должны видеть общие черты в этих трех типах контейнеров ключ=значение и, вероятно, иметь кучу вопросов. Пока отложите вопросы, так как следующее упражнение пополнит ваш «объектно-ориентированный словарный запас». В этом упражнении от вас требуется, чтобы вы просто набрали этот код и выполнили, чтобы приобрести некоторый опыт работы с классами, прежде чем двигаться дальше.

ex40.py

```
1  class Song(object):
2
3  def __init__(self, lyrics):
4     self.lyrics = lyrics
5
6  def sing_me_a_song(self):
7     for line in self.lyrics:
8          print line
9
10 happy_bday = Song([u"Не могу я тебе в день рождения",
11                    u" Дорогие подарки дарить,",
12                    u" Но зато в эти ночи весенние"])
13
14 bulls_on_parade = Song([u"Далеко-далеко на лугу пасется кто?",
15                    u"Пейте, дети, молоко, будете здоровы!"])
16
17 happy_bday.sing_me_a_song()
18
19 bulls_on_parade.sing_me_a_song()
```

Внимание! Если вам все же лень вручную набирать код примеров из этой книги, все файлы с кодом вы можете скачать по адресу **https://eksmo.ru/files/Shaw_Python.zip**.

Результат выполнения

Сеанс упражнения 40

```
$ python ex40.py
Не могу я тебе в день рождения
Дорогие подарки дарить,
Но зато в эти ночи весенние
Далеко-далеко на лугу пасется кто?
Пейте, дети, молоко, будете здоровы!
```

Практические задания

1. По подобию напишите еще несколько песен и убедитесь, что вы понимаете, что передаете список строк в качестве лирики.

2. Поместите текст песни в отдельную переменную, а затем передайте эту переменную в класс.

3. Попробуйте свои силы и совершите дополнительные действия с тремя контейнерами. Не волнуйтесь, если вы не имеете представления о том, что происходит: просто поэкспериментируйте и понаблюдайте за результатами.

4. Выполните поиск в Интернете по запросу «объектно-ориентированное программирование» и попытайтесь разобраться в том, что вы читаете. Не волнуйтесь, если прочитанное не имеет абсолютно никакого смысла для вас. Половина этого материала не имеет никакого смысла и для меня.

Распространенные вопросы

Почему так необходим параметр `self`, когда я выполняю `__init__` и другие функции для классов?

Если параметр `self` не указан, то код наподобие `cheese = 'Frank'` неоднозначен. Исходя из этого кода неясно, что вы имеете в виду — атрибут `cheese` *экземпляра* или локальную переменную с именем `cheese`. С кодом `self.cheese = 'Frank'` такой проблемы нет: вы ясно указали, что имеете в виду экземпляр атрибута `self.cheese`.

Поговорим об ООП

В этом упражнении мы поговорим об объектно-ориентированном программировании. Для начала я приведу небольшой список терминов с определениями, которые необходимо знать по этой теме. Следом вы увидите несколько предложений с пропусками, которые вы должны будете заполнить. И, наконец, вам потребуется выполнить большую группу упражнений, чтобы закрепить полученные знания.

Терминология

- **Класс** — используется в Python для создания объектов.
- **Объект** — два значения: базовый тип объекта и любой экземпляр какого-либо объекта.
- **Экземпляр** — вы получаете, когда инструктируете Python создать класс.
- **`def`** — инструкция для определения функции внутри класса.
- **`self`** — внутри функции в классе `self` представляет собой переменную для экземпляра/объекта, к которому осуществляется доступ.
- **Наследование** — концепция, согласно которой один класс может наследовать характеристики другого класса, так же как вы от ваших родителей.
- **Композиция** — концепция, согласно которой класс может состоять из других классов как компонентов. Аналогично, автомобиль имеет колеса или двигатель.
- **Атрибут** — свойство класса в композициях. Как правило, атрибуты являются переменными.
- **`is-a`** — тип взаимосвязи в ООП, когда что-либо *наследует* некие характеристики от других объектов. Далее по тексту данный тип

взаимосвязи упоминается под терминами *наследовать* и *наследование*. Пример: «семга — это разновидность рыбы».

- **has-a** — тип взаимосвязи в ООП, когда что-либо *скомбинировано* из других объектов или имеет признак. Далее по тексту данный тип взаимосвязи подразумевается под терминами *скомбинирован* и *композиция*. Пример: «у семги есть рот».

Не пожалейте время, чтобы сделать карточки для заучивания этих терминов. Как обычно, заучивание покажется бессмысленным, пока вы не закончите это упражнение, но базовые термины нужно знать в первую очередь.

Чтение кода

Ниже я привел несколько фрагментов кода на языке Python слева и описание кода справа.

```
class X(Y)
```
"создается класс с именем X, наследующим Y."

```
class X(object): def __init__(self, J)
```
"класс X комбинирует `__init__` с параметрами `self`, J."

```
class X(object): def M(self, J)
```
"класс X комбинирует функцию с именем M с параметрами `self`, J."

```
foo = X()
```
"создается `foo` как экземпляр класса X."

```
foo.M(J)
```
"из `foo` получается функция M, а затем вызывается с параметрами `self`, J."

```
foo.K = Q
```
"из `foo` получается атрибут K, а затем устанавливается равным Q."

В этих строках кода вы видите X, Y, M, J, K, Q и `foo`, имена которых можно заменить пробелами. Предложения выше я могу также написать следующим образом.

1. Создается класс с именем ???, наследующим Y.
2. Класс ??? комбинирует `__init__` с параметрами `self` и ???.

3. Класс ??? комбинирует функцию с именем ??? с параметрами `self` и ???.

4. Создается `foo` как экземпляр класса ???.

5. Из `foo` получается функция ???, а затем вызывается с параметрами ??? и ???.

6. Из `foo` получается атрибут ???, а затем устанавливается равным ???.

И вновь запишите их на карточки и заучите. Напишите фрагмент кода Python на одной стороне и описание на другой. Вы должны научиться произносить описание в точности, каждый раз, когда видите сторону с кодом. Не примерно, а в точности!

Смешанное упражнение

Далее я объединил задачи по проверке терминологии и выражений. Выполнение этого упражнения заключается в следующем.

1. Возьмите карточку с выражением и прочитайте код. Произнесите описание, которое вы выучили.

2. Переверните карточку и прочитайте описание. Встречая каждый термин в предложении, берите соответствующую карточку с термином.

3. Повторите все встреченные в предложении термины.

4. Продолжайте тренироваться, пока не надоест. Сделайте перерыв и повторите упражнение.

Перевод с кода на русский язык

Теперь я подготовил небольшое программное упражнение с помощью кода Python, который будет без перерыва запрашивать у вас чтение кода. Это простой сценарий, в котором используется библиотека под названием `urllib`, позволяющая загрузить составленный мной список слов с сервера. Его код приведен ниже. Этот же код содержится в файле *oop_test.py*, с которым вы будете работать позднее.

ex41.py

```
import random
from urllib import urlopen
import sys

WORD_URL = "http://learncodethehardway.org/words.txt"
WORDS = []

PHRASES = {
    "class %%%(%%%):":
    u"Создается класс с именем %%%, наследующим %%%.",
    "class %%%(object):\n\tdef __init__ (self, ***) ":
    u"Класс %%% комбинирует __init__ с параметрами self, ***.",
    "class %%%(object):\n\tdef ***(self, @@@) ":
    u"Класс %%% комбинирует функцию с именем *** с параметрами
    self, @@@.",
    "*** = %%%() ":
    u"Создается *** как экземпляр класса %%%.",
    "***.***(@@@) ":
    u"Из *** получается функция ***, а затем вызывается
    с параметрами self, @@@.",
    "***.*** = '***'":
    u"Из *** получается атрибут ***, а затем устанавливается равным
     '***'."
}

# тренировка запоминания фраз
PHRASE_FIRST = False
if len(sys.argv) == 2 and sys.argv[1] == "russian":
    PHRASE_FIRST = True

# подгружаем слова с сервера
for word in urlopen(WORD_URL).readlines():
    WORDS.append(word.strip())


def convert(snippet, phrase):
    class_names = [w.capitalize() for w in
    random.sample(WORDS, snippet.count("%%%"))]
    other_names = random.sample(WORDS, snippet.count("***"))
    results = []
    param_names = []

```

```
40      for i in range(0, snippet.count("@@@")):
41          param_count = random.randint(1,3)
42          param_names.append(', '.join(random.sample(WORDS,
            param_count)))
43
44      for sentence in snippet, phrase:
45          result = sentence[:]
46
47          # вымышленные имена классов
48          for word in class_names:
49              result = result.replace("%%%", word, 1)
50
51          # вымышленные прочие имена
52          for word in other_names:
53              result = result.replace("***", word, 1)
54
55          # список вымышленных параметров
56          for word in param_names:
57              result = result.replace("@@@", word, 1)
58
59          results.append(result)
60
61      return results
62
63
64  # выполнение, пока не нажато сочетание клавиш CTRL+Z
65  try:
66  while True:
67      snippets = PHRASES.keys()
68      random.shuffle(snippets)
69
70      for snippet in snippets:
71          phrase = PHRASES[snippet]
72          question, answer = convert(snippet, phrase)
73          if PHRASE_FIRST:
74              question, answer = answer, question
75
76          print question
77
78          raw_input("> ")
79          print u"ОТВЕТ: %s\n\n" % answer
80  except EOFError:
81  print u"\nПока"
```

Запустите этот сценарий и попытайтесь перевести «объектно-ориентированные фразы» на русский язык. Как вы видите, словарь `PHRASES` содержит обе формы, и вам просто нужно ввести правильный ответ.

Перевод с русского языка в код

Затем вы должны запустить этот же сценарий в «русском» варианте, чтобы выполнить упражнение в обратном порядке.

```
$ python oop_test.py russian
```

Помните, что в генерируемых выражениях используются случайные слова, не несущие смысловой нагрузки. В каком-то смысле это благотворно скажется на развитии навыка чтения кода в сравнении с тем случаем, когда специально подобранные имена переменных и классов «что-то подсказывают». Не обращайте внимания на генерируемые слова, это просто шаблонные выражения.

Дополнительное упражнение по чтению кода

Теперь вы готовы выполнить новое упражнение, позволяющее развить навыки чтения кода, используя выражения, с которыми познакомились ранее. Ваша задача — взять несколько *ру*-файлов с классами, а затем выполнить следующие действия.

1. Каждому классу присвойте подходящее имя и убедитесь, что другие классы успешно наследуют его.

2. Затем перечислите все функции в классах и поддерживаемые параметры.

3. Составьте список всех используемых атрибутов.

4. Для каждого атрибута используйте класс.

Цель упражнения состоит в том, чтобы научиться читать реальный код и находить в нем шаблонные выражения, которые вы изучили в этом упражнении.

Если вы будете тренироваться достаточно долго, эти шаблонные выражения начнут бросаться вам в глаза при чтении кода. А ведь раньше они просто казались вам бессмысленным набором букв!

Распространенные вопросы

Что за код `result = sentence[:]`?

Это способ копировать списки в языке Python. Синтаксис `[:]` используется, чтобы эффективно разделить список на фрагменты, с самого первого элемента до самого последнего.

Этот сценарий слишком сложный!

К этому моменту вы должны быть в состоянии напечатать этот код и успешно выполнить. В коде есть несколько маленьких хитростей, но совершенно никаких сложностей. В случае возникновения проблем вспомните все, что вы узнали к этому моменту об отладке сценариев, и попробуйте справиться со сложностями.

УПРАЖНЕНИЕ 42

Композиция, наследование, объекты и классы

Очень важно понимать разницу между классом и объектом. Проблема заключается в том, что нет никакой реальной «разницы» между классом и объектом. На самом деле, это то же самое, только в разные моменты времени. Я продемонстрирую пример с помощью дзен-коана:

```
В чем разница между рыбой и семгой?
```

Возможно, этот вопрос собьет вас с толку. Присядьте и задумайтесь на минутку. Я имею в виду, что рыба и семга различны, но, с другой стороны, это одно и то же, верно? Семга — это разновидность рыбы, потому это одно и то же. Но так как семга — конкретный вид рыбы, он на самом деле отличается от всех других видов рыб. Это то, что делает ее семгой, а не палтусом. Так что семга и рыба одинаковы, но различны. Фантастика.

Этот вопрос сбивает с толку, потому что большинство людей так не задумываются о реальных вещах, но при этом интуитивно понимают их. Вам не нужно думать о разнице между рыбой и семгой, потому что вы знаете, как они связаны между собой. Вы знаете, что семга является рыбой и что существуют и другие виды рыбы, не задумываясь об этом.

Давайте рассмотрим еще один пример: вы поймали три семги и, поскольку вам так хочется, вы решили дать им имена — Федя, Петя и Кирилл. Теперь задумайтесь над этим вопросом:

```
В чем разница между Кириллом и семгой?
```

Опять же, это очень странный вопрос, хотя и проще, чем вопрос про разницу между рыбой и семгой. Вы знаете, что Кирилл — это имя семги, и поэтому никакой разницы нет. Кирилл — это просто специфический «экземпляр» семги. Федя и Петя — также экземпляры семги. Что я имею в виду, когда говорю

«экземпляры»? Я имею в виду, что они были «созданы» из некоторых представителей семги и теперь представляют реальные объекты, которым присущи атрибуты семги.

Теперь головосносящая идея: рыба — это класс и семга — это класс, а Кирилл — это объект. Задумайтесь на секунду. Теперь давайте разберем все вышесказанное очень медленно и посмотрим, что получается.

Рыба — это класс, а это означает, что это не реальная вещь, а слово, которым мы обозначаем экземпляры вещей с похожими атрибутами. Есть плавник? Жабры? Живет в воде? Прекрасно! Вероятно, это рыба.

И тут приходит преподаватель философии и говорит: «Нет, мой юный друг, эта рыба на самом деле Salmo salar, также известная как семга». Этот преподаватель только уточнил информацию по поводу рыбы и создал новый класс под названием «Семга», который имеет более специфичные атрибуты. Заостренная форма головы, нежно-розовый цвет мяса, крупный размер, более крупная чешуя, обитает в океане или пресной воде, вкусная? Замечательно! Вероятно, это семга.

И, наконец, приходит повар и говорит преподавателю философии: «О-о-о, я вижу тут прекрасную семгу. Я назову эту рыбину Кириллом и приготовлю из нее шикарное блюдо под соусом». Теперь у вас есть экземпляр семги (который также является экземпляром рыбы) по имени Кирилл, которая превратилась в нечто реальное, что насытило вас. Так рыба стала объектом.

Итак: Кирилл — это представитель семги, которая является рыбой. Объект — это класс класса.

Пример кода

Это довольно странная концепция, но, если быть честным, вам стоит беспокоиться об этом, только когда вы создаете новые классы и когда используете их. Я покажу вам две хитрости, при помощи которых можно понять, класс перед вами или объект.

Во-первых, вам нужно выучить два понятия: «наследование» и «композиция». Наследование — это когда речь идет об объектах и классах, связанных друг с другом внутриклассовыми отношениями. При композиции речь идет об объектах и классах, которые связаны только потому, что ссылаются друг на друга.

Теперь изучите приведенный ниже фрагмент кода и замените каждый комментарий ## ?? собственным, указывая, какой тип взаимосвязей используется в следующей строке — композиция или наследование, и что это за взаимосвязи. Я привел несколько примеров, поэтому вы просто должны написать пропущенные комментарии.

Помните, что наследование — это связь между рыбой и семгой, в то время как композиция — связь между семгой и жабрами.

ex42.py

```
## Animal наследует object
class Animal(object):
    pass

##??
class Dog(Animal):

  def __init__(self, name):
      ##??
      self.name = name

##??
class Cat(Animal):

  def __init__(self, name):
      ##??
      self.name = name

##??
class Person(object):

  def __init__(self, name):
      ##??
      self.name = name

  ## Person - композиция животного некоторого вида
  self.pet = None

##??
class Employee(Person):

  def __init__(self, name, salary):
```

```
33          ##?? хмм, что за странная магия?
34          super(Employee, self).__init__(name)
35          ##??
36          self.salary = salary
37
38  ##??
39  class Fish(object):
40      pass
41
42  ##??
43  class Salmon(Fish):
44      pass
45
46  ##??
47  class Halibut(Fish):
48      pass
49
50
51  ## rover наследует Dog
52  rover = Dog("Rover")
53
54  ##??
55  satan = Cat("Satan")
56
57  ##??
58  mary = Person("Mary")
59
60  ##??
61  mary.pet = satan
62
63  ##??
64  frank = Employee("Frank", 120000)
65
66  ##??
67  frank.pet = rover
68
69  ##??
70  flipper = Fish()
71
72  ##??
73  crouse = Salmon()
74
75  ##??
76  harry = Halibut()
```

О синтаксисе `class имя(object)`

Помните, как я настаивал, чтобы вы всегда использовали код `class имя(object)`, и не рассказывал почему? Теперь пришло время рассказать, потому что вы только что узнали о разнице между классом и объектом. Я не рассказывал об этом до сих пор, поскольку вы бы только запутались и не смогли научиться использовать эти концепции.

Так случилось, что исходный метод использования классов в Python был существенно изменен из-за того, что был заметно ограничен в плане объектно-ориентированных возможностей. Для совместимости со старыми версиями Python было решено использовать две объектные модели: «старые» («классические») классы и «новые». Начиная с версии Python 3 «старые» классы больше не поддерживаются, поэтому вам следует использовать новую, правильную версию.

В новой версии реализуется метод «класс наследует объект». При этом решено, что слово `object` должно указываться в нижнем регистре, чтобы стать «классом», унаследованным при создании класса. Непонятно, да? Класс наследуется от класса с именем `object`, чтобы создать класс, который не является объектом.

Вот поэтому я не мог рассказать вам об этом ранее. Теперь вы можете попытаться понять концепцию класса `object`, если вам хочется.

Тем не менее я предлагаю вам этого не делать. Просто полностью забудьте про «старые» классы в пользу новых и запомните, что Python всегда требует `object`, когда вам нужно создать класс.

Практические задания

1. Найдите ответ на вопрос, почему разработчики Python добавили этот странный класс `object` и что это значит.

2. Можно ли использовать `class` как объект?

3. Измените код с животными, людьми и рыбами в этом упражнении, добавив функции с действиями. Что происходит, если функции помещаются в «базовый класс», например как животное `Animal` вместо `Dog`.

4. Найдите код других программистов и проработайте все наследования и композиции.

5. Создайте новые взаимосвязи множественного наследования, используя списки и словари.

6. Существует ли «множественная композиция»?

Распространенные вопросы

Что делать с комментариями `##???`

Вам нужно «заполнить пропуски» собственными комментариями, указывая, какая концепция используется — наследование или композиция. Перечитайте текст этого упражнения и изучите другие комментарии, чтобы понять, что я имею в виду.

Для чего нужен код `self.pet = None`?

Он присваивает значение по умолчанию `None` атрибуту `self.pet` этого класса.

Для чего нужен код `super(Employee, self).__init__(name)`?

Это способ выполнить метод `__init__` родительского класса. Выполните в Интернете поиск по запросу «python метод super» и прочитайте про этот метод.

УПРАЖНЕНИЕ 43

Основы объектно-ориентированного анализа и дизайна

В этом упражнении я опишу процесс проектирования и разработки программ с помощью Python, в частности с применением объектно-ориентированного программирования (ООП). Под словом «процесс» я подразумеваю набор шагов, который вам нужно выполнить. Но обратите внимание, что вы не должны слепо копировать их в своих проектах, и не думайте, что они будут полностью работоспособны в вашем случае. Это просто хорошая отправная точка для реализации многих задач программирования, которую не следует рассматривать как единственный способ решения такого рода задач. Этот процесс — только один из способов, которому вы можете следовать. Общий план процесса выглядит следующим образом.

1. Напишите или зарисуйте свою задачу.
2. Извлеките ключевые концепции из шага 1 и исследуйте их.
3. Сформируйте иерархию классов и связи объектов для данных концепций.
4. Напишите код классов и протестируйте их.
5. Вернитесь к коду, исправьте ошибки и доработайте его.

Этот процесс представляет собой «нисходящий» (сверху вниз) подход. То есть вы начинаете с самого абстрактного уровня, поверхностной идеи, а затем медленно прорабатываете ее, пока идея не окрепнет и вы не сможете приступить к набору кода.

Лично я сначала пытаюсь просто описать эту задачу и обдумываю ее. Возможно, я даже составляю одну-две диаграммы или что-то вроде схемы или даже отправляю себе несколько электронных писем, описывающих задачу. Так я могу извлечь ключевые понятия из задачи, а также исследовать этот вопрос.

Потом я просматриваю все эти заметки, схемы, зарисовки и описания и извлекаю ключевые понятия. Существует очень простой способ сделать это: составьте список всех существительных и глаголов, использованных в вашем текстовом и графическом материале, а затем нарисуйте (запишите) связи между ними. Так я получаю подходящий список имен для классов, объектов и функций, требуемый на следующем шаге. Я беру полученный список концепций, а затем анализирую те, которые не понимаю, чтобы уточнить их.

После того как список концепций готов, я создаю простую иерархическую схему (дерево) концепций и прослеживаю, как они соотносятся в виде классов. К примеру, я беру список существительных и задаю себе вопрос: «Похожа ли эта концепция на другие существительные? Если да, значит, у них есть общий родительский класс. Как его назвать?» Продолжайте делать так до тех пор, пока не проработаете всю иерархию классов, весь список (или диаграмму) существительных. Затем переходите к глаголам, прорабатывайте имена функций для каждого класса и размещайте их в дереве.

Когда с иерархией классов покончено, я сажусь составлять некоторый базовый каркас кода, который содержит только классы и их функции. Затем я пишу тестовый сценарий, который выполняет этот код, и проверяю, чтобы классы, созданные мной, работали правильно. Иногда я сразу пишу тестовый сценарий, а иногда работаю поэтапно — пишу небольшой тест, затем немного кода, вновь небольшой тест, а затем код и так далее, пока все кирпичики не будут проверены.

И, наконец, я циклично прорабатываю написанный код, исправляя ошибки и уточняя детали, прежде чем выполнять следующие реализации. Если я сталкиваюсь с трудностями в какой-либо конкретной позиции кода из-за сложности реализации концепции или неожиданной проблемы, то работаю только с этим фрагментом кода, чтобы исправить его, прежде чем продолжить.

Теперь и вы пройдете через процесс создания игрового движка и самой игры для данного упражнения.

Анализ простого игрового движка

Игра, которую я хочу создать, называется «Готоны с планеты Перкаль 25» и представляет собой небольшой квест. Начав с одной лишь идеи, пришедшей мне в голову, я могу развить ее и воплотить игру в жизнь.

Запись или зарисовка задачи

Я написал небольшой текст про игру:

«Инопланетяне (Готоны) захватили космический корабль. На этом корабле игровой персонаж должен пробраться через лабиринт отсеков и сбежать в спасательной капсуле на планету. Игровой процесс будет похож на приключенческие игры, наподобие Zork или Adventure, с текстовым интерфейсом и смешными выражениями. Игра будет включать в себя движок, запускающий карту сцен. В каждой сцене будет выводиться собственное описание, когда игрок попадет туда, а затем будет осуществляться переход в другую сцену».

Я отлично понимаю, как будет работать игра, поэтому опишу каждую сцену.

- **Смерть.** Когда игрок умирает, должна выводиться какая-нибудь забавная фраза.

- **Центральный коридор.** Здесь будет начинаться игра. Тут будут находиться один из Готонов и наш герой. Игрок должен победить Готона с помощью шутки, прежде чем продолжить играть.

- **Оружейная лаборатория.** Здесь герой получает нейтронную бомбу, чтобы заминировать корабль, прежде чем попасть в спасательную капсулу. Потребуется угадать числовой код доступа к бомбе.

- **Топливный отсек.** Другая сцена битвы с Готонами, где герой устанавливает бомбу.

- **Отсек спасательных капсул.** Здесь герой убегает с корабля, но только если выбрал капсулу с правильным номером.

Теперь я могу зарисовать схему сцен и, может быть, добавить больше текста к каждой сцене — все, что приходит на ум во время анализа задачи.

Извлечение ключевых концепций и их анализ

Теперь у меня есть достаточно информации, чтобы извлечь некоторые из существительных и проанализировать соответствующую иерархию классов. Сначала я составляю список всех существительных.

- Инопланетянин
- Игрок
- Корабль
- Лабиринт
- Комната
- Сцена
- Готон
- Спасательная капсула
- Планета
- Карта
- Движок
- Смерть
- Центральный коридор
- Оружейная лаборатория
- Топливный отсек

Также можно выписать все глаголы и проанализировать, годятся ли они в качестве имен функций, но пока я пропущу этот момент.

Теперь вы можете также исследовать каждую из этих концепций и все, что вы не учли. Я работал над играми подобного типа, поэтому хорошо знаю, как они устроены. В своем примере я мог бы исследовать, как корабли спроектированы, или как работают бомбы. Возможно, я изучил бы некоторые технические задачи, такие как сохранение состояния игры в базе данных. После того как провел исследование, я начал бы вновь с шага 1 и на основе полученной дополнительной информации переписал описание и извлек новые концепции.

Формирование иерархии классов и схемы объектов на основе концепций

После того как все данные для формирования иерархии классов получены, задайте вопрос: «Что на что похоже?» Я также спросил бы: «Каким основным словом можно обозначить концепции?»

Сразу же я вижу, что «Комната» и «Сцена» по сути одно и то же, в зависимости от того, что должно происходить. Для этой игры я выбираю слово «Сцена». Потом вижу, что все специфические комнаты типа «Центрального коридора» по сути лишь сцены. Также мне ясно, что «Смерть» — это тоже «Сцена»: это подтверждает мой выбор «Сцены» вместо «Комнаты», так как в игре присутствует сцена смерти, но никак не комната. «Лабиринт» и «Карта» в целом тоже похожи. Я возьму слово «Карта», поскольку использовал его чаще. Я не буду разрабатывать систему боя, поэтому опущу слова «Инопланетянин» и «Игрок», сохранив их на будущее. Также и «Планета» также может быть просто другой сценой, а не чем-то специфическим.

После этого мыслительного процесса я формирую иерархию классов, которая в моем текстовом редакторе выглядит примерно так.

- Карта
- Движок
- Сцена
    - Смерть
    - Центральный коридор
    - Оружейная лаборатория
    - Топливный отсек
    - Спасательная капсула

Также следует продумать, какие действия необходимы в каждом классе, основываясь на глаголах из описания. Например, я знаю, что из приведенного выше описания мне нужно «запустить» (`play`) движок, «перейти к следующей сцене» (`next_scene`) из карты, «открыть сцену» (`opening_scene`)

и «войти» (`enter`) на сцену. Я добавлю их следующим образом (в скобках использую аналоги на латинице так, как они применяются в коде):

- Карта (`Map`)
    - `next_scene`
    - `opening_scene`
- Движок (`Engine`)
    - `play`
- Сцена (`Scene`)
    - `enter`
- Смерть (`Death`)
- Центральный коридор (`CentralCorridor`)
- Оружейная лаборатория (`LaserWeaponArmory`)
- Топливный отсек (`TheFuelcell`)
- Спасательная капсула (`EscapePod`)

Обратите внимание на то, что я вложил пункт `enter` в «Сцену», так как знаю, что все сцены ниже будут наследовать его, а затем замещать.

Кодинг классов и тестовый запуск

После того как дерево классов и некоторые из функций описаны, я создаю исходный файл сценария в своем редакторе и пробую написать код игры. Обычно я просто копирую и вставляю дерево, показанное выше, в исходный файл, а затем редактирую его, создавая классы. Ниже представлен небольшой пример, как это может выглядеть первоначально, включая простой проверочный код в конце файла.

ex43_classes.py

```
class Scene(object):

    def enter(self):
        pass


class Engine(object):

    def __init__(self, scene_map):
        pass

    def play(self):
        pass

class Death(Scene):

    def enter(self):
        pass

class CentralCorridor(Scene):

    def enter(self):
        pass

class LaserWeaponArmory(Scene):

    def enter(self):
        pass

class TheFuelcell(Scene):

    def enter(self):
        pass

class EscapePod(Scene):

    def enter(self):
        pass


class Map(object):

```

```
43      def __init__(self, start_scene):
44          pass
45
46      def next_scene(self, scene_name):
47          pass
48
49      def opening_scene(self):
50          pass
51
52
53  a_map = Map('central_corridor')
54  a_game = Engine(a_map)
55  a_game.play()
```

В этом файле можно заметить, что я просто скопировал иерархию классов, а затем добавил некоторый код в конце, чтобы запустить сценарий и просмотреть, работает ли эта базовая структура. В следующих разделах этого упражнения вы наберете оставшуюся часть кода игры и заставите работать сценарий в соответствии с описанием игры.

