Зед А. Шоу

Легкий способ выучить Python 3

Уникальная методика обучения программированию для начинающих


Мировой
компьютерный
бестселер

Zed A. Shaw

Python 3

Addison
Wesley

Зед А. Шоу

Легкий способ выучить Python 3

Москва
2019

УДК 004.43
ББК 32.973.26-018.1
Ш81

Zed A. Shaw
LEARN PYTHON 3 THE HARD WAY:
A Very Simple Introduction to the Terrifying Beautiful World of Computers and Code.
1st edition

Authorized translation from the English language edition, entitled Learn Python 3 the Hard Way: A very simple introduction to the terrifying beautiful world of computers and code 1st edition; ISBN 0134692888; by Zed A. Shaw; published by Pearson Education, Inc., publishing as Addison-Wesley Professional.

Шоу, Зед.

Ш81 Легкий способ выучить Python 3 / Зед Шоу ; [пер. с англ. М. А. Райтмана]. — Москва : Эксмо, 2019. — 368 с. — (Мировой компьютерный бестселлер).

ISBN 978-5-04-093536-9

Зед Шоу — автор всемирно известной методики самостоятельного обучения языкам программирования The Hard Way (в дословном переводе: «Сложный способ»). Со свойственным Зеду юмором он дал такое название собственному методу не только «шутки ради», но еще и чтобы сразу направить мысли читателей в правильное русло. «Самостоятельное изучение языков программирования, — говорит Шоу, — это непрестанная работа над собой. Ведь главный ваш враг в освоении любой новой информации и в получении новых навыков — вы сами». Мегауспешная серия самоучителей теперь и в России!

УДК 004.43
ББК 32.973.26-018.1

ISBN 978-5-04-093536-9

Содержание

Предисловие

Эта простая книга предназначена для обучения вас программированию с нуля. Хотя ее название[1] звучит как «трудный способ изучения кодинга», на самом деле это не так. Слово «трудный» используется потому, что в книге применена техника обучения, называемая инструкцией. Инструкции заключаются в последовательности созданных мной упражнений, направленных на закрепление навыков программирования через повторение. Такой метод обучения весьма результативен для новичков, которые не имеют знаний и должны приобрести базовые навыки, прежде чем смогут понять более сложные темы. Данный метод используется во всех сферах, от боевых искусств и музыки до изучения элементарной математики и обучения чтению.

Эта книга заложит и укрепит навыки программирования на Python с помощью техник практики и запоминания, позволяя переходить 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 даст вам «черный пояс программиста». Это означает, что вы приобретете достаточно навыков, чтобы начать программировать.

Если вы будете усердно работать, не станете торопиться и приобретете эти навыки, вы научитесь кодингу.

Новое в этом издании

В этой книге описана работа с версией Python 3.6. Я выбрал эту версию Python, потому что в ней реализована новая улучшенная система форматирования строк, которой проще пользоваться, чем предыдущей версией 4 (или 3, уже не помню, так как их было много). Новички могут столкнуться с некоторыми трудностями в освоении версии Python 3.6, но в этой книге я покажу, как с ними справляться. Особенно неприятная трудность заключается в том, что в некоторых случаях Python 3.6 выдает очень скудные сообщения об ошибках, но я помогу вам с ними разобраться.

Также я дополнил видеоуроки, вложив в них свой более чем пятилетний опыт обучения пользователей работе с Python. Вы можете посмотреть эти видеоролики на сайте **informit.com/title/9780134692883**. Ранее, в видеороликах было записано, как я выполняю упражнения. В новых видеороликах, помимо этого,

[1] В оригинале книга называется Learn Python 3 the Hard way. — Прим. ред.

вы увидите, как сделать каждое упражнение неправильно, а затем все исправить. Так вы приобретете навык отладки кода. Он научит вас исправлять проблемы, с которыми вы столкнетесь, а также, покажет, как Python выполняет созданные вами программы. Цель данной методологии заключается в том, чтобы дать вам представление о способах выполнения Python вашего кода, чтобы было проще разобраться, почему он не работает. Вы также узнаете много полезных подсказок для отладки неработающих программ.

И наконец, в этой книге полностью от начала и до конца поддерживается операционная система Microsoft Windows 10. В предыдущем издании основное внимание уделялось системам типа Unix, таким как macOS и Linux, а Windows была не в приоритете. Перед тем, как я сел за эту книгу, корпорация Microsoft начала серьезно относиться к инструментам разработки с открытым исходным кодом и программистам, и я не мог проигнорировать это событие и не применить среду разработки под Windows для программирования на языке Python. Видеоролики записаны в среде Microsoft Windows и отражают выполнение различных сценариев, но показана работа и в операционных системах macOS и Linux для полноты восприятия. Я расскажу вам обо всех шагах на каждой платформе, опишу инструкции по установке и дам все необходимые советы.

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 на первых порах, но все равно придерживайтесь данного курса. Если вы будете читать эту книгу и выполнять упражнения пару часов в сутки, то приобретете отличную основу для перехода к изучению другой книги. С помощью этой книги вы, может, и не получите «реальных» навыков программирования, но точно освоите основы, необходимые для изучения языка.

Суть этой книги в том, чтобы научить вас трем самым основным навыкам, необходимым каждому начинающему программисту: чтению и вводу, вниманию к деталям и обнаружению различий.

Чтение и ввод

Очевидно, что, если у вас есть проблемы с вводом команд, вам будет сложно разобраться с кодом. Особенно, если вы набираете явно лишние символы в исходном коде. Без этого простого навыка вы не сможете даже на самом элементарном уровне разобраться в том, как работает программное обеспечение.

Набирая и выполняя примеры программ, вы запоминаете имена элементов языка программирования и учитесь читать код.

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 важнейшие элементы результата вашего труда. В программировании отсутствие данного навыка чревато багами, а также трудностями, возникающими у пользователей программного обеспечения.

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

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 использовать их. Сначала вы должны пойти трудным путем – научиться определять различия наглядно, а потом уже можно использовать инструменты.

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 замечать опечатки, ошибки, баги и другие проблемы.

Хватит смотреть, спрашивайте

Когда вы пишете код, обязательно случаются баги. «Баг» – это дефект, ошибка или проблема в коде, который вы написали. Легенда гласит, что термин закрепился в речи программистов из-за того, что некое насекомое залетело в один из первых компьютеров и привело к выходу его из строя. Чтобы ввести компьютер в строй, было необходимо его починить или отладить. В мире программ существует много багов. Крайне много.

Как и то легендарное насекомое, ваши баги будут прятаться где-то в коде, и вам нужно будет их разыскать. Вы не можете просто сидеть перед монитором своего компьютера, разглядывая код, в надежде, что ответ придет к вам сам. Так вы не получите необходимой информации – данных.

Вам нужно встать и найти насекомое.

Чтобы это сделать, вам придется выполнить свой код и понять, что происходит (или не происходит), или взглянуть на проблему под другим углом. В этой книге я часто говорю вам «хватит смотреть, спрашивайте». Я расскажу, как сделать так, чтобы по коду вы могли определить, что происходит, и вывести полученную информацию в возможное решение проблемы. Я также покажу, как разными способами «смотреть» на код, чтобы у вас было больше информации и понимания.

Никакого копирования/вставки

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

Дополнительные видеоролики

К книге прилагается обширный набор видеороликов, демонстрирующих, как работает код, и, самое главное, как его «сломать». Видео – идеальный способ продемонстрировать множество намеренно допущенных типичных ошибок и способов их исправления.

Я также продемонстрирую некоторые приемы отладки. В своих видеороликах я покажу вам, как «заканчивать смотреть на код и начать спрашивать» о том, что пошло не так. Вы можете смотреть эти видеоролики в Интернете на сайте **informit.com/title/9780134692883**.

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, что, чтобы достичь хороших результатов, нужно практиковаться каждый день, даже если в определенный день хочется прогулять занятие (что бывает частенько) или же процесс обучения продвигается с трудом. Продолжайте пытаться, и в конечном счете, все станет проще и веселее.

Пока я писал эту книгу и аналогичное руководство по языку Ruby, я открыл для себя рисование и живопись. Я влюбился в изобразительное искусство в возрасте 39 лет и стал проводить каждый день, изучая его таким же образом, как обучался игре на гитаре, музыке и программированию. Я собрал коллекцию обучающих книг, делал все, что там было написано, рисовал каждый день, и сосредотачивался на том чтобы получать удовольствие от процесса обучения. Я ни в коем случае не «художник», но теперь могу заявить, что умею рисовать. Так же как я изучал искусство, я обучаю и вас с помощью этой книги. Если раздробить задачу на маленькие упражнения и уроки, и выполнять их каждый день, вы можете научиться абсолютно всему. Если вы сосредоточитесь на медленном постижении знаний и получении удовольствия от процесса обучения, вы выиграете, независимо от того, насколько хорошо будете в своем деле преуспевать.

Когда вы изучите эту книгу и продолжите программировать, помните, что начинание на первых порах может быть трудным. Возможно, вы боитесь неудачи, поэтому сдаетесь при первой трудности. А может вам не хватает самодисциплины, и вы ничего не делаете, так как «это скучно». Или, может, вы уверяете себя в своей «одаренности», поэтому все, что может выставить вас глупым или неопытным, не стоит того, чтобы попытаться. Может быть, вы конкурентоспособны и несправедливо сравнивать вас с кем-то вроде меня, кто занимается программированием в течение более 20 лет.

Независимо от причины 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 человеческого языка. Вы будете сражаться со словами и 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». Если вы продолжите выполнять упражнения и пытаться понять их, вы достигнете его. Конечно, вы не станете гуру кодинга, но, по крайней мере, разберетесь, как работает язык программирования.

Если вы бросите обучение, то никогда не достигнете этого результата. Вы столкнетесь с первой трудностью (которая лишь первая) и прекратите обучение. Если вы продолжите заниматься, продолжите вводить код, пытаясь понять его и научиться читать, вы, в конечном счете, получите искомые навыки. Но если прочитали всю книгу и до сих пор не понимаете, как работает код, вы, по крайней мере, попытались. Вы можете сказать, что старались изо всех сил, и даже больше, но не получилось. Но вы, по крайней мере, попытались. И вы можете гордиться этим!

Благодарности

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

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

Спасибо.

Настройка

В этом упражнении нет примеров кода. Оно предназначено для подготовки компьютера к установке программного обеспечения Python. Вы должны в точности следовать приведенным инструкциям. Если у вас возникнут сложности с выполнением описанных инструкций, посмотрите мои видеоролики для вашей платформы.

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

---

macOS

Для успешного выполнения этого упражнения совершите следующие действия:

Перейдите на страницу **www.python.org/downloads/** и загрузите файл Python 3 последней версии по ссылке Mac OS X 64-bit/32-bit installer. Установите это программное обеспечение так же, как любое другое.

1. Перейдите на сайт **atom.io**, найдите ссылку на скачивание дистрибутива текстового редактора Atom, скачайте и установите его. Если пользоваться программой Atom вам неудобно, просмотрите список аналогичных текстовых редакторов в конце данного упражнения.

2. Поместите значок приложения Atom на панель **Dock** для быстрого доступа к редактору.

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

4. Поместите значок программы Терминал (Terminal) на панель **Dock** для быстрого доступа к редактору.

5. Запустите программу Терминал (Terminal).

6. В окне программы Терминал (Terminal) выполните команду `python3.6`. После ввода имени команды нажмите клавишу **@**. Так вы можете выполнять любые команды.

7. Выполните команду `quit()`, а затем нажмите 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 с помощью команды `ls`, чтобы увидеть сохраненный файл.

macOS: результат

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

```
$ python3.6
```

```
  Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32
bit (Intel)] on win32
  Type "help", "copyright", "credits" or "license" for more information.
  >>>
  ~ $ mkdir lpthw
  ~ $ cd lpthw
  lpthw $ ls
  # ... Используйте текстовый редактор для создания файла test.
  txt....
  lpthw $ ls
  test.txt
  lpthw $
```

Windows

1. Перейдите на сайт **atom.io** с помощью веб-браузера, скачайте текстовый редактор Atom и установите его. Вам не нужно обладать правами администратора для этого.

2. Для быстрого доступа к Atom поместите ярлык этой программы на Рабочий стол и/или Панель задач. Обе настройки можно выполнить во время установки. Если производительности вашего компьютера для работы программы Atom недостаточно, просмотрите список аналогичных текстовых редакторов в конце данного упражнения.

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. Загрузите дистрибутив Python 3 последней версии со страницы **www.python.org/downloads/** и установите его. Убедитесь, что при установке установлен флажок **Add Python 3.6 to the PATH** (Добавить Python 3.6 к системной переменной PATH).

6. В оболочке PowerShell запустите среду Python. Для этого просто наберите имя `python` и нажмите клавишу **Enter**. Если запуск среды Python не происходит (команда `python` не распознается оболочкой командной строки), переустановите ее и убедитесь, что установлен флажок **Add Python 3.6 to the PATH** (Добавить Python 3.6 к системной

переменной PATH). Эту настройку вы можете случайно пропустить, поэтому будьте внимательны.

7. 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.

8. Разберитесь, как в оболочке командной строки (программе PowerShell) 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. Запустите команду меню Сохранить (Save) или Сохранить как (Save As) и выберите этот каталог.

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

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

С этого момента и далее в книге, если я пишу словосочетание «оболочка командной строки» или слово «консоль», речь идет о PowerShell, ее и следует использовать. Для запуска Python3.6 вы можете просто набрать команду `python`.

Windows: результат

```
> python
>>> quit()
> mkdir lpthw
> cd lpthw
...Используйте редактор для создания файла test.txt в папке lpthw ...
>
> dir
 Volume in drive C is
 Volume Serial Number is 085C-7E02
```

```
Directory of C:\Users\you\lpthw

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, что если вы работаете в Linux, то умеете устанавливать пакеты, поэтому приведу лишь краткие инструкции:

1. Запустите менеджер пакетов Linux и с помощью него установите среду разработки Python 3.6. Если не получается, загрузите дистрибутив Python 3 последней версии со страницы **www.python.org/downloads/** и установите его вручную.

2. Запустите менеджер пакетов Linux и с помощью него установите текстовый редактор Atom. Если пользоваться программой Atom вам неудобно, просмотрите список аналогичных текстовых редакторов в конце данного упражнения.

3. Для быстрого доступа к редактору Atom поместите ярлык этой программы на панель быстрого запуска.

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

5. Также поместите и ярлык оболочки командной строки на панель быстрого запуска.

6. Запустите оболочку командной строки.

7. В окне оболочки командной строки выполните команду `python3.6`. После ввода имени команды нажмите клавишу **Enter**. Если выполнить команду не получается, попробуйте ввести просто `python`.

8. 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.

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

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

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

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

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

Linux: результат

```
$ python
>>> quit()
$ mkdir lpthw
$ cd lpthw
# ... Используйте текстовый редактор 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.

Поиск в Интернете

Значительная часть этой книги учит искать информацию по программированию в Интернете. Я буду говорить вам «найдите это в Интернете», и ваша задача будет состоять в том, чтобы воспользоваться поисковой системой и найти ответ. Я не даю вам готовые ответы сразу, потому что хочу научить вас самостоятельно расширять свои знания, чтобы вы не нуждались в этой книге после ее прочтения. Если вы можете найти ответ на свой вопрос в Интернете, то вы на один шаг ближе к моменту, когда перестанете нуждаться в моих подсказках, что и является моей целью.

Благодаря поисковым системам, таким как Google, вы с легкостью сможете найти все необходимое. Если я говорю: «найдите в Интернете список функций Python», вы делаете следующее:

1. Загружаете веб-сайт **google.com**.
2. Вводите в строке поиска запрос типа *python3 список функций*.
3. Просматриваете перечисленные в результатах поиска сайты, пока не найдете ответ.

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, уделите время практике в выполнении описанных действий, потому что, если вы не умеете делать эти базовые операции, вам будет очень сложно при настоящем программировании в будущих упражнениях.

Если кто-то посоветует вам остановиться на каком-то определенном упражнении из этой книги или пропустить некоторые из них, вам следует проигнорировать такие советы. Каждый, кто скрывает от вас свои знания, или, что еще хуже, «преподносит все на блюдце» вместо мотивации вас на приложение собственных усилий, просто выстраивает нити зависимости от себя. Не слушайте их и в любом случае выполняйте упражнения, чтобы научиться самостоятельно развивать свои навыки.

Программисты в конечном итоге, могут сказать вам, что следует использовать операционную систему macOS или Linux. Если программисту по душе

красивые шрифты и типографика, он посоветует вам приобрести компьютер Mac. Если он любит держать все под контролем и отращивает огромную бороду, он предложит установить Linux. Мой же совет звучит так: нужно использовать любой компьютер, который есть у вас в настоящий момент. 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. Записывать упражнения с помощью текстового редактора.
2. Выполнять упражнения, которые вы записали.
3. Решать проблемы с ними в случае сбоя.
4. Повторять операции для запоминания.

Все остальное будет только отвлекать, поэтому придерживайтесь озвученного плана.

Альтернативные текстовые редакторы

Текстовый редактор – необходимый атрибут любого программиста, но вам, как новичку, нужен самый простой текстовый редактор для набора кода. Такие редакторы отличаются от тех, что предназначены для записи обычных текстов, потому что обрабатывают уникальные характеристики компьютерного кода. В этой книге я рекомендую использовать Atom – бесплатный и кроссплатформенный редактор.

Но, в некоторых случаях, Atom может некорректно работать на вашем компьютере, поэтому предлагаю несколько альтернативных вариантов:

Название	Платформа	Сайт
Visual Studio Code	Windows, macOS, Linux	**code.visualstudio.com**
Notepad++	Windows	**notepad-plus-plus.org**
gEdit	Linux, macOS, Windows	**github.com/GNOME/gedit**
Textmate	macOS	**github.com/textmate/textmate**

SciTE	Windows, Linux	**www.scintilla.org/SciTE.html**
jEdit	Linux, macOS, Windows	**www.jedit.org**

Редакторы размещены в порядке, начиная с наиболее стабильных в работе. Имейте в виду, что некоторые из них могут зависать, не запускаться или работать некорректно на вашем компьютере. Если вы безрезультатно попробуете один из них, испытайте другой. Кроме того, в столбце Платформа я разместил названия операционных систем в аналогичном порядке – первыми идут те, в которых указанная программа точно будет работать. Поэтому, если вы пользуетесь компьютером под управлением операционной системы Windows, ищите те редакторы, у которых в столбце Платформа операционная система Windows указана первой.

Если вы уже имеете опыт работы с Vim или Emacs, то можете использовать их. Если вам незнакомы эти названия, то вы вполне обойдетесь без них. Программисты могут убеждать вас использовать Vim или Emacs, но это только затруднит обучение. У вас задача изучить Python, а не Vim или Emacs. Если, завершая сеанс работы с Vim, вы не сможете выйти из этой программы, наберите команду `:q!` или `ZZ`. Если тот человек, который рекомендовал вам использовать Vim, не упомянул про это, вы поймете, почему его не стоит слушать.

Не используйте интегрированную среду разработки (IDE) для выполнения упражнений из этой книги. Использование среды разработки приведет к тому, что вы не сможете работать с новыми языками программирования, пока та или иная компания не решит продать вам среду разработки для нужного языка. Это означает, что вы не сможете использовать этот новый язык, пока он не станет достаточно популярным, чтобы наработать выгодную клиентскую базу для компании-производителя среды разработки. Если вы уверены в себе, то можете работать в специализированном текстовом редакторе для программистов (например, Vim, Emacs, Atom и т. п.), тогда вам не придется ничего дожидаться. В некоторых ситуациях среды разработки хороши (например, если ведется работа с существующей огромной базой кода), но сильное увлечение ими ограничивает ваше развитие.

Вам также не стоит использовать IDLE – специальную интегрированную среду разработки для языка Python. В ней много ограничений, и к тому же, не совсем адекватная цена за полную версию. Все, что вам необходимо, – это простой редактор, оболочка командной строки и Python.

УПРАЖНЕНИЕ 1

Первая программа

Надеюсь, вы хорошо 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.

Внимание! Если вы пропустили упражнение 0, вы недобросовестно выполняете задания из книги. Вы пытаетесь работать в IDLE или IDE? Я упоминал в упражнении 0, что не следует использовать их. Если вы пропустили упражнение 0, будьте добры, вернитесь и прочтите его.

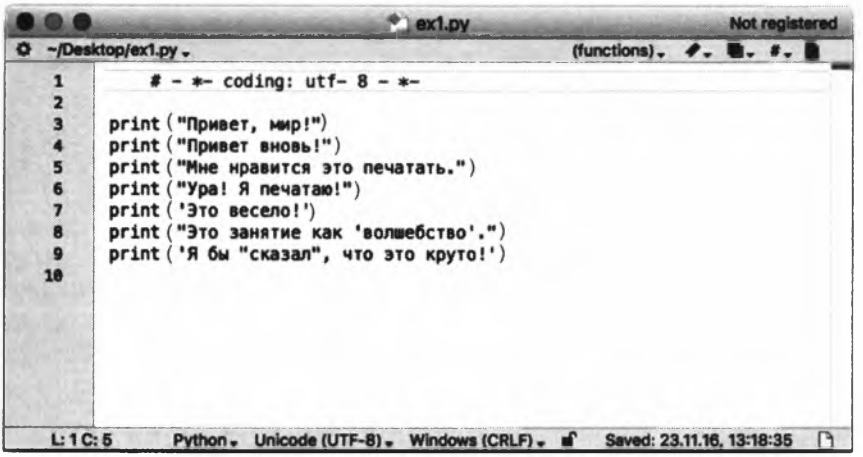
Введите следующий текст в файл с именем *ex1.py*. Важно соблюдать расширение, так как Python лучше всего работает с файлами, имена которых заканчиваются на *.py*.

ex1.py

```
1 print("Привет, мир!")
2 print("Привет вновь!")
3 print("Мне нравится это печатать.")
4 print("Ура! Я печатаю!")
5 print('Это весело!')
6 print("Это занятие как 'волшебство'.")
7 print('Я бы "сказал", что это круто!')
```

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_python3.zip**.