Исправление ошибок и доработка кода

Заключительный этап в процессе — не столько шаг, сколько цикл. Вы никогда не выполните эту задачу за один проход. На протяжении всего процесса работы вы вновь и вновь будете возвращаться к коду и дорабатывать его на основе информации, полученной на последующих шагах. Бывает так, что, перейдя к шагу 3, я понимаю, что мне нужно вернуться к шагам 1 и 2, поработать на них. А иногда, получив вдохновение и добравшись почти до конца, я пишу финальную версию кода, но потом возвращаюсь и прорабатываю предыдущие шаги, чтобы убедиться, что учел все возможности.

Другая идея в этом процессе заключается в том, что вы не просто прорабатываете каждый уровень от начала и до конца, а потом переходите к следующему. Вы можете прервать процесс, выделить его фрагмент и выполнить все шаги исключительно для него, если столкнетесь с той или иной проблемой. Скажем, если я не знаю, как реализовать метод `Engine.play`, я могу прерваться и выполнить весь описанный процесс только для этой одной функции, чтобы выяснить, как ее реализовать.

Нисходящий подход против восходящего

Процесс, который я только что описал, как правило, называется «нисходящим» подходом, так как начинается с самых абстрактных концепций (верхний уровень) и прорабатывается вплоть до фактической реализации (нижний уровень). Я предлагаю использовать именно этот подход, который я описал выше. Однако вы должны знать, что существует еще один способ решения задач в программировании — когда начало идет от кода вверх, до абстрактных понятий. Этот альтернативный вариант называется «восходящим» подходом. Ниже представлены общие шаги, которым нужно следовать при восходящем подходе.

1. Возьмите небольшой фрагмент кода проекта и попробуйте его выполнить.

2. Доработайте код, создав классы и автоматизированные тесты.

3. Извлеките ключевые концепции из используемого кода и проанализируйте их.

4. Опишите, для чего предназначен код.

5. Вернитесь к шагам 1–2 и доработайте код. Возможно, понадобится полностью удалить его и начать сначала.

6. Переходите к другому фрагменту кода проекта.

Такой подход, я считаю, больше подойдет опытным программистам, способным решать задачи сразу с помощью кода. Восходящий подход очень удобен, если у вас проработаны небольшие фрагменты общей головоломки, но нет общей картины. Раздробление проекта на небольшие части и анализ их кода помогают медленно, но верно добираться до решения задачи. Тем не менее помните, что путь решения, вероятно, будет запутанным и нестандартным, поэтому моя версия этого подхода предусматривает переход к предыдущим шагам и анализ с внедрением новой полученной информации.

Код игры «Готоны с планеты Перкаль 25»

Стоп! Я собираюсь продемонстрировать вам мое окончательное решение вышеуказанной задачи, но я не хочу, чтобы вы просто вводили код. Возьмите созданный мной каркас кода, который я привел выше, а затем попробуйте

создать его рабочую версию на основе описания. После того как вы решите эту задачу, вернитесь к данному упражнению и посмотрите, как это сделал я.

Я разделю готовый код файла *ex43.py* на фрагменты и прокомментирую каждый из них, вместо того чтобы привести полный листинг.

ex43.py

```
# -*- coding: utf- 8 -*-

import codecs, sys
outf = codecs.getwriter('cp866')(sys.stdout, errors='replace')
sys.stdout = outf
```

Первым делом в начале файла необходимо указать специальный код, чтобы русский текст отображался правильно.

ex43.py

```
from sys import exit
from random import randint
```

Это лишь базовые инструкции импорта для игры, ничего сложного.

ex43.py

```
class Scene(object):

def enter(self):
    print u"Эта сцена еще не настроена. Создайте подкласс
        и реализуйте функцию enter()."
        exit(1)
```

Как вы видели в каркасе кода, там есть базовый класс `Scene`, который будет включать общие элементы для всех сцен в игре. В этой простой программе их задача невелика, это скорее демонстрация процесса создания базового класса.

ex43.py

```
class Engine(object):

def __init__(self, scene_map):
   self.scene_map = scene_map

def play(self):
   current_scene = self.scene_map.opening_scene()

   while current_scene!= last_scene:
        next_scene_name = current_scene.enter()
        current_scene = self.scene_map.next_scene(next_scene_name)

   # не забудьте вывести последнюю сцену
   current_scene.enter()
```

У меня также есть класс `Engine`, и здесь можно увидеть, как я использую методы `map. opening_scene` и `map.next_scene`. Так как я предварительно планировал игру, я мог лишь предполагать, что напишу какие-либо методы, а затем использовать их, прежде чем будет создан класс `Map`.

ex43.py

```
class Death(Scene):

quips = [
   u"Вы погибли. Как это ни печально.",
   u"Ваша мать будет грустить по вам... надо было быть умнее.",
   u"Надо же было быть таким придурком.",
   u"Даже мой маленький щенок соображает лучше."
   ]

def enter(self):
   print Death.quips[randint(0, len(self.quips)- 1)]
    exit(1)
```

Первая сцена в моей игре необычна и носит название `Death` (то есть смерть), код которой наиболее прост из числа того, что вы можете написать.

ex43.py

```
1   class CentralCorridor(Scene):
2
3   def enter(self):
4     print u"Готоны с планеты Перкаль 25 захватили ваш корабль
          и уничтожили "
5     print u"всю команду. Вы — единственный, кто остался в живых. "
6     print u"Вам нужно выкрасть нейтронную бомбу в оружейной
          лаборатории, "
7     print u"заложить ее в топливном отсеке и покинуть
          корабль в спасательной "
8     print u"капсуле прежде, чем он взорвется."
9     print "\n"
10    print u"Вы бежите по центральному коридору в оружейную
          лабораторию, когда перед вами "
11    print u"появляется Готон с красной чешуйчатой кожей,
          гнилыми зубами и в костюме клоуна. "
12    print u"Он с ненавистью смотрит на вас и, преградив
          дорогу в лабораторию, "
13    print u"вытаскивает бластер, чтобы уничтожить вас."
14
15    action = raw_input("> ").decode(sys.stdin.encoding or
      locale.getpreferredencoding(True))
16
17    if action == "стрелять!":
18        print u"Вы быстро выхватываете свой бластер
              и начинаете палить по Готону."
19        print u"Его клоунский наряд крутится и мешает
              лучам попадать в "
20        print u"его тело. Все ваши выстрелы лазером
              потерпели неудачу, и заряд бластера иссяк. "
21        print u"Костюм Готона, который купила его мать,
              безнадежно испорчен, поэтому "
22        print u"он в ярости выхватывает бластер
              и стреляет вам в голову. "
23        print u"Вы убиты."
24        return 'death'
25
26    elif action == u"проскочить!":
27        print u"Словно боксер мирового класса, вы
              уворачиваетесь и проскальзываете справа "
28        print u"от Готона, краем глаза видя, что его
              бластер направлен вам в голову. "
```

```
29          print u"И тут вы подскальзываетесь и врезаетесь
                в металлическую стену. От удара "
30          print u"вы теряете сознание. "
31          print u"Придя в сознание, вы успеваете
                почувствовать, что Готон топчется на вашей "
32          print u"голове и пожирает вас."
33          return 'death'
34
35      elif action == u"пошутить!":
36          print u"К счастью, вы знакомы с культурой Готонов
                и знаете, что их способно рассмешить. "
37          print u"Вы рассказываете бородатый анекдот: "
38          print u"Неоколонии, изоморфно релятивные
                к мультиполосным гиперболическим параболоидам."
39          print u"Готон замирает, старается сдержать смех,
                а затем начинает безудержно хохотать."
40          print u"Пока он смеется, вы достаете бластер
                и стреляете Готону в голову. "
41          print u"Он падает, а вы перепрыгиваете его
                и бежите в оружейную лабораторию."
42          return 'laser_weapon_armory'
43
44        else:
45          print u"ТАК НЕЛЬЗЯ ПОСТУПИТЬ!"
46          return 'central_corridor'
```

После этого я создал класс `CentralCorridor`, который представляет собой начало игры. Я создаю сцены игры прежде, чем будет написан код класса `Map`, поскольку дальше мне нужно будет сослаться на них.

ex43.py

```
1   class LaserWeaponArmory(Scene):
2
3       def enter(self):
4           print u"Вы вбегаете в оружейную лабораторию и начинаете
            обыскивать комнату, "
5           print u"спрятались ли тут другие Готоны. Стоит мертвая тишина. "
6           print u"Вы бежите в дальний угол комнаты и находите
            нейтронную бомбу "
7           print u"в защитном контейнере. На лицевой стороне
            контейнера расположена "
```

```
8           print u"панель с кнопками, и вам надо ввести правильный
            код, чтобы достать бомбу. "
9           print u"Если вы 10 раз введете неправильный код,
            контейнер заблокируется, и вы "
10          print u"не сможете достать бомбу. Код состоит из трех цифр."
11          code = u" %d %d %d" % (randint(1,9), randint(1,9),
                randint(1,9))
12      guess = raw_input("[keypad]> ")
13      guesses = 0
14
15          while guess!= code and guesses < 10:
16              print u"ВЖЖЖИИИК!"
17              guesses += 1
18              guess = raw_input("[keypad]> ")
19
20          if guess == code:
21              print u"Контейнер открывается со щелчком
                и выпускает сизый газ. "
22              print u"Вы вытаскиваете нейтронную бомбу и бежите
                в топливный отсек, "
23              print u"чтобы установить бомбу в нужном месте."
24              return 'the_fuelcell'
25          else:
26              print u"Вы слышите, как замок жужжит последний
                раз, а затем чувствуете "
27              print u"горелый запах — замок расплавился. "
28              print u"Вы остаетесь в оружейной лавке, пока
                наконец Готоны не взорвут "
29              print u"ваш корабль со своего и вы не умрете."
30              return 'death'
31
32
33
34  class TheFuelcell(Scene):
35
36      def enter(self):
37          print u"Вы вбегаете в топливный отсек с нейтронной
            бомбой и видите "
38          print u"пятерых Готонов, безуспешно пытающихся управлять "
39          print u"кораблем. Один уродливее другого, и все в клоунских "
40          print u"костюмах, как и Готон, убитый вами. Они не достают
            оружие, "
41          print u"так как видят бомбу у вас в руках и не хотят, чтобы "
42          print u"вы установили ее."
```

```
43
44          action = raw_input("> ").decode(sys.stdin.encoding or
            locale.getpreferredencoding(True))
45
46          if action == u"бросить бомбу":
47              print u"Вы в панике активируете и бросаете бомбу в толпу
                Готонов, "
48              print u"а затем прыгаете к двери шлюза. Сразу после
                этого "
49              print u"один из Готонов стреляет вам в спину. Умирая, "
50              print u"вы видите, как другие Готоны тщетно пытаются
                деактивировать "
51              print u"бомбу. Умирая, вы осознаете, что Готоны тоже
                погибнут."
52              print u"Ваше сознание угасает."
53              return 'death'
54
55      elif action == u"установить бомбу":
56          print u"Вы указываете бластером на бомбу в ваших руках."
57          print u"Готоны поднимают лапы вверх и в страхе потеют."
58          print u"Вы осторожно, не отворачиваясь, подходите к двери и "
59          print u"аккуратно устанавливаете бомбу, держа Готонов
                под прицелом. "
60          print u"Вы запрыгиваете в шлюз и закрываете его ударом
                по кнопке, "
61          print u"а затем бластером расплавляете замок, чтобы Готоны
                не смогли "
62          print u"открыть дверь. Теперь вам нужно залезть
                в спасательную капсулу "
63          print u"и удрать с корабля к чертям собачьим."
64          return 'escape_pod'
65      else:
66          print u"ТАК НЕЛЬЗЯ ПОСТУПИТЬ!"
67          return u"the_fuelcell"
68
69
70  class EscapePod(Scene):
71
72      def enter(self):
73          print u"Вы мчитесь по отсеку со спасательными капсулами.
            Некоторые из них "
74          print u"могут быть повреждены и взорвутся во время полета.
            Всего капсул "
```

```
75          print u"пять, и у вас нет времени, чтобы осматривать каждую
            из них "
76          print u"на отсутствие повреждений."
77          print u"Задумавшись на секунду, вы решаете сесть в капсулу под "
78          print u"номером… "
79          print u"Какой номер вы выбираете?"
80
81          good_pod = randint(1,5)
82          guess = raw_input("[pod #]> ")
83
84
85          if int(guess)!= good_pod:
86              print u"Вы запрыгиваете в капсулу номер %s и нажимаете
                кнопку отстыковки." % guess
87              print u"Капсула вылетает в космическое пространство,
                а затем "
88              print u"взрывается с яркой вспышкой и разбрасывая
                осколки."
89              print u"Вы умираете."
90              return 'death'
91          else:
92              print u"Вы запрыгиваете в капсулу номер %s и нажимаете
                кнопку отстыковки." % guess
93              print u"Капсула вылетает в космическое пространство,
                а затем"
94              print u"отправляется к планете неподалеку. Вы смотрите
                в иллюминатор и видите, как ваш "
95              print u"корабль взрывается. Его осколки повреждают
                топливный
                отсек корабля "
96              print u"Готонов, и тот тоже разлетается в клочья. "
97              print u"Победа за вами!"
98
99
100             return 'finished'
101
102 class Finished(Scene):
103
104     def enter(self):
105         print u"Вы победили! Отличная работа!"
106         return 'finished'
```

Это код оставшихся сцен игры, и, так как я знал, что они понадобятся, и думал о том, как они будут взаимодействовать, я могу сразу привести весь их код.

Кстати, я не сразу набирал весь этот код. Помните, я упоминал, что следует работать с ним поэтапно, проверять по одному фрагменту. Здесь я сразу привел финальный результат после многих проверок.

ex43.py

```
class Map(object):

scenes = {
   'central_corridor': CentralCorridor(),
   'laser_weapon_armory': LaserWeaponArmory(),
   'the_fuelcell': TheFuelcell(),
   'escape_pod': EscapePod(),
   'death': Death(),
   'finished': Finished(),
   }

def __init__(self, start_scene):
   self.start_scene = start_scene

def next_scene(self, scene_name): val = Map.scenes.get(scene_name)
   return val

def opening_scene(self):
   return self.next_scene(self.start_scene)
```

Теперь перед вами класс `Map`, и, как вы можете видеть, в нем хранятся все сцены — их имена перечислены в словаре, а затем я ссылаюсь на этот словарь с помощью `Map.scenes`. Это также объясняет, почему `Map` расположен после `scenes` — словарь ссылается на сцены, поэтому они должны к этому моменту существовать.

ex43.py

```
a_map = Map('central_corridor')
a_game = Engine(a_map)
a_game.play()
```

И, наконец, я написал код запуска игры, который создает `Map`, затем передавая эту функцию в `Engine` перед вызовом `play`, чтобы игра заработала.

Результат выполнения

Сначала попробуйте пройти эту игру самостоятельно. Если вы зашли в тупик, просмотрите описание игры или немного измените код, чтобы «обмануть» игру. Хотя читерство — это не очень хорошо. Также вы можете подсмотреть мою реализацию кода, а затем вернуться к работе над своим проектом. Но для начала обязательно попытайтесь решить проблему самостоятельно.

Когда я попробовал пройти игру, то получил следующий вывод.

Сеанс упражнения 43

```
$ python ex43.py

--------------
Готоны с планеты Перкаль 25 захватили ваш корабль и уничтожили
всю команду. Вы — единственный, кто остался в живых.
Вам нужно выкрасть нейтронную бомбу в оружейной лаборатории,
заложить ее в топливном отсеке и покинуть корабль в спасательной
капсуле прежде, чем он взорвется.

Вы бежите по центральному коридору в оружейную лабораторию, когда
перед вами появляется Готон с красной чешуйчатой кожей, гнилыми
зубами и в костюме клоуна. Он с ненавистью смотрит на вас
и, преградив дорогу в лабораторию, вытаскивает бластер, чтобы
уничтожить вас.
> стрелять!
Вы быстро выхватываете свой бластер и начинаете палить по Готону.
Его клоунский наряд крутится и мешает лучам попадать в его тело.
Все ваши выстрелы лазером потерпели неудачу, и заряд бластера иссяк.
Костюм Готона, который купила его мать, безнадежно испорчен,
поэтому он в ярости выхватывает бластер и стреляет вам в голову.
Вы убиты.
Вы погибли. Как это ни печально.
```

Практические задания

1. В мой код закралась ошибка. Почему замок допускает 11 попыток?

2. Объясните, как осуществляется переход в следующую комнату (сцену).

3. Добавьте чит-коды в игру, чтобы пройти больше сложных сцен. Я могу сделать это двумя словами в одной строке кода.

4. Вернитесь к моему описанию и анализу проекта, а затем попытайтесь реализовать небольшую боевую систему для героя и Готонов, с которыми он сталкивается.

5. На профессиональном языке существует понятие «конечный автомат». Почитайте про него. Конечный автомат может не понадобиться на данном этапе, но в любом случае изучите тему.

Распространенные вопросы

Где найти истории для моих будущих игр?

Вы можете написать их, как если бы рассказывали другу. Или вы можете использовать простые сцены из книг или фильмов, которые вам нравятся.

Наследование и композиция

В сказаниях, в которых мужественные герои побеждают злодеев, всегда присутствует какое-нибудь таинственное место. Это может быть пещера, темный лес, другая планета, что угодно, куда герою не следует идти. И, разумеется, вскоре после появления злодея герою необходимо пойти в этот дурацкий лес, чтобы добро победило зло. Кажется, что герой специально ищет обстоятельства, вынуждающие его рисковать жизнью в этом зловещем лесу.

Редко встречаются сказки про героев, которые достаточно умны, чтобы избежать всего этого. Вы никогда не услышите, чтобы герой молвил что-то вроде: «Минуточку, если я с риском для жизни выйду в открытое море, покинув мою родину, чтобы победить некоего уродливого принца, который похитил дочь короля, я же могу погибнуть! Думаю, что лучше останусь здесь и займусь сельским хозяйством». Если бы так происходило, не было бы никаких лавовых озер, смертей, воскрешений, боев на мечах, великанов, да и вообще сказочных историй. Из-за этого лес в таких историях похож на черную дыру, которая затягивает героя независимо от его действий.

В объектно-ориентированном программировании тот таинственный лес — это *наследование*. Опытные программисты знают, как избежать этого зла: они знают, что в глуши темного леса скрывается злая королева, *множественное наследование*. Она любит пожирать программное обеспечение и программистов своими массивными зубами сложности, пережевывая павших. Но лес настолько велик и заманчив, что почти каждый программист должен войти в него и попытаться победить злую королеву, прежде чем сможет назвать себя настоящим программистом. Вы просто не можете сопротивляться тяге таинственного леса, поэтому вступаете в него. После череды приключений вы научитесь держаться подальше от этого леса и брать с собой армию, если вдруг понадобится идти туда снова.

Таким веселым образом я пытаюсь сказать, что собираюсь научить вас чему-то, чего следует избегать, — *наследованию*. Программисты, которые сейчас бродят по лесу и сражаются с королевой, вероятно, скажут, что вам нужно пойти с ними. Они так говорят потому, что им нужна ваша помощь, так как созданное ими, вероятно, им не по силам. Поэтому вы всегда должны помнить следующее.

В большинстве случаев код наследования может быть упрощен или заменен композицией, а множественного наследования следует избегать любой ценой.

Что такое «наследование»?

Термин «наследование» используется для обозначения того, что какой-либо класс заимствует большинство функций (или все) от порождающего класса. Это происходит неявным образом всякий раз, когда вы пишете код `class Foo(Bar)`, что означает «Создать класс `Foo`, который является наследником класса `Bar`». В этом случае любые действия, которые вы производите над экземплярами `Foo`, работают так, как если бы они производились над экземпляром `Bar`. Это позволяет вам разместить общую функциональность в классе `Bar`, а затем специализировать ее в классе `Foo` так, как требуется.

При выполнении подобной специализации существуют три варианта взаимодействия между порождающим классом и его наследником.

1. При действиях над потомком подразумевается действие над родителем.
2. Действия над потомком переопределяют действие над родителем.
3. Действия над потомком видоизменяют действие над родителем.

Продемонстрирую все эти варианты по порядку и приведу программный код.

Неявное наследование

Сначала я покажу вам неявные действия, которые встречаются тогда, когда вы определяете функцию в порождающем классе, а *не* в потомке.

ex44a.py

```
1   class Parent(object):
2
3     def implicit(self):
4         print "PARENT implicit() "
5
6   class Child(Parent):
7       pass
8
9   dad = Parent()
10  son = Child()
```

```
11
12  dad.implicit()
13  son.implicit()
```

Инструкция `pass` используется в определении класса `class Child:`, чтобы сообщить языку Python о том, что вам необходим пустой блок. Так создается класс с именем `Child`, но при этом говорится, что к его определению нечего добавить. Вместо этого он унаследует поведение от класса `Parent`. При запуске этого кода вы получите следующее.

Сеанс упражнения 44a

```
$ python ex44a.py
РОДИТЕЛЬ implicit()
РОДИТЕЛЬ implicit()
```

Обратите внимание на то, что, хотя я вызываю метод `son.implicit()` в строке 13 и при этом в классе `Child` нет определения функции `implicit`, код работает и происходит вызов функции, определенной в классе `Parent`. Этим демонстрируется то, что при размещении функций в базовом классе (то есть, `Parent`) все подклассы (то есть, `Child`) автоматически получат эти функции. Это очень удобно при повторяющемся коде, который необходим вам в нескольких классах.

Явное переопределение

Проблема с неявным вызовом функций состоит в том, что иногда вам может понадобиться иное поведение потомка. В таком случае следует переопределить функцию внутри потомка, фактически заменив функциональность. Чтобы выполнить это, просто определите функцию с таким же именем в классе `Child`. Например, так.

ex44b.py

```
1   class Parent(object):
2
3       def override(self):
```

```
4           print "PARENT override() "
5
6   class Child(Parent):
7
8     def override(self):
9           print "CHILD override() "
10
11  dad = Parent()
12  son = Child()
13
14  dad.override()
15  son.override()
```

В этом примере в обоих классах есть функция `override`. Давайте посмотрим, что произойдет после запуска.

Сеанс упражнения 44b

```
$ python ex44b.py
РОДИТЕЛЬ override()
ПОТОМОК override()
```

Как видите, при выполнении строки 14 используется функция `Parent.override`, поскольку здесь переменная `(dad)` имеет значение `Parent`. Однако при выполнении строки 15 выводятся сообщения функции `Child.override`, так как `son` — это экземпляр класса `Child`, а в классе `Child` данная функция переопределена.

Прервитесь ненадолго и попробуйте поэкспериментировать с этими двумя вариантами, прежде чем продолжить чтение.

Видоизменение до или после

Третий вариант использования наследования — это особый случай переопределения, при котором вы желаете видоизменить поведение до или после запуска функции из класса `Parent`. Сначала вы переопределяете функцию так, как это сделано в последнем примере, а затем используете встроенную функцию `super` языка Python, чтобы использовать при вызове версию функции

из класса `Parent`. Вот пример того, как это делается, чтобы вы смогли уяснить приведенное описание.

ex44c.py

```
1  class Parent(object):
2
3      def altered(self):
4          print "PARENT altered() "
5
6  class Child(Parent):
7
8      def altered(self):
9          print "CHILD, BEFORE PARENT altered() "
10         super(Child, self).altered()
11         print "CHILD, AFTER PARENT altered() "
12
13 dad = Parent()
14 son = Child()
15
16 dad.altered()
17 son.altered()
```

Здесь важны строки 9–11; в них при вызове функции `son.altered()` в классе `Child` происходит следующее.

1. Поскольку я переопределил функцию `Parent.altered`, выполняется версия `Child.altered`, и строка 9 работает так, как и следовало ожидать.

2. В данном случае я хочу выполнить нечто до и после, и поэтому после строки 9 мне нужно использовать функцию `super`, чтобы добраться до версии `Parent.altered`.

3. В строке 10 я вызываю функцию `super(Child, self).altered()`, которая во многом подобна функции `getattr`, использованной вами в прошлом, но она учитывает наследование и обеспечит вам доступ к классу `Parent`. Следует понимать эту строку так: «Вызвать функцию `super` с аргументами `Child` и `self`, а затем вызвать функцию `altered` для возвращенного результата, каким бы он ни был».

4. На данном этапе выполняется версия `Parent.altered`, которая выводит сообщение `Parent`.

5. Наконец, происходит выход из функции `Parent.altered`, и работу продолжает функция `Child.altered`, выводя измененное сообщение.

Если вы запустите этот код, вы должны увидеть следующее.

Сеанс упражнения 44c

```
$ python ex44c.py
РОДИТЕЛЬ altered()
ПОТОМОК, ДО ВЫЗОВА altered() В РОДИТЕЛЕ
РОДИТЕЛЬ altered()
ПОТОМОК, ПОСЛЕ ВЫЗОВА altered() В РОДИТЕЛЕ
```

Комбинация взаимодействий

Для демонстрации всех вариантов привожу окончательную версию кода, в которой в одном файле показан каждый тип взаимодействия при наследовании.

ex44d.py

```
1   class Parent(object):
2
3     def override(self):
4         print "PARENT override() "
5
6     def implicit(self):
7         print "PARENT implicit() "
8
9     def altered(self):
10        print "PARENT altered() "
11
12  class Child(Parent):
13
14  def override(self):
15    print "CHILD override() "
16
```

```
17  def altered(self):
18      print "CHILD, BEFORE PARENT altered() "
19      super(Child, self).altered()
20      print "CHILD, AFTER PARENT altered() "
21
22  dad = Parent()
23  son = Child()
24
25  dad.implicit()
26  son.implicit()
27
28  dad.override()
29  son.override()
30
31  dad.altered()
32  son.altered()
```

Проанализируйте каждую строку этого кода и напишите комментарии, объясняющие, что выполняется в строке, и является ли это переопределением. После этого запустите код и проверьте, получилось ли то, что вы ожидали.

Сеанс упражнения 44d

```
$ python ex44d.py
РОДИТЕЛЬ implicit()
РОДИТЕЛЬ implicit()
РОДИТЕЛЬ override()
ПОТОМОК override()
РОДИТЕЛЬ altered()
ПОТОМОК, ДО ВЫЗОВА altered() В РОДИТЕЛЕ
РОДИТЕЛЬ altered()
ПОТОМОК, ПОСЛЕ ВЫЗОВА altered() В РОДИТЕЛЕ
```

Причины использования функции super()

Сказанное выше вполне соответствует здравому смыслу, но теперь мы попадаем в трудное положение, когда возникает *множественное наследование*. В этой ситуации вы определяете класс, который наследует поведение от одного или от *нескольких* классов, например, так.

```
class SuperFun(Child, BadStuff):
    pass
```

Это равносильно высказыванию «Создать класс с именем SuperFun, который одновременно наследует поведение классов Child и BadStuff».

В этом случае, когда вы осуществляете неявные действия с любым экземпляром SuperFun, язык Python должен отыскать допустимую функцию в иерархии обоих классов, Child и BadStuff, но ему необходимо сделать это в установленном порядке. Для этого применяется так называемый порядок применения методов (MRO, method resolution order), а также алгоритм, который называется C3.

Поскольку инструмент MRO сложен, а алгоритм четко определен, Python не доверяет его осуществление вам. Это доставило бы вам неудобства, не так ли? Вместо этого язык Python дает вам функцию super(), которая выполняет всю работу за вас в тех местах, где требуется альтернативный вариант работы, показанный выше на примере функции Child.altered. Когда есть функция super(), вам не нужно беспокоиться о том, как обеспечить правильность работы, поскольку язык Python найдет верную функцию за вас.

Использование функции super() с методом __init__

Чаще всего функция super() используется в функциях __init__ в базовых классах. Обычно это единственное место, где вам необходимо выполнить что-либо внутри потомка, а затем завершить инициализацию в родителе. Вот небольшой пример того, как это выполняется для класса Child.

```
class Child(Parent):

def __init__(self, stuff):
self.stuff = stuff
super(Child, self).__init__()
```

Это почти то же, что и в приведенном выше примере с Child.altered, за исключением того, что я определяю некоторые переменные внутри функции __init__ до того, как класс Parent инициализируется со своей функцией Parent.__init__.