Интерфейс текстового редактора Atom, независимо от операционной системы, будет выглядеть следующим образом:



Не беспокойтесь, если интерфейс редактора на вашем компьютере немного отличается. В вашем случае может быть несколько другой заголовок окна, цвета могут различаться, а в левой части окна вашего Atom вместо **ex01**, будет отображаться каталог, который вы использовали для сохранения ваших файлов. Все эти отличия допустимы. Сейчас самое 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. У меня в начале строки указано слово `print`, и код выглядит точно так же, как записано в файле *ex1.py*. «Точно так же» означает абсолютное совпадение, а не «что-то вроде того». Каждый символ должен быть введен в точности, чтобы код работал. Оформление кода цветом может различаться. Не обращайте внимания на цвет, важны набираемые символы.

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

```
python3.6 ex1.py
```

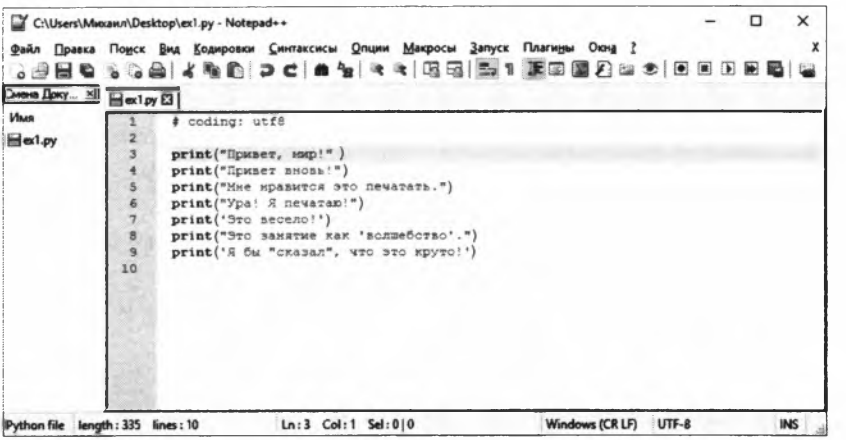
В операционной системе Windows всегда набирайте слово `python` вместо `python3.6`, как показано ниже:

```
python ex1.py
```

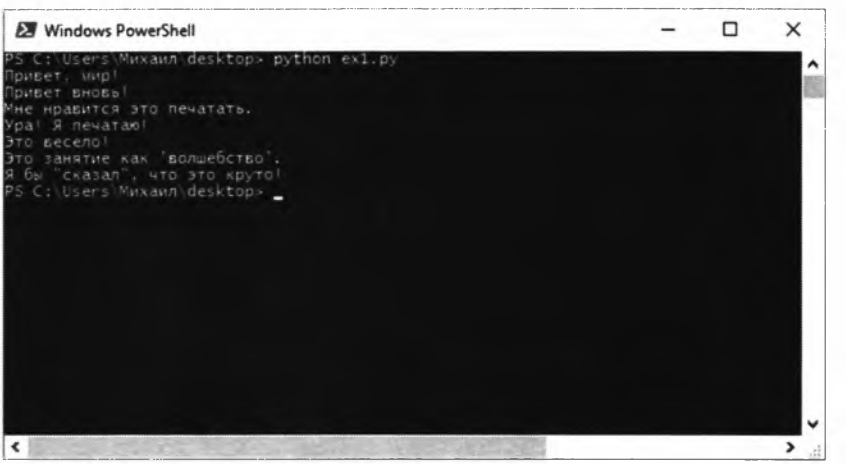
Если вы все сделали правильно, то увидите тот же результат, что представлен на рисунках ниже. Если нет – вы где-то ошиблись. И не надо думать, что ошибся компьютер.

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 `python3.6 ex1.py`, это не важно. Важно то, что вы вводите команду и должны увидеть результат ее выполнения, аналогичный моему.

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

```
$ python3.6 ex1.py
  File "ex1.py", line 3
    print ("Мне нравится это печатать.)
                                       ^
SyntaxError: EOL while scanning string literal
```

Важно, чтобы вы понимали, в чем причина ошибок, так как вы регулярно будете их делать. 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 3 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 двойной кавычки ".

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

Практические задания

Каждое упражнение также содержит практические задания. Практические задания необходимо выполнять для закрепления пройденного материала. Если вы не можете их выполнить в настоящий момент, 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 каких-либо исключений.

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

Распространенные вопросы

Эти вопросы задавались реальными студентами при комментировании этой книги.

Могу ли я использовать IDLE[2]?

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.

Как научить редактор подсвечивать код разными цветами, как в ваших примерах?

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?

---

[2] Integrated DeveLopment Environment – интегрированная среда разработки на языке Python. – (Прим. перев.)

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 – `python3.6 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 в том же каталоге, что и созданный вами файл. Убедитесь, что перешли в этот каталог с помощью команды `cd`. Например, если вы сохранили файл в каталоге папка/*ex1.py*, то нужно первым делом выполнить команду `cd папка`, прежде чем пытаться запустить команду `python3.6 ex1.py`. Если вы не умеете перемещаться по каталогам, изучите приложение, упоминаемое в первом вопросе.

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-либо информацию, а также для отключения (закомментирования) части кода вашей программы, если нужно, чтобы временно он 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("Это моя программа.") # и этот комментарий игнорируется.
5
6 # С помощью комментариев можно "отключать" части кода:
7 # print ("Код не работает.")
8
9 print("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

```
$ python3.6 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 может выглядеть несколько иначе на вашем компьютере. Важно то, что отображается после строки `$ python3.6 ...`.

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 направлении. Начните с последней строки и, проверяя каждое слово, разберитесь, что вы должны ввести.

3. Вы обнаружили ошибки? Устраните их.

4. Прочитайте вслух введенный ранее код, проговаривая название каждого символа. Вновь 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.

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 и занимались бы математикой, а не написанием кода внутриигровых покупок и рекламы.

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. После изучения кода этого упражнения, вернитесь к списку и разберитесь, для выполнения какого действия предназначен каждый арифметический оператор. Например, оператор + выполняет сложение.

ex3.py

```
1 print("Я подсчитаю свою живность:")
2
3 print("Куры", 25 + 30 / 6)
4 print("Петухи", 100 - 25 * 3 % 4)
5
6 print("А теперь подсчитаю яйца:")
7
```

```
 8 print(3 + 2 + 1 - 5 + 4 % 2 - 1 / 4 + 6)
 9
10 print("Верно ли, что 3 + 2 < 5 - 7?")
11
12 print(3 + 2 < 5 - 7)
13
14 print("Сколько будет 3 + 2?", 3 + 2)
15 print("А сколько будет 5 - 7?", 5 - 7)
16
17 print("Ой, я, кажется, запутался... Почему False?")
18
19 print("Еще раз посчитаю...")
20
21 print("Это больше?", 5 > -2)
22 print("А это больше или равно?", 5 >= -2)
23 print("А это меньше или равно?", 5 <= -2)
```

Убедитесь, что напечатали код в точности, как у меня, прежде чем запускать программу. Сравните каждую строку вашего файла и моего.

Результат выполнения

Сеанс упражнения 3

```
$ python3.6 ex3.py
Я подсчитаю свою живность:
Куры 30.0
Петухи 97
А теперь подсчитаю яйца:
6.75
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. Перед каждой строкой кода напишите комментарий (предваряя текст комментария символом #) для себя, указывая, что делает данный код.

2. Помните, вы запускали Python в упражнении 0? Вновь запустите Python, как там описано, и с помощью математических операторов используйте оболочку командной строки в качестве калькулятора.

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

4. 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?

Данная операция вычисляет остаток от деления. Допустим, *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 следующим образом: Скобки, Экспоненты (степени), Умножение, Деление, Сложение и Вычитание. Чтобы проще было запомнить, можно сократить до С.Э.У.Д.С.В. и придумать маленькое предложение: «Смотрите, Эти Умельцы Делают Скачущий Велосипед». В Python используется этот же порядок вычислений.

УПРАЖНЕНИЕ 4

Переменные и имена

В предыдущих упражнениях вы научились выполнять вычисления и выводить текст с помощью команды `print`. Теперь давайте разберемся с переменными. В программировании переменная является всего лишь именем для чего-нибудь, подобно тому, как именем «Зед» зовут «человека, который написал эту книгу». Программисты используют переменные, чтобы код был похож на человеческий язык, и его проще было читать. Если бы удобных имен переменных не существовало, программисты заблудились бы в дебрях собственного кода.

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

1. Перед каждой строкой кода напишите комментарий (предваряя текст комментария символом #) для себя, 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("В наличии", cars, "автомобилей.")
print("И только", drivers, "водителей вышли на работу.")
print("Получается, что", cars_not_driven, "машин пустуют.")
print("Мы можем перевезти сегодня", carpool_capacity, "человек.")
print("Сегодня нужно отвезти", passengers, "человек.")
print("В каждой машине будет примерно", average_passengers_per_car, "человека.")
```

Примечание. В именах некоторых переменных, например `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

```
$ python3.6 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 3.0 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. Обязательно ссылайтесь на 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. Напишите комментарий для каждой строки с присвоением значения переменной.

4. Знаете ли вы, что символ = (равно) называется оператором присваивания, и 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.

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

Все очень 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 до строки 1, 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.

УПРАЖНЕНИЕ 5

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) какой-либо текст, вы создаете строку. Строки позволяют работать с текстовой информацией. Вы выводите строки на экран, сохраняете в файлы, отсылаете на веб-серверы и т. д. и т. п.

Строки очень удобны. В этом упражнении вы узнаете, как формировать строки, в которые встроены переменные. Вы будете встраивать переменные внутрь строк с использованием специальных символов `{}`, окружая этими символами переменные. Также вам нужно будет начинать строку с переменными с буквы `f` (от слова *format* – форматирование), например, так: `f"Привет, {переменная}!"`. Эта маленькая буковка `f` перед символом " (двойные кавычки) и символы `{}` говорят 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

```
my_name = 'Зед Шоу'
my_age = 35 # это правда!
my_height = 188 # см
my_weight = 80 # кг
my_eyes = 'Голубые'
my_teeth = 'Белые'
my_hair = 'Каштановые'

print(f"Давайте поговорим о человеке по имени {my_name}.")
print(f"Его рост составляет {my_height} см.")
print(f"Он весит {my_weight} кг.")
print("На самом деле это не так и много.")
print(f"У него {my_eyes} глаза и {my_hair} волосы.")
print(f"Его зубы обычно {my_teeth}, хотя он и любит пить кофе.")

```

```
16 # эта строка кода довольно сложная, не ошибитесь!
17 total = my_age + my_height + my_weight
18 print(f" Если я сложу {my_age}, {my_height} и {my_weight}, то получу {total}.")
```

Результат выполнения

Сеанс упражнения 5

```
$ python3.6 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 префикс `my_`. Убедитесь, что вы изменили имена переменных во всем коде, а не только в строках, где присваивали им значения.

2. Попробуйте написать код, который преобразует сантиметры и килограммы в метры и тонны. 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 = 'Зед Шоу'`?

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

Как округлить число с плавающей точкой?

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

Почему я не могу понять этот пример?

Попробуйте изменить в этом сценарии значения переменных, указав собственные. Пример с данными о себе покажется вам более понятным. Кроме того, учитывая, что вы только начали изучать язык программирования, это нормально. Продолжайте, и чем больше вы выполните упражнений, тем понятнее будет код.

Строки и текст

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

Для начала разберемся со строками.

Строка, как правило, это какой-либо 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 '), для его вывода на экран.

Строка может содержать любое количество переменных, используемых в коде на языке Python. Напоминаю, что переменная – это строка кода с синтаксисом вида `имя = значение`. В этом упражнении, с помощью кода `types_of_people = 10` создается переменная с именем `types_of_people`, и ей присваивается значение, равное `10`. Вы можете поместить имя этой переменной в любую текстовую строку в виде `{types_of_people}`. Кроме того, вы увидите строки особого типа, называемые *f-строками*, которые выглядят так:

```
f"некоторый текст {переменная}"
f"другой текст {другаяпеременная}"
```

В языке Python также употребляется еще один тип форматирования с синтаксисом `.format()`, который применяется в строке 17. Иногда я использую его, когда хочу применить форматирование к уже созданной строке, например, в цикле. Мы рассмотрим эту тему немного позже.

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

ex6.py

```
1 types_of_people = 10
2 x = f"Существует {types_of_people} типов людей."
```

```
 3
 4 binary = "Python"
 5 do_not = "нет"
 6 y = f"Те, кто понимает {binary}, и те, кто - {do_not}."
 7
 8 print(x)
 9 print(y)
10
11 print(f"Я сказал: {x}")
12 print(f"А еще я сказал: '{y}'")
13
14 hilarious = False
15 joke_evaluation = "Разве это не смешно?! {}"
16
17 print(joke_evaluation.format(hilarious))
18
19 w = "Это часть строки слева..."
20 e = "а это справа."
21
22 print(w + e)
```

Результат выполнения

Сеанс упражнения 6

```
$ python3.6 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 обманываю вас.

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

Распространенные вопросы

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

В основном, для соблюдения форматирования, но я также использую одиночные кавычки внутри строк, заключенных в двойные кавычки. Взгляните на строки 6 и 15, чтобы понять, как я это делаю.

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.

УПРАЖНЕНИЕ 7

Еще о выводе

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

ex7.py

```
print("У Мэри был маленький барашек.")
print("Его шерсть была белой как {}.".format('снег'))
print("И всюду, куда Мэри шла,")
print("Маленький барашек всегда следовал за ней.")
print("." * 10) # что это могло значить?

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

# Обратите внимание на запятую в конце строки. Уберите ее. Что произошло?
print(end1 + end2 + end3 + end4 + end5, end=' ')
print(end6 + end7 + end8)
```

Результат выполнения

Сеанс упражнения 7

```
$ python3.6 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 последующих 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 в блокнот и разберитесь, какие ошибки вы чаще совершаете.

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

5. Помните, что все совершают ошибки. Программисты считают, что они никогда не ошибаются, и их действия совершенны, но это неправда. Программисты постоянно совершают ошибки.

Работа с ошибками

Вам понравилось «ломать» код в упражнении 6? Отныне вы будете выводить из строя весь написанный вами или вашими друзьями код. Я не буду приводить раздел «Работа с ошибками» в каждом упражнении, но упоминаю об этом практически во всех видеоуроках. Ваша цель заключается в том, чтобы найти как можно больше различных способов «сломать» код, пока вы не устанете или не исчерпаете все возможные варианты. В некоторых упражнениях я буду указывать специфичный способ «вывода из строя» кода, в остальных случаях «ломайте» скрипты самостоятельно.

Распространенные вопросы

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, либо комментарии объясняют, 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 таком случае, вам просто необходимо расшифровать код и перевести его на человеческий язык.

Равнозначно ли для строк использование одиночных и двойных кавычек, или они предназначены для разных целей?

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

Вывод, вывод

ex8.py

```
formatter = "{} {} {} {}"

print(formatter.format(1, 2, 3, 4))
print(formatter.format("раз", "два", "три", "четыре"))
print(formatter.format(True, False, False, True))
print(formatter.format(formatter, formatter, formatter, formatter))
print(formatter.format(
          "Спят в конюшне пони,",
          "начал пес дремать,",
          "только мальчик Джонни",
          "не ложится спать!"
))
```

Результат выполнения

Сеанс упражнения 8

```
$ python3.6 ex8.py
1 2 3 4
раз два три четыре
True False False True
{} {} {} {} {} {} {} {} {} {} {} {} {} {} {} {}
Спят в конюшне пони, начал пес дремать, только мальчик Джонни не ложится
спать!
```

В этом упражнении я использую элемент кода, называемый функцией, чтобы передать переменную `formatter` в другие строки. Когда я пишу `formatter.format(...)`, я указываю Python сделать следующее:

1. Взять строку `formatter`, определенную в строке 1.
2. Вызвать ее функцию `format`, то есть выполнить консольную команду по имени `format`.

3. Передать функции `format` четыре аргумента, которые совпадают с четырьмя группами символов `{}` в значении переменной `formatter`. Можно перефразировать как: «передать аргументы консольной команде `format`».

4. В результате вызова функции `format` для строки `formatter` получаем новую строку, в которой символы `{}` заменены на четыре переменные. Этот результат выведет функция `print`.

Думаю, для восьмого упражнения вы получили довольно много новой информации. Вполне ожидаемо, что вы не до конца понимаете все происходящее. Не волнуйтесь, в последующих упражнениях все должно постепенно проясниться. А пока попытайтесь изучить код и понять, как он работает, прежде чем переходить к следующему упражнению.

Практические задания

Проверьте свой код, запишите ошибки и постарайтесь не совершать их в следующем упражнении. Другими словами, повторите практические задания, приведенные в упражнении 7.

Распространенные вопросы

Почему значение «`pas`» нужно заключать в кавычки, а значения `False` и `True` – нет?

Потому, что Python распознает логические 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.

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. IDLE не позволит вам выполнить последующие упражнения в книге.

Вывод, вывод, вывод

К этому моменту вы уже должны были понять, что подход, реализуемый в этой книге, – научить вас чему-то новому с помощью более чем одного упражнения. Все начинается с набора кода, который вы, возможно, сразу не поймете, затем при помощи дополнительных упражнений, объясняются основные моменты. Если сейчас вы ничего не поймете, разберетесь позже, после того, как выполните больше упражнений. Записывайте, что вам непонятно, и продолжайте обучение.

ex9.py

```
# Несколько непривычный код - набирайте его в точности.

days = "Пн Вт Ср Чт Пт Сб Вс"
months = "Янв\nФевр\nМарт\nАпр\nМай\nИюнь\nИюль\nАвг"

print("Это дни недели: ", days)
print("А это месяцы: ", months)

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

Результат выполнения

Сеанс упражнения 9

```
$ python3.6 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

Проверьте свой код, запишите ошибки, и постарайтесь не совершать их в следующем упражнении. Вы «ломаете» код, а затем исправляете его? Другими словами, повторите практические задания, приведенные в упражнении 7.

Распространенные вопросы

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.

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

Нужно начать список названий месяцев с символов `\n`, как показано ниже:
`months = "\nЯнв\nФевр\nМарт\nАпр\nМай\nИюнь\nИюль\nАвг"`

Так ли плохо, если все мои ошибки исключительно орфографические?

Большинство программных ошибок в начале обучения (и даже с опытом) носят орфографический характер, либо представляют собой опечатки, но и они могут приводить к неработоспособности кода.

Управляющие последовательности

В упражнении 9 я продемонстрировал новые возможности форматирования вывода. Вы увидели два способа, позволяющих 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, – это 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. Мы рассмотрим некоторые из этих последовательностей, чтобы вы поняли принцип их работы.

Среди них большое значение имеет управляющая последовательность для экранирования одинарных (') или двойных кавычек ("). Допустим, у вас есть строка, заключенная в двойные кавычки, а вы хотите указать двойные кавычки внутри самой строки. Если вы напишете строку вроде `"я "понимаю" Володю"`, Python запутается, так как посчитает, что кавычка перед словом `"понимаю"` заканчивает строку. В таком случае нужен способ сообщить Python, что кавычки внутри «ненастоящие».

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

```
"Брюки Levi's с обхватом талии 30\"."  # экранирование двойной
кавычки в строке
'Брюки Levi\'s с обхватом талии 30".'  # экранирование одиночной
кавычки в строке
```

Второй способ заключается в использовании тройных кавычек, """, в которые можно заключить любое количество текстовых строк. Этот вариант мы также рассмотрим в примере ниже.

ex10.py

```
1 d_artagnan = "\tМеня зовут д'Артаньян."
```

```
athos = "Эта строка\nразделена на две."
porthos = "Я \\ - \\ мушкетер!"

aramis = """
Подсказки:
\t* Граф из Гаскони
\t* Связан с Миледи
\t* Барон\n\t* Генерал Ордена
"""

print(d_artagnan)
print(athos)
print(porthos)
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_python3.zip**.

Результат выполнения

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

Сеанс упражнения 10

```
$ python3.6 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 табуляции (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*
\x*hh*	Символ ASCII в шестнадцатеричной нотации *h*h

Практические задания

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

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

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

Распространенные вопросы

Я до сих пор не до конца разобрался в последнем упражнении. Следует ли 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 (`\`). Подумайте, в каких случаях эта последовательность может понадобиться.

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

Потому, что вы используете обычный слеш (`/`), вместо обратного (`\`). Это разные символы, которые служат для выполнения различных задач.

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

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

Что лучше использовать, `'''` или `"""`?

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

Получение ответов на вопросы

Настало время набирать темп программирования. Вы должны вводить много кода, чтобы набрать опыт непосредственно в наборе текста программ, но печатать простой код довольно скучно. Поэтому сейчас мы попробуем запросить у пользователей сведения о них. Это немного сложнее, потому 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

```
1 print("Сколько тебе лет?", end=' ')
2 age = input()
3 print("Каков твой рост?", end=' ')
4 height = input()
5 print("Сколько ты весишь?", end=' ')
6 weight = input()
7
8 print(f"Итак, тебе {age} лет, в тебе {height} см роста и {weight} кг веса.")
```

Внимание! Обратите внимание на то, что в конце каждой строки со значением в аргументах команды `print` указан символ `,` (запятая). Это необходимо для того, чтобы добавить символ перехода на новую строку.

Результат выполнения

Сеанс упражнения 11

```
$ python3.6 ex11.py
Сколько тебе лет? 38
Каков твой рост? 185
Сколько ты весишь? 70
Итак, тебе 38 лет, в тебе 185 см роста и 70 кг веса.
```

Практические задания

1. Выполните поиск в Интернете и выясните, для чего предназначена функция `input()` языка Python.

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

3. Измените нашу «анкету», чтобы задавать пользователям другие вопросы.

Распространенные вопросы

Как мне получить от пользователя число, чтобы выполнить определенные вычисления?

Эта задача немного сложнее. Попробуйте использовать инструкцию `x = int(input())`, которое принимает число в виде строкового значения с помощью функции `input()`, а затем преобразует его в целочисленное значение с помощью функции `int()`.

Запрос ввода

При вводе функции `input()` вы набираете символы скобок – `(` и `)`. Это подобно форматированию с дополнительными переменными, например `f"{x} {y}"`. Функцию `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 = input("Имя? ")
```

Пользователь увидит запрос «Имя?» и сможет ввести ответ, который будет присвоен в качестве значения переменной `y`. Таким образом, мы можем задать пользователю вопрос и получить ответ.