Композиция

Применять наследование удобно, однако другой вариант выполнения того же самого — это всего лишь использовать дополнительные классы и модули, не полагаясь на неявное наследование. Если проанализировать три способа задействовать наследование, то можно увидеть, что в двух из них необходимо написать новый код, который заменяет или модифицирует функциональность. Это с легкостью можно воспроизвести посредством простого вызова функций в другом классе. Вот пример того, как это выполнить.

ex44e.py

```
class Other(object):

    def override(self):
        print "OTHER override() "

    def implicit(self):
        print "OTHER implicit() "

    def altered(self):
        print "OTHER altered() "

class Child(object):

    def __init__(self):
        self.other = Other()

    def implicit(self):
        self.other.implicit()

    def override(self):
        print "CHILD override() "

    def altered(self):
        print "CHILD, BEFORE OTHER altered() "
        self.other.altered()
        print "CHILD, AFTER OTHER altered() "

son = Child()

```

```
30  son.implicit()
31  son.override()
32  son.altered()
```

В этом коде я не использую имя `Parent`, поскольку это *не* взаимосвязь родитель — потомок в виде наследования. Это взаимосвязь в виде композиции, где класс `Child` обладает классом `Other` для выполнения своей работы. После запуска этого кода получаем следующий результат.

Сеанс упражнения 44e

```
$ python ex44e.py
КЛАСС OTHER, implicit()
ПОТОМОК override()
ПОТОМОК, ДО ВЫЗОВА altered() В КЛАССЕ OTHER
КЛАСС OTHER, altered()
ПОТОМОК, ПОСЛЕ ВЫЗОВА altered() В КЛАССЕ OTHER
```

Как видите, бóльшая часть кода в классах `Child` и `Other` повторяется, чтобы выполнять одно и то же. Единственное различие заключается в том, что мне пришлось определить функцию `Child.implicit`, которая выполняет лишь одно действие. После этого я мог бы задаться вопросом: была ли надобность в классе `Other` и не мог ли я просто сделать отдельный модуль с именем `other.py`?

Наследование или композиция: что выбрать?

В конечном итоге вопрос о выборе наследования или композиции упирается в попытку решения проблемы многократного использования кода. Вам не хотелось бы, чтобы код дублировался в вашем программном обеспечении, поскольку это неопрятно и неэффективно. Наследование решает эту проблему, предлагая вам механизм подразумеваемых функций в базовых классах. Композиция справляется с задачей, предоставляя вам модули и возможность простого вызова функций в других классах.

Если оба варианта решают проблему повторного использования, то какой из них предпочтительнее и для каких ситуаций? Ответ чрезвычайно субъективен, однако поделюсь с вами тремя рекомендациями на этот счет.

1. Любой ценой старайтесь избегать множественного наследования, поскольку это слишком сложно для регулярного применения. Если этого нельзя избежать, будьте готовы к изучению иерархии классов и потратьте время на то, чтобы выяснить, откуда все берется.

2. Используйте композицию для «упаковки» кода в модули, которые используются в различных независимых приложениях и ситуациях.

3. Применяйте наследование, только если существуют четко связанные, повторно используемые фрагменты кода, которые укладываются в одну общую концепцию, или же если вы вынуждены так поступить по какой-то причине.

Не будьте, однако, рабами этих правил. Следует помнить о том, что объектно-ориентированное программирование — это всецело общественная договоренность, которую выработали программисты для хранения и распространения кода. И так как это общественная договоренность, но кодифицированная в языке Python, люди, с которыми вы работаете, могут вынудить вас отойти от приведенных правил. В этом случае выясните их метод работы и адаптируйтесь к ситуации.

Практические задания

В этом упражнении всего одно задание на отработку навыков, зато большое. Посетите страницу **www.python.org/dev/peps/pep-0008** и начните применять изложенную там информацию в своем коде. Вы заметите, что некоторые моменты там отличаются от написанного в этой книге, но теперь вы сможете понять приведенные там рекомендации. В примерах кода далее эти указания либо соблюдаются, либо нет, в зависимости от того, насколько запутанным он становится. Советую и вам поступать так же, поскольку понятный код важнее, чем знание правил «для посвященных».

Распространенные вопросы

Как мне научиться лучше решать проблемы, с которыми мне не приходилось встречаться раньше?

Единственный способ научиться лучше решать проблемы — это решить *самостоятельно* столько проблем, сколько сможете. Как правило, при встрече

с трудным вопросом люди спешат поскорее найти ответ. Это оправдано, если вам необходимо выдать результат оперативно. Попробуйте, однако, найти самостоятельное решение, если вы располагаете временем. Остановитесь и как можно дольше концентрируйтесь только на этой задаче, проверьте все возможные варианты, пока вы не решите ее или не сдадитесь. После этого найденные вами решения станут более качественными, и в конечном итоге вы научитесь лучше решать проблемы.

Не являются ли объекты всего лишь копиями классов?

Для некоторых языков программирования (типа JavaScript) это верно. Эти языки называют языками прототипов, и немногочисленные различия между объектами и классами в них проявляются только в использовании. Однако в языке Python классы выступают в роли шаблонов, которые «чеканят» новые объекты, подобно производству монет при помощи чекана (шаблона).

Разработка игры

Вы должны учиться самостоятельно решать новые задачи. Я надеюсь, что, читая эту книгу, вы поняли, что вся необходимая информация доступна в Интернете. Вам просто нужно найти ее, используя правильные поисковые запросы. Помните об этом в данном упражнении, где вы начнете работу над большим проектом и попробуете довести его до рабочего состояния. Ниже изложены основные требования.

1. Создайте собственную игру, отличающуюся от той, которую привел я в качестве примера.

2. Используйте несколько файлов с кодом и команду `import`, чтобы импортировать их. Убедитесь, что вы понимаете, как правильно использовать команду `import`.

3. Создавайте по одному классу для каждой сцены (комнаты) и присваивайте им имена, которые соответствуют предназначению классов (например, `GoldRoom` или `KoiPondRoom`).

4. Ваш персонаж должен будет знать об этих сценах, поэтому создайте соответствующий класс. Существует много способов, однако попробуйте сделать так, чтобы каждая сцена возвращала информацию о том, какая сцена следует далее, или определите переменную, которая хранит эти данные.

Обратите внимание, что я не буду подсказывать вам. Потратьте неделю на выполнение этого задания и создайте лучшую игру, какую только можете. Используйте классы, функции, словари, списки — все, что только может понадобиться для создания прекрасной игры. Цель этого занятия заключается в том, чтобы научить структурировать классы, которые необходимы другим классам в других файлах.

Помните, я не буду рассказывать, как именно писать код, поскольку вы должны сделать это самостоятельно. Постарайтесь разобраться без чужой помощи. Программирование — это, по сути, поиск решения задачи, а это означает работу методом проб и ошибок, экспериментирование, провалы и победы или даже удаление проекта полностью и начало с нуля. Если вы попали в тупик, обратитесь за помощью к программистам и покажите им свой код. Если некоторые из них скупы на комментарии, не тратьте на них

время, а сосредоточьте внимание на людях, которые предлагают свою помощь. Продолжайте дорабатывать и исправлять код, пока игра не будет работать идеально.

Удачи в работе с игрой, и до встречи через неделю!

Проверка созданной игры

Теперь следует оценить игру, которую вы только что разработали. Может быть, вы сделали только половину, прежде чем зашли в тупик. Или закончили игру, но она работает лишь частично. В любом случае мы проработаем множество аспектов программирования, которые вы должны знать к этому моменту, и проверим, правильно ли вы использовали их в своей игре. Мы изучим, как правильно форматировать и использовать классы.

Вы задаетесь вопросом: почему сначала вы должны попробовать выполнить задание самостоятельно, а затем уже я расскажу вам, как делать это правильно? С этой страницы я буду приучать вас к самостоятельности. Я держал вас за руку все это время и не могу делать это далее. Теперь, вместо того чтобы я рассказывал вам, как и что делать, выполнять упражнения вы будете по своему усмотрению, а затем изучать способы улучшения кода, написанного вами.

Вам будет довольно сложно первое время, и за результат необходимо будет бороться, причем, вероятно, он вас расстроит, но придерживайтесь данной стратегии несмотря ни на что — и в конце концов вы научитесь самостоятельно решать проблемы. Вы начнете искать творческие пути решения проблем, а не просто копировать их из учебников.

Оформление функций

Все правила оформления функций, которым я научил вас ранее, применимы, включая следующее.

- По разным причинам программисты называют функции в составе классов *методами*. В целом это только маркетинг, просто имейте в виду — каждый раз, когда вы будете говорить «функция», программисты будут раздражающе поправлять вас и говорить «метод». Если они совсем достанут вас, попросите их продемонстрировать познания в математике и объяснить, чем «метод» отличается от «функции».

- Когда вы работаете с классами, то основную часть своего времени тратите на описание того, как классы «делают что-либо». Вместо того чтобы именовать функцию в соответствии с тем, что она выполняет, называйте ее так, как будто это команда, которую вы даете классу. Например, функция `pop` «говорит», по сути, следующее: «Эй, список, удали-ка это». Она не называется `remove_from_end_of_list`, поскольку, хоть и выполняет именно это, она не является командой для списка.

- Код функций должен быть небольшим и простым. По какой-то причине, когда люди осваивают классы, они забывают об этом.

Оформление классов

- Имена ваших классов должны быть написаны в «горбатом» регистре, например `SuperGoldFactory` вместо `super_gold_factory`.

- Старайтесь не усложнять используемые функции `__init__`. В противном случае их сложнее использовать.

- Имена прочих функций помимо строчных букв должны содержать «символ подчеркивания», например `my_awesome_hair`, но не `myawesomehair` или `MyAwesomeHair`.

- Будьте последовательны в организации аргументов функций. Если ваш класс предназначен для выполнения действий с пользователями, собаками и кошками, сохраняйте этот порядок следования во всем коде, даже если это не критично. Если в вашем коде одна функция принимает `(dog, cat, user)`, а другая — отдает `(user, cat, dog)`, код программы использовать будет трудно.

- Старайтесь не использовать глобальные и извлеченные из модулей переменные. Они должны быть автономны.

- Тупая логичность — это бич мелких умов. Логика — это хорошо, но тупо следовать какой бы то ни было идее только потому, что так делают все, — дурной тон. Задумайтесь.

- Всегда, всегда используйте формат `class` *Имя* `(object)`, иначе проблем не избежать.

Оформление кода

- Добавляйте пустые строки, чтобы людям было удобно читать ваш код. Существует много плохих программистов, которые умеют писать качественный код, но которые не добавляют пробелы. Это плохой стиль на любом языке, потому что человеческий глаз и мозг используют пустые пространства и строки для визуального выделения элементов.

- Если вы не можете прочитать код вслух, вероятно, он труден для восприятия. Если у вас возникли проблемы с каким-то несложным фрагментом кода, попробуйте прочитать его вслух. Помимо того что вы поймете, насколько код прост или сложен для восприятия, это поможет определить проблемные места.

- Старайтесь писать код так, как это делают другие программисты на Python, пока не определитесь с собственным стилем.

- После того как выработаете собственный стиль, не торопитесь использовать его. Работа с кодом других людей обязательна для каждого программиста, а другие люди могут иметь очень плохой стиль. Поверьте мне, вы, вероятно, тоже пишете код в неудачном стиле и даже не осознаете этого.

- Если вы встретили код в стиле, который вам нравится, попробуйте написать свою программу, имитируя этот стиль.

Оформление комментариев

- Вам обязательно встретятся программисты, которые сочтут, что ваш код вполне самодостаточен и комментарии не требуются. Скорее всего, это будет сказано официальным тоном, примерно так: «Я никогда не пишу комментарии. Ч. т. д.». Такие программисты либо консультанты, которые берут дополнительную плату за консультацию других людей, не понимающих код, либо некомпетентны и сами не понимают свой код. Никогда не работайте с такими людьми. Игнорируйте их и пишите комментарии.

- В комментариях описывайте, *что* и *почему* происходит в коде. Код сам уже говорит о том, *как* вы что-то сделали, но *почему* вы так поступили — еще более важно.

- При написании документирующих комментариев для ваших функций указывайте, где их можно использовать. Небольшой комментарий, кто и что может сделать с помощью этой функции, очень помогает.

- И, наконец, комментарии следует поддерживать в актуальном состоянии и не охватывать ими чрезмерно большие фрагменты кода. Комментируйте относительно небольшие фрагменты кода, а если вы внесли изменения в код, проверьте комментарии, чтобы убедиться, что они все еще актуальны.

Выставление оценки

Я хочу, чтобы прямо сейчас вы притворились мной. Примите очень строгий вид, распечатайте свой код, возьмите красную ручку и отметьте каждую найденную ошибку, из числа упомянутых как в этом упражнении, так и в других упражнениях и книгах, которые вы читали. После завершения проверки исправьте все найденные ошибки и упущения. Затем повторите проверку пару раз, разыскивая код, который можно улучшить. Применяйте на практике все советы, которые я вам предоставил, при анализе кода.

Цель этого упражнения заключается в тренировке вашего внимания к деталям в классах. После того как вы закончите с кодом из этого упражнения, возьмите другой и сделайте то же самое. Проанализируйте распечатку какого-либо его фрагмента и найдите все ошибки, в том числе и в плане оформления. Затем внесите исправления и проверьте, не нарушился ли процесс выполнения программы.

Я хочу, чтобы вы проверяли и исправляли код в течение недели, не отвлекаясь ни на что другое. Ваш собственный код и код других программистов. Это очень тяжелая работа, но, когда вы справитесь с ней, ваш мозг будет натренирован, как руки боксера.

УПРАЖНЕНИЕ 46

Каркас проекта

Вы начнете с создания подходящей схемы каталогов/файлов, она станет основой для работы нового проекта. Схема будет включать его макет, автоматизированные тесты, модули и установочные сценарии. Когда вам понадобится создать новый проект, просто скопируйте этот каталог, переименуйте и отредактируйте файлы, чтобы начать работу.

Установка пакетов Python

Прежде чем приступать к этому упражнению, необходимо установить программное обеспечение для Python, используя систему управления пакетами под названием pip. И здесь возникает проблема. Описать этот процесс будет затруднительно. Существует так много способов установки программного обеспечения на компьютер, что я потратил бы 10 страниц на пошаговое руководство.

Вместо детального описания я расскажу вам, что нужно установить и как заставить это работать. Так для вас откроется целый мир программного обеспечения, которое вы сможете использовать.

Установите следующие пакеты Python.

1. pip с сайта **pypi.python.org/pypi/pip**
2. distribute с сайта **pypi.python.org/pypi/distribute**
3. nose с сайта **pypi.python.org/pypi/nose**
4. virtualenv с сайта **pypi.python.org/pypi/virtualenv**

Нужно не просто скачать и установить эти пакеты вручную, как обычное программное обеспечение. Прочитайте раздел Installation на странице соответствующего пакета в Интернете, посмотрите, как другие люди устанавливают эти пакеты в вашей операционной системе, или же выполните поиск в Интернете по запросу «установка пакетов в python с помощью pip» (учитывайте при этом, что пользуетесь версией Python 2, а не 3). Процесс установки будет отличаться в различных версиях Linux, macOS и, безусловно, Windows.

Я предупреждаю вас: это вам не понравится. Между собой мы называем такой процесс «бритьем яка». «Бритье яка» — это любая рутинная, отупляющая, раздражающе скучная и утомительная работа, которую вам нужно сделать, прежде чем перейти к чему-то более увлекательному. Вы хотите создавать крутые проекты Python, но не можете начать делать это, пока не создали схему каталогов. И вы не можете создать схему каталогов, пока не установили несколько нужных пакетов Python. И вы не можете установить пакеты, пока не установите инсталлятор пакетов. И, в свою очередь, вы не можете установить инсталлятор пакетов, пока не выясните, как программное обеспечение устанавливается в вашей операционной системе, и так далее.

Вы справитесь с этим, так или иначе. Считайте, что это ваш экзамен, позволяющий попасть в клуб программистов. Каждый программист выполняет эти надоедливые, утомительные задачи, прежде чем сможет работать над чем-то поистине интересным.

Примечание. Иногда установщик Python не присваивает значение `C:\Python27\Scripts` системной переменной `PATH`. Если так и произошло в вашем случае, вернитесь назад и добавьте этот путь так же, как и `C:\python27` в упражнении 0 с помощью `[Environment]::SetEnvironmentVariable("Path", "$env:Path;C:\Python27\Scripts", "User")`.

Подготовка схемы каталогов проекта

Первым делом создайте схему каталогов, по очереди выполнив следующие команды в оболочке командной строки.

```
$ mkdir projects
$ cd projects/
$ mkdir skeleton
$ cd skeleton
$ mkdir bin
$ mkdir NAME
$ mkdir tests
$ mkdir docs
```

Я использую каталог с именем *projects*, в котором хранятся все мои рабочие проекты. Внутри него расположен каталог *skeleton*, содержащий основные

файлы проекта. Каталог *NAME* каждый раз переименовывается согласно основному модулю проекта, для которого используется схема каталогов.

Далее нужно создать некоторые исходные файлы. Вот как это выполняется в операционной системе Linux/macOS.

```
$ touch NAME/__init__.py
$ touch tests/__init__.py
```

Аналогичные шаги в оболочке Windows PowerShell.

```
$ new-item -type file NAME/__init__.py
$ new-item -type file tests/__init__.py
```

Эти команды создают пустые файлы модулей Python, в которые мы можем поместить наш код. Затем нужно создать файл *setup.py*, который в дальнейшем мы можем использовать для инсталляции нашего проекта, если это потребуется.

setup.py

```
1   try:
2   from setuptools import setup
3   except ImportError:
4      from distutils.core import setup
5
6   config = [
7      'description': 'My Project',
8      'author': 'My Name',
9      'url': 'URL to get it at.',
10     'download_url': 'Where to download it.',
11     'author_email': 'My email.',
12     'version': '0.1',
13     'install_requires': ['nose'],
14     'packages': ['NAME'],
15     'scripts': [],
16     'name': 'projectname'
17     ]
18
19  setup(**config)
```

Отредактируйте код в этом файле, добавив собственную информацию о проекте. Наконец, создайте простой шаблонный файл для тестов с именем *tests/NAME_tests.py*.

NAME_tests.py

```
1  from nose.tools import *
2  import NAME
3
4  def setup():
5     print u"УСТАНОВКА!"
6
7  def teardown():
8     print u"ЗАВЕРШЕНИЕ!"
9
10 def test_basic():
11    print u"ВЫПОЛНЕНИЕ!"
```

Окончательная структура каталогов

Когда вы закончите формирование структуры каталогов с файлами, она должна выглядеть так.

```
$ ls -R
NAME       bin        docs        setup.py        tests

./NAME:
__init__.py

./bin:

./docs:

./tests:
NAME_tests.py  __init__.py
```

Пример показан на платформе Unix, в операционной системе Windows структура идентична. Вот как она будет выглядеть в форме дерева.

```
setup.py
NAME/
   __init__.py
bin/
docs/
tests/
   NAME_tests.py
   __init__.py
```

И теперь вы должны выполнять свои команды, применимые к этому каталогу, из этой позиции. Если команда `ls -R` не позволяет отобразить такую структуру, вы находитесь в неправильном каталоге. К примеру, люди обычно переходят в каталог *tests/* и пытаются запустить там файлы, которые, разумеется, не будут работать. Для выполнения тестов приложения вы должны выйти из каталога *tests/* и подняться на уровень выше. Скажем, вы попробуете следующее.

```
$ cd tests/        # НЕПРАВИЛЬНО! НЕПРАВИЛЬНО!   НЕПРАВИЛЬНО!
$ nosetests

----------------------------------
Ran 0 tests in 0.000s

OK
```

Это ошибка! Чтобы избежать ее, вы должны выйти из каталога *tests/*, выполнив следующие действия.

```
$ cd .. # выходим из каталога tests/
$ ls # ПРАВИЛЬНО! Теперь вы в правильном каталоге
NAME  bin   docs    setup.py       tests
$ nosetests
.
----------------------------------
Ran 1 test in 0.004s

OK
```

Помните об этом, потому что люди совершают эту ошибку довольно часто.

Проверка проекта

После выполнения всех описанных в этом упражнении действий следующая команда должна работать.

```
$ nosetests
.
----------------------------------
Ran 1 test in 0.007s

OK
```

Я объясню, для чего предназначена команда `nosetests`, в следующем упражнении, а сейчас обратите внимание: если вы не видите результат, то, вероятно, где-то закралась ошибка. Убедитесь, что вы поместили файлы *__init__.py* в папки *NAME* и *tests* и создали файл *tests/NAME_tests.py*.

Использование каркаса

На данный момент вы проделали основную часть занудной работы по подготовке проекта. Каждый раз при создании нового проекта выполняйте следующие действия.

1. Создайте копию вашей схемы каталогов проекта. Присвойте ей имя после создания проекта.

2. Переименуйте каталог *NAME* в соответствии с названием вашего проекта или основного модуля.

3. Измените код в файле *setup.py*, добавив всю необходимую информацию для вашего проекта.

4. Переименуйте файл *tests/NAME_tests.py*, используя имя создаваемого модуля.

5. Дважды проверьте работоспособность, вновь используя команду `nosetests`.

6. Начинайте разработку.

Обязательно к выполнению

В этом упражнении нет практических заданий, но следующие пункты необходимо выполнить.

1. Поищите информацию об использовании всех созданных файлов.

2. Поищите информацию о том, как использовать файл *setup.py*. Внимание: это фрагмент программного обеспечения с не очень хорошо оформленным кодом, поэтому его весьма непривычно использовать.

3. Создайте проект и напишите код модуля, а затем проверьте его работоспособность.

4. Поместите сценарий в каталог *bin*, откуда вы сможете его запустить. Поищите информацию о том, как создать сценарий Python, который можно запустить на используемой вами платформе.

5. Укажите ссылку на сценарий, помещенный в каталог *bin*, в файле *setup.py*, чтобы он копировался при инсталляции.

6. Запустите файл *setup.py*, чтобы установить созданный вами модуль, проверьте его работоспособность, а затем используйте систему pip, чтобы удалить его.

Распространенные вопросы

Указанные инструкции применимы для операционной системы Windows?

Применимы, но в зависимости от используемой версии Windows вам, возможно, понадобится немного поколдовать над установкой, чтобы все работало. Продолжайте исследовать и тестировать проект, пока не получите успешный результат, или попросите более опытного программиста на Python помочь вам.

У меня не получается выполнить команду `nosetests` в операционной системе Windows.

Иногда установщик Python не присваивает значение `C:\Python27\Scripts` системной переменной `PATH`. Если так и произошло у вас, вернитесь назад и добавьте этот путь так же, как и `C:\python27` в упражнении 0.

Как правильно оформлять словарь `config` в файле *setup.py*?

Прочитайте документацию по системе distutils на странице **docs.python.org/distutils/setupscript.html**.

У меня не получается загрузить модуль `NAME`. Выводится ошибка `ImportError`.

Проверьте, создан ли файл *NAME/__init__.py*. Если вы пользуетесь операционной системой Windows, проверьте, не назвали ли вы его случайно именем *NAME/__init__.py.txt*. Дополнительное расширение присваивают некоторые текстовые редакторы.

Зачем во все проекты нужно добавлять папку *bin/*?

Это стандартный каталог для размещения сценариев, выполняемых в оболочке командной строки, но не модулей.

Можете ли вы привести пример реального проекта?

Существует много проектов, написанных на языке Python, которые используют описываемый каркас, например следующий, созданный мной: **https://gitorious.org/python-modargs**.

После выполнения команды `nosetests` отображается результат только одного теста. Так и должно быть?

Да, именно так. На моем компьютере результат такой же.

Автоматическое тестирование

Постоянный ввод команд для тестирования работоспособности игры не доставляет большого удовольствия. И вы наверняка задаетесь вопросом, нельзя ли написать небольшие сценарии, которые проверяют код? Можно! Затем, когда вносятся изменения или добавляется код в программу, понадобится просто «запустить тестирование» и проверить, по-прежнему ли работоспособна ваша программа. Такие автоматизированные тесты не смогут перехватить все ошибки, но в любом случае сэкономят ваше время, избавляя вас от ввода команд и проверки по нескольку раз кода вручную.

Начиная со следующего, каждое упражнение будет включать раздел «Что нужно тестировать?» вместо «Результат выполнения». Вы будете писать автоматизированные тесты для всех ваших программ, начиная с сегодняшнего дня, и так вы станете, я надеюсь, еще более крутым программистом.

Я не буду даже пытаться объяснить, почему вы должны писать автоматизированные тесты. Скажу только, что вы становитесь программистом, а программисты автоматизируют выполнение скучных и утомительных задач. Проверка кода программного обеспечения, безусловно, скучное и утомительное занятие, поэтому вы можете написать небольшой сценарий, делающий это за вас.

Кроме того, разрабатывая модульные тесты, вы получаете опыт в программировании. Эту книгу вы изучаете, чтобы разрабатывать программы. Теперь нужно сделать следующий шаг и написать код, который проверяет ваши программы. Так вы будете иметь четкое представление о том, что именно вы написали. Вы усвоите, что именно делает ваш код и почему он работает, а также потренируете внимание к деталям.

Создание примера для тестирования

Мы возьмем фрагмент очень простого кода и напишем один несложный тест. Разработка его будет вестись на основе вашего каркаса проекта.

Во-первых, создайте проект *ex47* из каркаса проекта. Ниже представлены шаги, которые следует выполнить. Я привожу краткие описательные

инструкции, а не команды для ввода, так как вы уже должны сами уметь работать в оболочке командной строки.

1. Скопируйте каталог *skeleton* в папку *ex47*.
2. Переименуйте все объекты с именем *NAME* на *ex47*.
3. Замените слово `NAME` во всех файлах на `ex47`.
4. И, наконец, удалите все файлы с расширением *.pyc*.

Вернитесь к упражнению 46, если вы зашли в тупик или испытываете трудности с выполнением упражнения, и, если требуется, попрактикуйтесь в его выполнении.

Примечание. Помните о выполнении команды `nosetests` для запуска тестирования. Вы можете использовать команду `python ex46_tests.py`, но это будет не так просто, и понадобится выполнить ее для каждого тестового файла.

Затем создайте простой файл *ex47/game.py*, в который поместите код для проверки. Это будет очень простенький класс, который мы будем проверять с помощью автоматизированного теста.

game.py

```
1   class Room(object):
2
3       def __init__(self, name, description):
4           self.name = name
5           self.description = description
6           self.paths = {}
7
8       def go(self, direction):
9           return self.paths.get(direction, None)
10
11      def add_paths(self, paths):
12          self.paths.update(paths)
```

Теперь создайте модульный тест.

ex47_tests.py

```
from nose.tools import *
from ex47.game import Room


def test_room():
    gold = Room("GoldRoom",
                """This room has gold in it you can grab. There's a
                door to the north.""")
    assert_equal(gold.name, "GoldRoom")
    assert_equal(gold.paths, {})

def test_room_paths():
    center = Room("Center", "Test room in the center.")
    north = Room("North", "Test room in the north.")
    south = Room("South", "Test room in the south.")

    center.add_paths({'north': north, 'south': south})
    assert_equal(center.go('north'), north)
    assert_equal(center.go('south'), south)

def test_map():
    start = Room("Start", "You can go west and down a hole.")
    west = Room("Trees", "There are trees here, you can go east.")
    down = Room("Dungeon", "It's dark down here, you can go up.")

    start.add_paths({'west': west, 'down': down})
    west.add_paths({'east': start})
    down.add_paths({'up': start})

    assert_equal(start.go('west'), west)
    assert_equal(start.go('west').go('east'), start)
    assert_equal(start.go('down').go('up'), start)
```

Этот файл импортирует класс `room`, созданный вами в модуле `ex47.game`, а затем проверяет его. Для этого используется набор тестов, которые представляют собой функции с именами, начинающимися с `test_`. Внутри каждого теста используется некоторый код, создающий сцену или группу сцен,

а затем проверяющий их работоспособность. Сначала код проверяет основные функции сцены, затем пути и, наконец, карту целиком.