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

ex12.py

```
1 age = input("Сколько тебе лет? ")
2 height = input("Каков твой рост? ")
3 weight = input("Сколько ты весишь? ")
4
5 print(f"Итак, тебе {age} лет, в тебе {height} см роста и {weight} кг веса.")
```

Результат выполнения

Сеанс упражнения 12

```
$ python3.6 ex12.py
Сколько тебе лет? 38
Каков твой рост? 185
Сколько ты весишь? 70
Итак, тебе 38 лет, в тебе 185 см роста и 70 кг веса.
```

Практические задания

1. В оболочке командной строки, в которой вы запускаете сценарии Python, введите команду `pydoc input`. Прочитайте показанное

сообщение. Если вы работаете в операционной системе Windows, выполните команду `python3.6 -m pydoc input`.

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

3. Найдите в Интернете 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 то, 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 команду `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 команды `pydoc` будет выдавать ошибку. Вы можете пропустить эти практические задания и выполнить поиск документации по Python в Интернете.

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

Вы можете его использовать, однако в этом случае результат вызова функции `input()` не будет сохранен в переменной, что приведет к неожиданным последствиям. Попробуйте этот способ, а затем выведите на экран введенные данные. Попытайтесь выяснить, почему этот способ не работает.

Параметры, распаковка, переменные

В этом упражнении мы рассмотрим еще один метод ввода, который можно использовать для передачи переменных сценарию (сценарий или скрипт – другое название ваших файлов с расширением *.py*). Как вы уже знаете, набрав команду `python3.6 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, мы затем его рассмотрим:

ex13.py

```
1 from sys import argv
2 # Прочитайте раздел "Результат выполнения", чтобы понять, как
3 # запустить этот скрипт
4 script, first, second, third = argv
5 print("Этот сценарий называется:", script)
6 print("Моя первая переменная называется:", first)
7 print("Моя вторая переменная называется:", second)
8 print("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 аргумента во многих языках программирования. Эта переменная содержит аргументы, которые вы передаете сценарию Python при запуске. В упражнениях далее вы будете работать с ней и разберетесь, как она работает.

Строка 4 распаковывает переменную `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 его и назначь его в качестве значений 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 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».

Внимание! До этого вы запускали сценарии Python без аргументов. Если вы введете только команду `python3.6 ex13.py`, то совершите ошибку! Обратите внимание на то, как я запускаю сценарий. Такой способ запуска необходим всегда, когда используется переменная `argv`.

Результат выполнения

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

Сеанс упражнения 13

```
$ python3.6 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

```
$ python3.6 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

$ python3.6 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 вроде следующего:

Сеанс упражнения 13

```
$ python3.6 ex13.py first 2nd
Traceback (most recent call last):
  File "ex13.py", line 3, in <module>
    script, first, second, third = argv
ValueError: not enough values to unpack (expected 4, got 3)
```

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 не менее 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 объяснить?

2. Напишите сценарий с меньшим количеством аргументов и сценарий, имеющий больше аргументов. Присваивайте переменным подходящие имена.

3. Совместите функцию `input()` с `argv` в сценарии, который требует дополнительный пользовательский ввод. Не переусердствуйте. Используйте переменную `argv` для получения от пользователя одних данных, а функцию `input()` – других.

4. Помните, что модули предоставляют вам возможности программирования. Модули. 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 `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 `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 `int(input())`.

Как использовать оболочку командной строки?

Вы должны уже уметь это делать. В противном случае, см. приложение в конце этой книги.

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

Это просто. Укажите две строки в конце этого сценария, применив функцию `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 `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 `input()` вместе, чтобы спросить у пользователя нечто специфичное. Вам понадобится это для следующего упражнения, в котором мы научимся считывать и записывать файлы. В этом упражнении мы применим функцию `input()` несколько иначе, выводя простое приглашение >. Это похоже на старенькую игру вроде Zork или Adventure.

ex14.py

```
from sys import argv

script, user_name = argv
prompt = '> '

print(f"Привет, {user_name}, я - сценарий {script}.")
print("Я хочу задать тебе несколько вопросов.")
print(f"Я тебе нравлюсь, {user_name}?")
likes = input(prompt)

print(f"Где ты живешь, {user_name}?")
lives = input(prompt)

print("На каком компьютере ты работаешь?")
computer = input(prompt)

print(f"""
Итак, ты ответил {likes} на вопрос, нравлюсь ли я тебе.
Ты живешь {lives}. Не представляю, где это.
И у тебя есть компьютер {computer}. Прекрасно!
""")
```

Мы создаем переменную `prompt`, связанную с нужным нам запросом, и передаем ее функции `input()` вместо постоянного ее ввода. Теперь, если мы хотим запросить другие данные, мы можем быстро изменить их и вновь запустить сценарий. Очень удобно.

Результат выполнения

При запуске сценария не забудьте передать ему свое имя в качестве аргумента переменной `argv`.

Сеанс упражнения 14

```
$ python3.6 ex14.py Михаил
Привет, Михаил, я - сценарий ex14.py.
Я хочу задать тебе несколько вопросов.
Я тебе нравлюсь, Михаил?
> Ну да
Где ты живешь, Михаил?
> в Москве
На каком компьютере ты работаешь?
> Tandy 1000

Итак, ты ответил Ну да 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 нечто совершенно другое.

3. Добавьте еще один аргумент и примените его в сценарии так же, как в предыдущем упражнении использовался код типа `first, second = argv`.

4. Разберитесь, как я объединил многострочный текст `"""` с символами 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 `python3.6`, а затем введете `python3.6 ex14.py Михаил`, то потерпите неудачу, потому попытаетесь запустить Python из уже запущенного Python. Завершите работу Python, а затем введите `python3.6 ex14.py Михаил`.

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 все понять и запомнить.

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.

Как запустить этот сценарий в IDLE?

Не используйте IDLE.

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 запускаю сценарий.

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

УПРАЖНЕНИЕ 15

Чтение файлов

Всего, что вы уже знаете о функции `input()` и переменной `argv`, достаточно для чтения файлов. Возможно, вам понадобится разобраться в этом упражнении, чтобы понять суть, так что выполняйте его тщательно и фиксируйте проблемы и вопросы. Работая с файлами, вы легко можете стереть их содержимое, если не будете соблюдать осторожность.

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, но обычный текстовый документ, содержимое которого мы будем считывать в нашем сценарии. Ниже представлено содержимое этого файла:

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

Все, что мы хотим сделать, это с помощью нашего сценария открыть файл 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 каких-либо 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 `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(f"Содержимое файла {filename}:")
8 print(txt.read())
9
```

```
10 print("Снова введите имя файла:")
11 file_again = input("> ")
12
13 txt_again = open(file_again)
14
15 print(txt_again.read())
```

В этом файле происходит несколько интересных вещей, поэтому давайте его быстренько разберем:

- В строках 1–3 используется переменная `argv`, позволяющая получить имя файла. Далее вы видите строку 5, в которой используется новая команда – `open`. Не откладывая, выполните команду `pydoc open` и прочитайте справку о команде `open`. Обратите внимание на то, как, подобно вашим скриптам и функции `input()`, команда `open` принимает параметр и возвращает значение, которое вы можете присвоить собственной переменной. Таким образом, мы открыли файл.

- Строка 7 выводит коротенький текст, а в строке 8 вы видите что-то совсем новое и интересное. Из переменной `txt` мы вызываем функцию `read()`. В результате выполнения функции `open()` вы получаете файл, к которому также можно применить команду. Вы пишете код, используя `.` (точку), затем имя команды и параметры, по аналогии с функциями `open()` и `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

Внимание! Будьте внимательны! До сих пор для запуска сценария вы использовали только его имя, однако теперь, когда вы применяете переменную `argv`, вам требуется добавить аргументы. Если вы посмотрите на первую строку приведенного ниже примера, вы увидите, что для запуска сценария я использую команду `python ex15.py ex15_sample.txt`. Обратите внимание на дополнительный аргумент `ex15_sample.txt` после имени сценария

`ex15.py`. Если вы не введете его, то получите сообщение об ошибке, поэтому будьте внимательны!

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

Сеанс упражнения 15

```
$ python3.6 ex15.py ex15_sample.txt
Содержимое файла ex15_sample.txt:
У Мэри был маленький барашек
Его шерсть была белой как снег
И всюду, куда Мэри шла, маленький барашек всегда следовал за ней

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

Практические задания

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

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 в Интернете по запросу «Python открытие файлов»

3. В этом упражнении я использовал слово «команды» для обозначения функций (также называемых «методами»). С функциями и методами мы разберемся позднее в этой книге.

4. Удалите строки 10—15, в которых используется функция `input()`, и выполните сценарий снова.

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

6. Выполните команду `python3.6` и попробуйте открыть текстовый файл прямо из командной строки. Изучите, как открывать файлы и читать их содержимое в оболочке командной строки при запущенном Python.

7. 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». Это можно представить как старый ленточный накопитель, который использовался в больших ЭВМ 1950-х годов, или как DVD-проигрыватель сегодня. Вы можете перемещаться по ним, а затем «считывать», но не содержимое.

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

Первым делом, в оболочке командной строки просто введите команду `python3.6` и нажмите клавишу **Enter**. Теперь вы запустили Python, как мы делали это ранее. После этого, вы можете вводить код, и Python будет выполнять его маленькими порциями. Поэкспериментируйте. Чтобы завершить работу, выполните команду `quit()` и нажмите клавишу **Enter**.

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

Python не ограничивает многократное открытие файла, иногда это даже необходимо.

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 и я. Не нужно подставлять имена файлов; вы дадите Python команду подставить имя.

Чтение и запись файлов

Если вы выполнили практические задания из прошлого упражнения, вы должны были увидеть все виды команд (методов/функций) для работы с файлами. Ниже представлен список команд, которые вы должны запомнить:

- `close` – закрывает файл; аналогично команде Файл Þ Сохранить (File Þ Save) в меню оконного редактора;
- `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 стереть важные данные;
- `write(stuff)` – записывает данные в файл;
- `seek(0)` – перемещает указатель текущей позиции чтения/записи в начало файла.

Можно запомнить назначение каждой из команд, представив себе виниловую пластинку, аудио- или видеокассету, DVD или компакт-диск. На заре компьютерной эры данные хранились на подобных носителях, поэтому многие из операций, применяемых к файлам, по-прежнему напоминают операции с линейной системой хранения. Ленточные и DVD-накопители предполагают «нахождение» конкретной позиции чтения/записи. Современные операционные системы и носители данных размывают границу между оперативной памятью и дисковыми накопителями, однако мы по-прежнему представляем себе магнитную ленту, чтение и запись на которой осуществляются с помощью перемещаемой головки.

На данный момент, 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:

ex16.py

```
from sys import argv

script, filename = argv

print(f"Я собираюсь стереть файл {filename}.")
print("Если вы не хотите стирать его, нажмите сочетание клавиш CTRL+C (^C).")
print("Если хотите стереть файл, нажмите клавишу Enter.")

input("?")

print("Открытие файла...")
target = open(filename, 'w')

print("Очистка файла. До свидания!")
target.truncate()

print("Теперь я запрашиваю у вас три строки.")

line1 = input("строка 1: ")
line2 = input("строка 2: ")
line3 = input("строка 3: ")

print("Это я запишу в файл.")

target.write(line1)
target.write("\n")
target.write(line2)
target.write("\n")
target.write(line3)
target.write("\n")

print("И наконец, я закрою файл.")
target.close()
```

Это объемный код, вероятно, самый большой из введенных вами. Поэтому набирайте его 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 следующее. Во-первых, результат работы вашего скрипта:

Сеанс упражнения 16

```
$ python3.6 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 маленький барашек
строка 2: Его шерсть была белой как снег
строка 3: И всюду, куда Мэри шла, маленький барашек всегда следовал за ней
Это я запишу в файл.
И наконец, я закрою файл.
```

Во-вторых, в вашем редакторе (например, Atom) откройте созданный файл (в моем случае, *test.txt*) 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 похожий сценарий, в котором будут использоваться 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, так это или нет.

Распространенные вопросы

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

См. практическое задание N° 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(имяфайла)`, чтобы открыть файл в режиме чтения `'r'`?

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

Еще о файлах

Давайте выполним еще несколько действий с файлами. Мы напишем сценарий Python для копирования содержимого из одного файла в другой. Пример будет очень коротким, но послужит для вас источником идей о других операциях, которые вы можете делать 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(f"Копирование данных из файла {from_file} в файл {to_file}")
 7
 8 # Можете следующие две строки кода разместить в одну?
 9 in_file = open(from_file)
10 indata = in_file.read()
11
12 print(f"Исходный файл имеет размер {len(indata)} байт")
13
14 print(f"Целевой файл существует? {exists(to_file)}")
15 print("Готов, нажмите клавишу Enter для продолжения или CTRL+C для отмены.")
16 input()
17
18 out_file = open(to_file, 'w')
19 out_file.write(indata)
20
21 print("Отлично, все сделано.")
22
23 out_file.close()
24 in_file.close()
```

Вы должны сразу обратить внимание, что мы импортируем функцию по имени `exists`. Она возвращает значение `True`, если файл существует, основываясь на его имени, которое получает в качестве аргумента. Либо возвращает значение `False`, если файл не существует. Мы будем использовать

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

Использование инструкции `import` позволяет получить доступ к огромному разнообразию бесплатного кода, написанного другими (как правило, более опытными) программистами, что может сэкономить вам много времени.

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

```
$ # сначала создаем простой файл
$ echo "Это тестовый файл." > test.txt
$ # теперь просматриваем его
$ cat test.txt
Это тестовый файл.
$ # теперь запускаем наш сценарий с этим файлом
$ python3.6 ex17.py test.txt new_file.txt
Копирование данных из файла test.txt в файл new_file.txt
Исходный файл имеет размер 42 байт
Целевой файл существует? False
Готов, нажмите клавишу Enter для продолжения или CTRL+C для отмены.

Отлично, все сделано.
```

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 в конце книги.

Практические задания

1. 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.

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

3. 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 в качестве простого способа вывода содержимого файла на экран. Выполните команду `man cat`, чтобы прочитать информацию про нее.

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

5. Прочитайте об инструкции `import` и запустите Python, чтобы испытать ее в действии. Попробуйте импортировать что-нибудь и посмотрите, получится ли это у вас. Ничего страшного, если не получится.

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.

Мне кажется, это упражнение слишком трудное.

Это совершенно нормально. Вы можете не «втянуться» в 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 все упражнения, особенно те, с которыми у вас возникают затруднения. Потерпите.

Для чего нужна функция `len()`?

Она получает длину переданной ей строки и возвращает это значение в виде числа. Поэкспериментируйте с этой функцией.

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

Вы, наверное, сделали что-то, вроде этого: `indata = open(from_file).read()`. В таком случае вам не нужна функция `in_file.close()` в конце сценария. Python должен закрыть файл сразу после выполнения этой строки кода.

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

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

Имена, переменные, код, функции

Длинное название для нашего упражнения, не так ли? Я собираюсь познакомить вас с функциями! Каждый программист должен знать о функциях и о том, как они работают и для чего предназначены. Самое простое объяснение вы получите прямо сейчас. Функции выполняют три вещи:

1. Они присваивают имена фрагментам кода так, как переменные именуют строки и числа.
2. Они принимают аргументы, как 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, как они связаны между собой.

ex18.py

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

# ок, здесь вместо *args мы делаем следующее
def print_two_again(arg1, arg2):
    print(f"arg1: {arg1}, arg2: {arg2}")

# принимает только один аргумент
def print_one(arg1):
    print(f"arg1: {arg1}")

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

```
16     print("А я ничего не получаю.")
17
18
19 print_two("Михаил","Райтман")
20 print_two_again("Михаил","Райтман")
21 print_one("Первый!")
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`[3].

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 довольно непростой способ создать функцию. В Python мы можем пропустить распаковку аргументов и указать желаемые имена прямо внутри скобок `()`. Этот прием демонстрирует функция `print_two_again`.

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

---

[3] От англ. define – определение. *(Прим. перев.)*.

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

Внимание! Это очень важно. Не расстраивайтесь, если сейчас 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, то должны увидеть следующее:

Сеанс упражнения 18

```
$ python3.6 ex18.py
arg1: Михаил, arg2: Райтман
arg1: Михаил, arg2: Райтман
arg1: Первый!
А я ничего не получаю.
```

Это наглядно демонстрирует работу функций. 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. Все ли отступы строк кода внутри функции состоят из четырех пробелов? Ни больше, ни 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 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 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.

УПРАЖНЕНИЕ 19

Функции и переменные

Функции могут показаться очень сложными, но не стоит волноваться. Просто продолжайте выполнять упражнения, учитывайте контрольный список из упражнения 18, и в конечном счете 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 этом упражнении мы решим эту проблему:

ex19.py

```
def cheese_and_crackers(cheese_count, boxes_of_crackers):
    print(f"У нас есть {cheese_count} сырков!")
    print(f"У нас есть {boxes_of_crackers} пачек чипсов!")
    print("Этого достаточно для вечеринки!")
    print("Погнали!\n")


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


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


print("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 передать функции значения. Также можем передать ей переменные. Или же передать вычисления. И, разумеется, объединить вычисления и переменные.

В каком-то смысле, аргументы функции похожи на символ =, используемый при создании переменных. По факту, если вы можете использовать символ =, чтобы присвоить имя значению, 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

```
$ python3.6 ex19.py
Мы можем непосредственно передать числа функции:
У нас есть 20 сырков!
У нас есть 30 пачек чипсов!
Этого достаточно для вечеринки!
Погнали!

ИЛИ, мы можем использовать переменные из нашего сценария:
У нас есть 10 сырков!
У нас есть 50 пачек чипсов!
Этого достаточно для вечеринки!
Погнали!

Мы даже можем выполнять вычисления внутри функции:
У нас есть 30 сырков!
У нас есть 11 пачек чипсов!
Этого достаточно для вечеринки!
Погнали!

И объединять переменные с вычислениями:
У нас есть 110 сырков!
У нас есть 1050 пачек чипсов!
Этого достаточно для вечеринки!
Погнали!
```

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 ли 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. Подойдите творчески.

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 она делает. Еще одна уловка – читать код вслух. Или можно распечатать код, а рядом писать комментарии и зарисовывать процесс выполнения.

Как поступить, если я хочу запросить у пользователя количество сырков и чипсов?

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

Приводит ли создание переменной `amount_of_cheese` к изменению переменной `cheese_count` в функции?

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

Это плохо, если глобальные переменные (например, `amount_of_cheese`) имеют то же имя, что и переменные внутри функции?

Да, поскольку в этом случае вы не вполне уверены в том, о какой именно переменной говорите. Однако иногда у вас может возникать необходимость в использовании одного и того же имени или так может получиться случайно. Просто по возможности постарайтесь этого избегать.

Существует ли предельное количество аргументов, которые может иметь функция?

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, далее в книге вы создадите игру, в которой используется этот прием.

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

```
from sys import argv

script, input_file = argv

def print_all(f):
    print(f.read())

def rewind(f):
    f.seek(0)

def print_a_line(line_count, f):
    print(line_count, f.readline())

current_file = open(input_file)

print("Первым делом выведем этот файл целиком:\n")

print_all(current_file)

print("Теперь отмотаем назад, словно это кассета.")

rewind(current_file)

print("Выведем три строки:")

current_line = 1
print_a_line(current_line, current_file)

current_line = current_line + 1
print_a_line(current_line, current_file)

current_line = current_line + 1
```

```
33 print_a_line(current_line, current_file)
```

Обратите особое внимание на то, как мы передаем номер текущей строки при каждом вызове функции `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_python3.zip**.

Результат выполнения

Сеанс упражнения 20

```
$ python3.6 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 все вызовы функций и проверьте их объявление (`def`), чтобы убедиться, что вы передаете 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. Подробности вы узнаете позже.

Почему запуск функции `seek(0)` не присваивает переменной `current_line` значения 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`.

Откуда функции `readline()` известно, где находится каждая строка?

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.

Почему отображаются пустые строки между строками в файле?

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`.

УПРАЖНЕНИЕ 21

Что возвращают функции

Ранее вы использовали символ =, чтобы именовать переменные и присваивать им значения в виде чисел или строк. Теперь мы продолжим использовать оператор = и научимся присваивать переменным значения, получаемые от функции, с помощью нового ключевого слова `return` языка Python. Кое на что потребуется обратить пристальное внимание, но об этом позже. Сначала наберите следующий код:

ex21.py

```
def add(a, b):
    print(f"СЛОЖЕНИЕ {a} + {b}")
    return a + b

def subtract(a, b):
    print(f"ВЫЧИТАНИЕ {a} - {b}")
    return a - b

def multiply(a, b):
    print(f"УМНОЖЕНИЕ {a} * {b}")
    return a * b

def divide(a, b):
    print(f"ДЕЛЕНИЕ {a} / {b}")
    return a / b


print("Давайте выполним несколько вычислений с помощью функций!")

age = add(30, 5)
height = subtract(190, 4)
weight = multiply(35, 2)
iq = divide(250, 2)

print(f"Возраст: {age}, Рост: {height}, Вес: {weight}, IQ: {iq}")


# Головоломка в качестве дополнительного задания, введите код в любом случае.
print("Это головоломка.")
```

```
30
31 what = add(age, subtract(height, multiply(weight, divide(iq, 2))))
32
33 print("Получается: ", what, "Вы можете это вычислить вручную?")
```

Мы создали функции для математических операций сложения, вычитания, умножения и деления. Обратите внимание на последнюю строку кода функции `add`, в которой указано `return a + b`. В ней происходит следующее:

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 далее в коде.

Как и в случае многих других понятий в этой книге, вы должны разбираться в этой теме медленно и досконально. Чтобы лучше разобраться, попробуйте решить дополнительную головоломку и узнать кое-что интересное.