Из важных функций следует отметить `assert_equal`, проверяющие объявленные вами переменные и пути на сцене. Если результат ошибочен, `nosetests` выведет сообщение об ошибке, и вы сможете исправить код.

Руководство по тестированию

Ниже приведены принципы, которыми следует руководствоваться при разработке тестов.

Файлы с тестами помещайте в каталог *tests/* и присваивайте им имена наподобие *ИМЯ_tests.py*; в противном случае команда `nosetests` не сможет их запустить. Кроме того, такие имена файлов не позволят спутать тесты с другими сценариями.

Пишите отдельный тестовый файл для каждого создаваемого вами модуля.

Составляйте краткий код тестов (функций) и не волнуйтесь, если они будут немного неаккуратными. В тестовых файлах допустим некоторый беспорядок.

Но даже в этом случае старайтесь быть аккуратными и удаляйте любой повторяющийся код. Создавайте вспомогательные функции, позволяющие избавиться от дублирующегося кода. Вы еще поблагодарите меня, когда внесете изменения в программу, а затем понадобится скорректировать код тестов. Дублирующийся код затруднит изменение тестов.

И, наконец, не полагайтесь полностью на тесты. Иногда лучший способ переделать что-то — просто удалить код и начать все сначала.

Результат выполнения

Сеанс упражнения 47

```
$ nosetests
...
----------------------------------
Ran 3 tests in 0.008s

OK
```

Выше вы видите результат выполнения команды, если все работает правильно. Попробуйте внести в код ошибку, чтобы просмотреть, как она будет обнаружена при тестировании, а затем исправьте ее.

Практические задания

Поищите в Интернете информацию о команде `nosetests`, а также поищите альтернативные возможности.

Поищите в Интернете информацию о «доктестах» Python и проверьте, может быть, они более удобны.

Расширьте возможности тестируемой сцены и переделайте код игры. Протестируйте код.

Распространенные вопросы

У меня выводится ошибка синтаксиса при выполнении команды `nosetests`. Почему это происходит?

Прочитайте текст ошибки и номер строки кода и исправьте ее. Такие инструменты, как `nosetests`, выполняют и проверяют ваш код, поэтому будут обнаруживать синтаксические ошибки так же, как и интерпретатор Python.

У меня не получается импортировать модуль `ex47.game`.

Проверьте наличие файла *ex47/__init__.py*. Вернитесь к упражнению 46, чтобы узнать, как его создать. Если причина не в этом и файл на месте, выполните следующую команду в операционной системе macOS/Linux:

```
export PYTHONPATH=$PYTHONPATH:.
```

Или эту в операционной системе Windows:

```
$env: PYTHONPATH = "$env: PYTHONPATH;."
```

И, наконец, убедитесь, что вы выполняете тестирование с помощью инструмента `nosetests`, а не просто интерпретатора Python.

При выполнении команды `nosetests` возникает ошибка `UserWarning`.

Вы, скорее всего, установили две версии Python или не инсталлировали пакет distribute. Вернитесь к упражнению 46 и установите пакет `distribute` или `pip`, как там описано.

УПРАЖНЕНИЕ 48

Расширенный пользовательский ввод

Ваша игра, надеюсь, работает как задумывалось, но вот пользовательский ввод совершенно не защищен от ошибок. На каждой сцене срабатывает собственный, очень точный набор фраз, которые интерпретатор принимает только в том случае, если игрок набрал их абсолютно точно. А вам наверняка хотелось бы, чтобы пользователи могли набирать фразы по-разному. К примеру, чтобы указанные ниже фразы работали одинаково.

- Открыть дверь.
- Открою дверь.
- Войду в дверь.

Эти две фразы должны также срабатывать одинаково.

- Ударю медведя.
- Ударю медведя по морде.

Пользователю будет удобно написать действие на «человеческом» языке, а код игры примет этот ввод за корректный. Для реализации такого поведения мы разработаем специальный модуль. Код модуля будет содержать несколько классов, необходимых для совместной обработки пользовательского ввода и обеспечения безошибочной работы игры. Упрощенная версия может учитывать следующие правила.

- Слова, разделенные пробелами.
- Фразы, составленные из слов.
- Грамматические конструкции, структурирующие предложения.

Начнем с выяснения того, как получить ввод от пользователя и какие слова нужно использовать.

Игровой словарь

В нашей игре мы создадим словарь, включающий следующие слова.

- **Направление.** North, south, east, west, down, up, left, right, back.
- **Глаголы.** Go, stop, kill, eat.
- **Стоп-слова.** The, in, of, from, at, it.
- **Существительные.** Door, bear, princess, cabinet.
- **Числа.** Любая строка длиной от 0 до 9 символов.

В случае с существительными есть небольшая проблема, так как каждая сцена может использовать собственный набор существительных, но давайте сейчас просто начнем работу и вернемся к этой проблеме позже.

Разделение предложений

После того как словарь составлен, нам понадобится способ разделить предложения. В нашем случае мы представляем предложение «словами, разделенными пробелами», поэтому понадобится сделать следующее.

```
stuff = raw_input('> ')
words = stuff.split()
```

Да, это действительно весь код, который нам сейчас нужен, но он реально будет работать очень хорошо.

Кортежи

Выяснив, как разбить предложение на слова, мы должны обработать список слов и указать «тип» каждого из них. Для реализации этого мы применим удобную структуру языка Python, называемую «кортеж». Кортеж — это

простой список, который вы не можете изменить. Он создается помещением данных, разделенных запятой, в скобки `()`, следующим образом.

```
first_word = ('direction', 'north')
second_word = ('verb', 'go')
sentence = [first_word, second_word]
```

Так создается пара (ТИП, СЛОВО), после чего можно анализировать и обрабатывать слова.

Выше приведен лишь пример, но, по сути, это конечный результат. Вам нужно принять «сырой» ввод от пользователя, разделить его на слова с помощью функции `split`, а затем проанализировать эти слова и определить их тип, после чего наконец создать из них предложение.

Анализ ввода

Теперь вы готовы написать код собственного анализатора. Он будет принимать строки «сырого» ввода от пользователя и возвращать предложение, состоящее из списка кортежей с парами (ТОКЕН, СЛОВО). Если слово не входит в словарь, анализатор должен вернуть СЛОВО и пометить ТОКЕН как ошибочный. Такие токены ошибок сообщат пользователям о неправильном вводе.

И вот где начинается праздник. Я не собираюсь рассказывать вам, как это сделать. Вместо этого я разработаю модульный тест, а вы — анализатор, причем так, чтобы он успешно проходил тестирование.

Исключения и числа

Существует одна маленькая проблема, с которой я помогу вам справиться. Она касается преобразования чисел. Чтобы решить эту задачу, мы немного смошенничаем и применим исключения. Исключение — это особая ситуация, которая может возникнуть при выполнении той или иной функции. Исключение возникает, когда функция сталкивается с ошибкой. После этого вы должны обработать это исключение. Например, вы ввели следующую команду в Python.

```
~/projects/simplegame $ python
Python 2.6.5 (r265:79063, Apr 162010, 13:57:41)
[GCC4.4.3] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>> int("hell")
Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'hell'
>>
```

`ValueError` — это исключение, вызванное функцией `int()` потому, что вы передали `int()` нечисловое значение. Функция `int()` могла бы вернуть значение, чтобы сообщить об ошибке, но, так как она возвращает только целые числа, этого не происходит. Она не может вернуть `-1`, так как это число. Вместо попытки выяснить, что возвращать при возникновении ошибки, функция `int()` вызывает исключение `ValueError`, и вы должны обработать его.

Обработка исключения выполняется путем использования ключевых слов `try` и `except`.

```
def convert_number(s):
  try:
    return int(s)
  except ValueError:
    return None
```

Вы помещаете уязвимый для ошибок код в блок `try`, а код, выполняемый в случае ошибки, в блок `except`. В этом случае мы указали, что функция `int()` должна возвращать целое число. Если возникает ошибка, ее следует перехватить и вернуть `None`.

В коде анализатора, разрабатываемого вами, вы должны использовать такой код, чтобы проверить, что введенные данные представляют собой число.

Что нужно тестировать?

Ниже представлен код, который вы должны использовать.

ex48.py

```
from nose.tools import *
from ex48 import lexicon


def test_directions():
    assert_equal(lexicon.scan("north"), [('direction', 'north')])
    result = lexicon.scan("north south east")
    assert_equal(result, [('direction', 'north'),
                          ('direction', 'south'),
                          ('direction', 'east')])

def test_verbs():
    assert_equal(lexicon.scan("go"), [('verb', 'go')])
    result = lexicon.scan("go kill eat")
    assert_equal(result, [('verb', 'go'),
                          ('verb', 'kill'),
                          ('verb', 'eat')])


def test_stops():
    assert_equal(lexicon.scan("the"), [('stop', 'the')])
    result = lexicon.scan("the in of")
    assert_equal(result, [('stop', 'the'),
                          ('stop', 'in'),
                          ('stop', 'of')])


def test_nouns():
    assert_equal(lexicon.scan("bear"), [('noun', 'bear')])
    result = lexicon.scan("bear princess")
    assert_equal(result, [('noun', 'bear'),
                          ('noun', 'princess')])

def test_numbers():
    assert_equal(lexicon.scan("1234"), [('number', 1234)])
    result = lexicon.scan("3 91234")
```

```
37      assert_equal(result, [('number', 3),
38                            ('number', 91234)])
39
40
41  def test_errors():
42      assert_equal(lexicon.scan("ASDFADFASDF"), [('error', 'ASDFADFASDF')])
43      result = lexicon.scan("bear IAS princess")
44      assert_equal(result, [('noun', 'bear'),
45                            ('error', 'IAS'),
46                            ('noun', 'princess')])
```

Вы создаете новый проект на основе схемы каталогов и вводите код тестов для него вручную. Разработайте свой анализатор так, чтобы он выполнял проверку проекта. Заострите внимание на деталях и убедитесь, что все работает должным образом.

Советы по разработке

Выполняйте проверки по очереди, одну за другой. Не усложняйте код; поместите все слова в вашем словаре в списки, которые находятся в модуле *lexicon.py*. Не редактируйте существующий список вводимых слов, а создайте новый список с кортежами. Кроме того, используйте ключевое слово `in` для обработки этих списков, чтобы проверять, входит ли введенное слово в словарь. Пользуйтесь словарями.

Практические задания

1. Улучшите модульный тест, чтобы охватить бóльшую часть словаря.

2. Добавьте значения в словарь, а затем обновите модульный тест.

3. Сделайте так, чтобы анализатор обрабатывал ввод данных пользователя в любом регистре. Обновите тест и проверьте работоспособность этого метода.

4. Найдите другой способ преобразования чисел.

5. Мой код занимает 37 строк. Ваш длиннее? Или короче?

Распространенные вопросы

Почему у меня продолжают возникать ошибки импорта?

Ошибки импорта обычно возникают в следующих четырех случаях: 1) вы не поместили файл *__init__.py* в каталог с модулями; 2) вы находитесь в неправильном каталоге; 3) вы импортировали неправильный модуль, поскольку ошиблись при вводе команды; 4) параметру `PYTHONPATH` не присвоено значение `.`, поэтому вы не можете загрузить модули из текущего каталога.

В чем разница между конструкциями `try-except` и `if-else`?

Конструкции `try-except` используются только для обработки исключений, возникающих при работе модулей. Они никогда не применяются в качестве альтернативы конструкции `if-else`.

Можно ли не прерывать игровой процесс, пока пользователь не ввел данные?

Я полагаю, что вы хотите, чтобы пользователей атаковали монстры, если пользователи не реагируют достаточно быстро? Это вполне реально, но необходимые модули и методы не описываются в этой книге.

Формирование предложений

Результат работы нашего маленького анализатора представляется списком, который выглядит следующим образом.

```
>>> from ex48 import lexicon
>>> print lexicon.scan("go north")
[('verb', 'go'), ('direction', 'north')]
>>> print lexicon.scan("kill the princess")
[('verb', 'kill'), ('stop', 'the'), ('noun', 'princess')]
>>> print lexicon.scan("eat the bear")
[('verb', 'eat'), ('stop', 'the'), ('noun', 'bear')]
>>> print lexicon.scan("open the door and smack the bear in the nose")
[('error', 'open'), ('stop', 'the'), ('noun', 'door'), ('error', 'and'),
('error', 'smack'), ('stop', 'the'), ('noun', 'bear'), ('stop',
'in'), ('stop', 'the'), ('error', 'nose')]
>>>
```

Теперь давайте превратим это в класс предложения для игры. Если вы помните уроки в начальной школе, структура предложения может быть представлена следующим образом:

```
Подлежащее Сказуемое Дополнение
```

Становится все сложнее, и вас, вероятно, это начинает раздражать, особенно если вы были невнимательны на уроках русского языка. Но все, что нам нужно сделать, — это превратить вышеупомянутые списки кортежей в замечательный объект `sentence`, включающий подлежащее, сказуемое и дополнение.

Соответствия и считывание

Нам понадобятся четыре способа.

1. Способ циклической обработки списка кортежей. Это несложно.

2. Способ поиска «соответствий» кортежей разного типа на основе установки «подлежащее, сказуемое и дополнение».

3. Способ «считывания» подходящего кортежа, чтобы мы могли делать определенный выбор.

4. Способ «пропуска» ненужных элементов, таких как стоп-слова.

Мы поместим код этих функций в файл с именем *ex48/parser.py* для тестирования. Мы используем функцию `peek`, чтобы считывать следующий элемент в нашем списке кортежа, а затем функцию `match`, чтобы извлечь один из них и обработать. Рассмотрим первую функцию `peek`.

```
def peek(word_list):
    if word_list:
        word = word_list[0]
        return word[0]
    else:
        return None
```

Все очень просто. Теперь — функция `match`.

```
def match(word_list, expecting):
    if word_list:
        word = word_list.pop(0)

        if word[0] == expecting:
            return word
        else:
            return None
    else:
        return None
```

Опять же ничего сложного. И, наконец, функция `skip`.

```
def skip(word_list, word_type):
    while peek(word_list) == word_type:
        match(word_list, word_type)
```

Вы должны понимать, как работают эти функции. Убедитесь, что понимаете.

Строение предложений

С помощью наших функций теперь мы можем начать строить объекты `sentence` из списка кортежей. Это делается следующим образом.

1. Определение следующего слова с помощью функции `peek`.
2. Если определенное слово подходящее, вызывается функция, обрабатывающая данную часть грамматики, допустим, с именем `parse_subject`.
3. Если этого не произойдет, инициируется (`raise`) ошибка, с которой вы познакомитесь в этом упражнении.
4. Когда все готово, должен быть получен объект `sentence` для использования в нашей игре.

Лучший способ продемонстрировать процесс — показать вам код. И вот чем это упражнение отличается от предыдущего: вы напишете модульный тест, проверяющий мой код анализатора.

Ниже представлен этот код, предназначенный для разбора простых предложений с помощью модуля `ex48.lexicon`.

ex49.py

```
class ParserError(Exception):
    pass


class Sentence(object):

  def __init__(self, subject, verb, object):
      # запомните, мы берем кортежи ('noun','princess')
      и конвертируем их
      self.subject = subject[1]
      self.verb = verb[1]
      self.object = object[1]


def peek(word_list):
  if word_list:
      word = word_list[0]
```

```
17          return word[0]
18      else:
19          return None
20
21
22  def match(word_list, expecting):
23      if word_list:
24          word = word_list.pop(0)
25
26          if word[0] == expecting:
27              return word
28          else:
29              return None
30      else:
31          return None
32
33
34  def skip(word_list, word_type):
35      while peek(word_list) == word_type:
36          match(word_list, word_type)
37
38
39  def parse_verb(word_list):
40      skip(word_list, 'stop')
41
42      if peek(word_list) == 'verb':
43          return match(word_list, 'verb')
44      else:
45          raise ParserError("Expected a verb next.")
46
47
48  def parse_object(word_list):
49      skip(word_list, 'stop')
50      next = peek(word_list)
51
52      if next == 'noun':
53          return match(word_list, 'noun')
54      if next == 'direction':
55          return match(word_list, 'direction')
56      else:
57          raise ParserError("Expected a noun or direction next.")
58
59
60  def parse_subject(word_list, subj):
```

```
61      verb = parse_verb(word_list)
62      obj = parse_object(word_list)
63
64      return Sentence(subj, verb, obj)
65
66
67  def parse_sentence(word_list):
68      skip(word_list, 'stop')
69
70      start = peek(word_list)
71
72      if start == 'noun':
73          subj = match(word_list, 'noun')
74          return parse_subject(word_list, subj)
75      elif start == 'verb':
76          # assume the subject is the player then
77          return parse_subject(word_list, ('noun', 'player'))
78        else:
79          raise ParserError("Must start with subject, object, or verb
            not: %s" % start)
```

Пара слов об исключениях

Вы уже немного знаете об исключениях, но не умеете генерировать их. Пример кода демонстрирует это в самом начале, в классе `ParserError`. Обратите внимание на использование в коде классов, чтобы применить тип `Exception`. Также обратите внимание на использование ключевого слова `raise`, позволяющего сгенерировать исключение.

При тестировании вы захотите обрабатывать эти исключения. Я покажу вам, как это сделать.

Что нужно тестировать?

Для упражнения 49 напишите полноценный тест, полностью проверяющий приведенный код. Тест должен быть помещен в файл *tests/parser_tests.py* по аналогии с предыдущим упражнением. Должны возникать исключения при конструировании неудачных предложений.

Проверяйте исключения с помощью функции `assert_raises`, описанной в документации к фреймворку nose. Разберитесь, как использовать эту функцию, чтобы написать тест, который может быть провален: это очень важно при тестировании. Подробнее об этой функции (и других) читайте в документации к nose.

По завершении вы должны знать, как работает данный фрагмент кода и как написать тест для проверки кода других разработчиков, даже если этого пока не требуется. Поверьте, это очень важный навык.

Практические задания

1. Измените код методов `parse_` и попытайтесь поместить их в класс, вместо того чтобы использовать их только как методы. Какой вариант вам больше нравится?

2. Разработайте синтаксический анализатор, более устойчивый к ошибкам, чтобы избежать праведного гнева пользователей при вводе нераспознаваемых слов.

3. Улучшите грамматическую часть, внедрив обработку дополнительных типов данных, таких как числа.

4. Задумайтесь, как можно использовать класс `sentence` в игре, чтобы реализовать дополнительные забавные возможности, связанные с вводом пользователя.

Распространенные вопросы

Функция `assert_raises` не работает как нужно.

Убедитесь, что вы используете синтаксис `assert_raises(exception, callable, parameters)` вместо `assert_raises(exception, callable(parameters))`. Обратите внимание, что во втором случае происходит вызов функции, а затем результат передается `assert_raises`, что неправильно. Вместо этого вы должны передать `assert_raises` функцию для вызова *и* соответствующие аргументы.

Ваш первый веб-сайт

Заключительные три упражнения очень сложны и займут много вашего времени. В текущем упражнении вы разработаете простую веб-версию одной из ваших игр. Прежде чем приступать к этому упражнению, вы *должны* успешно выполнить упражнение 46 и установить систему pip, позволяющую управлять пакетами, а также уметь создавать схему каталогов проекта. Если у вас есть затруднения, вернитесь к упражнению 46 и выполните его еще раз.

Установка фреймворка lpthw.web

Прежде чем приступить к созданию веб-приложения, вам понадобится установить «веб-фреймворк» под названием *lpthw.web*. Термин «фреймворк» обычно означает «некий пакет, который упрощает для меня некоторые действия». В мире веб-приложений разработчики создают «веб-фреймворки» во избежание сложных проблем, с которыми они могут столкнуться при создании собственных сайтов. Они распространяют такие общие решения в виде пакетов, которые можно скачать и использовать при разработке ваших собственных проектов.

В нашем упражнении мы будем использовать фреймворк *lpthw.web*, помимо которого существует множество других. Сначала изучите фреймворк *lpthw.web*, а затем переходите к другому, когда наберетесь опыта (или просто продолжайте использовать *lpthw.web*, так как его возможностей вполне достаточно).

Используя pip, установите фреймворк *lpthw.web*.

```
$ sudo pip install lpthw.web
[sudo] password for zedshaw:
Downloading/unpacking lpthw.web
  Running setup.py egg_info for package lpthw.web

Installing collected packages: lpthw.web
  Running setup.py install for lpthw.web

Successfully installed lpthw.web
Cleaning up...
```

Приведенный код будет работать в операционной системе Linux/macOS. Если вы пользуетесь операционной системой Windows, просто удалите из команды ключевое слово `sudo` следующим образом: `pip install lpthw.web`. Если у вас не получается установить данный фреймворк, вернитесь к упражнению 46 и проверьте, все ли вы делаете правильно.

Внимание! Другие программисты Python могут сказать вам, что *lpthw.web* лишь форк (ответвление) другого веб-фреймворка под названием *web.py* и что *web.py* гораздо «волшебнее». В этом случае ответьте им, что платформа Google AppEngine первоначально использовала *web.py* и ни один программист на Python не жаловался на избыток волшебства, поскольку все они работали с Google. То, что хорошо для Google, будет хорошим стартом и для вас. Раз так, давайте просто продолжим учиться программированию и будем игнорировать их попытки внушить нам какую-то идею вместо того, чтобы научить нас чему-то полезному.

Создание простого проекта

Для начала создадим простенькое веб-приложение «Привет, мир!» и каталог для этого проекта с помощью фреймворка *lpthw.web*. Сначала подготовим каталог проекта.

```
$ cd projects
$ mkdir gothonweb
$ cd gothonweb
$ mkdir bin gothonweb tests docs templates
$ touch gothonweb/__init__.py
$ touch tests/__init__.py
```

Вы возьмете игру из упражнения 43 и превратите ее в веб-приложение. Но сначала нужно создать базовое приложение *lpthw.web*. Напишите следующий код в файле *bin/app.py*.

ex50.py

```
1   import web
2
3   urls = (
4     '/', 'index'
```

```
    )

app = web.application(urls, globals())

class index:
    def GET(self):
        greeting = "Hello world!"
        return greeting

if __name__ == "__main__":
app.run()
```

Затем запустите приложение примерно так.

```
$ python bin/app.py
http://0.0.0.0:8080/
```

И хотя вам захочется поступить так...

```
$ cd bin/ # НЕПРАВИЛЬНО! НЕПРАВИЛЬНО! НЕПРАВИЛЬНО!
$ python app.py # НЕПРАВИЛЬНО! НЕПРАВИЛЬНО! НЕПРАВИЛЬНО!
```

Это неправильно. Ни в одном проекте Python вы не должны переходить в нижележащий каталог для запуска файлов. Вы должны оставаться в общем каталоге и запускать любые файлы оттуда, чтобы все системные компоненты могли получить доступ ко всем модулям и файлам. Перечитайте упражнение 46, чтобы разобраться в структуре каталогов проекта и способах управления им, если вам что-то неясно.

И, наконец, запустите веб-браузер и перейдите по адресу **http://localhost:8080/**. Вы должны увидеть следующее. Во-первых, в вашем браузере появится надпись Hello World!. Во-вторых, в оболочке командной строки появится следующий вывод.

```
$ python bin/app.py
http://0.0.0.0:8080/
127.0.0.1:59542 - - [13/Jun/201111:44:43] "http/1.1 GET /" — 200 OK
```

```
    127.0.0.1:59542 - - [13/Jun/201111:44:43] "http/1.1 GET /favicon.
ico" -      404 Not Found
```

Это так называемые журнальные сообщения, выводимые *lpthw.web*, чтобы вы могли убедиться, что сервер работает, и увидеть, что происходит «за кадром» браузера. Журнальные сообщения используются для отладки в случаях, если возникают проблемы с запуском. Например, в листинге выше сказано, что браузер пытался получить доступ к файлу */favicon.ico*, но, так как он не существует, программа вернула код статуса `404 Not Found`.

Я не объясняю, как работают все эти компоненты веб-приложения, потому что хочу, чтобы вы сначала попробовали создать его, а затем изучить на протяжении следующих двух упражнений. Чтобы вы все поняли, я «сломаю» веб-приложение *lpthw.web*, а затем реструктурирую его для понимания, как он настроен.

Что происходит?

Вот что происходит во время запуска веб-приложения в браузере.

1. Ваш браузер устанавливает сетевое соединение с вашим собственным компьютером, называемым локальным (`localhost`). Также для соединения используется порт 8080.

2. После того как соединение осуществлено, браузер отправляет HTTP-запрос к приложению *bin/app.py* и запрашивает URL-адрес `/`, который обычно является первым URL-адресом на любом сайте.

3. Внутри файла *bin/app.py* расположен список URL-адресов с соответствующими классами. Единственное соответствие, используемое в нашем файле, — это `'/', 'index'`. Оно означает, что всякий раз, когда в браузере осуществляется переход по адресу `/`, *lpthw.web* находит класс `index` и загружает его для обработки запроса.

4. Теперь, когда *lpthw.web* обнаружил класс `index`, вызывается метод `index.GET` для экземпляра этого класса, чтобы фактически обработать запрос. Соответствующая функция выполняется, и возвращается строка, которую *lpthw.web* должен передать в браузер.

5. И, наконец, *lpthw.web* обрабатывает запрос и отправляет ответ браузеру, в котором вы его и видите.

Убедитесь, что вы действительно поняли принцип работы. Составьте схему того, как данные из вашего браузера передаются *lpthw.web*, затем методу `index.GET` и обратно в браузер.

Работа над ошибками

Во-первых, удалите строку 11, где вы присваиваете значение переменной `greeting`; затем нажмите кнопку **Обновить** (Refresh) в браузере. Вы должны увидеть страницу с исчерпывающей информацией о полученной ошибке и неработоспособности вашего приложения. Вы знаете, что ошибка возникла из-за отсутствия переменной `greeting`, тем не менее прочитайте информацию на странице, сгенерированной *lpthw.web*, чтобы отследить ошибку. Выполните следующие действия.

1. Разверните каждую строку **Local vars** (щелкните мышью по черному треугольнику) и посмотрите, о каких переменных и в каких строках идет речь.

2. Взгляните на раздел **Request Information** и найдите информацию, с которой вы уже знакомы. Это данные, которые ваш веб-браузер передает приложению *gothonweb*. Как правило, в процессе работы эти данные вам не видны, поэтому изучите этот раздел, чтобы понять, какая информация передается.

3. Попробуйте «сломать» веб-приложение другими способами и исследовать, что происходит. Не забудьте также изучать журнальные сообщения в оболочке командной строки, так как туда *lpthw.web* выводит другие данные, помогающие отслеживать ошибки.

Создание базовых шаблонов

Вы разобрали приложение *lpthw.web* и наверняка заметили, что Hello World! — не слишком интересная HTML-страница. Это веб-приложение, и оно нуждается в надлежащем HTML-ответе. Чтобы добиться этого, вы создадите простой шаблон, форматирующий текст крупным шрифтом зеленого цвета. Также мы переведем текст приветствия на русский язык.

Первый шаг заключается в создании файла *templates/index.html* со следующим кодом.

index.html

```
$def with (greeting)
<html>
    <head>
       <meta charset="utf-8">
       <title>Готоны с планеты Перкаль 25</title>
    </head>
    <body>

   $if greeting:
       Я просто хочу сказать <em style="color: green; font-size:
       2em;">$greeting</em>.
    $else:
       <em>Привет,</em> мир!

    </body>
</html>
```

Если вы знакомы с языком HTML, код покажется вам знакомым. Если нет, то изучите основы HTML и попробуйте сверстать несколько веб-страниц вручную, чтобы понять принцип работы этого языка. Данный HTML-файл представляет собой *шаблон*, означающий, что *lpthw.web* будет заполнять «промежутки» в тексте в соответствии с переменными, указанными в шаблоне. Вместо каждой переменной `$greeting` в шаблоне вы будете видеть соответствующий контент из базового файла веб-приложения.

Чтобы приложение *bin/app.py* поддерживало шаблон, а также для локализации приветствия вам нужно добавить некоторый код, сообщающий *lpthw.web*, где расположен шаблон и как визуализировать его в браузере. Измените код в файле *bin/app.py* следующим образом.

app.py

```
1   # -*- coding: utf- 8 -*-
2
3   import web
4
5   urls = (
6      '/', 'Index'
7     )
8
9   app = web.application(urls, globals())
10
```

```
render = web.template.render('templates/')

class Index(object):
    def GET(self):
        greeting = u"Привет, мир"
        return render.index(greeting = greeting)

if __name__ == "__main__":
app.run()
```

Обратите пристальное внимание на строку с кодировкой в начале файла, новую переменную `render` и изменения в последней строке блока `index.GET`, касающиеся возвращения функции `render.index()` и передачи переменной `greeting`.

Внимание! Если вам все же лень вручную набирать код примеров из этой книги, все файлы с кодом вы можете скачать по адресу **https://eksmo.ru/files/Shaw_Python.zip**.

После того как внесены все изменения, перезагрузите веб-страницу в браузере, и вы увидите другое сообщение с приветствием, набранным шрифтом зеленого цвета. Вы также можете просмотреть исходный код страницы с помощью соответствующей команды в браузере, чтобы убедиться, что перед вами действительно HTML-файл.

Вы могли не понять, зачем нужны внесенные изменения, поэтому позвольте мне объяснить, как работает шаблон.

1. В файл *bin/app.py* вы добавили новую переменную, `render`, которой назначен объект `web.template.render`.

2. Этот объект `render` «знает», что нужно загружать файлы *.html* из каталога *templates/*, поскольку вы передали ему эту информацию в качестве параметра.

3. Далее по коду, когда браузер встречает строку `index.GET`, вместо простого возвращения строки `greeting`, как это было ранее, вы

вызываете метод `render.index` и передаете ему приветствие в качестве переменной.

4. Метод `render.index` является своего рода *волшебной* функцией, в которой объект `render` «видит», что вы запрашиваете элемент `index`, переходит в каталог *templates/*, ищет страницу с именем *index.html*, а затем визуализирует ее.

5. В файле *templates/index.html*, как вы видите, начальное определение указывает, что данный шаблон принимает параметр `greeting` по аналогии с функцией. Кроме того, код в этом шаблоне чувствителен к отступам, как и сценарии Python, поэтому убедитесь, что вы указали их правильно.

6. И, наконец, в файле *templates/index.html* есть HTML-код, который ищет переменную `greeting` и, если она обнаружена, выводит одно сообщение с помощью кода `$greeting` (сообщение по умолчанию).

Чтобы погрузиться глубже в это упражнение, измените значение переменной `greeting` и соответствующий HTML-код, чтобы просмотреть результат. Кроме того, создайте еще один шаблон с именем *templates/foo.html* и визуализируйте его с помощью метода `render.foo()` вместо `render.index()`. Так вы увидите, как имя функции, которую вы вызываете в методе `render`, сопоставляется с файлом *.html* в каталоге *templates/*.

Практические задания

1. Прочитайте документацию на сайте **webpy.org** по фреймворку, аналогичному *lpthw.web*.

2. Попрактикуйтесь со всеми полученными знаниями, в том числе поэкспериментируйте с примерами кода.

3. Прочитайте о стандартах HTML5 и CSS3 и попробуйте создать дополнительные файлы *.html* и *.css*.

4. Если у вас есть друг, который знаком с Django и готов помочь вам в нем разобраться, после прочтения книги выполните упражнения 50, 51 и 52 в Django.

Распространенные вопросы

Я не могу подключиться к http://localhost:8080.

В качестве альтернативы попробуйте подключиться по адресу **http://127.0.0.1:8080/**.

В чем разница между *lpthw.web* и *web.py*?

Ее нет. Я просто «заблокировал» конкретную версию *web.py*, чтобы использовать ее для обучения с помощью этой книги, и назвал ее *lpthw.web*. Более поздние версии фреймворка *web.py* могут отличаться от текущей версии.

Я не могу найти файл *index.html* (или какой-то другой).

Вы, вероятно, сначала выполнили команду `cd bin/`, а затем пытаетесь работать над проектом. Не поступайте так. Все команды и инструкции предполагают, что вы находитесь в каталоге уровнем выше *bin/*, поэтому, если вы не можете выполнить команду `python bin/app.py`, вы находитесь в неправильном каталоге.

Почему мы присваиваем `greeting=greeting`, когда обращаемся к шаблону?

Вы не присваиваете `greeting`; вы устанавливаете именованный параметр для обращения к шаблону. Это действие влияет только на вызов функции шаблона.

Я не могу использовать порт 8080 на моем компьютере.

Вероятно, у вас установлена антивирусная программа, которая использует этот порт. Попробуйте другой порт.

После установки *lpthw.web* возникает ошибка `ImportError "No module named web"`.

Вы, скорее всего, установили несколько версий Python и используете не ту из них, или установка прошла неудачно из-за использования старой версии pip. Попробуйте удалить *lpthw.web* и установить его заново. Если это не сработает, убедитесь, что вы используете правильную версию Python.

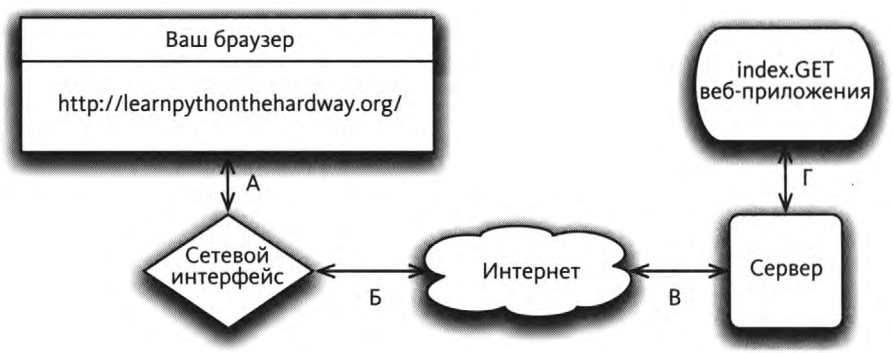
Получение ввода из браузера

Думаю, вам было интересно увидеть в окне браузера текст «Привет, мир!». И будет еще более интересно, если позволить пользователю передавать текст в приложение с помощью веб-формы. В этом упражнении мы улучшим наше веб-приложение, добавив веб-формы и возможность хранения информации о пользователях в «сеансах».

Как устроена Всемирная паутина

Время скучной теории. Вы должны узнать немного больше о том, как устроена Всемирная паутина, прежде чем сможете разработать веб-форму. Ниже представлено краткое, но точное описание процесса работы, которое поможет определить источник проблемы, если что-то пойдет не так с вашим приложением. Кроме того, работать с веб-формами будет проще, если знать, как они функционируют.

Начнем разбор полетов с простой схемы, которая иллюстрирует процесс передачи данных при веб-запросе.



Для упрощения я отметил этапы прохождения запроса буквами.

1. Вы вводите URL-адрес **http://learnpythonthehardway.org/** в адресной строке браузера, и он посылает запрос на сетевой интерфейс (А) вашего компьютера.

2. Ваш запрос передается через Интернет (Б) на удаленный компьютер (В), на котором мой сервер принимает запрос.

3. После того как сервер принял его, мое веб-приложение получает его (Г), и мой код Python запускает обработчик `index.GET`.

4. Ответ отправляется с моего сервера Python, когда выполняется команда `return`, и возвращается в ваш браузер (Г).

5. Сервер с запущенным сайтом принимает ответ в автономном режиме (Г), а затем отправляет его обратно через Интернет (В).

6. Ответ от сервера через Интернет передается на сетевой интерфейс компьютера (Б), а затем и в браузер (А).

7. И, наконец, ваш браузер отображает ответ.

В этом описании используется несколько терминов, которые вы должны знать, чтобы успешно работать с веб-приложениями.

Браузер. Программное обеспечение, которое вы, вероятно, используете каждый день. Большинство людей не знают, как на самом деле работает браузер. Они просто называют браузер «Интернетом». Работа браузера состоит в приеме адресов (например, **http://learnpythonthehardway.org**), которые вы вводите в строке URL-адреса, и дальнейшем использовании этой информации для отправки запросов на сервер по указанному адресу.

Адрес. Это, как правило, URL (Uniform Resource Locator — Единый указатель ресурса), например **http://learnpythonthehardway.org**, который указывает, куда браузер должен обращаться. Первая часть (**http**) указывает на используемый протокол (в данном случае HyperText Transfer Protocol — Протокол передачи гипертекста). Также используется протокол **ftp://ibiblio.org** (File Transfer Protocol — Протокол передачи файлов). Вторая часть (**learnpythonthehardway.org**) представляет собой «имя хоста», используемое человеком в качестве читабельного адреса и подставляемое вместо числового IP-адреса (похожего на телефонный номер компьютера в Интернете). Наконец, URL-адреса могут иметь **путь** наподобие **/book** примерно такой: **http://learnpythonthehardway.org/book**. Путь указывает на файл или какой-либо другой ресурс *на* сервере для извлечения с помощью запроса. Существует много других частей, но это основные из них.

Соединение (подключение). После того как браузер «узнал», что вы хотите использовать такой-то протокол (**http**), такой-то сервер (**learnpythonthehardway.org**) и получить определенный ресурс на этом сервере, он должен установить соединение. Браузер просто запрашивает у операционной системы возможность открыть «порт» на компьютере — обычно 80. Когда порт открывается, начинается передача данных через Интернет между вашим и серверным компьютером с сайтом **learnpythonthehardway.org**. Точно так же происходит и локальное соединение по адресу **http://localhost:8080**, только в этом случае вы инструктируете браузер подключиться к вашему собственному компьютеру (локальному) и использовать порт 8080, а не 80. Вы также можете обратиться по адресу **http://learnpythonthehardway.org:80** и получить тот же результат, за исключением явного использования порта 80 вместо того, который установлен по умолчанию.

Запрос. Браузер осуществляет подключение с помощью указанного вами адреса. Теперь он должен запросить ресурсы (которые вы хотите получить) на удаленном сервере. Если вы указали путь */book* в конце URL-адреса, значит, вы хотите получить доступ к файлу (ресурсу) в каталоге */book*; при этом большинство серверов перенаправит на файл */book/index.html*, если он существует. То, что делает браузер, чтобы получить этот ресурс, называется отправкой запроса на сервер. Я не буду вдаваться в подробности, как именно это происходит, нужно просто понять, что браузер должен послать определенные данные в качестве запроса к серверу. Интересно, что эти «ресурсы» необязательно должны быть файлами. Например, если веб-приложение в браузере запрашивает какие-либо данные, сервер возвращает некий сгенерированный контент.

Сервер. Сервер — это компьютер, к которому обращается браузер и который знает, как реагировать на запросы файлов/ресурсов, поступающие от вашего браузера. Большинство веб-серверов просто отправляют файлы, и в реальности это бóльшая часть трафика. Но в этом упражнении вы создаете сервер, который будет принимать запросы на ресурсы, а затем возвращать строки, которые вы будете использовать в разработке. Когда разработка будет завершена, вам будет казаться, что в браузер поступают файлы, хотя на самом деле это лишь код. Как можно видеть из упражнения 50, для создания ответа используется небольшой фрагмент кода.

Ответ. Это некий HTML- (CSS-, JavaScript- или графический) контент, который сервер передает обратно в браузер в ответ на его запрос.

Файлы же сервер просто считывает с жесткого диска и отправляет в браузер, при этом упаковывая контент с использованием специального «заголовка», по которому браузер определяет, что получает. В случае вашего приложения вы также будете передавать данные, включая заголовок, но генерировать эти данные на лету с помощью кода на языке Python.

Это самый быстрый курс, обучающий тому, как веб-браузер получает доступ к информации на серверах в Интернете. Теории достаточно для того, чтобы понять это упражнение, но, если у вас остаются вопросы, почитайте по этой теме столько, сколько нужно, чтобы разобраться в ней. Как вариант, возьмите диаграмму, приведенную в начале упражнения, и разделите веб-приложение из упражнения 50 на соответствующие части. Если у вас это получается, значит, вы поняли, как устроена Всемирная паутина.

Принцип работы веб-формы

Лучший способ научиться работать с веб-формами — это написать код, позволяющий принимать вводимые в элементы формы данные, а потом узнать, как с этими данными работать. Откройте файл *bin/app.py* и напишите в нем следующий код.

form_test.py

```
import web

urls = (
   '/hello', 'Index'
  )


app = web.application(urls, globals())

render = web.template.render('templates/')

class Index(object):
   def GET(self):
        form = web.input(name=u"Неизвестный")
        greeting = u"Привет, %s" % form.name

```

```
17          return render.index(greeting = greeting)
18
19  if __name__ == "__main__":
20  app.run()
```

Перезапустите приложение (в оболочке командной строки нажмите сочетание клавиш **Ctrl+C**, а затем запустите приложение снова), чтобы убедиться, что загружается измененное приложение; а затем с помощью браузера перейдите по адресу **http://localhost:8080/hello**. Вы должны увидеть текст «Я просто хочу сказать *Привет, Неизвестный*». Затем измените URL-адрес в вашем браузере следующим образом: **http://localhost:8080/hello?name=Петька**, и вы увидите текст «Я просто хочу сказать *Привет, Петька*». Вы можете изменить часть адреса **name=Петька**, подставив собственное имя. Теперь программа поздоровается с вами.

Давайте разберем изменения, которые я внес в сценарий.

1. Вместо использования обычной строки `greeting` я применил метод `web.input`, позволяющий получить данные от браузера. Эта функция по умолчанию принимает значение в формате «ключ=значение». Она анализирует часть URL-адреса **name=Петька**, которую вы передаете, а затем возвращает подходящий объект, представляющий эти значения.

2. Затем я конструирую элемент `greeting` из нового атрибута `form.name` объекта `form`, о чем вам должно быть известно.

3. Все остальное в файле осталось без изменений.

Вы также не ограничены только одним параметром в URL-адресе. Внесите изменения в этот пример, используя две переменные следующим образом: **http://localhost:8080/hello?name=Петька&greet=Здравствуй**. Затем подкорректируйте код в файле *app.py*, внедрив `form.name` и `form.greet` следующим образом.

```
greeting = " %s, %s" % (form.greet, form.name)
```

После этого, попробуйте перейти по указанному URL-адресу. Если вы удалите часть **&greet=Здравствуй**, отобразится сообщение об ошибке. Поскольку параметру `greet` не присвоено значение по умолчанию в строке

`web.input(name=u"Неизвестный")`, это обязательный элемент. Вернитесь к коду приложения и укажите данное значение по умолчанию в вызове `web.input`, чтобы убедиться, что ошибка исправлена. Также вы можете поступить следующим образом: указать значение `greet=None`, чтобы проверить, существует ли приветствие, и в противном случае отобразить более подходящее сообщение об ошибке.

```
form = web.input(name=u"Неизвестный", greet=None)

if form.greet:
   greeting = " %s, %s" % (form.greet, form.name)
   return render.index(greeting = greeting)
else:
   return u"ОШИБКА: требуется приветствие с помощью greet."
```

Создание HTML-форм

Передача параметров в URL-адресе — действенный способ, но не очень удобный для пользователей. То, что вам действительно нужно, это «форма с методом POST», которая представляет собой специальный HTML-файл, содержащий код элемента `<form>`. Эта веб-форма будет принимать данные от пользователя, а затем передавать их в веб-приложение так же, как и в предыдущем примере.

Давайте создадим такой файл и протестируем его в работе. Ниже представлен код нового HTML-файла *hello_form.html*, который нужно поместить в каталог *templates/*.

hello_form.html

```
<html>
    <head>
   <meta charset="utf-8">
        <title>Пробная веб-форма</title>
    </head>
<body>

<h1>Заполните эту форму</h1>

```

```
<form action="/hello" method="POST">
  Приветствие: <input type="text" name="greet">
    <br/>
  Ваше имя: <input type="text" name="name">
    <br/>
  <input type="submit">
</form>

</body>
</html>
```

А в файл *bin/app.py* внесите следующие изменения.

post_form.py

```
import web

urls = (
  '/hello', 'Index'
 )

app = web.application(urls, globals())

render = web.template.render('templates/')

class Index(object):
  def GET(self):
      return render.hello_form()

  def POST(self):
      form = web.input(name=u"Неизвестный", greet=u"Привет")
      greeting = " %s, %s" % (form.greet, form.name)
      return render.index(greeting = greeting)

if __name__ == "__main__":
app.run()
```

После внесения изменений, которые я продемонстрировал выше, просто перезапустите веб-приложение в своем браузере.

На этот раз вы увидите форму с полями ввода «Приветствие:» и «Ваше имя:». Когда вы нажмете кнопку «Отправить запрос» в веб-форме, вы увидите такое же приветствие, что и ранее. При этом обратите внимание на URL-адрес в адресной строке браузера: используется значение **http://localhost:8080/hello**, несмотря на переданные в параметрах значения.

За это в файле *hello_form.html* отвечает строка кода `<form action="/hello" method="POST">`. Данный код сообщает браузеру следующее.

1. Следует выполнить сбор данных от пользователя с помощью элементов веб-формы.

2. Передать собранные данные на сервер, используя запрос типа POST, который представляет собой другой вариант браузерного запроса, «скрывающего» данные из элементов формы.

3. Передать данные по URL-адресу **/hello** (как указано в коде `action="/hello"`).

После этого можно увидеть, что имена двух элементов `input` совпадают с именами переменных в измененном коде. Также обратите внимание, что вместо метода GET внутри класса `index` я использую другой метод, POST. Измененное приложение работает следующим образом.

1. Браузер сначала обращается к веб-приложению в каталоге */hello*. Оно передает запрос по методу GET, поэтому запускается функция `index.GET` и возвращает форму `hello_form`.

2. Пользователь заполняет веб-форму в браузере, а браузер передает данные из элемента `form` по методу POST.

3. Веб-приложение инициирует выполнение метода `index.POST`, а не `index.GET` для обработки этого запроса.

4. Данный метод `index.POST` передает обратно страницу приветствия, как и ранее. На самом деле, здесь ничего не изменилось, код лишь перемещен в новую функцию.

Для практики откройте файл *templates/index.html* и добавьте ссылку для возврата к состоянию **/hello**, чтобы можно было заполнять форму и просматривать результат. Разберитесь, как эта ссылка функционирует, позволяя вам циклично переключаться между файлами *templates/index.html* и *templates/hello_form.html*, и что происходит при выполнении этой последней версии кода Python.

Подготовка макета шаблона

Когда вы приступите к работе над игрой в следующем упражнении, вам понадобится создать множество небольших HTML-страниц. Писать каждый раз страницы с нуля вам быстро надоест. К счастью, вы можете создать «макет» шаблона (оформление оболочки), который отформатирует все ваши страницы с использованием одинаковых колонтитулов. Опытные программисты всегда избавляются от повторного выполнения одной и той же работы, поэтому макеты имеют для них важное значение.

Внесите в файл *templates/index.html* следующие изменения.

index_laid_out.html

```
$def with (greeting)

$if greeting:
   Я просто хочу сказать <em style="color: green; font-size:
   2em;">$greeting</em>.
$else:
   <em>Привет,</em> мир!
```

А файл *templates/hello_form.html* измените так.

hello_form_laid_out.html

```
<h1>Пробная веб-форма</h1>

<form action="/hello" method="POST">
   Приветствие: <input type="text" name="greet">
     <br/>
   Ваше имя: <input type="text" name="name">
     <br/>
   <input type="submit">
</form>
```

Все, что мы сделали, — удалили шаблонный контент из верхней и нижней части страниц, который на них отображается. Мы поместим его в единый

файл *templates/layout.html*, который теперь будет участвовать в работе веб-приложения.

После того как вы внесли изменения в два файла выше, создайте новый файл *templates/layout.html* со следующим кодом.

layout.html

```
$def with (content)

<html>
<head>
  <meta charset="utf-8">
  <title>Готоны с планеты Перкаль 25</title>
</head>
<body>

$: content

</body>
</html>
```

Этот файл выглядит как обычный шаблон, за исключением того, что он передает контент из других шаблонов и используется для оборачивания его. Код, помещенный в этот файл, не требуется в других шаблонах. Следует также обратить внимание на строку `$: content`, так как этот код несколько отличается от других переменных в шаблоне.

В заключение измените в файле *app.py* строку кода, которая отвечает за объект `render`.

```
render = web.template.render('templates/', base="layout")
```

Этот код сообщает фреймворку *lpthw.web* о необходимости использовать файл *templates/layout.html* в качестве базового шаблона для всех остальных шаблонов. Перезапустите приложение и просмотрите, что получилось. А затем попытайтесь изменить расположение компонентов макета без правки других шаблонов.

Разработка автоматических тестов для веб-форм

Вы легко можете проверить веб-приложение с помощью браузера, нажав кнопку **Обновить** (Refresh), но давайте не забывать, что мы с вами — программисты. Зачем эти повторяющиеся действия, когда мы можем написать код, позволяющий проверить наше приложение? Далее вы напишете небольшой тест для вашего веб-приложения, пользуясь знаниями, полученными во время изучения упражнения 47. Если вы не помните о чем шла речь в упражнении 47, прочитайте его еще раз.

Вам понадобится выполнить настройку, чтобы Python смог загрузить файл *bin/app.py* для тестирования. Когда мы перейдем к упражнению 52, вы измените это поведение, но сейчас создайте пустой *bin/__init__.py*, чтобы Python считал *bin/* каталогом.

Я также создал небольшую функцию для lpthw.web, анализирующую поведение вашего веб-приложения, которую назвал `assert_response`. Создайте тестовый файл *tests/tools.py* со следующим содержимым.

tools.py

```
from nose.tools import *
import re

def assert_response(resp, contains=None, matches=None, headers=None,
    status="200"):

    assert status in resp.status, "Expected response %r not in %r" %
        (status, resp.status)

    if status == "200":
        assert resp.data, "Response data is empty."

    if contains:
        assert contains in resp.data, "Response does not contain %r" %
        contains

    if matches:
        reg = re.compile(matches)
        assert reg.matches(resp.data), "Response does not match %r" %
        matches
```

```
17
18      if headers:
19          assert_equal(resp.headers, headers)
```

Теперь вы можете написать автоматизированный тест для последней версии *bin/app.py* созданного файла. Создайте новый файл с именем *tests/tools.py* и напишите в нем следующий код.

app_tests.py

```
1   from nose.tools import *
2   from bin.app import app
3   from tests.tools import assert_response
4
5   def test_index():
6       # check that we get a 404 on the / URL
7       resp = app.request("/")
8       assert_response(resp, status="404")
9
10      # test our first GET request to /hello
11      resp = app.request("/hello")
12      assert_response(resp)
13
14      # make sure default values work for the form
15      resp = app.request("/hello", method="POST")
16      assert_response(resp, contains="Nobody")
17
18      # test that we get expected values
19      data = ['name': 'Zed', 'greet': 'Hola']
20      resp = app.request("/hello", method="POST", data=data)
21      assert_response(resp, contains="Zed")
```

И, наконец, используйте команду `nosetests` для тестирования веб-приложения.

```
$ nosetests
.
----------------------------------
Ran 1 test in 0.059s

OK
```

Все, что здесь происходит, — это *импорт* всего приложения целиком из модуля *bin/app.py*, а затем запуск его вручную. Фреймворк *lpthw.web* содержит очень простой API для обработки запросов, который выглядит следующим образом.

```
app.request(localpart='/', method='GET', data=None,
            host='0.0.0.0:8080', headers=None, https=False)
```

Это означает, что вы можете передать данные URL-адреса в качестве первого параметра, а затем изменить метод запроса, а также посылаемые данные формы, включая адрес хоста и заголовки. Работа делается без использования фактического веб-сервера, поэтому вы можете выполнять проверку с помощью автоматизированных тестов, а также использовать свой браузер для тестирования запущенного сервера.

Для проверки ответов, получаемых от этой функции, используйте код `assert_response` из модуля `tests.tools`:

```
assert_response(resp, contains=None, matches=None, headers=None,
            status="200")
```

В ответ передаются данные, полученные при вызове `app.request`, а затем добавляются элементы, которые необходимо проверить. Используйте параметр `contains`, чтобы получить в ответе определенные значения. Используйте параметр `status` для проверки определенных ответов. На самом деле в этой маленькой функции используется довольно много данных, поэтому я рекомендую вам изучить ее.

В автоматизированном тесте *tests/app_tests.py* я сначала проверяю, что URL-адрес / возвращает ошибку `404 Not Found`, так как файл на самом деле не существует. Затем я проверяю, что параметр `/hello` работает как с GET, так и с POST методами веб-формы. Проверка должна быть достаточно простой, даже если вы полностью не понимаете, как она выполняется.

Постарайтесь как следует разобраться в работе последней версии приложения, особенно как работает автоматизированное тестирование. Поймите, как я импортировал приложение *bin/app.py* и запустил его непосредственно для автоматизированного тестирования. Это важный момент, который необходим для полноценного обучения.

Практические задания

1. Найдите и прочитайте дополнительный материал о языке HTML и попробуйте улучшить макет нашей простой веб-формы. Для удобства можно накидать эскиз на бумаге, а затем реализовать его с помощью HTML.

2. Это задание может показаться сложным, тем не менее попытайтесь создать форму загрузки файла, чтобы можно было выгрузить изображение на сервер и сохранить его на диск.

3. Это еще более крышесносящее занятие, но поищите документацию RFC по HTTP (она описывает, как работает HTTP) и прочитайте столько, сколько сможете. Это, правда, скучно, но обязательно пригодится.

4. Еще одно сложное занятие, но, возможно, кто-нибудь из ваших друзей сможет помочь вам создать веб-сервер, например Apache, Nginx или thttpd. Попробуйте выложить на него пару *.html* и *.css* файлов. Не беспокойтесь, если у вас ничего не получится. Веб-серверы — это неприятно.

5. Сделайте перерыв и попробуйте разработать столько различных веб-приложений, сколько сможете. Вы должны обязательно прочитать о сеансах в фреймворке *web.py* (это то же самое, что и *lpthw.web*), чтобы научиться сохранять статус пользователя.

Распространенные вопросы

Возникает ошибка `ImportError "No module named bin.app"`.

Повторюсь, это происходит потому, что вы перешли в неправильный каталог, не создали файл *bin/__init__.py* или не установили значение переменной `PYTHONPATH=.` в вашей оболочке командной строки. Всегда помните об этих способах решения проблемы, так как эти причины возникают очень часто, а ваши вопросы по поводу этой ошибки только замедляют работу.

При работе с шаблоном отображается сообщение `__template__() takes no arguments (1 given)`.

Вы, скорее всего, забыли добавить строку `$def with (greeting)` или аналогичное объявление переменной в верхней части кода шаблона.

УПРАЖНЕНИЕ 52

Онлайн-игра

Вот мы и добрались до конца книги, и в этом упражнении я собираюсь действительно бросить вам вызов.

Когда вы закончите, то будете достаточно компетентным начинающим программистом на языке Python. Вам понадобится прочитать еще несколько книг и разработать несколько проектов, но вы будете иметь все необходимые навыки, чтобы успешно их завершить. Для этого вам понадобятся время, мотивация и ресурсы.

В этом упражнении мы создадим не законченную игру, а «движок», который позволит запускать игру из упражнения 47 в браузере. Ваши действия будут включать переработку кода из упражнения 43, применение структуры из упражнения 47, добавление автоматизированных тестов и, наконец, разработку веб-движка, который позволит запускать игру.

Это упражнение огромное, и я предполагаю, что вы потратите от недели до нескольких месяцев на его выполнение, прежде чем закончите. Лучше всего разделить его на несколько небольших фрагментов и постепенно выполнять их, чтобы выполнить упражнение до конца.

Доработка игры из упражнения 43

Вы вносили изменения в проект *gothonweb* на протяжении двух последних упражнений, и займетесь тем же самым и в этом упражнении. Так вы нарабатываете навык под названием «рефакторинг», или, по-другому, «переработка кода». Так программисты именуют процесс использования старого кода, его изменения, для внедрения новых возможностей или его очистки. Вы занимались этим, не представляя себе, что это вторая природа разработки программного обеспечения.

На данном этапе вы возьмете идеи управляемой карты сцен из упражнения 47 и игру из упражнения 43, а затем объедините их вместе, чтобы создать новую игровую структуру. Она будет основана на том же контенте, просто «переработана», чтобы иметь более подходящую структуру.

Сначала возьмите код из файла *ex47/game.py* и скопируйте его в файл *gothonweb/map.py*, а файл *tests/ex47_tests.py* в *tests/map_tests.py*, после чего выполните команду `nosetests`, чтобы убедиться, что все работает.

Примечание. С этого момента я не буду приводить результат прогона теста. Предполагается, что вы самостоятельно сделали это и вернетесь к соответствующим упражнениям, если у вас возникнет ошибка.