Результат выполнения

Сеанс упражнения 21

```
$ python3.6 ex21.py
Давайте выполним несколько вычислений с помощью функций!
СЛОЖЕНИЕ 30 + 5
ВЫЧИТАНИЕ 190 - 4
УМНОЖЕНИЕ 35 * 2
ДЕЛЕНИЕ 250 / 2
Возраст: 35, Рост: 186, Вес: 70, IQ: 125.0
Это головоломка.
ДЕЛЕНИЕ 125.0 / 2
УМНОЖЕНИЕ 70 * 62.5
ВЫЧИТАНИЕ 186 - 4375.0
```

```
СЛОЖЕНИЕ 35 + -4189.0
Получается:  -4154.0 Вы можете это вычислить вручную?
```

Практические задания

1. Если вы не до конца понимаете, как работает команда `return`, попробуйте написать несколько собственных функций, возвращающих 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 выяснить нормальную формулу, которая воссоздавала бы тот же набор операций.

3. После того как получите формулу 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, относясь к нему как к игре. Решение головоломки типа этой превращает программирование в удовольствие, поэтому я и в дальнейшем буду давать вам небольшие задачи по ходу книги.

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 я могу использовать функцию `input()`, чтобы указать собственные значения?

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

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

Для начала возьмите выражение `24 + 34/100 - 1023`. Преобразуйте его так, чтобы можно было использовать функции. Теперь придумайте собственные математические выражения 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 их название (имя) и что они делают. Если вы не можете найти имя символа в этой книге, выполните поиск в Интернете. Если вы не знаете, для чего используется то или иное ключевое слово или символ, поищите о нем справочные сведения и попробуйте использовать его в коде.

Вы можете встретить слова или символы, информацию о которых не сможете найти, или не сможете понять, как они работают. В этом случае, сохраните эти 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 может показаться 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 по 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 проще.

УПРАЖНЕНИЕ 23

Строки, байты и кодировки символов

Для выполнения этого упражнения вам потребуется скачать с сайта **https://eksmo.ru/files/shaw_python3.zip** созданный автором текстовый файл под названием *languages.txt* (вы его найдете в папке *Examples\ex23*). Этот файл, содержащий список языков мира, был создан для демонстрации следующих концепций:

1. Как современные компьютеры хранят языки для отображения и обработки текста, и как в Python 3 для этого используются так называемые строки.

2. Как «кодировать» и «декодировать» строки Python в байты.

3. Как справляться с ошибками, возникающими при работе со строками и байтами.

4. Как читать и понимать код, даже если вы видите его впервые.

Кроме того, мы кратко коснемся инструкции `if` и структуры данных `list`, предусмотренных в Python 3 для обработки списков элементов. Вам не нужно досконально осваивать этот код или сразу же вникать в эти концепции. Вы попрактикуетесь в следующих упражнениях. На данный момент вам нужно получить представление о том, что предстоит пройти, и изучить четыре темы из предыдущего списка.

Внимание! Это трудное упражнение! В нем будет много информации, которая касается глубинных принципов работы компьютера. Трудность этого упражнения также связана со сложностью понятия строк и их использования в Python. Я рекомендую вам пройти это упражнение максимально медленно. Выпишите все незнакомые вам слова и выясните их значения. При необходимости изучайте по одному абзацу за раз. Чтобы не терять время, параллельно с этим вы можете выполнять другие упражнения. Работайте над ним столько, сколько потребуется.

Предварительное исследование

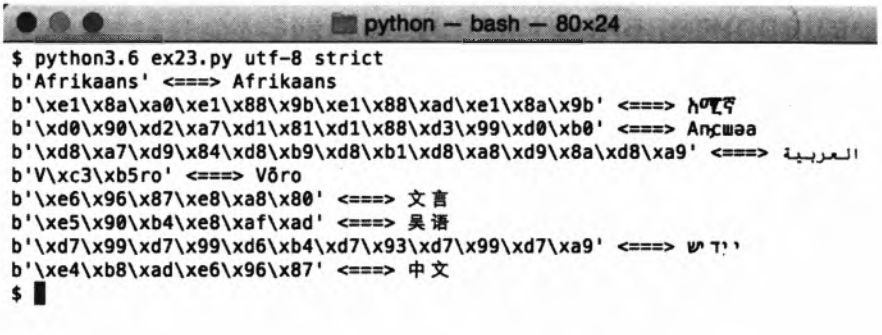
Сейчас я покажу вам, как исследовать фрагмент кода для выявления его секретов. Чтобы этот код сработал, вам понадобится файл *languages.txt*, поэтому убедитесь в том, что вы его скачали. Этот файл содержит список названий языков мира в кодировке UTF-8.

ex23.py

```
import sys
script, encoding, error = sys.argv


def main(language_file, encoding, errors):
    line = language_file.readline()

    if line:
        print_line(line, encoding, errors)
        return main(language_file, encoding, errors)


def print_line(line, encoding, errors):
    next_lang = line.strip()
    raw_bytes = next_lang.encode(encoding, errors=errors)
    cooked_string = raw_bytes.decode(encoding, errors=errors)

    print(raw_bytes, "<===>", cooked_string)


languages = open("languages.txt", encoding="utf-8")

main(languages, encoding, error)
```

Составьте список всего, что вы видите впервые. Таких элементов может оказаться довольно много, поэтому просмотрите файл несколько раз.

После этого запустите сценарий и поэкспериментируйте с ним. Вот часть команд, которые я использовал для тестирования:



Внимание! Как видите, здесь я использую изображения для демонстрации того, что должно отобразиться на вашем экране. После проведенного исследования выяснилось, что у многих на компьютере не отображается текст в кодировке UTF-8, поэтому я использовал изображения, чтобы вам было понятно, чего ожидать. Даже моя собственная система верстки (LaTeX) не способна обрабатывать текст в этой кодировке, из-за чего мне пришлось использовать изображения. Если вы не видите этого, скорее всего, ваша оболочка командной строки Python не отображает текст в кодировке UTF-8, и вам следует постараться исправить эту проблему.

В приведенных примерах используются кодировки `utf-8`, `utf-16` и `big5` для демонстрации преобразования и типов ошибок, с которыми вы можете столкнуться. В Python 3 все вышеупомянутое называется «кодеками» (codec), однако мы используем параметр `encoding` (кодировка). В конце этого упражнения приведен список доступных кодировок на случай, если вам захочется поэкспериментировать. Чуть позже я объясню, что означают выведенные на экран данные. Пока просто постарайтесь уловить принцип работы, чтобы мы могли продолжить обсуждение.

После нескольких запусков сценария просмотрите список символов и постарайтесь угадать, что делает каждый из них. Запишите свои догадки и проверьте их, выполнив поиск в Интернете. Не волнуйтесь, если вы понятия не имеете о том, как их искать. Просто попробуйте.

Переключатели, общепринятые обозначения и кодировки

Прежде чем углубляться в этот код, вам следует получить базовое представление о том, как данные хранятся на компьютере. Современные компьютеры невероятно сложны, однако на базовом уровне они подобны огромному массиву переключателей. С помощью электричества компьютеры включают или выключают эти переключатели. Включенный переключатель может соответствовать `1`, а выключенный – `0`. Единица – это энергия, электричество, мощность, содержание, состояние «включено». Ноль – означает выключено, готово, прошло, отключено, обесточено. Эти единицы и нули называются битами.

Компьютер, который позволяет работать только с `1` и `0`, был бы ужасно неэффективным и невероятно раздражающим. Однако при помощи `1` и `0` компьютеры кодируют большие числа. Компьютеру требуется 8 единиц и нулей для кодирования 256 чисел (`0–255`). Что же такое кодировка? Это всего лишь общепринятый стандарт представления числа в виде последовательности битов. Это придуманная людьми система общепринятых обозначений, в которой, например, `00000000` означает `0`, `11111111` – `255`, а `00001111` – `15`. На заре компьютерной эры велись длинные дискуссии даже насчет того, каким должен быть порядок этих битов, поскольку даже в этом вопросе были разногласия.

Сегодня последовательность из 8 битов (`1` и `0`) мы называем «байтом». Раньше у каждого был свой взгляд на то, что следует считать байтом, поэтому вы все еще можете встретить людей, считающих, что это понятие должно быть гибким, а байт должен состоять из 9, 7 или 6 битов, однако сейчас мы просто говорим, что байт соответствует 8 битам. Это наше соглашение, которое определяет кодировку байта. Для кодирования больших чисел есть и другие соглашения, предполагающие использование 16, 32, 64 и даже большего количества битов, если вам необходимо работать с действительно большими числами. Существуют целые группы людей, которые занимаются только тем, что спорят об этих стандартах, а затем реализуют их в качестве кодировок, которые, в конечном счете, управляют состоянием переключателей внутри компьютера.

Определившись с байтами, мы можем хранить и отображать текст, решив перед этим вопрос кодирования букв с помощью чисел. На заре компьютерной эры существовало множество соглашений относительно того, как 8 или 7 (плюс-минус) битов можно поставить в соответствие символам из хранившегося в компьютере списка. Наибольшую популярность приобрел Американский стандартный код для обмена информацией (American Standard Code for Information Interchange) или ASCII. Этот стандарт предполагает соответствие

чисел буквам. Число `90` соответствует букве `Z`. В битах это число выражается как `1011010` и сопоставляется с таблицей ASCII, хранящейся в компьютере.

Прямо сейчас вы можете ввести следующий код на языке Python:

```
>>> 0b1011010
90
>>> ord('Z')
90
>>> chr(90)
'Z'
>>>
```

Сначала я записываю число `90` в двоичном формате, далее я получаю число, соответствующее букве `'Z'`, а затем преобразую число в букву `'Z'`. Не старайтесь все это запомнить. Кажется, мне пришлось делать это всего лишь дважды за все время использования Python.

Имея таблицу ASCII для кодирования символов при помощи 8 битов (байта), мы можем «составлять» из них слова. Таким образом, если я захочу написать свое имя, Zed A. Shaw, я просто воспользуюсь последовательностью байтов: `[90, 101, 100, 32, 65, 46, 32, 83, 104, 97, 119]`. Большинство текстов на первых компьютерах представляли собой просто последовательность байтов, хранящуюся в памяти, которую компьютер использовал для вывода текста на экран. Повторяю, что это просто принятая всеми последовательность, которая обусловливает состояние переключателей.

Проблема кодировки ASCII заключается в том, что она кодирует только буквы английского алфавита и, возможно, нескольких родственных ему языков. Помните о том, что байт может кодировать 256 чисел (`0-255` или `00000000-11111111`). Однако количество символов, используемых в языках мира, намного превышает число 256. Разные страны создали собственные кодировки для своих языков, однако, несмотря на свою работоспособность, большинство из таких кодировок поддерживают лишь один язык. Это означает, что при необходимости вставить английское название книги в предложение на тайском языке вы столкнетесь с проблемой. Для этого вам потребуется одна кодировка для тайского языка и еще одна для английского.

Для решения этой проблемы была создана кодировка Unicode. Ее название созвучно со словом «encode» (кодировать), и она призвана служить «универсальной кодировкой» для всех языков. По своей сути, Unicode представляет собой то же самое, что и таблица ASCII, отличаясь значительно увеличенным

объемом. Для кодирования символа в системе Unicode может использоваться 32 бита, что многократно превышает количество существующих символов. Использование 32-битных значений позволяет хранить 4,294,967,295 символов (2^32), чего вполне достаточно для всех языков Земли, а, возможно, и всей Вселенной. В настоящее время мы используем свободное пространство для таких важных вещей, как язык идеограмм и смайликов.

Теперь у нас есть возможность закодировать любой нужный нам символ, однако 32 бита — это 4 байта (32/8 == 4), а значит, у нас остается слишком много невостребованного места в любом тексте, который необходимо закодировать. Мы также можем воспользоваться 16 битами (2 байта), но даже в этом случае у нас останется слишком много свободного места в большей части текстов. Решение заключается в том, чтобы использовать 8-битную кодировку, а при необходимости кодирования большего количества символов применять расширенную кодировку. Это означает, что у нас появляется еще одна общепринятая кодировка, которая представляет собой своеобразную сокращенную версию, позволяющую использовать 8 бит для кодирования большинства распространенных символов, а при необходимости прибегать к 16 или 32-битной кодировке.

В языке Python для кодирования текста используется кодировка UTF-8, что расшифровывается как Unicode Transformation Format 8 Bits (Формат преобразования Юникода, 8-бит). Это формат преобразования символов Unicode в последовательность байтов, представляющую собой последовательность битов, которая управляет состоянием переключателей. Вы можете использовать и другие кодировки, однако стандартом в настоящее время является UTF-8.

Анализ результата выполнения кода

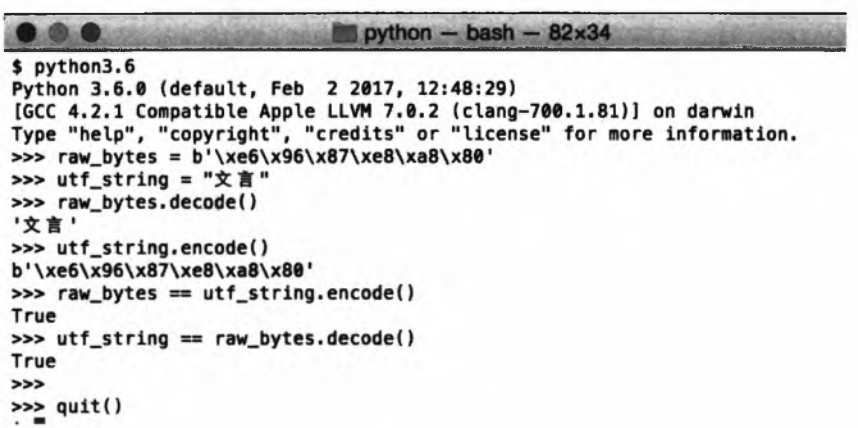
Теперь давайте проанализируем результат выполнения приведенных ранее команд. Возьмем только первую команду и первые несколько строк, выведенных на экран:

```
python — bash — 80×24
$ python3.6 ex23.py utf-8 strict
b'Afrikaans' <===> Afrikaans
b'\xe1\x8a\xa0\xe1\x88\x9b\xe1\x88\xad\xe1\x8a\x9b' <===> አማርኛ
b'\xd0\x90\xd2\xa7\xd1\x81\xd1\x88\xd3\x99\xd0\xb0' <===> Аҧсшәа
b'\xd8\xa7\xd9\x84\xd8\xb9\xd8\xb1\xd8\xa8\xd9\x8a\xd8\xa9' <===> العربية
b'V\xc3\xb5ro' <===> Võro
b'\xe6\x96\x87\xe8\xa8\x80' <===> 文言
b'\xe5\x90\xb4\xe8\xaf\xad' <===> 吴语
b'\xd7\x99\xd7\x99\xd6\xb4\xd7\x93\xd7\x99\xd7\xa9' <===> ייִדיש
b'\xe4\xb8\xad\xe6\x96\x87' <===> 中文
$
```

Сценарий *ex23.py* берет байты из байтовой строки `b''` и преобразует их в UTF-8 или другую выбранную вами кодировку. Слева приводятся числа, соответствующие каждому байту UTF-8 (в шестнадцатеричном формате), а справа — символы кодировки UTF-8. Другими словами, слева от <===> находятся числовые или «необработанные» байты, используемые в Python для хранения строки. Чтобы интерпретатор Python обрабатывал их как байты, мы используем символы `b''`. Затем эти необработанные байты отображаются справа в обработанном виде, и вы наблюдаете в своем терминале соответствующие им символы.

Анализ кода

Мы уже имеем представление о строках и последовательностях байтов. В языке Python строка (`string`) – это закодированная с использованием кодировки UTF-8 последовательность символов для отображения текста или работы с ним. Байты (`bytes`) представляют собой необработанную последовательность байтов, которую Python использует для хранения этой строки UTF-8. Чтобы сообщить интерпретатору Python о том, что вы работаете с необработанными байтами, в начале такой строки нужно использовать символы `b'`. Все это основано на соглашениях относительно обработки текста в Python. Вот пример того, как Python кодирует строки и декодирует байты.



Запомните: если у вас есть необработанные байты, вы должны использовать функцию `.decode()` для того, чтобы в результате получить строку. Необработанные байты представляют собой простую последовательность байтов — обычных чисел, поэтому вам необходимо сообщить Python о необходимости «преобразовать ее в строку UTF». Если у вас есть строка, которую вы хотите отправить, сохранить, поделиться ею или выполнить над ней какую-либо другую операцию, скорее всего, у вас это получится, но иногда Python будет

выдавать ошибку, сообщающую о том, что он не знает как «закодировать» вашу строку. Опять же, Python имеет встроенный стандарт кодировки, но не знает о том, какой именно стандарт нужен вам. В таком случае, для получения желаемого результата вам нужно использовать функцию `.encode()`.

Чтобы это запомнить (хотя я практически всегда пользуюсь подсказкой), можно использовать аббревиатуру DBES (Decode Bytes, Encode Strings), что означает Декодирование Байтов, Кодирование Строк. Когда у вас есть байты, а вам нужна строка, декодируйте байты. Если у вас есть строка, и вам нужно преобразовать ее в байты, закодируйте строку.

Давайте разберем код сценария *ex23.py* строку за строкой:

1–2 Обычная обработка аргумента командной строки, о которой вы уже знаете.

5 Здесь я начинаю вводить основную часть кода в функции, которая называется `main`. Она будет вызываться в конце сценария.

6 Первое, что делает данная функция, – это считывает одну строку из файла со списком языков, который мы ей передали. Вы уже делали это раньше, поэтому ничего нового для вас здесь быть не должно. Это обычная функция `readline()`, используемая при работе с текстовыми файлами.

8 Здесь я использую кое-что новенькое. Вы будете изучать это во второй половине книги, так что считайте это своеобразным анонсом к тому, с чем вам предстоит познакомиться. Это инструкция `if`, которая позволяет принимать решения в коде Python. Вы можете проверить переменную на истинность, на основании чего выполнить фрагмент кода, либо пропустить его. В данном примере я проверяю, имеет ли переменная `line` какое-либо содержимое. Функция `readline()` возвращает пустую строку по достижении конца файла, а фрагмент кода `if line` просто проверяет наличие этой пустой строки. Если функция `readline()` возвратит нам значение, условие окажется истинным, и будет выполнен код под ней (текст с отступом в строках 9–10). Если условие окажется ложным, интерпретатор Python пропустит строки 9–10.

9 Для вывода этой строки на экран я вызываю отдельную функцию. Это упрощает мой код и делает его более простым для понимания. Если я захочу узнать о том, как работает эта функция, я могу перейти к ней и подробнее ее изучить. Как только я узнаю, что делает функция `print_line()`, мне можно будет просто запомнить ее название и не думать о подробностях.

10 Здесь «происходит волшебство». Я снова вызываю функцию `main()` внутри функции `main()`. На самом деле это вовсе не волшебство, поскольку в программировании нет никакой магии. Вам доступна вся необходимая информация. Похоже, я вызываю функцию внутри нее самой, что может показаться недопустимым. Спросите себя, почему это должно быть недопустимым? Не существует никаких технических препятствий для вызова в этом месте какой-либо функции, даже функции `main()`. Если функция предполагает переход к началу сценария, где было определено ее имя, то вызов этой функции в конце самой себя приведет просто... к возврату в самое начало и очередному ее запуску. Это создаст цикл. Теперь вернитесь к строке 8, и вы увидите инструкцию `if`, благодаря которой этот цикл не будет выполняться вечно. Постарайтесь разобраться в этой концепции как можно тщательнее, однако не расстраивайтесь, если она покажется вам не вполне понятной.

13 Здесь начинается определение функции `print_line()`, которая отвечает за кодирование каждой строки из файла *languages.txt*.

14 Здесь я просто удаляю символы `\n` в конце строки `line`.

15 Я наконец-то беру название языка, полученное из файла *languages.txt*, и преобразую его в необработанные байты. Помните аббревиатуру DBES: декодирую байты, кодирую строки. Переменная `next_lang` представляет собой строку, поэтому, для получения необработанных байтов мне нужно вызвать функцию `.encode()` и закодировать строки. Я передаю функции `encode()` информацию о нужной мне кодировке и о том, как следует обрабатывать ошибки.

16 На данном этапе происходит операция, обратная той, которую выполняет код в строке 15, — я создаю переменную `cooked_string` из `raw_bytes`. Помните, что согласно принципу DBES, мы декодируем байты, а `raw_bytes` - это байты, поэтому я вызываю функцию `.decode()`, чтобы получить строку Python. Эта строка должна совпадать с переменной `next_lang`.

18 Я просто вывожу обе переменные на экран, чтобы вы увидели, как они выглядят.

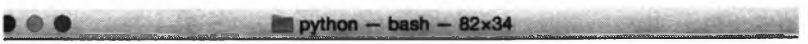
21 Я закончил с определением функций, и теперь мне нужно открыть файл *languages.txt*.

23 В конце сценария вызывается функция `main()` со всеми правильными параметрами, чтобы запустить работу цикла. Помните о том, что это означает возврат к строке 5, где была определена функция `main()`, а в строке 10

функция `main()` вызывается снова, создавая цикл. Благодаря содержащемуся в строке 8 фрагменту кода `if line:` наш цикл не будет работать вечно.