После того как код из упражнения 47 скопирован, пришло время его переработать, чтобы внедрить карту из упражнения 43. Я начну с установки базовой структуры, и тогда вы сможете завершить файлы *map.py* и *map_tests.py*.

Разметьте базовую структуру карты с помощью класса `Room`, как показано ниже.

map.py

```
class Room(object):

    def __init__(self, name, description):
        self.name = name
        self.description = description
        self.paths = {}

    def go(self, direction):
        return self.paths.get(direction, None)

    def add_paths(self, paths):
        self.paths.update(paths)


central_corridor = Room("Central Corridor",
"""
Готоны с планеты Перкаль 25 захватили ваш корабль и уничтожили
всю команду. Вы — единственный, кто остался в живых.
Вам нужно выкрасть нейтронную бомбу в оружейной лаборатории,
заложить ее в топливном отсеке и покинуть корабль в спасательной
капсуле прежде, чем он взорвется.

Вы бежите по центральному коридору в оружейную лабораторию, когда
перед вами появляется Готон с красной чешуйчатой кожей, гнилыми
зубами и в костюме клоуна.
```

```
Он с ненавистью смотрит на вас и, преградив дорогу в лабораторию,
вытаскивает бластер, чтобы уничтожить вас.
""")


laser_weapon_armory = Room("Laser Weapon Armory",
"""
К счастью, вы знакомы с культурой Готонов и знаете, что их способно
рассмешить.
Вы рассказываете бородатый анекдот:
Неоколонии, изоморфно релятивные к мультиполосным гиперболическим
параболоидам.
Готон замирает, старается сдержать смех, а затем начинает
безудержно хохотать.
Пока он смеется, вы достаете бластер и стреляете Готону в голову.
Он падает, а вы перепрыгиваете его и бежите в оружейную лабораторию.

Вы вбегаете в оружейную лабораторию и начинаете обыскивать комнату,
спрятались ли тут другие Готоны. Стоит мертвая тишина.
Вы бежите в дальний угол комнаты и находите нейтронную бомбу
в защитном контейнере. На лицевой стороне контейнера расположена
панель с кнопками, и вам надо ввести правильный код, чтобы достать
бомбу.
Если вы 10 раз введете неправильный код, контейнер заблокируется
и вы не сможете достать бомбу. Код состоит из трех цифр.
""")


the_bridge = Room("The Bridge",
"""
Контейнер открывается со щелчком и выпускает сизый газ.
Вы вытаскиваете нейтронную бомбу и бежите в топливный отсек,
чтобы установить бомбу в нужном месте.

Вы вбегаете в топливный отсек с нейтронной бомбой и видите
пятерых Готонов, безуспешно пытающихся управлять
кораблем. Один уродливее другого, и все в клоунских
костюмах, как и Готон, убитый вами. Они не достают оружие,
так как видят бомбу у вас в руках и не хотят, чтобы
вы установили ее.
""")


escape_pod = Room("Escape Pod",
```

```
65  """
66  Вы указываете бластером на бомбу в ваших руках.
67  Готоны поднимают лапы вверх и в страхе потеют.
68  Вы осторожно, не отворачиваясь, подходите к двери и
69  аккуратно устанавливаете бомбу, держа Готонов под прицелом.
70  Вы запрыгиваете в шлюз и закрываете его ударом по кнопке,
71  а затем бластером расплавляете замок, чтобы Готоны не смогли
72  открыть дверь. Теперь вам нужно залезть в спасательную капсулу
73  и удрать с корабля к чертям собачьим.
74
75  Вы мчитесь по отсеку со спасательными капсулами. Некоторые из них
76  могут быть повреждены и взорвутся во время полета. Всего капсул
77  пять, и у вас нет времени, чтобы осматривать каждую из них
78  на отсутствие повреждений.
79  Задумавшись на секунду, вы решаете сесть в капсулу под
80  номером...
81  Какой номер вы выбираете?
82  """)
83
84
85  the_end_winner = Room("The End",
86  """
87  Вы запрыгиваете в капсулу номер 2 и нажимаете кнопку отстыковки.
88  Капсула вылетает в космическое пространство, а затем
89  отправляется к планете неподалеку. Вы смотрите в иллюминатор
    и видите, как ваш
90  корабль взрывается. Его осколки повреждают топливный отсек корабля
91  Готонов, и тот тоже разлетается в клочья.
92  Победа за вами!
93  """)
94
95
96  the_end_loser = Room("The End",
97  """
98  Вы запрыгиваете в капсулу со случайным номером и нажимаете кнопку
    отстыковки.
99  Капсула вылетает в космическое пространство, а затем
100 взрывается с яркой вспышкой и разбрасывая осколки.
101 Вы умираете.
102 """
103 )
104
105 escape_pod.add_paths([
106   '2': the_end_winner,
```

```
107    '*': the_end_loser
108 ])
109
110 generic_death = Room("death", "You died.") # Вы умерли
111
112 the_bridge.add_paths([
113    'throw the bomb': generic_death, # бросить бомбу
114    'slowly place the bomb': escape_pod # установить бомбу
115 ])
116
117 laser_weapon_armory.add_paths([
118    '0132': the_bridge,
119    '*': generic_death
120 ])
121
122 central_corridor.add_paths([
123    'shoot!': generic_death, # стрелять!
124    'dodge!': generic_death, # проскочить!
125    'tell a joke': laser_weapon_armory # пошутить!
126 ])
127
128 START = central_corridor
```

Обратите внимание, что есть несколько проблем, связанных с нашим классом `Room` и этой картой.

Мы должны поместить текст, который был в конструкциях `if-else`, *перед* входом на сцену как часть каждой сцены. Это означает, что вы не можете получить случайный доступ к карте, что не очень хорошо. Вы исправите эту проблему далее в этом упражнении.

В исходной игре выполнялся код, определяющий верный код контейнера с бомбой и номер исправной спасательной капсулы. В этом варианте игры мы просто укажем определенные значения по умолчанию и будем работать с ними, но позднее в практических заданиях вы вернетесь к этой проблеме.

Я создал код `generic_death`, выполняемый при любом неправильном решении в игре, который вы должны закончить самостоятельно. Вам нужно будет просмотреть код игры и добавить все исходы игрового процесса, а также все их проверить.

Я ввел новый тип перехода, помеченный "*", который будет использоваться в качестве «ловушки» для действий в движке.

После того как код игры написан, создайте новый автоматизированный тест в файле *tests/map_test.py*. Пример кода приведен выше.

map_tests.py

```
from nose.tools import *
from gothonweb.map import *

def test_room():
    gold = Room("GoldRoom",
        """This room has gold in it you can grab. There's a
        door to the north.""")
    assert_equal(gold.name, "GoldRoom")
    assert_equal(gold.paths, {})

def test_room_paths():
    center = Room("Center", "Test room in the center.")
    north = Room("North", "Test room in the north.")
    south = Room("South", "Test room in the south.")

    center.add_paths({'north': north, 'south': south})
    assert_equal(center.go('north'), north)
    assert_equal(center.go('south'), south)

def test_map():
    start = Room("Start", "You can go west and down a hole.")
    west = Room("Trees", "There are trees here, you can go east.")
    down = Room("Dungeon", "It's dark down here, you can go up.")

    start.add_paths({'west': west, 'down': down})
    west.add_paths({'east': start})
    down.add_paths({'up': start})

    assert_equal(start.go('west'), west)
    assert_equal(start.go('west').go('east'), start)
    assert_equal(start.go('down').go('up'), start)

def test_gothon_game_map():
    assert_equal(START.go('shoot!'), generic_death)
```

```
35      assert_equal(START.go('dodge!'), generic_death)
36
37      room = START.go('tell a joke')
38      assert_equal(room, laser_weapon_armory)
```

Ваша задача в этой части упражнения — закончить карту и разработать автоматизированный тест, поддерживающий локализованную версию игры и полностью проверяющий всю карту. Он должен включать в себя проверку всех объектов `generic_death`. Проверьте, чтобы тестирование выполнялось качественно и тест был как можно более полным. Это необходимо, так как далее мы будем вносить изменения в эту карту, а тест будет использоваться для проверки работоспособности.

Сеансы и отслеживание пользователей

В определенных позициях в вашем веб-приложении нужно будет отслеживать некоторую информацию и связывать ее с браузером пользователя. Всемирная паутина (из-за протокола HTTP) является «бессеансовой», что означает, что каждый отправляемый запрос не зависит от каких-либо других выполняемых запросов. Если вы запрашиваете страницу А, передаете данные, а затем переходите по ссылке на страницу Б, все данные, которые вы передали на странице А, попросту исчезают.

Решение этой проблемы заключается в создании небольшого хранилища данных (как правило, в базе данных или на диске), в котором используется уникальный номер для каждого браузера. Оно позволит отслеживать, что делает этот браузер. В простом фреймворке *lpthw.web* это реализуется довольно легко. Ниже приведен пример, демонстрирующий, как это делается.

session_sample.py

```
1   import web
2
3   web.config.debug = False
4
5   urls = (
6      "/count", "count",
7      "/reset", "reset"
8     )
```

```
app = web.application(urls, locals())
store = web.session.DiskStore('sessions')
session = web.session.Session(app, store, initializer={'count': 0})

class count:
    def GET(self):
        session.count += 1
        return str(session.count)

class reset:
    def GET(self):
        session.kill()
        return ""

if __name__ == "__main__":
    app.run()
```

Также вам понадобится создать каталог *sessions/*, в котором приложение разместит хранилище сеансов. Создайте его, запустите приложение и перейдите по ссылке `/count`. Обновляйте содержимое и наблюдайте за увеличением значения счетчика. Если вы закроете браузер, информация о вас будет *утеряна*, а это именно то, что нам нужно для игры. Существует способ сохранения информации после закрытия окна браузера, но в этом случае тестирование и разработка приложения существенно усложняются. Если затем перейти по ссылке `/reset` и вернуться к `/count`, счетчик будет сброшен, потому что вы прервали сеанс.

Постарайтесь разобраться в этом коде, чтобы понимать, как сеанс начинается со значением `count`, равным 0. Также изучите файлы в каталоге *sessions/*, открывая их. Ниже показан сеанс Python, в котором я открыл и расшифровал один из таких файлов.

```
>>> import pickle
>>> import base64
>>> base64.b64decode(open("sessions/XXXXX").read())
" (dp1\nS'count'\np2\nI1\nsS'ip'\np3\nV127.0.0.1\np4\
  nsS'session_id'\np5\nS'XXXX'\np6\ns."
>>>
>>> x = base64.b64decode(open("sessions/XXXXX").read())
>>>
```

```
>>> pickle.loads(x)
{'count': 1, 'ip': u'127.0.0.1', 'session_id': 'XXXXX'}
```

На самом деле, сеансы — это просто словари, которые записываются на диск с помощью библиотек `pickle` и `base64`. Существует множество способов хранения и управления сеансами, поэтому не так важно знать, как они работают. Это необходимо только для отладки или очистки сеанса.

Разработка движка

Итак, у вас теперь есть работающая карта игры и хороший модульный тест для нее. Теперь вам нужно разработать небольшой игровой движок: он станет запускать сцены, собирать данные, вводимые игроком, и отслеживать игровой процесс. Мы применим сеансы, которыми вы только что научились управлять, чтобы построить простой игровой движок со следующим функционалом.

1. Запуск новой игры для новых пользователей.
2. Предоставление пользователю сцены.
3. Прием пользовательского ввода.
4. Применение пользовательского ввода в игровом процессе.
5. Отображение результатов и поддержка работы игры до тех пор, пока пользователь не погибнет.

Для достижения нашей цели мы возьмем старый добрый файл *bin/app.py* и создадим полностью рабочий игровой движок, поддерживающий пользовательские сеансы. Также будут использованы простенькие HTML-файлы, а вам потребуется завершить данное задание. Ниже представлен код простого игрового движка.

app.py

```
1   import web
2   from gothonweb import map
3
4   urls = (
5      '/game', 'GameEngine',
6      '/', 'Index',
```

```
    )

app = web.application(urls, globals())

# небольшой трюк для поддержки сеансов в режиме отладки
if web.config.get('_session') is None:
    store = web.session.DiskStore('sessions')
    session = web.session.Session(app, store,
                initializer=['room': None])
    web.config._session = session
else:
    session = web.config._session

    render = web.template.render('templates/', base="layout")


class Index(object):
    def GET(self):
        # используется для "настройки" сеанса с начальными значениями
        session.room = map.START
        web.seeother("/game")


class GameEngine(object):

    def GET(self):
        if session.room:
            return render.show_room(room=session.room)
        else:
            # зачем это? нужно ли?
            return render.you_died()

def POST(self):
    form = web.input(action=None)

    # это баг, сможете его исправить?
    if session.room and form.action:
        session.room = session.room.go(form.action)

    web.seeother("/game")

if __name__ == "__main__":
    app.run()
```

Несмотря на наличие незнакомого кода в этом сценарии, удивительно, что целый движок веб-игры помещается в небольшом файле. Самый значимый «трюк» в этом сценарии — строки, предназначенные для работы с сеансами в отладочном режиме. В противном случае каждый раз при обновлении страницы игра бы переставала работать из-за завершения сеанса.

Перед тем как запускать файл *bin/app.py*, вам нужно изменить переменную окружения PYTHONPATH. Не знаете, что это такое? Переменная PYTHONPATH — это просто список имен каталогов, определяемых пользователем и системой, в которых располагаются файлы с программным кодом на языке Python. Для изменения используйте следующую команду.

```
export PYTHONPATH=$PYTHONPATH:.
```

В оболочке командной строки Windows PowerShell введите:

```
$env: PYTHONPATH = "$env: PYTHONPATH;."
```

Это нужно сделать лишь один раз, и вы уже делали ранее, но если вы получаете сообщение об ошибке импорта, то, вероятно, вы пропустили этот шаг или выполнили его неправильно.

Теперь нужно удалить файлы *templates/hello_form.html* и *templates/index.html* и создать два шаблона, упомянутых в приведенном выше коде. Первый из них — это *templates/show_room.html*.

show_room.html

```
$def with (room)

<h1> $room.name </h1>

<pre>
$room.description
</pre>

$if room.name == "death":
   <p><a href="/">Играть сначала?</a></p>
```

```
$else:
      <p>
      <form action="/game" method="POST">
        - <input type="text" name="action"> <input type="SUBMIT">
      </form>
      </p>
```

Это шаблон для отображения сцены в игровом процессе. Далее вам нужно сообщить пользователям о завершении игры, если они добрались до конца карты и завершили игровой процесс неудачно. Для этого используется файл *templates/you_died.html*.

you_died.html

```
<h1>Вы умерли!</h1>

<p>От вас осталась лишь кучка пыли.</p>
<p><a href="/">Играть сначала</a></p>
```

Поместив эти файлы на свои места, теперь вы должны выполнить следующие действия.

1. Запустите файл *tests/app_tests.py* и успешно протестируйте игру. Обратите внимание, что вы сможете совершить только пару щелчков мышью в игре, так как задействованы пользовательские сеансы.

2. Удалите файлы из каталога *sessions/* и убедитесь, что игра запустилась сначала.

3. Выполните команду `python bin/app.py` и проверьте игровой процесс.

Вы должны уметь обновлять игру и исправлять в ней баги, а также изменять на свой вкус игровые HTML-файлы и движок, реализуя новые возможности.

Ваш выпускной экзамен

Вы почувствовали, какое огромное количество информации на вас выплеснулось? Это прекрасно, но я хочу кое-что скорректировать, пока вы оттачиваете свои навыки. В завершение этого упражнения я предоставлю вам финальный набор заданий, которые вы можете выполнить на свое усмотрение. Вы увидите, что написанный вами код неидеален; это только первая версия программы. Теперь ваша задача состоит в том, чтобы сделать игру более полной, выполнив следующее.

1. Устраните все ошибки, о которых я упомянул в коде, а также те, которые вы найдете самостоятельно.

2. Улучшите автоматизированные тесты, чтобы более детально проверять приложения, и разработайте такой тест, который бы проверял приложение во время его работы.

3. Улучшите HTML-код.

4. Исследуйте систему авторизации и продумайте возможность регистрации игроков в приложении, чтобы пользователи могли авторизовываться и вести счет.

5. Доработайте карту игры, сделав ее как можно более крупной и функционально насыщенной.

6. Предоставьте пользователям справочную систему, которая позволит им спросить, что делать в той или иной сцене в игре.

7. Добавьте любые другие возможности, которые вам хочется реализовать в игре.

8. Создайте несколько «карт» — пусть пользователи выбирают игру, которую они хотят запустить. Ваш движок *bin/app.py* должен поддерживать возможность запуска любой предоставленной карты сцен, чтобы поддерживать несколько игр.

9. И, наконец, примените знания, полученные в упражнениях 48 и 49, чтобы улучшить обработку пользовательского ввода. У вас есть бóльшая часть необходимого кода; вам лишь нужно поработать над грамматикой и подключить все это к вашей веб-форме и `GameEngine`.

Удачи!

Распространенные вопросы

Я использую сеансы в моей игре и не могу проверить их с помощью `nosetests`.

Вам нужно прочитать о сеансах по адресу **webpy.org/cookbook/session_with_reloader**.

Возникает ошибка `ImportError`.

Неправильный каталог. Неверная версия Python. Значение переменной окружения `PYTHONPATH` не присвоено. Отсутствует файл *__init__.py*. Ошибка в коде `import`.

Дальнейшее обучение

Вы совсем еще не программист. Мне нравится думать об этой книге как о вашем руководстве для получения «черного пояса программиста». Вы знаете достаточно, чтобы изучить следующую книгу по программированию и справиться с нею на отлично. Эта книга учит вас получению нужных навыков и вниманию, необходимым, чтобы прочитать другие книги по языку Python и действительно чему-то научиться. Обучение даже может показаться более простым.

Я рекомендую вам изучить следующие проекты и попробовать использовать их в своей работе.

- Веб-фреймворк Django (**docs.djangoproject.com/en/1.10/**). Вы можете разрабатывать веб-приложения с помощью этого популярного веб-фреймворка (**djangoproject.com**).

- SciPy (**www.scipy.org**). Если вы занимаетесь научной или инженерной деятельностью, интересуетесь математикой и инструментарием Dexy (**dexy.it**), вы можете научиться создавать профессиональную документацию с помощью библиотеки SciPy.

- PyGame (**www.pygame.org/news.html**). Набор модулей, предназначенный для разработки игр с графикой и звуком.

- Pandas (**pandas.pydata.org**). Программная библиотека на языке Python для обработки и анализа данных.

- Natural Language Tool Kit (**nltk.org**). Инструментарий для анализа рукописного текста и разработки таких компонентов, как спам-фильтры и чат-боты.

- Requests (**docs.python-requests.org/en/latest/index.html**). Библиотека для управления HTTP-запросами.

- SimpleCV (**simplecv.org**). Инструмент программного захвата с веб-камеры для «прозрения» компьютера.

- Scrapy (**scrapy.org**). Фреймворк для сбора данных с веб-сайтов.

- Panda3D (**www.panda3d.org**). Игровой движок-фреймворк для трехмерной визуализации и разработки игр.

- Kivy (**kivy.org**). Графический фреймворк для разработки пользовательских интерфейсов для настольных и мобильных платформ.

- SciKit-Learn (**scikit-learn.org/stable**). Библиотека для машинного обучения.

- Ren'Py (**renpy.org).** Простой и гибкий движок для разработки визуальных интерактивных игр, подобных созданной в этой книге, но с графикой.

- Learn C the Hard Way (**c.learncodethehardway.org**). После того, как вы познакомитесь с Python, попробуйте изучить язык C и алгоритмы с помощью другой моей книги. Не торопитесь; язык C существенно отличается от Python, но отлично подходит для изучения.

Выберите один из представленных выше проектов и найдите учебники и документацию по выбранной теме. По мере обучения *вводите код вручную* и выполняйте. Именно так поступаю я. И все программисты поступают так. Прочитать документацию недостаточно, чтобы освоить программное обеспечение; вы должны пробовать работать с ним. После того как вы освоите учебник и любую другую найденную документацию, попробуйте разработать какой-нибудь проект. Все что угодно, даже если этот проект уже был кем-то написан. Просто разработайте что-нибудь.

Вероятно, написанный вами код пестрит ошибками. Это неплохо. Я совершаю ошибки каждый раз, когда начинаю изучать новый для меня язык программирования. Никто не застрахован от ошибок, даже опытные программисты. Если они уверяют в обратном, то они лгут.

Как изучить любой язык программирования

Теперь вы узнаете, как изучить большинство языков программирования, которыми вы можете заинтересоваться в будущем. Эта книга организована в соответствии с тем, как я и многие другие программисты изучаем новые языки. Ниже продемонстрирован процесс, которому я обычно следую.

1. Беру книгу или ищу какую-нибудь обзорную публикацию о языке, который хочу изучить.

2. Пролистываю книгу и набираю вручную весь код, а затем его выполняю.

3. Читаю книгу; изучаю, как работает код; делаю заметки.

4. Пишу на изучаемом языке несколько программ, похожих на те, что я разработал на других языках программирования.

5. Читаю код других разработчиков и пытаюсь скопировать их шаблоны.

С помощью этой книги я заставил вас пройти через этот процесс очень медленно и небольшими шажками. Другие книги организованы иначе, а это означает, что вы должны воспринимать материал так, как показал вам я. Для этого лучше всего пролистывать книгу и составлять список всех основных фрагментов кода. Превратите этот список в набор упражнений на основе каждой главы, а затем просто выполняйте их по порядку.

Описанный процесс также применим ко всем новым технологиям при условии, что по изучаемой теме есть книги. Если подходящей книги нет, придерживайтесь указанного выше процесса, используя документацию из Интернета или исходный код примеров для первоначального вхождения в тему.

Каждый новый язык, который вы осваиваете, повышает ваш уровень программирования, и чем больше вы узнаете, тем проще становится учиться. Освоив три или четыре языка, вы сможете изучать подобные языки за неделю. Теперь, когда вы изучили Python, вы сравнительно быстро можете освоить Ruby и JavaScript. Так происходит потому, что многие языки имеют схожие концепции, и как только вы изучите концепции одного языка, вам будет проще освоить их в другом.

Последнее, о чем нужно помнить при изучении нового языка: не будьте глупым туристом. Глупый турист тот, кто отправляется путешествовать в другую страну и жалуется, что местная еда отличается от привычной. «Почему я не могу купить вкусный гамбургер в этой дурацкой стране?!» Когда вы изучаете новый язык, учитывайте, что какие-то процедуры в ней выполняются не глупо, а просто иначе. Примите это, и вы легко сможете изучить его.

После изучения языка не будьте рабом чужих привычек. Иногда люди, изучающие программирование, делают просто идиотские вещи, потому что «все всегда делают так». Если вам больше по душе собственный стиль программирования, ваш стиль лучше и вы знаете, как другие программисты выполняют те же задачи, не стесняйтесь нарушать их правила, если это положительно влияет на разрабатываемое программное обеспечение.

Я обожаю изучать новые языки программирования. Я считаю себя программистом-«антропологом» и изучаю языки, на которых программисты «разговаривают друг с другом» с помощью компьютера. И мне это нравится! Возможно, я покажусь вам странным, так что просто учите языки программирования, потому что вам так хочется.

Наслаждайтесь! Это действительно будет интересно.

Совет бывалого программиста

Вы прочли эту книгу и решили заниматься программированием дальше. Возможно, это станет вашей профессией, а может быть, останется хобби. Вам нужны будут чьи-то советы, чтобы вы точно знали, что не сбились с пути, и могли получать максимум удовольствия от своего недавно обретенного увлечения. Я занимаюсь программированием уже давно. Настолько давно, что для меня это занятие стало невероятно скучным. Когда я писал эту книгу, я знал около 20 языков программирования и мог выучить новый язык за день — максимум за неделю, в зависимости от его сложности. И все же в итоге мне стало просто скучно, и все это перестало меня увлекать. Это не значит, что я считаю программирование скучным или что вы будете считать программирование скучным. Просто сейчас это занятие мне не кажется интересным. Во время своего долгого обучения я понял, что важны не сами языки, а то, что ты с ними делаешь. В сущности, я всегда это знал, но постоянно отвлекался на языки и забывал об этом. Теперь я об этом помню постоянно и советую помнить и вам. Не важно, какой язык программирования вы знаете и используете. Не идите на поводу у культа языков программирования, поскольку так вы не сможете разглядеть их реального предназначения — они просто ваш инструмент, помогающий делать что-то интересное.

Программирование как интеллектуальная деятельность — это единственный вид творчества, который позволяет создавать интерактивные предметы искусства. Вы можете создавать проекты, которыми будут играть другие люди, и вы можете косвенно общаться с ними. Ни один другой вид искусства не обладает подобной интерактивностью. Кино направлено лишь в сторону зрителя (и нет обратной связи). Живопись статична. Программа сочетает в себе все.

Программирование как профессию можно считать интересным лишь в самой небольшой степени. Это довольно неплохая работа, но можно зарабатывать примерно те же деньги и получать больше удовольствия, если держать, например, ресторан быстрого питания. Гораздо выгоднее иметь в своем арсенале программирование и применять его как секретное оружие в другой профессии.

Людей, которые умеют программировать, в мире технологий пруд пруди, их никто не ценит. Умеющих программировать людей в сфере биологии, медицины, управления, социологии, физики, истории и математики уважают, они могут использовать свои умения во благо этих сфер. Конечно, эти советы бессмысленны. Если вам понравилось учиться писать программы, попытайтесь направить это на улучшение своей жизни — любым доступным вам

способом. Вперед, исследуйте этот интересный, удивительный, новый интеллектуальный маршрут, который за последние 50 лет исследовали столь многие. Попробуйте и вы.

И наконец, я бы сказал, что, научившись создавать программы, вы изменитесь и станете другими — не лучше и не хуже, а просто другими. Возможно, люди вдруг станут обращаться с вами грубо из-за того, что вы можете создавать программы, например, будут называть вас «ботаном». Возможно, люди не захотят с вами спорить, потому что вы сможете разбить вдребезги все их аргументы. Или даже вы вдруг заметите, что их раздражает и настораживает в вас одно то, что вы знаете, как устроен компьютер.

Тут у меня всего один совет: пусть идут лесом. В мире должно быть больше странных людей, которые знают, как все устроено, и которым нравится это узнавать. Если к вам будут так относиться, просто помните, что это ваш путь, а не их. Отличаться от других — это не преступление, а люди, которые пытаются убедить вас в обратном, просто завидуют. Вы научились делать то, что они не смогут сделать даже в самом прекрасном сне.

Вы умеете программировать. Они нет. И это офигенно здорово.

Экспресс-курс по оболочке командной строки

Это приложение представляет собой супербыстрый курс по обучению работе с оболочкой командной строки. Обучение займет пару дней и не направлено на то, чтобы овладеть *всеми* секретами оболочки командной строки.

Введение в оболочку командной строки

Это приложение представляет собой ускоренный курс по обучению работе с оболочкой командной строки, в которой ваш компьютер с помощью команд сможет выполнять определенные задачи. Курс, разумеется, не так подробен и детализирован, как другие мои книги. Его цель — научить вас использовать компьютер, как это делают программисты. Когда вы дочитаете приложение, то сможете использовать большинство основных команд, с которыми программисты сталкиваются ежедневно. Вы разберетесь с каталогами и некоторыми другими концепциями.

Единственный совет, который я собираюсь дать вам, это:

```
Соберитесь и введите все описываемые команды!
```

Прошу простить, но вам действительно нужно это сделать. Если вам присущ иррациональный страх перед оболочкой командной строки, единственный способ победить его — это просто перебороть его.

Вы не сломаете свой компьютер. Вас не бросят в катакомбы под штаб-квартирой Microsoft в Редмонде. Ваши друзья не будут смеяться над вами, считая вас идиотом. Просто игнорируйте любые глупые и странные причины, по которым вы должны избегать оболочки командной строки.

Для чего это нужно? Потому что, если вы хотите научиться кодингу, вы должны уметь и обращаться с оболочкой командной строки. Языки программирования — продвинутые способы управления компьютером. Оболочка

командной строки — младший брат языков программирования. Изучая оболочку командной строки, вы учитесь контролировать работу компьютера с помощью языка программирования. Как только вы научитесь этому, сможете перейти к написанию кода и почувствуете, что на самом деле владеете куском металла, которым научились управлять.

Как использовать данное приложение

Рекомендуемый способ использования этого приложения заключается в следующем.