Углубляемся в кодирование

Теперь мы можем использовать наш небольшой сценарий для изучения других кодировок. На следующем изображении показан результат использования различных кодировок и того, как их можно «повредить». Сначала я применяю кодировку UTF-16 для того, чтобы вы увидели, чем она отличается от UTF-8. Вы также можете использовать кодировку UTF-32, чтобы увидеть, насколько больше места она занимает, и получить представление о пространстве, которое экономится с помощью кодировки UTF-8. После этого я пробую кодировку Big5, что, как видно, Python совсем не одобряет. Он выводит сообщение об ошибке, согласно которому Big5 не может кодировать некоторые символы в позиции 0 (это очень полезное сообщение). Одно из решений — дать Python задание «заменить» любые неподходящие для кодировки Big5 символы. Это уже следующий пример, демонстрирующий, как интерпретатор Python помещает символ ? везде, где находит символ, не соответствующий системе кодирования Big5.




```
 python3.6 ex23.py utf-16 strict
'\xff\xfeA\x00f\x00r\x00i\x00k\x00a\x00a\x00n\x00s\x00' <===> Afrikaans
'\xff\xfe\xa0\x12\x1b\x12-\x12\x9b\x12' <===> ᅠᄚᄛ
'\xff\xfe\x10\x04\xa7\x04A\x04H\x04\xd9\x040\x04' <===> Anɕɯəa
"\xff\xfe'\x06D\x069\x061\x06(\x06J\x06)\x06" <===> العربية
'\xff\xfeV\x00\xf5\x00r\x00o\x00' <===> Võro
'\xff\xfe\x87e\x00\x8a' <===> 文言
'\xff\xfe4T\xed\x8b' <===> 吳语
'\xff\xfe\xd9\x05\xd9\x05\xb4\x05\xd3\x05\xd9\x05\xe9\x05' <===> ש ך''
'\xff\xfe-N\x87e' <===> 中文
 python3.6 ex23.py big5 strict
'Afrikaans' <===> Afrikaans
raceback (most recent call last):
 File "ex23.py", line 23, in <module>
   main(languages, encoding, error)
 File "ex23.py", line 10, in main
   return main(language_file, encoding, errors)
 File "ex23.py", line 9, in main
   print_line(line, encoding, errors)
 File "ex23.py", line 15, in print_line
   raw_bytes = next_lang.encode(encoding, errors=errors)
nicodeEncodeError: 'big5' codec can't encode character '\u12a0' in position 0: il
egal multibyte sequence
 python3.6 ex23.py big5 replace
'Afrikaans' <===> Afrikaans
'????' <===> ????
'??\xc7\xda\xc7\xe1?\xc7\xc8' <===> ??cɯ?a
'???????' <===> ???????
'V?ro' <===> V?ro
'\xa4\xe5\xa8\xa5' <===> 文言
'??' <===> ??
'??????' <===> ??????
'\xa4\xa4\xa4\xe5' <===> 中文
```

Ломаем код

Вот некоторые идеи:

1. Найти строки текста, закодированные с помощью других кодировок, и поместить их в файл *ex23.py*, чтобы увидеть, к каким ошибкам это приведет.

2. Выяснить, что произойдет при использовании несуществующих кодировок.

3. Усложняем задание: перепишите код, используя байты `b''` вместо строк UTF-8, создав сценарий, решающий, фактически, противоположную задачу.

4. Если вы сможете это сделать, значит, сможете и «сломать» сценарий, убрав некоторые байты, чтобы посмотреть, к чему это приведет. Сколько байтов нужно удалить, чтобы Python выдал сообщение об ошибке? Сколько байтов нужно удалить, чтобы нарушить процесс вывода строки, но пройти систему декодирования Python?

5. Используйте то, что вы узнали при выполнении пункта 4, чтобы попробовать повредить файлы. С какими ошибками вы столкнулись? При каких повреждениях файл все еще может пройти систему декодирования Python?

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

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

print("--------------------------------")
print(poem)
print("--------------------------------")


five = 10 - 2 + 3 - 6
print(f"Здесь должна быть пятерка: {five}")

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


start_point = 10000
```

```
30 beans, jars, crates = secret_formula(start_point)
31
32 # помните, что это еще один способ форматирования строки
33 print("Начиная с: {}".format(start_point))
34 # так же, как со строкой f""
35 print(f"У нас есть {beans} бобов, {jars} банок и {crates} ящиков.")
36
37 start_point = start_point / 10
38
39 print("Мы также можем сделать это таким образом:")
40 formula = secret_formula(start_point)
41 # простой способ применить список к форматируемой строке
42 print("У нас есть {} бобов, {} банок и {} ящиков.".format(*formula))
```

Результат выполнения

Сеанс упражнения 24

```
$ python3.6 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, 5000.0 банок и 50.0 ящиков.
Мы также можем сделать это таким образом:
У нас есть 500000.0 бобов, 500.0 банок и 5.0 ящиков.
PS C:\Users\zed\temp\examples>
```

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 значение другой переменной 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 Всемирной паутины.

УПРАЖНЕНИЕ 25

И еще практика

Попрактикуемся в применении функций и переменных, чтобы убедиться, что вы их хорошо знаете. Это упражнение должно быть понятным для вас. Тем не менее, оно будет немного отличаться от предыдущих. Вы не будете запускать программу целиком. Вместо этого вы импортируете сценарий в Python и запустите функции вручную.

ex25.py

```
def break_words(stuff):
    """Эта функция разбирает текст на слова."""
    words = stuff.split(' ')
    return words

def sort_words(words):
    """Сортирует слова."""
    return sorted(words)

def print_first_word(words):
    """Выводит первое слово после извлечения."""
    word = words.pop(0)
    print(word)

def print_last_word(words):
    """Выводит последнее слово после извлечения."""
    word = words.pop(-1)
    print(word)

def sort_sentence(sentence):
    """Принимает целое предложение и возвращает отсортированные слова."""
    words = break_words(sentence)
    return sort_words(words)

def print_first_and_last(sentence):
    """Выводит первое и последнее слова предложения."""
    words = break_words(sentence)
    print_first_word(words)
    print_last_word(words)

```

```
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)
```

Во-первых, попробуйте запустить сценарий, как обычно, с помощью команды `python3.6 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 для вычислений. После выполнения команды `python3.6` в оболочке командной строки вы увидите примерно следующее:

```
$ python3.6
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

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

Сеанс упражнения 25 Python

```
1 import ex25
2 sentence = "ты увидишь лучшее в мире привидение с мотором."
3 words = ex25.break_words(sentence)
4 words
5 sorted_words = ex25.sort_words(words)
6 sorted_words
7 ex25.print_first_word(words)
8 ex25.print_last_word(words)
9 words
```

```
10 ex25.print_first_word(sorted_words)
11 ex25.print_last_word(sorted_words)
12 sorted_words
13 sorted_words = ex25.sort_sentence(sentence)
14 sorted_words
15 ex25.print_first_and_last(sentence)
16 ex25.print_first_and_last_sorted(sentence)
```

В моем примере это выглядит следующим образом:

Сеанс упражнения 25 Python

```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import ex25
>>> sentence = "ты увидишь лучшее в мире привидение с мотором."
>>> words = ex25.break_words(sentence)
>>> words
['ты', 'увидишь', 'лучшее', 'в', 'мире', 'привидение', 'с', 'мотором.']
>>> sorted_words = ex25.sort_words(words)
>>> sorted_words
['в', 'лучшее', 'мире', 'мотором.', 'привидение', 'с', 'ты', 'увидишь']
>>> ex25.print_first_word(words)
ты
>>> ex25.print_last_word(words)
мотором.
>>> words
['увидишь', 'лучшее', 'в', 'мире', 'привидение', 'с']
>>> ex25.print_first_word(sorted_words)
в
>>> ex25.print_last_word(sorted_words)
увидишь
>>> sorted_words
['лучшее', 'мире', 'мотором.', 'привидение', 'с', 'ты']
>>> sorted_words = ex25.sort_sentence(sentence)
>>> sorted_words
['в', 'лучшее', 'мире', 'мотором.', 'привидение', 'с', 'ты', 'увидишь']
>>> ex25.print_first_and_last(sentence)
ты
мотором.
>>> ex25.print_first_and_last_sorted(sentence)
в
```

```
увидишь
```

По мере выполнения и изучения каждой из этих строк кода, убедитесь, что вы понимаете, какие функции выполняются в сценарии *ex25.py*. Если ваш результат существенно отличается от моего, или у вас возникают ошибки, вам следует отладить свой код, завершить работу Python и начать заново.

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, и позднее вы узнаете подробнее о них.

3. Многократный ввод `ex25.` может утомить. Избежать этого можно при помощи строки импорта вроде `from ex25 import *`, которую можно расшифровать как «импортируй все из модуля ex25». Начните заново и убедитесь, что это предоставляет вам быстрый доступ к функциям.

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` при вводе некоторых функций.

Вы, наверное, указали функцию, в коде которой отсутствует команда `return` в конце. Прочитайте код файла в обратном порядке, как я вас 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` без расширения.

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 давать ей команды, в ней будут сохраняться результаты этих команд. Это похоже на то, как работали файлы и 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. Программируя, 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 вероятность появления ошибок в его коде. 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 вам сообщит сам интерпретатор Python. В то же время, некоторые ошибки являются математическими, и вы должны найти их самостоятельно. К другим ошибкам относятся недостатки форматирования и опечатки в строках кода.

Все эти ошибки очень распространены; их совершают все программисты. В том числе и очень опытные.

В этом упражнении ваша задача заключается в исправлении кода предоставленного 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 и, если вы затруднились 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_python3.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 все его возможности.

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, а также узнали довольно много математических возможностей Python.

Теперь вы будете изучать логику. Вам не придется постигать сложные теории, которые так любят ученые. Речь идет о базовой логике, которой достаточно для работы обычных программ. Программисты используют ее каждый день.

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

Я предупреждаю, что 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[4]) на другой. Затем вы вытаскиваете карточку, смотрите на логическую операцию и должны немедленно сказать: «Истина!» или «Ложь!». Продолжайте практиковаться, пока выучите все таблицы.

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

[4] True или False. – *Прим. перев.*

чтобы 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 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 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 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. Найду операторы сравнения на равенство и проверю их истинность:
    1. `3 != 4` истинно: `True and not ("testing" != "test" or "Python" == "Python")`
    2. `"testing" != "test"` истинно: `True and not (True or "Python" == "Python")`

    3. `"Python" == "Python"`: `True and not (True or True)`

2. Найду операции `and`/`or` внутри скобок и решу их:

    1. `(True or True)` истинно: `True and not (True)`

3. Найду все операции `not and` и обращу их:

    1. `not (True)` ложно: `True and False`

4. Найду все оставшиеся операции `and`/`or` и решу их:

    1. `True and False` ложно

Решив все это, в итоге я получу 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 продемонстрированной 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:

```
$ python3.6
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit
(Intel)] on win32
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 название 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 и 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'`, или `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 осуждается 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("Слишком много кошек! Мир обречен!")

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

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

if people > dogs:
    print("Не все так плохо!")


dogs += 5

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

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


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

Результат выполнения

Сеанс упражнения 29

```
$ python3.6 ex29.py
Слишком много кошек! Мир обречен!
Не все так плохо!
Людей больше или столько же, сколько собак.
Людей меньше или столько же, сколько собак.
Людей столько же, сколько собак.
```

Практические задания

В этом разделе «Практические задания» 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:

1. Как влияет конструкция `if` на код, расположенный далее?
2. Почему строка кода ниже конструкции `if` имеет отступ в четыре пробела?
3. Что произойдет, если удалить этот отступ?
4. Можете ли вы вставить другие логические выражения из упражнения 27 в конструкцию `if`? Попробуйте.
5. Что произойдет, если изменить исходные значения переменных `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 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 `if` создает так называемую «ветвь» в коде. Это своего рода квест, когда вы попадаете в одну локацию, если пойдете налево, или 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. Можно ли вставить другие логические выражения из упражнения 27 в конструкцию `if`? Попробуйте.

   Да, это возможно, и они могут быть настолько сложным, насколько вам по силам, хотя на самом деле сложный код, как правило, считается дурным стилем.

5. Что произойдет, если изменить исходные значения переменных `people`, `cats` и `dogs`?

Поскольку вы сравниваете числа, то если вы измените значения переменных, конструкции if по-прежнему будут проверяться на истинность, и блоки кода будут продолжать работать. Попробуйте подставить разные числа и представить, как работают блоки кода.

Сравните мои ответы с собственными и убедитесь, что вы поняли 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
trucks = 15


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

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

if people > trucks:
    print("Хорошо, давайте поедем на автобусе.")
else:
    print("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_python3.zip**.

Результат выполнения

Сеанс упражнения 30

```
$ python3.6 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 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 и запустит первый «истинный» блок. При этом будет выполнен только этот один 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. Изучив 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

```
print("""Ты находишься в темной комнате с двумя дверями.
В какую дверь ты войдешь - 1 или 2?""")

door = input("> ")

if door == "1":
    print("В этой комнате гигантский медведь поедает сырок 'Дружба'.")
    print("Твои действия?")
    print("1. Отберу сырок.")
    print("2. Заору медведю в ухо.")

    bear = input("> ")

    if bear == "1":
        print("Медведь вцепился тебе в лицо. Просто прекрасно!")
    elif bear == "2":
        print("Медведь укусил тебя за ногу. Просто прекрасно!")
    else:
       print(f"Прекрасно, это действие {bear} было единственным верным.")
        print("Медведь убежал прочь.")

elif door == "2":
    print("Ты смотришь в бесконечную пропасть глаз Ктулху. Твои действия?")
    print("1. Расскажу Ктулху про топи в Сибири.")
    print("2. Потреплю желтые пуговки на своей куртке.")
```

```
26      print("3. Попробую насвистеть песню 'Черный ворон'.")
27
28      insanity = input("> ")
29
30      if insanity == "1" or insanity == "2":
31          print("Ты спасся, потому что Ктулху превратился в желе.")
32          print("Просто прекрасно!")
33      else:
34          print("Безумие охватило тебя, и ты упал в бассейн с гнилью.")
35          print("Просто прекрасно!")
36
37  else:
38      print("Безумие охватило тебя, и ты разодрал себе лицо. Отличная работа!")
```

Ключевым моментом здесь является то, что вы поместили одни конструкции `if` внутрь других в виде исполняемого кода. 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 вам разобраться.

Результат выполнения

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

Сеанс упражнения 31

```
$ python3.6 ex31.py
Ты находишься в темной комнате с двумя дверьми.
В какую дверь ты войдешь - 1 или 2?
> 1
В этой комнате гигантский медведь поедает сырок 'Дружба'.
Твои действия?
1. Отберу сырок.
2. Заору медведю в ухо.
> 2
Медведь укусил тебя за ногу. Просто прекрасно!
```

Практические задания

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

2. Напишите совершенно новую игру. Возможно, вам эта не по душе, поэтому создайте свою собственную. Это ваш компьютер, делайте, что хотите.

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`) либо код `x in range(1, 10)`.

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

Элементарно: 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 данном упражнении мы будем использовать цикл `for`, который применим для создания и вывода различных списков. По мере выполнения упражнения вы поймете принцип его работы. 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 = ['шатенка', 'блондинка', 'рыжая']
eyes = ['карие', 'голубые', 'зеленые']
weights = [1, 2, 3, 4]
```

Список начинается с символа `[` (левая квадратная скобка), который «открывает» его. Затем вы перечисляете элементы списка через запятую, как вы и делали с аргументами функции. И наконец, список завершается символом `]` (правая скобка). Интерпретатор Python принимает этот список со всем его содержимым и присваивает его переменной.

Внимание! И вот теперь все усложняется для людей, которые не 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 и бумагу и разберите код на элементы, пока не поймете.

Теперь мы создадим несколько списков, используя циклы `for`, и выведем их:

ex32.py

```
the_count = [1, 2, 3, 4, 5]
fruits = ['яблоко', 'апельсин', 'персик', 'абрикос']
change = [1, 'червонец', 2, 'полтинник', 3, 'сотня']

# цикл for первого типа обрабатывает список
for number in the_count:
    print(f"Счетчик {number}")

# то же, что и выше
for fruit in fruits:
    print(f"Фрукт: {fruit}")

# также можно обрабатывать смешанные списки
# обратите внимание, что используются символы {}, так как неизвестен
# тип значения
for i in change:
    print(f"Я получил {i}")

# также мы можем создавать списки, начнем с пустого
elements = []

# затем используется функция range() для ограничения диапазона
for i in range(0, 6):
    print(f"Добавление {i} в список.")
    # append - функция для добавления элементов в список
    elements.append(i)

# теперь мы их выводим
for i in elements:
    print(f"Номер элемента: {i}")
```

Результат выполнения

Сеанс упражнения 32

```
$ python3.6 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 червонец
Я получил 2
Я получил полтинник
Я получил 3
Я получил сотня
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. Можно 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 и заново инициализируется при каждом запуске 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 последний. 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 оболочки командной строки.

Циклы `while`

Давайте познакомимся 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, ранее вы так же заучивали логические выражения, чтобы освоить их.

Вернемся к циклу `while`. Все, что он делает, это выполняет проверку, подобно конструкции `if`, но вместо однократного запуска блока кода, после его выполнения переходит в начало и повторяет выполнение кода. Код продолжает выполняться до тех пор, пока выражение не станет ложным.

Налицо проблема с циклами `while`: иногда они становятся бесконечными. Это замечательно, если ваше намерение состоит в том, чтобы цикл выполнялся до конца времен. В противном случае, почти всегда требуется, чтобы циклы рано или поздно завершались.

Чтобы избежать этих проблем, существуют правила, которым необходимо следовать:

1. Используйте циклы `while` как можно реже. Как правило, целесообразнее использовать цикл `for`.

2. Внимательно изучите код своих инструкций `while` и убедитесь, 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

```
i = 0
numbers = []

while i < 6:
    print(f"В начале значение i равно {i}")
    numbers.append(i)

    i = i + 1
    print("Текущие значения: ", numbers)
    print(f"В конце значение i равно {i}")


print("Значения: ")

for num in numbers:
    print(num)
```

Результат выполнения

Сеанс упражнения 33

```
$ python3.6 ex33.py
В начале значение i равно 0
Текущие значения:  [0]
В конце значение i равно 1
В начале значение i равно 1
Текущие значения:  [0, 1]
В конце значение i равно 2
В начале значение i равно 2
Текущие значения:  [0, 1, 2]
В конце значение i равно 3
В начале значение i равно 3
Текущие значения:  [0, 1, 2, 3]
В конце значение i равно 4
В начале значение i равно 4
Текущие значения:  [0, 1, 2, 3, 4]
В конце значение i равно 5
```

```
В начале значение i равно 5
Текущие значения:  [0, 1, 2, 3, 4, 5]
В конце 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 переменную, увеличивающую 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[5]? Что произойдет, если вы не удалите его?

Если программа ведет себя странно и непредсказуемо (что вполне вероятно в результате экспериментов), нажмите сочетание клавиш **Ctrl+C**, чтобы завершить ее работу.

[5] Число, на которое мы увеличиваем значение при каждой итерации цикла. *(прим.ред).*

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 большинства задач.

Циклы – это очень сложно. Как разобраться в них?

Основная причина, почему люди не понимают циклы, потому что не могут следовать «прыжкам», которые совершает исполнение кода. При выполнении, цикл обходит блок кода, а затем возвращается в начало блока. Чтобы понять принцип работы, вставьте инструкции `print` перед циклом, в начале цикла, в его середине и в конце. Изучите вывод и попытайтесь понять принцип работы.

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 = ['медведь', 'тигр', 'пингвин', '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 может показаться 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 четверо животных из нашего списка (`['медведь', 'тигр', 'пингвин', '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, и каким-то образом удалось управлять им. 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?»

Так 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 произвольный элемент из списка в любой момент. 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. Этот так называемые количественные числительные, которые вы можете выбрать в произвольном порядке, при этом 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 «количественному» путем вычитания единицы. «Третье» животное с индексом 2 – это пингвин. Вы должны запомнить это, потому что потратили всю свою жизнь, используя порядковые числа, и теперь должны научиться думать количественными. Просто вычтите 1, и все будет хорошо.

Запомните: порядковые == по порядку, с 1; количественные == в случайном порядке, с 0.

Давайте попрактикуемся. Возьмите следующий список животных и выполните упражнение, используя порядковое или количественное число животного. Помните, что если я говорю «первое», «второе» и так далее, то использую порядковый номер, поэтому вычитаем единицу. Если я указываю количественное число (0, 1, 2), используйте его без изменений.

```
animals = ['медведь', 'питон', 'пингвин', 'кенгуру', 'кит', '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. С учетом разницы между описываемыми типами чисел, можете ли вы объяснить, почему 2010 год в дате «1 января 2010 года» на самом деле 2010, а не 2009? (Подсказка: вы не можете выбирать год в случайном порядке.)

2. Напишите еще несколько списков и разработайте аналогичные индексы, пока вы не усвоите тему.

3. С помощью интерпретатора Python проверьте свои ответы на задания.

Внимание! Программисты посоветуют вам на эту тему почитать труды парня по фамилии Дейкстра. Я рекомендую избегать их советов, если вам хочется не терять время, а продолжать учиться программировать.

УПРАЖНЕНИЕ 35

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.

ex35.py

```
from sys import exit

def gold_room():
    print("Здесь полно золота. Сколько кг ты унесешь?")

    choice = input("> ")
    if "0" in choice or "1" in choice:
        how_much = int(choice)
    else:
        dead("Эй, ввести надо число.")

    if how_much < 50:
        print("Шикарно! Ты не жадный, поэтому выигрываешь!")
        exit(0)
    else:
        dead("Ах ты жадина!")


def bear_room():
    print("Здесь сидит медведь.")
    print("У медведя бочка с медом.")
    print("Медведь закрыл собой дверь выхода.")
    print("Как переместить медведя? Отобрать мед или подразнить медведя?")
    bear_moved = False

    while True:
        choice = input("> ")

        if choice == "отобрать мед":
            dead("Медведь посмотрел на тебя и ударил лапой по лицу.")
        elif choice == "подразнить медведя" and not bear_moved:
            print("Медведь отошел от двери.")
```

```
33              print("Вы можете войти в нее. Подразнить медведя или
34              войти в дверь?")
35              bear_moved = True
36          elif choice == "подразнить медведя" and bear_moved:
37              dead("Медведь разозлился и откусил тебе ногу.")
38          elif choice == "войти в дверь" and bear_moved:
39              gold_room()
40          else:
41              print("Введите одно из предложенных действий.")
42
43
44  def cthulhu_room():
45      print("На тебя смотрит великий и ужасный Ктулху.")
46      print("Он смотрит на тебя, и ты начинаешь сходить с ума.")
47      print("Убежать или начать сражаться?")
48
49      choice = input("> ")
50
51      if "убежать" in choice:
52          start()
53      elif "начать сражаться" in choice:
54          dead("Хм, возможно, даже удастся победить!")
55      else:
56          cthulhu_room()
57
58
59  def dead(why):
60      print(why, "Великолепно!")
61      exit(0)
62
63  def start():
64      print("Ты в темной комнате.")
65      print("Отсюда ведут две двери, налево и направо.")
66      print("Куда ты повернешь?")
67
68      choice = input("> ")
69
70      if choice == "налево":
71          bear_room()
72      elif choice == "направо":
73          cthulhu_room()
74      else:
75          dead("Ты ходишь из комнаты в комнату, пока не умираешь с голоду.")
76
```

```
77
78 start()
```

Результат выполнения

Ниже показано, как я попробовал поиграть:

Сеанс упражнения 35

```
$ python3.6 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 диаграммы продемонстрируйте игровой процесс.

2. Устраните все свои ошибки, включая опечатки.

3. Напишите комментарии к функциям, работа которых вам неясна. Помните про документирование?

4. Измените игровой процесс. Как бы вы могли упростить и расширить его?

5. Функция `gold_room()` реализует довольно странный способ ввода числа. Перечислите все недостатки этого способа? Можете ли вы

улучшить ее код, заменив простую проверку 1 или 0 в качестве числа? Использовать функцию `int()`?

Распространенные вопросы

Ничего не понимаю! Как работает эта программа!?

Каждый раз, когда вы затрудняетесь в понимании кода программного обеспечения, просто пишите комментарий выше каждой строки кода, объясняющий ее работу. После этого, исправьте неправильные комментарии, получив информацию о непонятной строке кода. 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 `input()` в некоторых случаях оформляется как `input('> ')`?

Данный параметр в функции **`input()`** позволяет вывести приглашение к вводу для пользователя.

УПРАЖНЕНИЕ 36

Разработка и отладка

Вы уже научились работать с конструкциями `if`. Теперь я собираюсь обучить вас правилам, позволяющим избегать неприятных ситуация при использовании циклов `for` и `while`. Также вы узнаете несколько приемов отладки и научитесь разбираться с распространенными программными проблемами. И, наконец, вы создадите небольшую игру, подобную рассмотренной в предыдущем упражнении, но с небольшими 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. По возможности, всегда используйте один уровень вложенности.

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 блока.

5. Ваши логические условия должны быть просты. Если простоту соблюсти не удается, произведите вычисления и запишите результаты в переменные в вашей функции, при этом используйте подходящие имена переменных.

Если вы будете следовать этим простым правилам, ваш код станет лучше, чем у большинства программистов. Вернитесь к предыдущему упражнения и взгляните, следовал ли я всем этим правилам. Если нет, то исправьте код.

Внимание! Никогда не следуйте слепо правилам в реальной жизни. Во время обучения вы должны следовать им, чтобы закрепить полу-

ченные знания, но в реальной жизни эти правила иногда использовать попросту глупо. Если вы считаете, что то или иное правило использовать не стоит, так и поступите.

Правила циклов

1. Используйте циклы `while`, только если требуется выполнять цикл бесконечно (то есть, по возможности, никогда). Это актуально только для Python; другие языки программирования отличаются.

2. Используйте циклы `for` во всех остальных случаях, особенно, если количество элементов, используемое 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 на нее неделю, чтобы сделать игру 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 будет выполнить, запустив вашу игру.

После этого начните набирать код. Если возникнут проблемы с достижением каких-либо целей на бумаге, измените рисунок и код.

Совет напоследок: все программисты иррационально паникуют перед началом каждого нового большого проекта. Они откладывают программирование, чтобы избежать этого страха, и, в конечном счете, не могут закончить (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 уже изученные символы и ключевые слова языка Python и рассмотреть новые, которые будут использоваться в будущих упражнениях. В этом разделе я перечислил все символы и ключевые слова Python, которые 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 слова может быть сложно найти в Интернете, но, тем не менее, попытайтесь это сделать.

Если по памяти вы ошиблись с предназначением того или иного символа/слова, возьмите пустую карточку и напишите на ней правильное определение. Затем тренируйтесь, пытаясь запомнить правильное определение. Если вы ранее не знали о том или ином символе/слове, также запишите его (с определением) на карточку и сохраните ее 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 для закрепления полученных знаний.

Ключевые слова

Ключевое слово	Описание	Пример
`and`	Логическое И.	`True and False == False`
`as`	Часть инструкции `with as`.	`with X as Y: pass`
`assert`	Утверждение (убеждение), что условие истинно.	`assert False, "Ошибка!"`
`break`	Немедленное прекращение выполнения цикла.	`while True: break`

class	Определение класса.	class Person(object)
continue	Переход на следующую итерацию цикла.	while True: continue
def	Определение функции.	def X(): pass
del	Удаление из словаря.	del X[Y]
elif	Условие «в противном случае, если».	if: X; elif: Y; else: J
else	Выполняется, если прочие условия не соблюдены.	if: X; elif: Y; else: J
except	Перехват исключения.	except ValueError, e: print(e)
exec	Выполнение строки в Python.	exec 'print("привет")'
finally	Вместе с инструкцией try: выполнение инструкции независимо от того, было вызвано исключение или нет.	finally: pass
for	Создание цикла.	for X in Y: pass
from	Импорт нескольких функций из модуля.	from x import Y
global	Создание глобальной переменной.	global X
if	Условие «если».	if: X; elif: Y; else: J
import	Импорт модуля.	import os
in	Часть цикла for; а также проверка X в Y.	for X in Y: pass also 1 in [1] == True
is	Как и == служит для проверки равенства.	1 is 1 == True
lambda	Определение анонимной функции.	s = lambda y: y ** y; s(3)
not	Логическое НЕ.	not True == False
or	Логическое ИЛИ.	True or False == True
pass	Ничего не делающая конструкция.	def empty(): pass
print	Вывод на экран указанной строки.	print('строка')
raise	Запускать исключения, когда все не так.	raise ValueError("Нет")
return	Выход из функции с возвращением значения.	def X(): return Y

try	Выполнение блока инструкций с перехватом исключений.	try: pass
while	Цикл «пока условие верно».	while X: pass
with	Менеджер контекста.	with X as Y: pass
yield	Определение функции-генератора.	def X(): yield Y; X().next()

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.

Ключевое слово	Описание	Пример
True	Логическое значение «Истина» (True)	True or False == True
False	Логическое значение «Ложь» (False)	False and True == False
None	Пустое значение	x = None
байты	Хранение байтов – текста, PNG, файла и т. д.	x = b"привет"
строки	Хранение текстовых данных	x = "привет"
числа	Хранение целые чисел	i = 100
числа с плавающей точкой	Хранение десятичных чисел	i = 10.389
списки	Хранение списков	j = [1,2,3,4]
словари	Хранение сопоставлений ключ=значение	e = {'x': 1, 'y': 2}

Управляющие последовательности

Напишите код с управляющими последовательностями, чтобы проверить свои знания.

Последовательность	Описание
\\	Обратный слеш
\'	Одинарная кавычка
\"	Двойная кавычка
\a	Звуковой сигнал
\b	Возврат на одну позицию
\f	Смена страницы
\n	Перевод строки
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция

Форматирование строк в старом стиле

То же касается и форматирования строк: используйте различные форматы, приведенные ниже, в своем коде, чтобы разобраться в их предназначении. В версии Python 2 эти символы форматирования применяются взамен f-строк. Попробуйте их использовать.

Последовательность	Описание	Пример
%d	Десятичные целые числа (не с плавающей запятой)	"%d" % 45 == '45'
%i	То же, что и %d	"%i" % 45 == '45'
%o	Восьмеричное целое число	"%o" % 1000 == '1750'
%u	Целое число без знака	"%u" % -1000 == '-1000'
%x	Шестнадцатеричное число в нижнем регистре	"%x" % 1000 == '3e8'
%X	Шестнадцатеричное число в верхнем регистре	"%X" % 1000 == '3E8'
%e	Число с плавающей точкой в экспоненциальной форме в нижнем регистре	"%e" % 1000 == '1.000000e+03'

`%E`	Число с плавающей точкой в экспоненциальной форме в верхнем регистре	`"%E" % 1000 == '1.000000E+03'`
`%f`	Число с плавающей точкой в десятичном представлении	`"%f" % 10.34 == '10.340000'`
`%F`	То же, что и `%f`	`"%F" % 10.34 == '10.340000'`
`%g`	Либо `%f`, либо `%e`, в зависимости от того, какое короче	`"%g" % 10.34 == '10.34'`
`%G`	То же, что и `%g`, но в верхнем регистре	`"%G" % 10.34 == '10.34'`
`%c`	Символ с кодом	`"%c" % 34 == '"'`
`%r`	Формат отладки – подобен `s`, но использует функцию `repr`, а не `str`	`"%r" % int == "<type 'int'>"`
`%s`	Строка	`"%s всем" % 'привет' == 'привет всем'`
`%%`	Символ `%`	`"%g%%" % 10.34 == '10.34%'`

Операторы

Часть операторов может быть незнакома 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, обязательно испытайте каждый из них в работе.

Оператор	Описание	Пример
`+`	Сложение	`2 + 4 == 6`
`-`	Вычитание	`2 - 4 == -2`
`*`	Умножение	`2 * 4 == 8`
`**`	Возведение в степень	`2 ** 4 == 16`
`/`	Деление	`2 / 4 == 0.5`
`//`	Целочисленное деление	`2 // 4 == 0`

%	Деление по модулю или интерполяция строк	`2 % 4 == 2`
<	Меньше	`4 < 4 == False`
>	Больше	`4 > 4 == False`
<=	Меньше или равно	`4 <= 4 == True`
>=	Больше или равно	`4 >= 4 == True`
==	Равно	`4 == 5 == False`
!=	Не равно	`4 != 5 == True`
( )	Круглые скобки	`len('hi') == 2`
[ ]	Скобки списка	`[1,3,4]`
{ }	Фигурные скобки	`{'x': 5, 'y': 10}`
@	Собачка (декоратор)	`@classmethod`
,	Запятая	`range(0, 10)`
:	Двоеточие	`def X():`
.	Точка	`self.x = 10`
=	Присвоение (равенство)	`x = 10`
;	Точка с запятой	`print("привет"); print("всем")`
+=	Прибавление и присвоение	`x = 1; x += 2`
-=	Вычитание и присвоение	`x = 1; x -= 2`
*=	Умножение и присвоение	`x = 1; x *= 2`
/=	Деление и присвоение	`x = 1; x /= 2`
//=	Деление и присвоение целой части	`x = 1; x //= 2`
%=	Деление по модулю и присвоение	`x = 1; x %= 2`
**=	Возведение в степень и присвоение	`x = 1; x **= 2`

Потратьте несколько дней на это 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 вы обнаружите. В реальности вы знаете достаточно, чтобы читать код, но, возможно, часть его фрагментов будет вам непонятна. В этом упражнении вы узнаете приемы, способствующие пониманию кода, написанного 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 листа бумаги, а не экрана компьютера. Распечатывайте пару-тройку страниц за один раз.

Во-вторых, во время чтения распечатки обратите особое внимание на следующие моменты:

1. Функции и что они делают.
2. Позиции присвоения значения каждой из переменных.
3. Все переменные с одинаковыми именами в разных частях кода программы. Они могут приводить к ошибкам, которые вы обнаружите 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 всех переменных построчно, функция 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 разберетесь в том, как работает код, вернитесь к компьютеру и прочитайте листинг еще раз. Возможно, вы увидите какой-либо 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.

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, ваши комментарии в будущем помогут разобраться в коде другому программисту.

Распространенные вопросы

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, чтобы вы нашли 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('привет')` 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 набор 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), то Python задействует его.

4. Далее Python обнаруживает круглую скобку (и «понимает», что это должно быть 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, 'привет')`, вместо написанного нами кода `mystuff.append('привет')`.

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:

```
$ python3.6
>>> class Thing(object):
...     def test(message):
...         print(message)
...
>>> a = Thing()
>>> a.test("привет")

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 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, это значит Python изменил `a.test("привет")` на `test(a, "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 = "Яблоки Апельсины Вороны Телефоны Свет Сахар"
2
3 print("Погодите, тут меньше 10 объектов. Давайте исправим это.")
4
5 stuff = ten_things.split(' ')
```

```
more_stuff = ["День", "Ночь", "Песня", "Мишка",
"Кукуруза", "Банан", "Девочка", "Мальчик"]

while len(stuff) != 10:
    next_one = more_stuff.pop()
    print("Добавляем: ", next_one)
    stuff.append(next_one)
    print(f"Теперь у нас {len(stuff)} объектов.")

print("Итак: ", stuff)

print("Давайте кое-что сделаем с нашими объектами.")

print(stuff[1])
print(stuff[-1]) # хм! интересно
print(stuff.pop())
print(' '.join(stuff)) # что? круто!
print('#'.join(stuff[3:5])) # просто супер!
```

Результат выполнения

Сеанс упражнения 38

```
$ python3.6 ex38.py
Погодите, тут меньше 10 объектов. Давайте исправим это.
Добавляем:  Мальчик
Теперь у нас 7 объектов.
Добавляем:  Девочка
Теперь у нас 8 объектов.
Добавляем:  Банан
Теперь у нас 9 объектов.
Добавляем:  Кукуруза
Теперь у нас 10 объектов.
Итак:  ['Яблоки', 'Апельсины', 'Вороны', 'Телефоны', 'Свет', 'Сахар',
'Мальчик', 'Девочка', 'Банан', 'Кукуруза']
Давайте кое-что сделаем с нашими объектами.
Апельсины
Кукуруза
Кукуруза
Яблоки Апельсины Вороны Телефоны Свет Сахар Мальчик Девочка Банан
Телефоны#Свет
```

Для чего нужны списки

Предположим, вы хотите создать компьютерную игру Go Fish. Если вы не знаете, что это за игра, прочитайте правила на сайте **oop.afti.ru/tasks/kartochnaya-igra-go-fish**. Для разработки этой игры, вам нужно будет разобраться с концепцией «колоды карт» и как представить ее в виде программы на языке Python. Вам нужно написать код Python, который будет работать с этой воображаемой колодой карт, причем так, чтобы для игрока эта колода ничем не отличалась от реальной. Вам понадобится структура «колоды карт» – то, что программисты называют структурой данных.

Что такое структура данных? Если задуматься, то структура данных – это лишь способ структурировать (упорядочить) какие-либо данные (факты). Да, это действительно так просто. Не смотря на то, что структуры данных могут быть очень сложными, это всего лишь способ сохранять данные внутри программы, чтобы вы имели к ним доступ. Структуры упорядочивают данные.

Мы рассмотрим их более подробно в следующем упражнении, а сейчас запомните, что списки – самые популярные структуры данных, которые используют программисты. Список – это упорядоченный перечень каких-либо элементов, которые вы хотите сохранить и получать к ним доступ случайным или линейным образом по номеру. Что?! Помните, что я говорил: это не значит, что списки в программировании сложнее, чем в обычном понимании, только потому, что программисты говорят: «список – это список». Давайте рассмотрим колоду карт как пример списка:

1. У вас есть стопка карт со значениями.

2. Эти карты в стопке (списке) перечислены, или упорядочены от верхней и до самой нижней карты.

3. Вы можете снимать карты случайным образом сверху, брать снизу и из средины.

4. Если вы хотите найти определенную карту, вам нужно взять колоду и снимать карту за картой.

Давайте разберем приведенное выше определение списка:

«упорядоченный перечень». Да, колода карт – это порядок, в котором есть первая карта и последняя.

«элементов, которые вы хотите сохранить». Да, карты – это то, что я собираюсь сохранять.

«и получать к ним доступ случайным образом». Да, если я хочу найти определенную карту, мне необходимо начать с начала и продолжать вытягивать карты по порядку.

«по номеру». Почти, как с колодой карт, если я скажу вам вытащить карту под номером 19, вам придется отсчитать 18 карт, пока вы не доберетесь до нужной. В наших списках в Python, компьютер может просто перейти к любому номеру, который вы ему зададите.

Это все, что касается списков как одной из концепций программирования. Каждая концепция в программировании похожа на вещи, которые мы видим в жизни. Если вы можете подобрать аналог в реальном мире, то с легкостью разберетесь, как работают структуры данных.

В каких случаях используются списки

Используйте списки каждый раз, когда востребованы преимущества структур данных:

1. Когда необходимо что-либо упорядочить. Помните, что списки не сортируются.

2. Если вам нужно получить произвольный доступ к содержимому по номеру. Помните, что нумерация количественных чисел начинается с 0.

3. Когда вам нужно пройтись по списку по порядку (от начала до конца). Помните, для чего нужны циклы `for`.

Во всех этих случаях следует использовать списки.

Практические задания

1. Возьмите каждую вызываемую функцию и разложите ее на шаги, как описано выше, чтобы понять, что делает Python. Например, `more_stuff.pop()` можно представить как `pop(more_stuff)`.

2. Переведите на человеческий язык оба представления вызова функции. Например, `more_stuff.pop()` можно прочитать как «вызвать `pop` для `more_stuff`». В свою очередь, `pop(more_stuff)` означает «вызвать `pop` с аргументом `more_stuff`». Так вы поймете, что, по сути, они делают одно и то же.

3. Поищите в Интернете информацию об объектно-ориентированном программировании. Сбиты с толку? Я, когда-то, был 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. 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 его везде, где только можно. Если вам все это кажется очень сложным, можно попробовать использовать «функциональное программирование».

6. Придумайте 10 примеров списков. Попробуйте написать скрипты для работы с ними.

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. Только глупцы все время соблюдают правила.

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. Перепишите код следующим образом: `' '.join(stuff)`.

Почему вы используете циклы `while`?

Попробуйте заменить их циклами `for` и посмотрите, будет ли вам удобнее.

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)`.

УПРАЖНЕНИЕ 39

Словари

Сейчас вы познакомитесь с еще одним представителем структур данных, встроенных в Python, – словарем. Словарь – это способ хранения данных, подобный списку, но позволяющий использовать не только числа для получения доступа к данным. Таким образом, словарь можно рассматривать как базу данных для хранения и организации информации.

Рассмотрим процесс работы со словарями в сравнении со списками. Как видно из примера ниже, список позволяет делать следующее:

Сеанс упражнения 39 в Python

```
>>> things = ['а', 'б', 'в', 'г']
>>> print(things[1])
б
>>> things[1] = 'я'
>>> print(things[1])
я
>>> things
['а', 'я', 'в', 'г']
```

Вы можете использовать числа для «индексации» элементов в списке, то есть с помощью этих чисел получать 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:

Сеанс упражнения 39 в Python

```
>>> stuff = {'имя': 'Михаил', 'возраст': 38, 'вес': 6 * 12 + 2}
>>> print(stuff['имя'])
Михаил
>>> print(stuff['возраст'])
38
```

```
>>> print(stuff['вес'])
74
>>> stuff['город'] = "Москва"
>>> print(stuff['город'])
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:

Сеанс упражнения 39 в Python

```
>>> stuff[1] = "Ура"
>>> stuff[2] = "Неа"
>>> print(stuff[1])
Ура
>>> print(stuff[2])
Неа
```

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 (или почти любой), но сейчас важнее всего понять общий принцип.

Разумеется, если в словарь можно было бы только добавлять записи, – это было бы довольно глупо, поэтому вы можете удалять записи, используя ключевое слово `del`:

Сеанс упражнения 39 в Python

```
>>> del stuff['город']
>>> del stuff[1]
>>> del stuff[2]
>>> stuff
{'имя': 'Михаил', 'возраст': 38, 'вес': 74}
```

Пример словаря

Теперь приступим к упражнению, которое вы должны изучить очень тщательно. Я хочу, чтобы вы напечатали код этого примера и попытались разобраться, как он работает. Обратите внимание на то, как я добавляю записи в словарь, извлекаю их, и на все операции, примененные в данном примере. Также,

обратите внимание на то, как в этом примере названиям стран сопоставляются аббревиатурами, а затем аббревиатуры городами в этих странах. Запомните, что сопоставление (или ассоциирование) – это ключевое понятие в словарях.

ex39.py

```
# схема связей аббревиатур с названиями стран
states = {
    'Россия': 'RU',
    'Германия': 'DE',
    'Узбекистан': 'UZ',
    'Зимбабве': 'ZW',
    'Турция': 'TR'
}

# создание базового набора стран и некоторых городов в них
cities = {
    'UZ': 'Газли',
    'TR': 'Сарыгерме',
    'DE': 'Мюнхен'
}

# добавление нескольких городов
cities['ZW'] = 'Гверу'
cities['RU'] = 'Москва'

# вывод некоторых городов
print('-' * 10)
print("В стране ZW есть город: ", cities['ZW'])
print("В стране RU есть город: ", cities['RU'])

# вывод некоторых стран
print('-' * 10)
print("Аббревиатура Турции: ", states['Турция'])
print("Аббревиатура Германии: ", states['Германия'])

# выполняется с учетом страны и словаря городов
print('-' * 10)
print("В Турции есть город: ", cities[states['Турция']])
print("В Германии есть город: ", cities[states['Германия']])

# вывод аббревиатур всех стран
print('-' * 10)
for state, abbrev in list(states.items()):
```

```
39      print(f"{state} имеет аббревиатуру {abbrev}")
40
41  # вывод всех городов в странах
42  print('-' * 10)
43  for abbrev, city in list(cities.items()):
44      print(f"В стране {abbrev} есть город {city}")
45
46  # а теперь сразу оба типа данных
47  print('-' * 10)
48  for state, abbrev in list(states.items()):
49      print(f"В стране {state} используется аббревиатура {abbrev}")
50      print(f"и есть город {cities[abbrev]}")
51
52  print('-' * 10)
53  # безопасное получение аббревиатуры страны, которая не представлена
54  state = states.get('США')
55
56  if not state:
57      print("Прошу прощения, США не существует или уничтожено.")
58
59  # получение города со значением по умолчанию
60  city = cities.get('US', 'не существует')
61  print(f"В стране 'US' есть город: {city}")
```

Результат выполнения

Сеанс упражнения 39

```
$ python3.6 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
----------
Россия имеет аббревиатуру RU
Германия имеет аббревиатуру DE
Узбекистан имеет аббревиатуру UZ
Зимбабве имеет аббревиатуру ZW
```

```
Турция имеет аббревиатуру TR
----------
В стране UZ есть город Газли
В стране TR есть город Сарыгерме
В стране DE есть город Мюнхен
В стране ZW есть город Гверу
В стране RU есть город Москва
----------
В стране Россия используется аббревиатура RU
и есть город Москва
В стране Германия используется аббревиатура DE
и есть город Мюнхен
В стране Узбекистан используется аббревиатура UZ
и есть город Газли
В стране Зимбабве используется аббревиатура ZW
и есть город Гверу
В стране Турция используется аббревиатура TR
и есть город Сарыгерме
----------
Прошу прощения, США не существует или уничтожено.
В стране 'US' есть город: не существует
```

Для чего нужны словари?