- Возьмите небольшой бумажный блокнот и ручку.
- Начните читать приложение с самого начала и выполняйте каждое упражнение точно так, как написано в книге.
- Если вы не поняли прочитанное, *сделайте пометку в блокноте.* Оставьте немного места, чтобы позднее написать решение.
- После того как вы выполнили упражнение, вернитесь к блокноту и изучите вопросы, которые записали. Попробуйте ответить на них, выполнив поиск в Интернете или спросив друзей, которые могут знать ответ. Отправьте мне электронное сообщение на адрес **help@learncodethehardway.org**, и я также попробую вам помочь.

Просто продолжайте двигаться дальше, выполняя упражнения, записывая возникающие вопросы, а затем возвращаясь и отвечая на них по возможности. К тому времени, как вы доберетесь до конца приложения, вы будете знать об использовании оболочки командной строки намного больше, чем предполагали.

Способы запомнить информацию

Стоит предупредить, что приведенную в приложении информацию нужно запоминать сразу. Это самый быстрый способ достичь результата, но у некоторых читателей может возникнуть страх и нежелание учить что-либо наизусть. Постарайтесь справиться с этими проблемами. Заучивание — один из важнейших навыков в обучении, поэтому вы должны справиться со страхом перед ним.

Вот как к этому следует подойти.

- Скажите себе, что вы *должны* сделать это. Не пытайтесь найти какие-либо упрощающие советы или обходные пути; просто сядьте и сделайте это.

- Запишите то, что следует запомнить, на нескольких карточках. Запишите половину нужной информации на одной стороне, а вторую половину — на другой.

- Каждый день в течение 15–30 минут тренируйте себя с помощью карточек, пытаясь вспомнить, что написано на каждой из них. Положите карточки, информацию на которых вы не запомнили, в отдельную стопку и тренируйтесь на них, пока не запомните. Затем смешайте все карточки и попробуйте вспомнить их все.

- Перед сном несколько минут тренируйтесь на тех карточках, которые не запомнили.

Существуют и другие способы. Например, вы можете записать информацию, которую нужно запомнить, на листе бумаги, заламинировать его и прикрепить на стену в ванной комнате. Во время душа, закрыв глаза, пытайтесь вспомнить информацию, поглядывая на лист при затруднении.

Если так поступать каждый день, вы сможете запомнить большинство команд, которые я описал в приложении, в течение недели, максимум месяца. После этого практически вся оставшаяся информация становится интуитивно понятной и ее заучивание упрощается. Цель не в запоминании абстрактных понятий, а скорее, во внедрении основ так, чтобы они были интуитивно понятны и вы не задумывались о них. После того как вы запомните эти основы, они перестанут мешать вам при изучении сложных абстрактных понятий.

Упражнение 1. Подготовка

Сейчас вы научитесь делать три вещи.

- Работать в оболочке командной строки (консоли, терминале, PowerShell).

- Понимать, что вы только что натворили.

- Выполнять дополнительные операции (на свое усмотрение).

В этом первом упражнении вы научитесь получать доступ к оболочке командной строки и запускать ее так, чтобы выполнять остальные упражнения из этого приложения.

Практикум

Далее вы научитесь получать доступ к вашей оболочке командной строки и разберетесь, как она работает.

macOS

В операционной системе macOS выполните следующие действия.

1. Нажав и удерживая клавишу ⌘, нажмите клавишу **Пробел**. На экране появится панель поиска синего цвета.

2. Введите слово **терминал** (terminal).

3. Щелкните мышью по значку приложения Терминал (Terminal) в виде черного окошка.

4. Откроется окно приложения Терминал (Terminal).

5. Перейдите к значку программы Терминал (Terminal) на панели **Dock** и, нажав и удерживая клавишу ^, щелкните мышью по ней, чтобы вызвать меню. Выберите в меню пункт **Параметры → Оставить в Dock** (Options → Keep in dock).

Теперь приложение Терминал (Terminal) запущено, а его значок доступен на панели **Dock** (Док), поэтому вы легко можете запустить его.

Linux

Предполагаю, что если вы пользуетесь операционной системой Linux, то уже знаете, как получить доступ к терминалу. Запустите менеджер окон и найдите приложение с названием вида Оболочка (Shell) или Терминал (Terminal).

Windows

В операционной системе Windows мы будем использовать средство PowerShell. Пользователи этой операционной системы привыкли работать

с программой под названием *cmd.exe*, но она работает не так, как PowerShell. Если вы пользуетесь операционной системой Windows 7 или более поздней версией, выполните следующие действия.

1. Нажмите кнопку **Пуск** (Start) или клавишу ⊞. В операционной системе Windows 10 нажмите кнопку в виде лупы в левой части панели задач.

2. В поле поиска начните ввод названия средства PowerShell.

3. Нажмите клавишу **Enter**.

Если вы пользуетесь операционной системой Windows версии ранее 7, вы должны серьезно рассмотреть вопрос обновления системы. Если обновление системы невозможно, скачайте средство PowerShell с сайта корпорации Microsoft. Выполните поиск во Всемирной паутине по запросу «PowerShell скачать» для вашей версии Windows. Дальнейшие действия по выполнению команд должны быть аналогичными, хотя я и не проверял ранние версии Windows XP.

Что вы изучили

Теперь вы знаете, как запустить оболочку командной строки, и можете выполнять остальные упражнения из этого приложения.

Примечание. Если у вас есть одаренный друг, пользующийся Linux, не обращайте внимания, если он рекомендует вам использовать приложение, отличное от Bash. Я учу людей пользоваться оболочкой Bash. Именно ею. Ваш друг может утверждать, что, используя zsh, вы станете гуру программирования и выиграете миллионы на фондовом рынке. Игнорируйте его. Ваша цель состоит в том, чтобы научиться пользоваться оболочкой командной строки, и на этом уровне не имеет значения, какую оболочку вы используете. Следующее предупреждение касается избегания IRC-каналов и прочих мест, где обитают «хакеры». Они считают забавным передать вам команды, которые могут вывести из строя ваш компьютер. Команда `rm -rf /` — классический пример того, что вы не должны набирать. Просто избегайте их. Если вам нужна помощь, убедитесь, что вы получаете ее от кого-то, кому доверяете, а не от незнакомых идиотов из Интернета.

Дополнительно

Это упражнение включает объемный дополнительный раздел. Остальные упражнения не так велики, и сейчас вы подготовитесь к остальной части приложения, заучив некоторые сведения. Доверьтесь мне, и дальнейшее обучение окажется более быстрым и простым.

Linux/macOS

Используя этот список команд, подготовьте картотеку с именами команд на одной стороне и их предназначением на другой. Заучивайте их каждый день, параллельно выполняя упражнения из этого приложения.

pwd	Вывести текущую директорию
hostname	Вывести сетевое имя компьютера
mkdir	Создать каталог
cd	Сменить каталог
ls	Вывести содержимое каталога
rmdir	Удалить каталог
pushd	Поместить каталог в стек
popd	Удалить каталог из стека
cp	Скопировать файл или каталог
mv	Переместить файл или каталог
less	Вывести постранично содержимое файла
cat	Вывести все содержимое файла
xargs	Выполнить аргументы
find	Искать файлы
grep	Искать внутри файлов
man	Вывести справочное руководство
apropos	Найти страницу справочного руководства
env	Просмотреть окружение
echo	Вывести определенные аргументы
export	Экспортировать/установить новую переменную окружения
exit	Выйти из оболочки
sudo	ОСТОРОЖНО! Установить права суперпользователя ОСТОРОЖНО!

Windows

Если вы пользуетесь операционной системой Windows, используйте следующий список команд.

pwd	Вывести текущую директорию
hostname	Вывести сетевое имя компьютера
mkdir	Создать каталог
cd	Сменить каталог
ls	Вывести содержимое каталога
rmdir	Удалить каталог
pushd	Поместить каталог в стек
popd	Удалить каталог из стека
cp	Скопировать файл или каталог
robocopy	Копировать с репликацией
mv	Переместить файл или каталог
more	Вывести постранично содержимое файла
type	Вывести все содержимое файла
forfiles	Обработать группу файлов
dir -r	Искать файлы
select-string	Искать внутри файлов
help	Вывести справочное руководство
helpctr	Найти страницу справочного руководства
echo	Вывести определенные аргументы
set	Экспортировать/установить новую переменную окружения
exit	Выйти из оболочки
runas	ОСТОРОЖНО! Установить права администратора

Учите, учите, учите! Учите, пока описание команды не будет отскакивать от зубов при виде ее названия. Затем разверните процесс: читайте описание команды и сразу вспоминайте ее имя. Так вы запомните их все.

Упражнение 2. Пути, папки и каталоги (pwd)

В этом упражнении вы узнаете, как отобразить (вывести) ваш рабочий каталог с помощью команды `pwd`.

Практикум

Я научу вас понимать «сеансы», которые продемонстрирую далее. Вам не нужно вводить всюду все, что показано здесь, только некоторые части.

- Вам не нужно вводить символы `$` (Unix) или `^gt;` (Windows). Это лишь демонстрация моего сеанса, чтобы вы могли увидеть результат действия команды.
- Вы вводите команды после символа `$` или >, а затем нажимаете клавишу **Enter**. Поэтому, если в моем примере указано `$ pwd`, вы просто набираете имя команды — `pwd` и нажимаете клавишу **Enter**.
- Вы увидите, что в моих примерах вывод сопровождается другими символами приглашения `$` или >. Так выглядит вывод (результат выполнения команды), и вы должны увидеть такой же результат.

Давайте попробуем выполнить простейшую команду, чтобы наглядно увидеть результат.

Сеанс упражнения 2

```
$ pwd
/Users/zedshaw
$
PS C:\Users\zed> pwd

Path
----
C:\Users\zed
PS C:\Users\zed>
```

Примечание. В этом приложении мне важно было сэкономить место на бумаге, поэтому мы сосредоточимся на важных деталях команд. Например, я удаляю первую часть приглашения (`PS C:\Users\zed` в примере выше) и оставляю только символ >. Таким образом, запрос на экране вашего компьютера будет выглядеть немного иначе, о чем не стоит волноваться. Запомните, что далее я буду указывать только символ >, обозначающий приглашение командной строки. Аналогично я делаю и для Unix-систем, но запросы в этих системах настолько разнообразны, что большинство людей привыкают, что символ $ обозначает приглашение.

Что вы изучили

Содержимое оболочки командной строки будет отличаться от моего. Вы увидите собственное имя пользователя перед символом `$`, как и имя вашего компьютера. В операционной системе Windows вид оболочки, вероятно, тоже будет выглядеть иначе. Суть в том, что вы видите следующее.

- Это приглашение командной строки (оболочки).
- Здесь вводится команда. В данном случае `pwd`.
- Выводится некий результат.

Повторите!

Вы только что узнали, для чего предназначена команда `pwd`, имя которой расшифровывается как print working directory — «вывести рабочий каталог». Что такое каталог? Это папка. «Папка» и «каталог» — одно и то же, и эти слова используются как синонимы. Когда вы запускаете графический файловый менеджер (например, Проводник Windows (Windows Explorer) или Finder) на вашем компьютере, вы можете перемещаться по папкам. Эти папки то же самое, что и «каталоги», с которыми мы собираемся работать.

Дополнительно

- Выполните команду `pwd` 20 раз и каждый раз говорите «вывести рабочий каталог».

- Запишите путь, который отображается в результате выполнения данной команды. Найдите его (папку по данному пути) с помощью графического файлового менеджера.

- Серьезно, выполните команду 20 раз и произнесите ее предназначение вслух. Не ворчите! Просто сделайте это.

Упражнение 3. Если вы заблудились

Когда вы выполняете все эти команды, то можете потеряться в дебрях каталогов. Вы можете не знать, где находитесь или где находится нужный файл, и не иметь ни малейшего представления, что делать дальше. Чтобы решить эту проблему, я собираюсь научить вас пользоваться командами, помогающими прояснить ситуацию.

Каждый раз, когда что-то идет не так, скорее всего, причина в том, что вы вводили команды, не зная, где находитесь. Решение — это выполнить команду для вывода текущего каталога. Так вы узнаете, где находитесь.

Следующим важным моментом является необходимость вернуться в безопасное место, ваш домашний каталог. Введите команду `cd ~` и вы перейдете в домашний каталог.

Таким образом, если вы заблудились, в любой момент введите:

```
pwd
cd ~
```

Первая команда, `pwd`, сообщает, где вы находитесь. Вторая команда, `cd ~`, перенаправляет вас в домашний каталог. Попробуйте еще раз.

Практикум

Прямо сейчас попробуйте определить, где вы находитесь, а затем вернитесь в домашний каталог, используя команды `pwd` и `cd ~`. Так вы сможете сориентироваться в иерархии каталогов.

Что вы изучили

Как вернуться в домашний каталог, если вы заблудились.

Упражнение 4. Создание каталога (mkdir)

В этом упражнении вы научитесь создавать новые каталоги (папки) с помощью команды `mkdir`.

Практикум

Запомните! Сначала нужно перейти в домашний каталог! Сделайте это, выполнив команду `pwd`, а затем `cd ~`, прежде чем выполнять это упражнение. Прежде чем выполнять любое упражнение из этого приложения, всегда переходите в домашний каталог!

Сеанс упражнения 4

```
$ pwd
$ cd ~
$ mkdir temp
$ mkdir temp/stuff
$ mkdir temp/stuff/things
$ mkdir -p temp/stuff/things/frank/joe/alex/john
$
```

Сеанс упражнения 4 в Windows

```
> pwd
> cd ~
> mkdir temp

Directory: C:\Users\zed

Mode                LastWriteTime         Length        Name
----                -------------         ------        ----
d----               12/17/2011 9:02 AM                  temp

> mkdir temp/stuff

Directory: C:\Users\zed\temp
```

```
Mode                 LastWriteTime         Length        Name
----                 -------------         ------        ----
d----                12/17/2011 9:02 AM                  stuff

> mkdir temp/stuff/things

    Directory: C:\Users\zed\temp\stuff

Mode                 LastWriteTime         Length        Name
----                 -------------         ------        ----
d----                12/17/2011 9:03 AM                  things

> mkdir temp/stuff/things/frank/joe/alex/john

    Directory: C:\Users\zed\temp\stuff\things\frank\joe\alex

Mode                 LastWriteTime         Length        Name
----                 -------------         ------        ----
d----                12/17/2011 9:03 AM                  john
```

Это — единственное упражнение, в коде которого я упомянул команды `pwd` и `cd ~`. В следующих упражнениях я их опустил, но вам следует их выполнять.

Что вы изучили

В коде команда `mkdir` встречается несколько раз. Всеми этими способами вы можете ее выполнить. Что она делает? Команда `mkdir` создает каталоги. Почему вы спрашиваете? Вы должны были создать свои карточки для заучивания и запомнить команды и их предназначение к этому времени. Если вы не знаете, что «команда `mkdir` создает каталоги», продолжайте тренироваться с карточками.

Что это значит «создать каталог»? Каталоги — это «папки». Это одно и то же. Код, который вы видите выше, позволяет создавать каталоги внутри других каталогов, вложенных в еще одни каталоги. Вся эта структура вложения называется «путем», позволяющим понять, что сначала следует каталог *temp*, затем каталог *stuff* и, наконец, каталог *things*. Это указатель для компьютера, определяющий, где в дереве папок (каталогов) расположены нужные вам данные. Все они составляют файловую систему на жестком диске вашего компьютера.

Примечание. В этом приложении я указываю символ / (называемый слешем или косой чертой) в качестве разделителя между папками для всех путей, так как в настоящее время они поддерживаются на всех компьютерах. Тем не менее, пользователи Windows должны учитывать, что также могут использовать символ \ (обратный слеш) в качестве разделителя. Некоторые пользователи Windows предпочитают использовать обратный слеш во всех командах, но это необязательно.

Дополнительно

- Понятие «пути» может сбить вас с толку на данном этапе, но не волнуйтесь. Мы будем работать с путями далее, и вы все поймете.
- Создайте еще 20 каталогов внутри каталога *temp* с различными уровнями вложенности. Посетите их с помощью файлового менеджера с графическим интерфейсом.
- Создайте каталог с пробелами в имени, поместив его в кавычки: `mkdir "My Folder"`.
- Если каталог с именем, который вы создаете, уже существует, вы увидите сообщение об ошибке. Используйте команду `cd` для перехода в другой рабочий каталог и попробуйте создать каталог в нем. В операционной системе Windows для этой цели отлично подойдет рабочий стол (каталог *desktop*).

Упражнение 5. Смена каталога (cd)

В этом упражнении вы узнаете, как перемещаться из одного каталога в другой с помощью команды `cd`.

Практикум

Я напомню вам основные инструкции, связанные с сеансами.

- Символ $ (Unix) или > (Windows) вводить не нужно.

- Вы сразу вводите команду, а затем нажимаете клавишу **Enter**. Если в моем примере указан код `$ cd temp`, вы вводите только `cd temp` и нажимаете клавишу **Enter**.

- Вывод отображается после того, как вы нажмете клавишу **Enter**, сразу после символа `$` или `>`.

- Сначала всегда переходите в домашний каталог! Выполняйте команду `pwd`, а затем `cd ~`, чтобы вернуться к исходной позиции.

Сеанс упражнения 5

```
$ cd temp
$ pwd
~/temp
$ cd stuff
$ pwd
~/temp/stuff
$ cd things
$ pwd
~/temp/stuff/things
$ cd frank/
$ pwd
~/temp/stuff/things/frank
$ cd joe/
$ pwd
~/temp/stuff/things/frank/joe
$ cd alex/
$ pwd
~/temp/stuff/things/frank/joe/alex
$ cd john/
$ pwd
~/temp/stuff/things/frank/joe/alex/john
$ cd ..
$ cd ..
$ pwd
~/temp/stuff/things/frank/joe
$ cd ..
$ cd ..
$ pwd
~/temp/stuff/things
$ cd ../../..
```

```
$ pwd
~/
$ cd temp/stuff/things/frank/joe/alex/john
$ pwd
~/temp/stuff/things/frank/joe/alex/john
$ cd ../../../../../../../
$ pwd
~/
$
```

Сеанс упражнения 5 в Windows

```
> cd temp
> pwd

Path
----
C:\Users\zed\temp

> cd stuff
> pwd

Path
----
C:\Users\zed\temp\stuff

> cd things
> pwd

Path
----
C:\Users\zed\temp\stuff\things

> cd frank
> pwd

Path
----
C:\Users\zed\temp\stuff\things\frank

> cd joe
> pwd

Path
```

```
----
C:\Users\zed\temp\stuff\things\frank\joe

> cd alex
> pwd

Path
----
C:\Users\zed\temp\stuff\things\frank\joe\alex

> cd john
> pwd

Path
----
C:\Users\zed\temp\stuff\things\frank\joe\alex\john

> cd ..
> cd ..
> cd ..
> pwd

Path
----
C:\Users\zed\temp\stuff\things\frank

> cd ../..
> pwd

Path
----
C:\Users\zed\temp\stuff

> cd ..
> cd ..
> cd temp/stuff/things/frank/joe/alex/john
> cd ../../../../../../../
> pwd

Path
----
C:\Users\zed

>
```

Что вы изучили

Все указанные каталоги вы создали в предыдущем упражнении, а в этом просто перемещались по ним с помощью команды `cd`. Как показано в сеансе выше, я также использую команду `pwd`, чтобы уточнить свое местонахождение, поскольку не помню его. Поэтому в листинге также приведен вывод команды `pwd`. Например, в строке 3 вы видите значение `~/temp`. Обратите внимание, это результат выполнения команды `pwd`. Не вводите его в своих сеансах!

Также вы должны понять, как я двигаюсь вверх по дереву каталогов.

Дополнительно

Очень важная часть понимания работы интерфейса командной строки (command line interface, CLI) на компьютере с графическим пользовательским интерфейсом (graphical user interface, GUI) заключается в изучении принципов их совместной работы. Когда я впервые познакомился с компьютером, не было никакого графического интерфейса и все операции выполнялись в среде DOS (с интерфейсом командной строки). Позже, когда компьютеры стали достаточно мощными, чтобы поддерживать графический интерфейс, мне было очень легко сопоставить каталоги из оболочки командной строки с окнами и папками графического интерфейса.

Сегодня большинство людей, однако, совершенно не понимают принципы работы оболочки командной строки, путей и каталогов. На самом деле, очень сложно научить их пользоваться интерфейсом командной строки, и единственный способ заключается в постоянной тренировке и выполнении команд, пока в один прекрасный день на пользователя не снизойдет озарение, как сопоставляются графический и командный интерфейсы.

Практиковаться можно, некоторое время перемещаясь по каталогам в графическом интерфейсе файлового менеджера, а затем обращаясь к ним через оболочку командной строки. Вот что следует выполнить.

- С помощью одной команды `cd` перейдите в каталог *joe*.
- С помощью одной команды `cd` вернитесь в каталог *temp*.
- Разберитесь, как с помощью одной команды `cd` перейти в домашний каталог.

- С помощью команды `cd` перейдите в каталог *Documents*, а затем найдите его с помощью графического файлового менеджера (Finder, Проводник (Windows Explorer) и т. д.).
- С помощью команды `cd` перейдите в каталог *Downloads*, а затем найдите его с помощью графического файлового менеджера.
- Выберите любой каталог в графическом файловом менеджере браузера, а затем перейдите к нему с помощью команды `cd`.
- Помните, что имена каталогов с пробелами следует помещать в кавычки. Это применимо к любой команде. Например, если у вас есть каталог *My Folder*, используйте команду `cd "My Folder"`.

Упражнение 6. Вывод содержимого каталога (`ls`)

В этом упражнении вы научитесь просматривать содержимое каталогов с помощью команды `ls`.

Практикум

Сначала убедитесь, что вы находитесь в каталоге уровнем выше папки `temp`. Если вы не знаете, где находитесь, выполните команду `pwd`, чтобы определить свое местонахождение, а затем перейдите в нужный каталог.

Сеанс упражнения 6

```
$ cd temp
$ ls
stuff
$ cd stuff
$ ls
things
$ cd things
$ ls
frank
$ cd frank
```

```
$ ls
joe
$ cd joe
$ ls
alex
$ cd alex
$ ls
$ cd john
$ ls
$ cd ..
$ ls
john
$ cd ../../../
$ ls
frank
$ cd ../../
$ ls
stuff
$
```

Сеанс упражнения 6 в Windows

```
> cd temp
> ls

    Directory: C:\Users\zed\temp

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----          12/17/2011 9:03 AM                stuff

> cd stuff
> ls

    Directory: C:\Users\zed\temp\stuff

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----          12/17/2011 9:03 AM                things

> cd things
> ls

Directory: C:\Users\zed\temp\stuff\things
```

```
Mode                 LastWriteTime         Length        Name
----                 -------------         ------        ----
d----                12/17/2011 9:03 AM                  frank

> cd frank
> ls

    Directory: C:\Users\zed\temp\stuff\things\frank

Mode                 LastWriteTime         Length        Name
----                 -------------         ------        ----
d----                12/17/2011 9:03 AM                  joe

> cd joe
> ls

    Directory: C:\Users\zed\temp\stuff\things\frank\joe

Mode                 LastWriteTime         Length        Name
----                 -------------         ------        ----
d----                12/17/2011 9:03 AM                  alex

> cd alex
> ls

    Directory: C:\Users\zed\temp\stuff\things\frank\joe\alex

Mode                 LastWriteTime         Length        Name
----                 -------------         ------        ----
d----                12/17/2011 9:03 AM                  john

> cd john
> ls
> cd ..
> ls

    Directory: C:\Users\zed\temp\stuff\things\frank\joe\alex

Mode                 LastWriteTime         Length        Name
----                 -------------         ------        ----
d----                12/17/2011 9:03 AM                  john
> cd ..
> ls
```

```
Directory: C:\Users\zed\temp\stuff\things\frank\joe

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----                12/17/2011 9:03 AM            alex

> cd ../../..
> ls

Directory: C:\Users\zed\temp\stuff

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----                12/17/2011 9:03 AM            things

> cd ..
> ls

Directory: C:\Users\zed\temp

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----                12/17/2011 9:03 AM            stuff

>
```

Что вы изучили

Выполнение команды `ls` позволяет вывести на экран содержимое каталога, в котором вы находитесь. Как видно из сеанса, я использую команду `cd` для перехода в различные каталоги, а затем вывожу их содержимое, чтобы определить имя каталога, в который перейду далее.

Команда `ls` поддерживает множество аргументов, и позднее вы узнаете о них, когда мы изучим команду `help`.

Дополнительно

- Введите каждую из команд, перечисленных в сеансе! Вы должны выполнить каждую из них, чтобы понять, как они работают. Просто

прочитать их вывод недостаточно. Я настойчиво рекомендую выполнить их!

- В среде Unix перейдите в каталог `temp` и выполните команду `ls -lR`.
- В операционной системе Windows добейтесь тех же результатов с помощью команды `dir -R`.
- Используйте команду `cd`, чтобы переходить в другие каталоги на вашем компьютере, а затем выполняйте команду `ls`, чтобы просмотреть их содержимое.
- Добавьте в свою записную книжку возникшие вопросы. Я уверен, что они возникли, поскольку даже я знаю не все об этой команде.
- Помните, что, если вы не можете определить свое местонахождение, используйте команды `ls` и `pwd`, чтобы решить эту проблему, а затем перейти в нужный каталог с помощью команды `cd`.

Упражнение 7. Удаление каталога (`rmdir`)

В этом упражнении вы научитесь удалять пустые каталоги.

Практикум

Сеанс упражнения 7

```
$ cd temp
$ ls
stuff
$ cd stuff/things/frank/joe/alex/john/
$ cd ..
$ rmdir john
$ cd ..
$ rmdir alex
$ cd ..
$ ls
joe
$ rmdir joe
$ cd ..
$ ls
```

```
frank
$ rmdir frank
$ cd ..
$ ls
things
$ rmdir things
$ cd ..
$ ls
stuff
$ rmdir stuff
$ pwd
~/temp
$
```

Внимание! Если вы выполняете команду `rmdir` в операционной системе macOS и операция завершается неудачей (каталог остается на месте), хотя вы уверены в том, что он пуст, это означает, что в каталоге находится файл с именем *.DS_Store*. В этом случае выполните команду `rm -rf <каталог>` (замените слово `<каталог>` именем удаляемого каталога).

Сеанс упражнения 7 в Windows

```
> cd temp
> ls

Directory: C:\Users\zed\temp

    Mode                 LastWriteTime         Length Name
    ----                 -------------         ------ ----
    d----           12/17/2011 9:03 AM                stuff

    > cd stuff/things/frank/joe/alex/john/
    > cd ..
    > rmdir john
    > cd ..
    > rmdir alex
    > cd ..
    > rmdir joe
    > cd ..
```

```
> rmdir frank
> cd ..
> ls

Directory: C:\Users\zed\temp\stuff

Mode                 LastWriteTime         Length         Name
----                 -------------         ------         ----
d----                12/17/20119:14 AM                    things

> rmdir things
> cd ..
> ls

Directory: C:\Users\zed\temp

Mode                 LastWriteTime         Length         Name
----                 -------------         ------         ----
d----                12/17/20119:14 AM                    stuff

> rmdir stuff
> pwd

Path
----
C:\Users\zed\temp

> cd ..
>
```

Что вы изучили

В этом сеансе команды перемешаны, поэтому убедитесь, что вы предельно внимательны и вводите их без ошибок. Каждая ошибка — это результат того, что вы были невнимательны. Если вы стали допускать много ошибок, сделайте перерыв или вообще закончите обучение на сегодня. Завтра вы вновь можете взяться за обучение.

В этом упражнении вы научились удалять каталоги. Это несложно. Просто перейдите в каталог уровнем выше, а затем выполните команду `rmdir <каталог>` (замените слово `<каталог>` именем удаляемого каталога), чтобы удалить указанный каталог.

Дополнительно

- Создайте не менее 20 каталогов и удалите их все.
- Создайте каталог с глубиной вложения не менее 10 уровней и удаляйте их по одному, как я продемонстрировал выше.
- При попытке удалить каталог с содержимым вы получите сообщение об ошибке. Я расскажу, как удалять такие каталоги, в последующих упражнениях.

Упражнение 8. Работа со стеком (pushd, popd)

В этом упражнении вы узнаете, как с помощью команды `pushd` сохранить ваше текущее местоположение в стек и перейти в новое расположение. После этого вы узнаете, как вернуться к сохраненному расположению (извлечь его из стека) с помощью команды `popd`.