Словарь – один из примеров структурирования данных, и так же, как и список, он часто используется в программировании. Словари применяются для сопоставления или ассоциирования данных, которые вы хотите сохранить, с ключами, необходимыми для извлечения этих данных. Повторюсь, программисты используют термин «словарь» для структур, аналогичных настоящим словарям, полных слов из реального мира.

Предположим, вы хотите найти значение слова «высокопревосходительство». Сегодня вы просто запустите браузер, перейдете на сайт поисковой системы и, указав запрос с этим словом, найдете нужный ответ. Поэтому можно считать, что поисковая система – это как бы огромная и колоссально расширенная версия словаря. Если бы не было поисковой системы, вот что вам пришлось бы делать:

1. Вы бы пошли в библиотеку за словарем.

2. Вы знаете, что слово «высокопревосходительство» начинается с буквы «В», поэтому вы бы открыли содержание и посмотрели, с какой страницы начинаются слова на букву «В».

3. Листали бы страницы, пока не нашли бы слова, начинающиеся с букв «Выс».

4. Листали бы дальше, пока не нашли бы нужное вам слово «высокопревосходительство», либо добрались бы до слов, начинающихся с букв «Вэ», и поняли, что нужно слова в этом словаре нет.

5. Если бы вы нашли запись, вы прочли бы обозначение этого слова.

Примерно так же работают словари. Вы сопоставляете со словом «высокопревосходительство» его значение. Словари в Python – это то же, что и обычные словари в реальном мире.

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 ограничением.

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 словарь в любых ситуациях, когда нужно, имея одно значение, «подсмотреть» другое. По сути, словари можно охарактеризовать как «обозреватели таблиц».

Для чего тогда используются списки?

Списки предназначены для хранения любой последовательности элементов, которые должны следовать по порядку, и доступ к ним 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 Всемирной паутиной, чтобы найти документацию е 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. Поехали!

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 `'яблоко'`, и вы хотите извлечь соответствующее ему значение, выполняется следующее:

ex40a.py

```
1 mystuff = {'яблоко': "Я - ЯБЛОКО!"}
2 print(mystuff['яблоко'])
```

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 содержащимися функциями или переменными.

2. Вы должны импортировать этот файл.

3. После импорта вы получите доступ к функциям и 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:

ex40a.py

```
1 # это записано в файле mystuff.py
2 def apple():
3     print("Я - ЯБЛОКО!")
```

После того, как модуль создан, его можно импортировать и получить доступ к функции `apple`:

ex40a.py

```
1 import mystuff
2 mystuff.apple()
```

Также я мог бы поместить в модуль переменную с именем `tangerine`, следующим образом:

ex40a.py

```
1 def apple():
2     print("Я - ЯБЛОКО!")
3
4 # это обычная переменная
5 tangerine = "Отражение мечты"
```

Доступ к ней осуществляется похожим образом:

ex40a.py

```
1 import mystuff
2
3 mystuff.apple()
4 print(mystuff.tangerine)
```

Возвращаясь к словарям, вы должны увидеть, насколько это похоже на использование словаря, хотя синтаксис несколько отличается. Давайте сравним:

ex40a.py

```
1 mystuff['яблоко'] # получаем apple из словаря
2 mystuff.apple() # получаем apple из модуля
3 mystuff.tangerine # аналогично, это обычная переменная
```

Так вы можете сформировать универсальный 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 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 поступил бы примерно так:

ex40a.py

```
1 class MyStuff(object):
2
3     def __init__(self):
4         self.tangerine = "Пусть бегут неуклюже..."
5
6     def apple(self):
7         print("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 смутит функция `__init__()` и код `self.tangerine`, используемый для инициализации переменной `tangerine`.

И вот почему классы используются вместо модулей: вы можете использовать упомянутый выше класс, создать 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 «создание 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, следующим образом:

ex40a.py

```
1 thing = MyStuff()
2 thing.apple()
3 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. Давайте разложим все по полочкам на примере приведенного выше кода для класса `MyStuff`:

1. Python ищет функцию `MyStuff()` и обнаруживает, что это объявленный вами класс.

2. Python создает пустой объект со всеми функциями, указанными вами в классе с помощью ключевого слова `def`.

3. Python затем ищет волшебную функцию `__init__` и, если она обнаружена, вызывает ее для инициализации созданного пустого объекта.

4. 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.

5. В примере я присвоил переменной `self.tangerine` текст из песни, а затем инициализировал этот объект.

6. Теперь Python берет этот созданный объект и присваивает его переменной `thing`, чтобы я смог работать с ней.

Это 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 похожими на модули. Если быть полностью корректным, 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:

ex40a.py

```
# 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 приведенный код и выполнили его, для приобретения опыта работы с классами, прежде чем двигаться дальше.

ex40.py

```
class Song(object):

    def __init__(self, lyrics):
        self.lyrics = lyrics

```

```
    def sing_me_a_song(self):
        for line in self.lyrics:
            print(line)

happy_bday = Song(["Не могу я тебе в день рождения",
                   "Дорогие подарки дарить,",
                   "Но зато в эти ночи весенние "])

bulls_on_parade = Song(["Далеко-далеко на лугу пасется кто?",
                        "Пейте, дети, молоко, будете здоровы!"])

happy_bday.sing_me_a_song()

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_python3.zip**.

Результат выполнения

Сеанс упражнения 40

```
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 ни малейшего 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 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 = 'Макс'` неоднозначен. Исходя из этого кода, не ясно, что вы имеете в виду – атрибут `cheese` экземпляра или локальную переменную с именем `cheese`. С кодом `self.cheese = 'Макс'` 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. Затем 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 класса и любой экземпляр какого-либо класса.
- Экземпляр – вы получаете, когда инструктируете Python создать объект класса.
- `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. Как правило, является переменной.
- `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 упоминается 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 вы должны 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` и `J`."
`foo = X()` "создается `foo` как экземпляр класса `X`."
`foo.M(J)` "из `foo` получается функция `M`, а затем вызывается с параметрами `self` и `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`; эти имена можно заменить на какие угодно. 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 ??? как экземпляр класса ???.
5. Из ??? получается функция ???, а затем вызывается с параметрами ??? и ???.
6. Из ??? 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

Теперь я объединил упражнения 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 ему карточку.
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 (описание русскими словами того, что происходит). Это простой сценарий, код которого приведен ниже. В нем используется библиотека под названием `urllib`, позволяющая загрузить с сервера составленный мной список слов. Этот же код содержится в файле *oop_test.py*, с которым вы будете работать позднее:

ex41.py

```
1 import random
2 from urllib.request import urlopen
3 import sys
4
5 WORD_URL = "http://learncodethehardway.org/words.txt"
6 WORDS = []
```

```
 7
 8 PHRASES = {
 9     "class %%%(%%%):":
10     "Создается класс с именем %%%, наследующим %%%.",
11     "class %%%(object):\n\tdef __init__(self, ***)" :
12     "Класс %%% комбинирует __init__ с параметрами self и параметрами ***.",
13     "class %%%(object):\n\tdef ***(self, @@@)":
14     "Класс %%% комбинирует функцию с именем *** с параметрами self и @@@.",
15     "*** = %%%()":
16     "Создается *** как экземпляр класса %%%.",
17     "***.***(@@@)":
18     "Из *** получается функция ***, а затем вызывается с параметрами self, @@@.",
19     "***.*** = '***'":
20     "Из *** получается атрибут ***, а затем устанавливается равным '***'."
21 }
22
23 # тренировка запоминания фраз
24 if len(sys.argv) == 2 and sys.argv[1] == "russian":
25     PHRASE_FIRST = True
26 else:
27     PHRASE_FIRST = False
28
29 # подгружаем слова с сервера
30 for word in urlopen(WORD_URL).readlines():
31     WORDS.append(str(word.strip(), encoding="utf-8"))
32
33
34 def convert(snippet, phrase):
35     class_names = [w.capitalize() for w in
36                    random.sample(WORDS, snippet.count("%%%"))]
37     other_names = random.sample(WORDS, snippet.count("***"))
38     results = []
39     param_names = []
40
41     for i in range(0, snippet.count("@@@")):
42         param_count = random.randint(1,3)
43         param_names.append(', '.join(
44         random.sample(WORDS, param_count)))
45
46     for sentence in snippet, phrase:
47         result = sentence[:]
48
49         # вымышленные имена классов
50         for word in class_names:
```

```
            result = result.replace("%%%", word, 1)

        # вымышленные прочие имена
        for word in other_names:
            result = result.replace("***", word, 1)

        # список вымышленных параметров
        for word in param_names:
            result = result.replace("@@@", word, 1)

        results.append(result)

    return results


# выполнение, пока не нажато сочетание клавиш CTRL+Z
try:
    while True:
        snippets = list(PHRASES.keys())
        random.shuffle(snippets)

        for snippet in snippets:
            phrase = PHRASES[snippet]
            question, answer = convert(snippet, phrase)
            if PHRASE_FIRST:
                question, answer = answer, question

            print(question)

            input("> ")
            print(f"ОТВЕТ: {answer}\n\n")
except EOFError:
    print("\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.

```
$ python3.6 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, по сравнению со случаями, когда специально подобранные имена переменных и классов «могут подсказать». 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 и использовать выражения, с которыми вы 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 будут бросаться вам в глаза при чтении кода. А ведь раньше они 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 ввести и успешно выполнить код. 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. Набирайте каждую строку кода предельно внимательно, убедитесь, что она в точности совпадает с моей, а все, чего не знаете, ищите в Интернете.

Мне все еще очень сложно!

Вы справитесь! Не торопитесь, если нужно, символ за символом с точностью введите код и выясните, как он работает.

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 может сбить 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, поэтому это одно и то же. Но в то же время, так как семга – конкретный вид рыбы, она 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 своего рода 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 следующим вопросом:

В чем разница между Кириллом *и* семгой?

Опять же, это очень странный вопрос, хотя и проще, чем вопрос о разнице 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 являются реальными объектами, которым присущи атрибуты семги.

Теперь диалектическая 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 о рыбе 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 (также являющийся экземпляром рыбы) по имени Кирилл, который превратился в нечто реальное, что насытило вас. Так рыба стала объектом.

Итак: Кирилл – это представитель семги, которая является рыбой. Объект – это экземпляр класс от класса.

Пример кода

Это довольно странная концепция, но если быть честным, вам стоит беспокоиться об этом, только когда вы создаете новые классы и используете их. Я продемонстрирую две уловки, способные помочь вам выяснить, 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 только 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):
        ## ?? хмм, что за чудеса?
        super(Employee, self).__init__(name)
        ## ??
```

```
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 ## barbos наследует Dog
52 barbos = Dog("Барбос")
53
54 ## ??
55 barsik = Cat("Барсик")
56
57 ## ??
58 mary = Person("Машка")
59
60 ## ??
61 mary.pet = barsik
62
63 ## ??
64 filka = Employee("Филька", 120000)
65
66 ## ??
67 filka.pet = barbos
68
69 ## ??
70 flipper = Fish()
71
72 ## ??
73 crouse = Salmon()
74
75 ## ??
76 harry = Halibut()
```

О синтаксисе `class имя(объект)`

В Python 3 вам не нужно добавлять значение `(объект)` после имени класса, но сообщество программистов на Python убеждено, что «использовать его лучше, чем не использовать», поэтому я и другие эксперты Python решили его все-таки использовать. Вы можете писать код, в котором используются как классы со значением `(объект)`, так и без него, и все они будут работать одинаково. На данном этапе, `(объект)` – это лишь дополнительная «писанина», которая никак не влияет на работу ваших классов.

В Python 2 была разница между двумя типами классов, но все это кануло в лету. Единственный плюс в использовании значения `(объект)` заключается в том, что вы помечаете для себя что «данный класс – это класс типа объекта». Все это может показаться вам непонятным, поэтому просто представьте код `class имя(объект)` как «базовый простой класс».

В конце концов, в будущем стиль программирования и предпочтения разработчиков на Python могут измениться, и использование значения `(объект)` может оказаться признаком моветона в программировании. Если это произойдет, просто перестаньте его указывать, или аргументируйте тем, что «гуру Python рекомендуют использовать его».

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»? Поищите информацию о «множественном наследовании» и о том, почему его следует избегать.

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, которые 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. Т. е. 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.

Если говорить обо мне, 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 ее.

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 базовый каркас кода, который содержит только классы и их функции. Затем я пишу тестовый сценарий, который выполняет этот код, и проверяю правильность работы созданных мной классов. Временами 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 продолжаю работу только с этим фрагментом кода, чтобы исправить его, прежде чем продолжить.

Теперь и вы пройдете через этот процесс создания игрового движка и самой игры.

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, я могу развить ее и воплотить игру в жизнь.

Запись или зарисовка задачи

Я написал небольшой текст об игре:

«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 в нее, 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 будет 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, в которой герой устанавливает бомбу.

- Отсек спасательных капсул. Здесь герой убегает с корабля, но только если он выбрал капсулу с правильным номером.

Теперь я могу зарисовать схему сцен и, может быть, добавить больше описательного 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 на данном этапе я пропущу этом момент.

Теперь вы можете также исследовать каждую из этих концепций и все, что вы не учли. Я разрабатывал игры подобного типа, поэтому хорошо знаю, как они работают. В своем примере я мог бы исследовать, как спроектированы корабли или как работают бомбы. Возможно, я исследовал бы некоторые технические задачи, такие, как сохранение состояния игры в базе данных. После того, как исследования мной проведены, я снова начинаю с шага 1 и, на основе полученной дополнительной информации, переписываю описание и извлекаю 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», что 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 выбираю 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» может быть просто другой сценой, а не чем-то специфическим.

После этих рассуждений 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 латинские аналоги так, как они используются 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 часть 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 небольшой 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 TheBridge(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()
```

Можно заметить, что в этом файле я просто скопировал иерархию классов, а затем добавил некоторый код в конце, чтобы запустить сценарий и проверить, работает ли эта базовая структура. В следующих разделах этого упражнения вы наберете оставшуюся часть кода игры и заставите сценарий работать в соответствии с описанием игры.

Исправление ошибок и доработка кода

Заключительный этап в процессе не столько шаг, сколько цикл `while`. 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 над ними. 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, но 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 назад 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. Дробление 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 я это сделал.

Я разделю готовый код файла *ex43.py* на фрагменты и прокомментирую каждый из них, вместо того, чтобы привести полный листинг.

ex43.py

```
1 from sys import exit
2 from random import randint
3 from textwrap import dedent
```

Это лишь базовые инструкции импорта для игры, ничего сложного. Единственная незнакомая строка – импорт функции `dedent` из модуля `textwrap`. Эта маленькая функция позволит нам описать сцены в виде строк, окруженных символами `"""` (три двойные кавычки подряд). Так мы можем использовать отступы в начале строк с текстом. Если указанную функцию не импортировать, мы не сможем использовать строки с символами `"""`, так как у них такие же отступы, что и у строк кода Python.

ex43.py

```
1 class Scene(object):
2
3     def enter(self):
4         print("Эта сцена еще не настроена.")
5         print("Создайте подкласс и реализуйте функцию enter().")
6         exit(1)
```

Как вы видели в каркасе кода, существует 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

```
1 class Engine(object):
2
3     def __init__(self, scene_map):
4         self.scene_map = scene_map
```

```
5
6     def play(self):
7         current_scene = self.scene_map.opening_scene()
8         last_scene = self.scene_map.next_scene('finished')
9
10        while current_scene != last_scene:
11            next_scene_name = current_scene.enter()
12            current_scene = self.scene_map.next_scene(next_scene_name)
13
14        # не забудьте вывести последнюю сцену
15        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

```
1 class Death(Scene):
2
3     quips = [
4         "Ты погиб.  Как это ни печально.",
5         "Твоя мать будет грустить по тебе... надо было быть умнее.",
6         "Надо же было быть таким придурком.",
7         "Даже мой маленький щенок соображает лучше.",
8         "Когда ж ты повзрослеешь, как говорил твой папка."
9
10    ]
11
12    def enter(self):
13        print(Death.quips[randint(0, len(self.quips)-1)])
14        exit(1)
```

Первая сцена в моей игре необычна и носит название `Death` (т.е. смерть), код которой наиболее прост.

ex43.py

```
1 class CentralCorridor(Scene):
2
3     def enter(self):
```

```
4       print(dedent("""
5           Готоны с планеты Перкаль 25 захватили ваш корабль
6           и уничтожили всю команду. Ты - единственный, кто остался
7           в живых. Тебе нужно выкрасть нейтронную бомбу в оружейной
8           лаборатории, заложить ее в топливном отсеке и покинуть
9           корабль в спасательной капсуле прежде, чем он взорвется.
10
11          Ты бежишь по центральному коридору в оружейную лабораторию,
12          когда перед тобой появляется готон с красной чешуйчатой
13          кожей, гнилыми зубами и в костюме клоуна. Он с ненавистью
14          смотрит на тебя и, перегородив дорогу в лабораторию,
15          вытаскивает бластер, чтобы отправить тебя к праотцам.
16          """))
17
18      action = input("> ")
19
20      if action == "стрелять!":
21          print(dedent("""
22              Ты быстро выхватываешь бластер и начинаешь палить по
23              готону. Его клоунский наряд крутится на теле, мешая
24              лучам попадать в цель. Все твои выстрелы потерпели
25              неудачу, и заряд бластера иссяк. Костюм готона, который
26              купила его мать, безнадежно испорчен, поэтому
27              инопланетянин в ярости выхватывает бластер и стреляет
28              тебе в голову. В панике ты пытаешься удрать и, резко
29              повернувшись, ударяешься головой о балку и теряешь
30              сознание. Ты пал смертью храбрых.
31              """))
32          return 'death'
33
34      elif action == "проскочить!":
35          print(dedent("""
36              Словно боксер мирового класса, ты уворачиваешься и
37              проскакиваешь мимо готона, краем глаза замечая, что
38              его бластер направлен тебе в голову. И тут ты
39              подскальзываешься и врезаешься в стену. От удара ты
40              теряешь сознание. Придя в сознание, ты успеваешь
41              почувствовать, что готон топчется на твоей голове и
42              пожирает тебя. Свет меркнет перед глазами.
43              """))
44          return 'death'
45
46      elif action == "пошутить!":
47          print(dedent("""
```

```
48                К счастью, ты знаком с культурой готонов и знаешь, что
49                может их рассмешить. Ты рассказываешь бородатый
50                анекдот: Неоколонии, изоморфно релятивные к
51                мультиполосным гиперболическим параболоидам,
52                теоретически катаральны. Готон замирает, старается
53                сдержать смех, а затем начинает безудержно хохотать.
54                Пока он смеется, ты достаешь бластер и стреляешь
55                готону в голову. Он падает, а ты перепрыгиваешь через
56                него и бежишь в оружейную лабораторию.
57                """))
58            return 'laser_weapon_armory'
59
60        else:
61            print("ТАК НЕЛЬЗЯ ПОСТУПИТЬ!")
62            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 мне нужно будет сослаться на них позднее. Кроме того, обратите внимание, как я использую функцию `dedent` в строке 4. Впоследствии, вы можете попытаться удалить ее и проанализировать результат.

ex43.py

```
1 class LaserWeaponArmory(Scene):
2
3     def enter(self):
4         print(dedent("""
5             Ты вбегаешь в оружейную лабораторию и начинаешь
6             обыскивать комнату, спрятались ли тут другие готоны.
7             Стоит мертвая тишина. Ты бежишь в дальний угол комнаты и
8             находишь нейтронную бомбу в защитном контейнере.
9             На лицевой стороне контейнера расположена панель с
10            кнопками, и тебе надо ввести правильный код, чтобы
11            достать бомбу. Если ты 10 раз введешь неправильный код,
12            контейнер заблокируется и ты не сможешь достать бомбу.
13            Учти, что код состоит из 3 цифр.
14            """))
15
16        code = f"{randint(1,9)}{randint(1,9)}{randint(1,9)}"
17        guess = input("[keypad]> ")
18        guesses = 0
19
```

```
20          while guess != code and guesses < 10:
21              print("ВЖЖЖИИИК!")
22              guesses += 1
23              guess = input("[keypad]> ")
24
25          if guess == code:
26              print(dedent("""
27                  Контейнер открывается со щелчком и выпускает сизый газ.
28                  Ты вытаскиваешь нейтронную бомбу и бежишь в
29                  топливный отсек, чтобы установить бомбу в нужном месте,
30                  активировать ее и успеть смотаться с корабля.
31                  """))
32              return 'the_bridge'
33          else:
34              print(dedent("""
35                  Ты слышишь, как замок жужжит последний раз, а затем
36                  чувствуешь запах гари - замок расплавился. Ты остаешься
37                  в оружейной лавке, пока наконец готоны не взрывают
38                  корабль выстрелом со своего судна и ты не умираешь.
39                  """))
40              return 'death'
41
42
43
44  class TheBridge(Scene):
45
46      def enter(self):
47          print(dedent("""
48              Ты вбегаешь в топливный отсек с нейтронной бомбой и
49              видишь пятерых готонов, безуспешно пытающихся управлять
50              кораблем. Один уродливее другого, и все в клоунских
51              костюмах, как и готон, убитый тобой. Они не достают
52              оружие, так как видят бомбу в твоих руках и не хотят,
53              чтобы ты взорвал ее. Преимущество явно на твоей стороне.
54              """))
55
56          action = input("> ")
57
58          if action == "бросить бомбу":
59              print(dedent("""
60                  Ты в панике активируешь бомбу и бросаешь ее в толпу
61                  готонов, а затем прыгаешь к двери шлюза.  Сразу
62                  после этого один из готонов стреляет тебе в спину.
63                  Умирая, ты видишь, как другие готоны тщетно пытаются
```

```
64                  деактивировать бомбу. Ты осознаешь, что готоны тоже
65                  погибнут. Свет меркнет перед глазами.
66                  """))
67              return 'death'
68
69          elif action == "установить бомбу":
70              print(dedent("""
71                  Ты указываешь бластером на бомбу в своих руках.
72                  Готоны поднимают лапы вверх и в страхе потеют.
73                  Ты осторожно, не отворачиваясь, подходишь к двери и
74                  аккуратно устанавливаешь бомбу, держа готонов на
75                  мушке. Ты запрыгиваешь в шлюз и закрываешь дверь
76                  ударом по кнопке, а затем бластером расплавляешь
77                  замок, чтобы готоны не смогли открыть её. Теперь
78                  тебе нужно залезть в спасательную капсулу и удрать с
79                  корабля к чертям собачьим.
80                  """))
81
82              return 'escape_pod'
83          else:
84              print("ТАК НЕЛЬЗЯ ПОСТУПИТЬ!")
85              return "the_bridge"
86
87
88  class EscapePod(Scene):
89
90      def enter(self):
91          print(dedent("""
92              Ты мчишься по отсеку со спасательными капсулами.
93              Некоторые из них могут быть повреждены и взорвутся во
94              время полета. Всего капсул пять, и у тебя нет времени, чтобы
95              осматривать каждую из них на отсутствие повреждений.
96              Задумавшись на секунду, ты решаешь сесть в капсулу под
97              номером...
98              Капсулу под каким номером ты выбираешь?
99              """))
100
101         good_pod = randint(1,5)
102         guess = input("[pod #]> ")
103
104
105         if int(guess) != good_pod:
106             print(dedent("""
107                 Ты запрыгиваешь в капсулу номер {guess} и нажимаешь кнопку
```

```
108                 отстыковки. Капсула вылетает в космическое
109                 пространство, а затем взрывается с яркой вспышкой,
110                 разбрасывая осколки. Ты умираешь.
111                 """))
112             return 'death'
113         else:
114             print(dedent("""
115                 Ты запрыгиваешь в капсулу номер {guess} и нажимаешь
116                 кнопку отстыковки. Капсула вылетает в космическое
117                 пространство, а затем отправляется к планете
118                 неподалеку. Ты смотришь в иллюминатор и видишь, как
119                 ваш корабль взрывается. Его осколки повреждают
120                 топливный отсек корабля готонов, и тот тоже
121                 разлетается в клочья. Победа за вами!
122                 """))
123
124             return 'finished'
125
126 class Finished(Scene):
127
128     def enter(self):
129         print("Ты победил! Отличная работа.")
130         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 набрать их код полностью.