Практикум

Сеанс упражнения 8

```
$ cd temp
$ mkdir -p i/like/icecream
$ pushd i/like/icecream
~/temp/i/like/icecream ~/temp
$ popd
~/temp
$ pwd
~/temp
$ pushd i/like
~/temp/i/like ~/temp
$ pwd
~/temp/i/like
$ pushd icecream
~/temp/i/like/icecream ~/temp/i/like ~/temp
$ pwd
~/temp/i/like/icecream
$ popd
~/temp/i/like ~/temp
$ pwd
```

```
~/temp/i/like
$ popd
~/temp
$ pushd i/like/icecream
~/temp/i/like/icecream ~/temp
$ pushd
~/temp ~/temp/i/like/icecream
$ pwd
~/temp
$ pushd
~/temp/i/like/icecream ~/temp
$ pwd
~/temp/i/like/icecream
$
```

Сеанс упражнения 8 в Windows

```
> cd temp
> mkdir -p i/like/icecream

    Directory: C:\Users\zed\temp\i\like

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----           12/20/201111:05 AM                icecream

> pushd i/like/icecream
> popd
> pwd

Path
----
C:\Users\zed\temp

> pushd i/like
> pwd

Path
----
C:\Users\zed\temp\i\like

> pushd icecream
```

```
> pwd
Path
----
C:\Users\zed\temp\i\like\icecream

> popd
> pwd

Path
----
C:\Users\zed\temp\i\like

> popd
>
```

Что вы изучили

Изучая эти команды, вы вторгаетесь во владения программистов, но команды настолько удобны, что я просто обязан научить вас пользоваться ими. Эти команды позволяют временно перейти в другой каталог, а затем вернуться обратно, легко переключаясь между ними.

Команда `pushd` помещает текущий каталог в стек для хранения и *перемещает* пользователя в указанный им другой каталог. Вы словно говорите компьютеру: «Сохрани мое текущее расположение и перейди в папку...».

Команда `popd` возвращает вас в каталог, помещенный в стек последним.

Кроме того, если в среде Unix запустить команду `pushd` без аргументов, вы сможете переключаться между текущим каталогом и каталогом, помещенным в стек последним. Это простой способ переключаться между двумя каталогами. К сожалению, он не работает в PowerShell.

Дополнительно

- Используйте описанные команды для перемещения по каталогам на вашем компьютере.
- Удалите каталоги *i/like/icecream* и создайте собственные, а затем перемещайтесь между ними.

- Проанализируйте вывод (результат работы) команд `pushd` и `popd`. Обратите внимание на то, что принцип их работы похож на работу стека.

- Как вы уже знаете, команда `mkdir -p` создает все каталоги указанного пути, даже если они не существуют. Ее я и выполнил первым делом в этом упражнении.

Упражнение 9. Создание пустых файлов (`touch`, `New-Item`)

В этом упражнении, вы узнаете, как сделать пустой файл с помощью команды `touch` (`new-item` в операционной системе Windows).

Практикум

Сеанс упражнения 9

```
$ cd temp
$ touch iamcool.txt
$ ls
iamcool.txt
$
```

Сеанс упражнения 9 в Windows

```
> cd temp
> New-Item iamcool.txt -type file
> ls

Directory: C:\Users\zed\temp

Mode                LastWriteTime         Length          Name
----                -------------         ------          ----
-a---               12/17/20119:03 AM                iamcool.txt

>
```

Что вы изучили

Вы узнали, как создавать пустые файлы. В среде Unix команда `touch` также изменяет дату и время файла. Я редко использую эту команду для других целей, кроме как создания пустых файлов. В операционной системе Windows используется другая команда, `New-Item`, которая работает аналогично, но позволяет также создавать новые каталоги.

Дополнительно

- Unix. Создайте каталог, перейдите в него, а затем создайте в нем пустой файл. Далее перейдите на один уровень вверх и выполните команду `rmdir` в текущем каталоге. Вы должны получить сообщение об ошибке. Разберитесь, почему возникла эта ошибка.

- Windows. Выполните то же самое, но сообщение об ошибке не появится. Вы увидите запрос подтверждения, действительно ли вы хотите удалить каталог.

Упражнение 10. Копирование файла (cp)

В этом упражнении вы узнаете, как скопировать файл из одного каталога в другой с помощью команды `cp`.

Практикум

Сеанс упражнения 10

```
$ cd temp
$ cp iamcool.txt neat.txt
$ ls
iamcool.txt neat.txt
$ cp neat.txt awesome.txt
$ ls
awesome.txt iamcool.txt neat.txt
$ cp awesome.txt thefourthfile.txt
$ ls
awesome.txt iamcool.txt neat.txt thefourthfile.txt
```

```
$ mkdir something
$ cp awesome.txt something/
$ ls
awesome.txt iamcool.txt neat.txt something thefourthfile.txt
$ ls something/
awesome.txt
$ cp -r something newplace
$ ls newplace/
awesome.txt
$
```

Сеанс упражнения 10 в Windows

```
> cd temp
> cp iamcool.txt neat.txt
> ls

Directory: C:\Users\zed\temp

Mode                 LastWriteTime         Length        Name
----                 -------------         ------        ----
-a---          12/22/2011  4:49 PM              0        iamcool.txt
-a---          12/22/2011  4:49 PM              0        neat.txt

> cp neat.txt awesome.txt
> ls

Directory: C:\Users\zed\temp

Mode                 LastWriteTime         Length        Name
----                 -------------         ------        ----
-a---          12/22/2011  4:49 PM              0        awesome.txt
-a---          12/22/2011  4:49 PM              0        iamcool.txt
-a---          12/22/2011  4:49 PM              0        neat.txt

> cp awesome.txt thefourthfile.txt
> ls

Directory: C:\Users\zed\temp

Mode                 LastWriteTime         Length        Name
----                 -------------         ------        ----
-a---          12/22/2011  4:49 PM              0        awesome.txt
```

```
-a---          12/22/2011  4:49 PM      0              iamcool.txt
-a---          12/22/2011  4:49 PM      0              neat.txt
-a---          12/22/2011  4:49 PM      0          thefourthfile.txt

> mkdir something

    Directory: C:\Users\zed\temp

Mode                 LastWriteTime         Length         Name
----                 -------------         ------         ----
d----          12/22/2011  4:52 PM                     something

> cp awesome.txt something/
> ls

    Directory: C:\Users\zed\temp

Mode                 LastWriteTime         Length         Name
----                 -------------         ------         ----
d----          12/22/2011  4:52 PM                     something
-a---          12/22/2011  4:49 PM      0              awesome.txt
-a---          12/22/2011  4:49 PM      0              iamcool.txt
-a---          12/22/2011  4:49 PM      0              neat.txt
-a---          12/22/2011  4:49 PM      0          thefourthfile.txt

> ls something

    Directory: C:\Users\zed\temp\something

Mode                 LastWriteTime         Length         Name
----                 -------------         ------         ----
-a---          12/22/2011  4:49 PM      0              awesome.txt

> cp -recurse something newplace
> ls newplace

    Directory: C:\Users\zed\temp\newplace

Mode                 LastWriteTime         Length         Name
----                 -------------         ------         ----
-a---          12/22/2011  4:49 PM      0              awesome.txt

>
```

Что вы изучили

Теперь вы умеете копировать файлы. Это просто: берется файл и копируется в новое расположение. В этом упражнении я также создал новый каталог и скопировал файл в него.

Я расскажу вам кое-что о программистах и системных администраторах. Они ленивые. И я ленивый. Мои друзья ленивы. Вот почему мы используем компьютеры. Мы хотим, чтобы компьютеры делали скучные вещи за нас. В этих упражнениях вы до сих пор набирали повторяющиеся скучные команды, которые уже вызубрили, но программисты обычно так не поступают. Как правило, если они обнаруживают что-то скучное и повторяющееся, то задумываются о том, как упростить задачу. Вы просто не знаете об этом.

Также отмечу, что программисты не так умны, как вы думаете. Если вы пытаетесь вспомнить имя команды, прежде чем ее ввести, то это не совсем правильно. Вместо этого попытайтесь себе представить, что она делает. Скорее всего, ее имя (аббревиатура) ассоциируется с тем действием, которое вам нужно выполнить. Если вы до сих пор не поняли это, поспрашивайте знакомых или выполните поиск во Всемирной паутине. За редким исключением, таким как `robocopy`, имена команд обозначают действия, которые они выполняют.

Дополнительно

- Скопируйте несколько каталогов с файлами, используя команду `cp -r`.
- Скопируйте файл в домашний каталог или на рабочий стол.
- Найдите скопированные файлы в графическом интерфейсе и откройте их в текстовом редакторе.
- Обратите внимание на то, что в некоторых случаях я указал символ / (слеш) после имени каталога. Так я указываю, что указанное имя обозначает каталог, и поэтому, если каталог не существует, отображается сообщение об ошибке.

Упражнение 11. Перемещение файла (mv)

В этом упражнении вы узнаете, как перемещать файлы из одного каталога в другой, используя команду mv.

Практикум

Сеанс упражнения 11

```
$ cd temp
$ mv awesome.txt uncool.txt
$ ls
newplace uncool.txt
$ mv newplace oldplace
$ ls
oldplace uncool.txt
$ mv oldplace newplace
$ ls
newplace uncool.txt
$
```

Сеанс упражнения 11 в Windows

```
> cd temp
> mv awesome.txt uncool.txt
> ls
    Directory: C:\Users\zed\temp

Mode                 LastWriteTime         Length        Name
----                 -------------         ------        ----
d----          12/22/2011  4:52 PM                      newplace
d----          12/22/2011  4:52 PM                      something
-a---          12/22/2011  4:49 PM      0               iamcool.txt
-a---          12/22/2011  4:49 PM      0               neat.txt
-a---          12/22/2011  4:49 PM      0          thefourthfile.txt
-a---          12/22/2011  4:49 PM      0               uncool.txt

> mv newplace oldplace
> ls

    Directory: C:\Users\zed\temp
```

```
Mode                 LastWriteTime         Length        Name
----                 -------------         ------        ----
d----                12/22/2011 4:52 PM                  oldplace
d----                12/22/2011 4:52 PM                  something
-a---                12/22/2011 4:49 PM      0           iamcool.txt
-a---                12/22/2011 4:49 PM      0           neat.txt
-a---                12/22/2011 4:49 PM      0           thefourthfile.txt
-a---                12/22/2011 4:49 PM      0           uncool.txt

> mv oldplace newplace
> ls newplace

    Directory: C:\Users\zed\temp\newplace

Mode                 LastWriteTime         Length        Name
----                 -------------         ------        ----
-a---                12/22/2011 4:49 PM      0           awesome.txt

> ls

    Directory: C:\Users\zed\temp

Mode                 LastWriteTime         Length        Name
----                 -------------         ------        ----
d----                12/22/2011 4:52 PM                  newplace
d----                12/22/2011 4:52 PM                  something
-a---                12/22/2011 4:49 PM      0           iamcool.txt
-a---                12/22/2011 4:49 PM      0           neat.txt
-a---                12/22/2011 4:49 PM      0           thefourthfile.txt
-a---                12/22/2011 4:49 PM      0           uncool.txt

>
```

Что вы изучили

Вы занимались перемещением файлов или, что точнее, их переименованием. Это просто: взять старое имя и назначить новое.

Дополнительно

- Переместите файл из каталога *newplace* в другой каталог, а затем верните его обратно.

Упражнение 12. Просмотр файла (`less`, `more`)

Для выполнения упражнения понадобится некоторая подготовка. Вам также понадобится текстовый редактор, который позволяет создавать и сохранять текстовые файлы (с расширением.*txt*). Выполните следующие действия.

Запустите текстовый редактор и напечатайте любой текст в созданном файле. В операционной системе macOS это может быть программа TextWrangler. В системе Windows — Блокнот (Notepad), а в Linux — Gedit. Любой текстовый редактор подойдет.

Сохраните файл на рабочем столе под именем *test.txt*.

Используйте в оболочке командной строки соответствующие команды, чтобы скопировать созданный файл в каталог *temp*, в котором вы работаете.

После этого можно приступать к выполнению упражнения.

Практикум

Сеанс упражнения 12

```
$ less test.txt
[здесь отображается содержимое файла]
$
```

Вот и все. Чтобы завершить работу команды `less`, введите букву `q` (от слова quit — выход).

Сеанс упражнения 12 в Windows

```
> more test.txt
[здесь отображается содержимое файла]
>
```

Примечание. Обратите внимание, что, чтобы вместо «кракозябр» отображался кириллический текст, файл необходимо сохранять в кодировке UTF-8 (выбирается в соответствующем раскрывающемся списке в диалоговом окне сохранения).

Примечание. В приведенном выше выводе команды строкой `[здесь отображается содержимое файла]` я сократил содержимое файла, которое там отобразилось. В вашем случае на месте этой строки отобразится текст, который вы напечатали в файле *test.txt*.

Что вы изучили

Перед вами один из способов просмотреть содержимое файла. Он полезен, поскольку, если файл содержит объемный текст, будет видна только одна «страница», которая вмещается в размеры экрана. В разделе «Дополнительно» далее вы еще немного потренируетесь с этой командой.

Дополнительно

- Откройте текстовый файл в редакторе и скопируйте/вставьте текст так, чтобы он занимал не менее 50–100 строк.
- Сохраните файл и скопируйте его в каталог *temp*, чтобы получить к нему доступ из оболочки командной строки.
- Выполните упражнение снова, но теперь вы увидите только часть текста. В среде Unix нажимайте клавишу **Пробел** или **W**, чтобы перемещаться по страницам. Клавиши со стрелками также можно использовать. В операционной системе Windows нажимайте клавишу **Пробел**, чтобы листать содержимое файла.
- Загляните в некоторые пустые файлы, созданные вами ранее.
- Команда `cp` перезаписывает файлы, если они существуют, поэтому будьте осторожны при копировании файлов.

Упражнение 13. Вывод содержимого файла (cat)

Вновь потребуется некоторая подготовка в плане создания файлов в одной программе и получения к ним доступа из командной строки. В том же текстовом редакторе, что и в предыдущем упражнении, создайте еще один файл с именем *test2.txt*, но на этот раз сохраните его непосредственно в каталог *temp*.

Практикум

Сеанс упражнения 13

```
$ less test2.txt
[здесь отображается содержимое файла]
$ cat test2.txt
Это глазки, чтобы видеть.
Это носик, чтоб дышать.
Это ушки, чтобы слышать.
Это ножки, чтоб бежать.
$ cat test.txt
Это ручки, чтобы маму
Очень крепко обнимать.
$
```

Сеанс упражнения 13 в Windows

```
> more test2.txt
[здесь отображается содержимое файла]
> cat test2.txt
Это глазки, чтобы видеть.
Это носик, чтоб дышать.
Это ушки, чтобы слышать.
Это ножки, чтоб бежать.
> cat test.txt
Это ручки, чтобы маму
Очень крепко обнимать.
>
```

Повторюсь, что в приведенном выше выводе команды строкой *[здесь отображается содержимое файла]* я сократил содержимое файла, которое там отобразилось.

Что вы изучили

Прекрасное стихотворение, не правда ли? Вернемся к делу.

Вы уже знаете первую команду, и с помощью нее я просто убедился, что файл на месте. Затем я выполнил команду `cat`. Она просто отображает все содержимое файла на экране без разбивки на страницы или остановки. Чтобы увидеть результат ее работы, попрактикуйтесь с файлом *test.txt* из прошлого упражнения, в который вы добавили множество строк текста.

Дополнительно

Создайте еще несколько текстовых файлов и потренируйтесь с командой `cat`.

Попробуйте выполнить команду `cat test.txt test2.txt` и посмотрите, что произойдет.

Попробуйте выполнить команду `cat test.txt, test2.txt` и посмотрите, что произойдет.

Упражнение 14. Удаление файла (`rm`)

В этом упражнении вы научитесь удалять (стирать) файлы с помощью команды `rm`.

Практикум

Сеанс упражнения 14

```
$ cd temp
$ ls
uncool.txt iamcool.txt neat.txt something thefourthfile.txt
$ rm uncool.txt
$ ls
iamcool.txt neat.txt something thefourthfile.txt
$ rm iamcool.txt neat.txt thefourthfile.txt
$ ls
something
$ cp -r something newplace
$
```

```
$ rm something/awesome.txt
$ rmdir something
$ rm -rf newplace
$ ls
$
```

Сеанс упражнения 14 в Windows

```
> cd temp
> ls

Directory: C:\Users\zed\temp

Mode                 LastWriteTime         Length         Name
----                 -------------         ------         ----
d----                12/22/20114:52 PM                    newplace
d----                12/22/20114:52 PM                    something
-a---                12/22/20114:49 PM      0             iamcool.txt
-a---                12/22/20114:49 PM      0             neat.txt
-a---                12/22/20114:49 PM      0             thefourthfile.txt
-a---                12/22/20114:49 PM      0             uncool.txt

> rm uncool.txt
> ls

Directory: C:\Users\zed\temp

Mode                 LastWriteTime         Length         Name
----                 -------------         ------         ----
d----                12/22/20114:52 PM                    newplace
d----                12/22/20114:52 PM                    something
-a---                12/22/20114:49 PM      0             iamcool.txt
-a---                12/22/20114:49 PM      0             neat.txt
-a---                12/22/20114:49 PM      0             thefourthfile.txt

> rm iamcool.txt
> rm neat.txt
> rm thefourthfile.txt
> ls

Directory: C:\Users\zed\temp

Mode                 LastWriteTime         Length         Name
----                 -------------         ------         ----
d----                12/22/20114:52 PM                    newplace
```

```
d----              12/22/2011 4:52 PM                something

> cp -r something newplace
> rm something/awesome.txt
> rmdir something
> rm -r newplace
> ls

>
```

Что вы изучили

Вы научились удалять файлы. Помните, что произошло, когда я попытался выполнить команду `rmdir` на каталоге с содержимым? Попытка провалилась, поскольку нельзя удалять каталоги с файлами внутри. Для удаления каталога сначала нужно удалить файл (файлы) или рекурсивно удалить все его содержимое. Это то, что вы сделали в самом конце упражнения.

Дополнительно

- Удалите все содержимое каталога *temp*, созданное вами на всех предыдущих упражнениях.
- Пометьте в записной книжке, что нужно быть осторожным при рекурсивном удалении файлов.

Упражнение 15. Выход из оболочки (`exit`)

Практикум

Сеанс упражнения 15

```
$ exit
```

Сеанс упражнения 15 в Windows

```
> exit
```

Что вы изучили

Последнее упражнение посвящено завершению работы с оболочкой командной строки. Это очень простое действие, но я также рекомендую вам изучить раздел «Дополнительно».

Дополнительно

В качестве последнего набора упражнений я предлагаю вам вооружиться своей записной книжкой и разобраться со следующими командами.

Команды для Unix:

- `xargs`
- `sudo`
- `chmod`
- `chown`

Команды для Windows:

- `forfiles`
- `runas`
- `attrib`
- `icacls`

Узнайте, для чего предназначены эти команды, потренируйтесь с ними, а затем добавьте их на карточки для заучивания.

Дальнейшее обучение

Вот экспресс-курс и пройден. На данный момент вы должны довольно сносно управлять файлами с помощью оболочки командной строки. Разумеется,

огромное количество команд и возможностей вам еще неизвестно, поэтому я рекомендую несколько достойных источников информации.

Руководства по Unix Bash

Оболочка командной строки, который вы пользовались, называется Bash. Это не лучшая оболочка, но она широко распространена и обладает множеством функций, поэтому отлично подойдет в качестве отправной точки. Ниже представлен краткий список ресурсов по теме.

Шпаргалка по Bash, **learncodethehardway.org/unix/bash_cheat_sheet.pdf**. Документ создан Рафаэлем и распространяется по свободной лицензии Creative Commons.

Руководство пользователя Bash, **www.gnu.org/software/bash/manual/bashref.html.**

Руководства по PowerShell

В операционной системе Windows используется только средство PowerShell. Ниже представлен краткий список ресурсов по PowerShell.

- Официальное руководство пользователя, **technet.microsoft.com/en-us/library/ee221100.aspx**.
- Руководство на русском языке, **technet.microsoft.com/ru-RU/library/bb978526**.
- Шпаргалка по PowerShell, **www.microsoft.com/en-us/download/details.aspx?id=30002**.
- Блог по PowerShell, **powershell.com/cs/blogs/ebook/default.aspx**.

Предметный указатель

Производственно-практическое издание

МИРОВОЙ КОМПЬЮТЕРНЫЙ БЕСТСЕЛЛЕР

Зед Шоу

ЛЕГКИЙ СПОСОБ ВЫУЧИТЬ PYTHON

Директор редакции *Е. Капьев*
Ответственный редактор *Е. Истомина*
Художественный редактор *А. Шуклин*

В оформлении обложки использована иллюстрация:
photovs / Shutterstock.com
Используется по лицензии от Shutterstock.com

ООО «Издательство «Э»
123308, Москва, ул. Зорге, д 1. Тел. 8 (495) 411-68-86

Өндіруші: «Э» АҚБ Баспасы, 123308, Мәскеу, Ресей, Зорге көшесі, 1 үй
Тел 8 (495) 411-68-86
Тауар белгісі «Э»
Қазақстан Республикасында дистрибьютор және өнім бойынша арыз-талаптарды қабылдаушының өкілі «РДЦ-Алматы» ЖШС, Алматы қ , Домбровский көш , 3«а», литер Б, офис 1
Тел 8 (727) 251-59-89/90/91/92, факс 8 (727) 251 58 12 вн 107
Өнімнің жарамдылық мерзімі шектелмеген
Сертификация туралы ақпарат сайтта Өндіруші «Э»

Сведения о подтверждении соответствия издания согласно законодательству РФ о техническом регулировании можно получить на сайте Издательства «Э»

Өндірген мемлекет Ресей
Сертификация қарастырылмаған

Подписано в печать 12.04.2017. Формат 70x100$^{1}/_{16}$.
Печать офсетная. Усл. печ. л. 28,52.
Тираж 2000 экз. Заказ № 5481.

Отпечатано в ОАО «Можайский полиграфический комбинат»
143200, г. Можайск, ул Мира, 93
www oaompk.ru, www ОАОМПК рф тел (495) 745-84-28, (49638) 20-685

12+

ISBN 978-5-699-98251-6

Оптовая торговля книгами Издательства «Э»:
142700, Московская обл , Ленинский р-н, г. Видное,
Белокаменное ш , д 1, многоканальный тел · 411-50-74

По вопросам приобретения книг Издательства «Э» зарубежными оптовыми покупателями обращаться в отдел зарубежных продаж
International Sales: International wholesale customers should contact Foreign Sales Department for their orders

По вопросам заказа книг корпоративным клиентам, в том числе в специальном оформлении, *обращаться по тел +7 (495) 411-68-59, доб 2261.*

Оптовая торговля бумажно-беловыми и канцелярскими товарами для школы и офиса:
142702, Московская обл., Ленинский р-н, г. Видное-2,
Белокаменное ш , д 1, а/я 5 Тел./факс +7 (495) 745-28-87 (многоканальный)

Полный ассортимент книг издательства для оптовых покупателей
Москва Адрес. 142701, Московская область, Ленинский р-н,
г. Видное, Белокаменное шоссе, д 1 Телефон +7 (495) 411-50-74
Нижний Новгород. Филиал в Нижнем Новгороде Адрес 603094,
г. Нижний Новгород, улица Карпинского, дом 29, бизнес-парк «Грин Плаза».
Телефон +7 (831) 216-15-91 (92, 93, 94)
Санкт-Петербург. ООО «СЗКО» Адрес 192029, г. Санкт-Петербург, пр Обуховской Обороны,
д. 84, лит. «Е». Телефон +7 (812) 365-46-03 / 04 **E-mail:** server@szko ru
Екатеринбург. Филиал в г Екатеринбурге Адрес 620024,
г Екатеринбург, ул. Новинская, д 2щ Телефон: +7 (343) 272-72-01 (02/03/04/05/06/08)
Самара. Филиал в г. Самаре Адрес: 443052, г Самара, пр-т Кирова, д 75/1, лит «Е»
Телефон· +7 (846) 269-66-70 (71 73) **E-mail:** RDC-samara@mail ru
Ростов-на-Дону. Филиал в г. Ростове-на-Дону Адрес: 344023,
г. Ростов-на-Дону, ул Страны Советов, 44 А Телефон: +7(863) 303-62-10
Центр оптово-розничных продаж Cash&Carry в г Ростове-на-Дону Адрес: 344023,
г. Ростов-на-Дону, ул. Страны Советов, д 44 В Телефон (863) 303-62-10 Режим работы с 9-00 до 19-00.
Новосибирск. Филиал в г. Новосибирске. Адрес 630015,
г. Новосибирск, Комбинатский пер., д 3 Телефон: +7(383) 289-91-42.
Хабаровск. Филиал РДЦ Новосибирск в Хабаровске. Адрес· 680000, г. Хабаровск,
пер.Дзержинского, д 24, литера Б, офис 1 Телефон +7(4212) 910-120
Тюмень. Филиал в г. Тюмени Центр оптово-розничных продаж Cash&Carry в г. Тюмени
Адрес. 625022, г. Тюмень, ул Алебашевская, 9А (ТЦ Перестройка+)
Телефон. +7 (3452) 21-53-96/ 97/ 98
Краснодар. Обособленное подразделение в г Краснодаре
Центр оптово-розничных продаж Cash&Carry в г. Краснодаре
Адрес: 350018, г. Краснодар, ул Сормовская, д 7, лит «Г» Телефон (861) 234-43-01(02)
Республика Беларусь. Центр оптово-розничных продаж Cash&Carry в г.Минске Адрес· 220014,
Республика Беларусь, г. Минск, проспект Жукова, 44, пом 1-17, ТЦ «Outleto».
Телефон: +375 17 251-40-23; +375 44 581-81-92. Режим работы с 10-00 до 22-00
Казахстан. РДЦ Алматы Адрес 050039, г Алматы, ул.Домбровского, 3 «А»
Телефон +7 (727) 251-58-12, 251-59-90 (91,92,99)
Украина. ООО «Форс Украина» Адрес· 04073, г.Киев, Московский пр-т, д 9.
Телефон: +38 (044) 290-99-44 **E-mail:** sales@forsukraine com

Полный ассортимент продукции Издательства «Э» можно приобрести в магазинах «Новый книжный» и «Читай-город».
Телефон единой справочной. 8 (800) 444-8-444. Звонок по России бесплатный.

В Санкт-Петербурге: в магазине «Парк Культуры и Чтения БУКВОЕД», Невский пр-т, д 46
Тел · +7(812)601-0-601, www bookvoed ru

Розничная продажа книг с доставкой по всему миру. Тел +7 (495) 745-89-14

ПРОГРАММИРОВАНИ

ДЛЯ НАЧИНАЮЩИХ

НАЧНИ ПРОГРАММИРОВАТЬ ПРЯМО СЕЙЧАС!

Вы выучите Python!

Хотите получить «черный пояс» по программированию? Зед Шоу гарантирует его вам!

В этой книге вы найдете

52 идеально продуманных упражнения,

а также теорию и ответы на часто задаваемые вопросы. Выполняйте задания, исправляйте ошибки, запускайте программы и наслаждайтесь результатом!

Вы узнаете:

- Как работает код
- Как выглядят хорошо написанные программы
- Как читать, писать и анализировать код
- Как находить и исправлять допущенные ошибки

Зед Шоу – программист, писатель, а еще он заядлый гитарист. Его книги прочли миллионы людей по всему миру. Написанные им программы используются в крупнейших международных компаниях. Его публикации все время цитируются многочисленными сообществами гиков в социальных сетях. Откройте для себя и вы этого интересного автора, чьи книги помогают людям исполнять свои мечты и обучаться программированию с нуля.

Эта книга посвящена языку Python 2.7 и будет особенно полезна тем, кто до сих пор никогда не пробовал самостоятельно писать программы. Простое и ясное изложение облегчит первые шаги тем, кто не привык к сухому академичному стилю. Важно, что в книге есть специальное приложение, посвященное основам работы в командной строке. Это издание будет неплохим началом пути в мир программирования, который не имеет конца.

Владислав Перлин, преподаватель центра «Специалист» при МГТУ им. Н. Э. Баумана

ISBN 978-5-699-98251-6

Addison Wesley