Кстати, я не сразу набирал весь этот код. Помните, я упоминал, что следует работать с ним поэтапно, проверять по одному фрагменту за раз. Здесь я сразу привел финальный результат после многих проверок.

ex43.py

```
1 class Map(object):
2
3     scenes = {
4         'central_corridor': CentralCorridor(),
5         'laser_weapon_armory': LaserWeaponArmory(),
6         'the_bridge': TheBridge(),
7         'escape_pod': EscapePod(),
8         'death': Death(),
9         'finished': Finished(),
10    }
11
```

```
    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` и затем передает 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 попробуйте решить проблему самостоятельно.

Когда я попробовал пройти игру (и проиграл), то получил следующий вывод:

Сеанс упражнения 43

```
$ python3.6 ex43.py

Готоны с планеты Перкаль 25 захватили ваш корабль и уничтожили
```

всю команду. Ты - единственный, кто остался в живых. Тебе нужно выкрасть нейтронную бомбу в оружейной лаборатории, заложить ее в топливном отсеке и покинуть корабль в спасательной капсуле прежде, чем он взорвется.

Ты бежишь по центральному коридору в оружейную лабораторию, когда перед тобой появляется готон с красной чешуйчатой кожей, гнилыми зубами и в костюме клоуна. Он с ненавистью смотрит на тебя и, перегородив дорогу в лабораторию, вытаскивает бластер, чтобы отправить тебя к праотцам.

> пошутить!

К счастью, ты знаком с культурой готонов и знаешь, что может их рассмешить. Ты рассказываешь бородатый анекдот:
Неоколонии, изоморфно релятивные к мультиполосным гиперболическим параболоидам, теоретически катаральны.
Готон замирает, старается сдержать смех, а затем начинает безудержно хохотать. Пока он смеется, ты достаешь бластер и стреляешь готону в голову. Он падает, а ты перепрыгиваешь через него и бежишь в оружейную лабораторию.

Ты вбегаешь в оружейную лабораторию и начинаешь обыскивать комнату, спрятались ли тут другие готоны. Стоит мертвая тишина. Ты бежишь в дальний угол комнаты и находишь нейтронную бомбу в защитном контейнере. На лицевой стороне контейнера расположена панель с кнопками и тебе надо ввести правильный код, чтобы достать бомбу. Если ты 10 раз введешь неправильный код, контейнер заблокируется и ты не сможешь достать бомбу. Учти, что код состоит из 3 цифр.

```
[keypad]> 123
ВЖЖЖИИИК!
[keypad]> 456
ВЖЖЖИИИК!
[keypad]> 789
ВЖЖЖИИИК!
[keypad]> 246
ВЖЖЖИИИК!
[keypad]> 753
ВЖЖЖИИИК!
[keypad]> 888
ВЖЖЖИИИК!
```

```
[keypad]> 777
ВЖЖЖИИИК!
[keypad]> 666
ВЖЖЖИИИК!
[keypad]> 444
ВЖЖЖИИИК!
[keypad]> 333
ВЖЖЖИИИК!
[keypad]> 222
```

Ты слышишь, как замок жужжит последний раз, а затем чувствуешь запах гари - замок расплавился. Ты остаешься в оружейной лавке, пока наконец готоны не взрывают корабль выстрелом со своего судна и ты не умираешь.

Даже мой маленький щенок соображает лучше.

Практические задания

1. Измените игру. Возможно, вам она не понравилась. Вам может показаться, что в ней слишком много насилия, или вам не по душе научная фантастика. Сделайте так, чтобы моя игра заработала, а затем измените ее, как вам того захочется. Это ваш компьютер, и вы можете делать все, что хотите.

2. В мой код закралась ошибка. Почему замок допускает 11 попыток угадать?

3. Объясните, как осуществляется переход в следующую комнату (сцену).

4. Добавьте чит-коды в игру, чтобы пройти больше сложных сцен. Я могу сделать это двумя словами в одной строке кода.

5. Вернитесь к моему описанию и анализу проекта, а затем попытайтесь реализовать небольшую боевую систему для героя и готонов, с которыми он сталкивается.

6. На профессиональном языке существует понятие «конечный автомат». Почитайте про него. Конечный автомат может не понадобиться вам 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. Либо вы можете использовать простые сцены из любимых книг или фильмов.

УПРАЖНЕНИЕ 44

Наследование и композиция

В сказаниях, в которых мужественные герои побеждают злодеев, всегда присутствует какое-либо 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 своей жизнью в этом зловещем лесу.

Редко попадались сказки о героях, которые достаточно умны, чтобы избежать всей ситуации целиком. 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, я лучше останусь здесь и займусь сельским хозяйством». Если бы так происходило, не было бы никаких озер лавы, смертей, воскрешений, боев на мечах, великанов, да и вообще сказочных историй. Из-за этого, лес в таких историях, кажется похожим 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, потому что 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 дурацкого леса и брать с собой армию, если когда-либо понадобится идти туда снова.

Все это – забавный способ сказать о том, что я собираюсь научить вас чему-то, что 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`». Когда вы поступаете так, язык обеспечивает то, что любые действия, которые вы производите над экземплярами `Foo`, работают так, как если бы они производились над экземпляром `Bar`. Это позволяет вам поместить общую функциональность в классе `Bar`, а затем специализировать ее в классе `Foo` так, как требуется.

При выполнении подобной специализации существует 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, когда вы определяете функцию в порождающем классе, а не в потомке.

ex44a.py

```
class Parent(object):

    def implicit(self):
        print("РОДИТЕЛЬ implicit()")

class Child(Parent):
    pass

dad = Parent()
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

```
$ python3.6 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 (т. е., `Parent`) все подклассы (т. е., `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, следующим образом:

ex44b.py

```
1 class Parent(object):
2
3     def override(self):
4         print("РОДИТЕЛЬ override()")
5
6 class Child(Parent):
```

```
 7
 8      def override(self):
 9          print("ПОТОМОК 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

```
$ python3.6 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` эта 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("РОДИТЕЛЬ altered()")
 5
 6 class Child(Parent):
 7
 8     def altered(self):
 9         print("ПОТОМОК, ДО ВЫЗОВА altered() В РОДИТЕЛЕ")
10         super(Child, self).altered()
11         print("ПОТОМОК, ПОСЛЕ ВЫЗОВА altered() В РОДИТЕЛЕ")
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 следующий вывод:

Сеанс упражнения 44c

```
$ python3.6 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

```
class Parent(object):

    def override(self):
        print("РОДИТЕЛЬ override()")

    def implicit(self):
        print("РОДИТЕЛЬ implicit()")

    def altered(self):
        print("РОДИТЕЛЬ altered()")

class Child(Parent):

    def override(self):
        print("ПОТОМОК override()")

    def altered(self):
        print("ПОТОМОК, ДО ВЫЗОВА altered() В РОДИТЕЛЕ")
        super(Child, self).altered()
```

```
20          print("ПОТОМОК, ПОСЛЕ ВЫЗОВА altered() В РОДИТЕЛЕ")
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, что у вас получилось то, что вы ожидали.

Сеанс упражнения 44d

```
$ python3.6 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 сложности, 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 практически 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, но есть и другой способ сделать то же самое – вам всего лишь нужно использовать дополнительные классы и модули, не полагаясь на неявное наследование. Если проанализировать три способа задействования наследования, то можно увидеть, что в двух из них необходимо написание нового кода, 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("КЛАСС OTHER override()")

    def implicit(self):
        print("КЛАСС OTHER implicit()")

    def altered(self):
        print("КЛАСС OTHER altered()")

class Child(object):

    def __init__(self):
        self.other = Other()

    def implicit(self):
        self.other.implicit()

    def override(self):
        print("ПОТОМОК override()")

    def altered(self):
        print("ПОТОМОК ДО ВЫЗОВА altered() В КЛАССЕ OTHER")
        self.other.altered()
        print("ПОТОМОК ПОСЛЕ ВЫЗОВА altered() В КЛАССЕ OTHER")

son = Child()

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

```
$ python3.6 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, значительная 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 счете, 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 неявных 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, я поделюсь с вами тремя рекомендациями:

1. Любой ценой старайтесь избегать множественного наследования, поскольку это слишком сложно и ненадежно. 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 каких-либо других 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 делать вследствие чего-либо, что вы используете.

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 к уклонению 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, поскольку упражнение большое. Посетите страницу **www.python.org/dev/peps/pep-0008** и начните использовать изложенные там сведения в своем коде. Вы заметите, что некоторые изложенные там моменты отличаются от того, что вы узнали из этой книги, однако теперь вы сможете понять приведенные там рекомендации и использовать их в собственном коде. В примерах кода в оставшейся части книги эти указания либо соблюдаются, либо нет, в зависимости от того, насколько запутанным становится код. Советую и вам поступать так же, поскольку понятный код является более важным, чем впечатление, которое вы произведете на всех своим знанием 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 столько проблем, сколько вы можете, самостоятельно. Как правило, при встрече с трудной проблемой люди спешат поскорее найти ответ. Это оправданно, если вы вынуждены оперативно выдать результат. Попробуйте, однако, найти самостоятельное решение, если вы располагаете временем. Остановитесь и как можно дольше концентрируйтесь только на задаче, проверьте все возможные способы, пока вы не решите ее или не сдадитесь. После этого найденные вами решения станут более качественными, и в конечном счете вы научитесь лучше решать проблемы.

Не являются ли объекты всего лишь копиями классов?

В некоторых языках 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).

УПРАЖНЕНИЕ 45

Разработка игры

Вы должны учиться самостоятельно решать возникающие 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, в котором 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 сделать это, 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 научиться 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, что 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 писать и исправлять код, пока игра не будет работать наилучшим образом.

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 книги я буду приучать вас к самостоятельности. Я держал вас за руку все это время и не могу делать это далее. Теперь я не стану рассказывать вам, как и что делать, вместо этого, вы будете по своему усмотрению выполнять упражнения, 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 решения из учебников.

Оформление функций

Применимы все правила оформления функций, которым я научил вас ранее, с добавлением следующих:

- По разным причинам программисты называют функции в составе классов методами. В целом, это только маркетинг, просто имейте в виду, что 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, будто это команда, которую вы даете классу. Например, функция `pop` «говорит», по сути, следующее: «Эй, список, удалика это». Она не называется `remove_from_end_of_list`, поскольку даже несмотря на то, что она 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, так что пишите `my_awesome_hair`, но не `myawesomehair` и не `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 – `(user, cat, dog)`, код программы использовать будет трудно.

- Старайтесь не использовать глобальные переменные, а также переменные из модулей. 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, это дурной тон. Задумайтесь.

- Всегда, всегда используйте формат `class имя(объект)`, 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 также поможет определить проблемные места.

- Старайтесь писать код так, как это делают другие программисты на Python, пока не определитесь со своим собственным стилем.

- После того, как вы 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 скажут, что ваш код вполне самодостаточен и комментарии не требуются. Скорее всего, это будет сказано официальным тоном, примерно так: «Я никогда не пишу комментарии. и 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 важнее.

- 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 и убедитесь, 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 других упражнениях и книгах, которые вы читали. После завершения проверки, исправьте все найденные ошибки и упущения. Затем повторите проверку пару раз, ища код, который можно улучшить. Применяйте на практике все советы, которые я вам предоставил, при анализе кода.

Цель этого упражнения заключается в тренировке вашего внимания на деталях в классах. После того, как вы закончите с кодом из этого упражнения, возьмите другой код и сделайте то же самое. Проанализируйте распечатку какого-либо его фрагмента и найдите все ошибки, в том числе и в оформлении. 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 у боксера.

Каркас проекта

В этом упражнении вы научитесь создавать каркас папки проекта. Эта схема станет основой, необходимой для обеспечения работы нового проекта. Она будет включать макет проекта, автоматизированные тесты, модули и установочные сценарии. Когда вам понадобится создать новый проект, просто скопируйте этот каталог, переименуйте и отредактируйте файлы, чтобы начать работу.

Установка в среде macOS/Linux

Прежде чем вы сможете начать это упражнение, вам нужно установить программное обеспечение для Python с помощью инструмента под названием pip3.6 (или просто pip) для установки новых модулей. Команда pip3.6 должна быть включена в вашу установку python3.6. Вы можете проверить это с помощью следующей команды:

```
$ pip3.6 list
pip (9.0.1)
setuptools (28.8.0)
$
```

Можно игнорировать любые предупреждения об устаревших версиях. Кроме того, вы можете увидеть сведения об установке других инструментов. Это неважно, нам нужны pip и setuptools. После этого нужно установить инструмент virtualenv:

```
$ sudo pip3.6 install virtualenv
Password:
Collecting virtualenv
  Downloading virtualenv-15.1.0-py2.py3-none-any.whl (1.8MB)
    100% |████████████████████████████████| 1.8MB 1.1MB/s
Installing collected packages: virtualenv
Successfully installed virtualenv-15.1.0
$
```

Это касается Linux или macOS. Если вы пользуетесь одной из этих ОС, то, чтобы удостовериться в успешной установке инструмента virtualenv, можно выполнить следующую команду:

```
$ whereis virtualenv
/Library/Frameworks/Python.framework/Versions/3.6/bin/virtualenv
```

В среде macOS вы увидите строку, наподобие показанной выше, но в Linux ситуация будет иной. В среде Linux, возможно, нужно будет использовать команду `virtualenv3.6` или же установить соответствующий пакет из окна менеджера пакетов.

После успешной установки инструмента virtualenv, с помощью него вы можете создать «виртуальную» установку Python (точнее, изолированное окружение), которая упростит управление версиями ваших пакетов для разных проектов. Для начала выполните указанные ниже команды, после чего я объясню их предназначение:

```
$ mkdir ~/.venvs
$ virtualenv --system-site-packages ~/.venvs/lpthw
$ . ~/.venvs/lpthw/bin/activate
(lpthw) $
```

Разберемся, что происходит в каждой строке:

1. В домашнем каталоге *~/* вы создаете папку с именем *.venvs*, в которой будут храниться все ваши виртуальные окружения.

2. Вы запускаете инструмент `virtualenv` и указываете параметр `--system-site-packages`, чтобы все установленные в системе библиотеки Python были доступны из окружения. Затем в каталоге *~/.venvs/lpthw* создается файл виртуального окружения.

3. Теперь мы входим в виртуальное окружение `lpthw`, используя оператор `.`, за которым следует строка `~/.venvs/lpthw/bin/activate`.

4. И, напоследок, вы видите в приглашении значение `(lpthw)`, оповещающее об использовании виртуального окружения `lpthw`.

Теперь вы можете видеть, что все успешно настроено.

```
(lpthw) $ which python
/Users/zedshaw/.venvs/lpthw/bin/python
(lpthw) $ python
Python 3.6.0rc2 (v3.6.0rc2:800a67f7806d, Dec 16 2016, 14:12:21)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> quit()
(lpthw) $
```

Как видно из вывода, теперь Python запускается из папки */Users/zedshaw/.venvs/lpthw/bin/python* вместо стандартного каталога установки. Также решена проблема ввода команды `python3.6`.

```
$ which python3.6
/Users/zedshaw/.venvs/lpthw/bin/python3.6
(lpthw) $
```

Аналогичный вывод будет и при выполнении команд `virtualenv` и `pip`.

Напоследок нужно установить инструмент тестирования nose, который мы будем использовать в этом упражнении.

```
$ pip install nose
Collecting nose
  Downloading nose-1.3.7-py3-none-any.whl (154kB)
    100% |████████████████████████████████| 163kB 1.4MB/s
Installing collected packages: nose
Successfully installed nose-1.3.7
(lpthw) $
```

Установка в среде Windows 10

Настройка рабочего окружения в операционной системе Windows 10 немного проще, чем в Linux или macOS, но только в том случае, если у вас установлена лишь одна версия Python. Если у вас установлены обе версии, скажем, Python 3.6 и Python 2.7, тогда я вряд ли смогу вам помочь, поскольку очень сложно управлять несколькими установками. Если вы следовали указаниям из данной книги и установили только Python версии 3.6, тогда вам нужно

сделать следующее. Для начала перейдите в корневую папку и убедитесь, что вы используете правильную версию Python.

```
> cd ~
> python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> quit()
```

Затем запустите инструмент pip, чтобы убедиться, что он установлен.

```
> pip list
pip (9.0.1)
setuptools (28.8.0)
```

Игнорируйте любые предупреждения об устаревшей версии, и ничего страшного, если у вас установлены другие пакеты. Затем, установите инструмент `virtualenv` для создания простых виртуальных окружений.

```
> pip install virtualenv
Collecting virtualenv
Using cached virtualenv-15.1.0-py2.py3-none-any.whl
Installing collected packages: virtualenv
Successfully installed virtualenv-15.1.0
```

После установки инструмента `virtualenv`, вам нужно создать папку *.venvs* и поместить в нее виртуальное окружение.

```
> mkdir .venvs
> virtualenv --system-site-packages .venvs/lpthw
Using base prefix 'c:\\users\\zedsh\\appdata\\local\\programs\\python\\python36'
New python executable in C:\Users\zedshaw\.venvs\lpthw\Scripts\python.exe
Installing setuptools, pip, wheel...done.
```

Первая команда создает папку *.venvs* для хранения виртуальных окружений. Вторая команда создает виртуальное окружение с именем `lpthw`.

Виртуальное окружение – это изолированное расположение, куда вы устанавливаете разные наборы пакетов для различных проектов.

После установки `virtualenv`, настройка окружения завершена, и вам необходимо его активировать.

```
> .\.venvs\lpthw\Scripts\activate
```

Будет запущен PowerShell-скрипт `activate`, который настроит виртуальное окружение `lpthw` в вашей оболочке командной строки. Каждый раз, когда вы будете писать программное обеспечение по примерам из этой книги, вы будете выполнять эту команду.

Обратите внимание, что в командной строке в приглашении появилось значение `(lpthw)`, указывающее, какое виртуальное окружение вы используете.

Наконец, вам необходимо установить инструмент `nose` для тестирования проектов в дальнейшем.

```
(lpthw) > pip install nose
Collecting nose
  Downloading nose-1.3.7-py3-none-any.whl (154kB)
    100% |████████████████████████████████| 163kB 1.4MB/s
Installing collected packages: nose
Successfully installed nose-1.3.7
(lpthw) >
```

Как вы видите, произошла установка пакета nose. Обратите внимание, что инструмент pip установил его в ваше виртуальное окружение *.venvs\lpthw*, а не в основной каталог для системных пакетов. Благодаря этому, вы можете использовать конфликтующие версии пакетов Python отдельно для каждого проекта, без изменения основной конфигурации системы.

Подготовка каркаса каталогов проекта

Первым делом создайте каркас каталогов, 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 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

```
try:
    from setuptools import setup
except ImportError:
    from distutils.core import setup

config = {
    'description': 'Мой проект',
    'author': 'Мое имя',
    'url': 'URL-адрес проекта',
    'download_url': 'Ссылка на скачивание',
```

```
    'author_email': 'Мой Email',
    'version': '0.1',
    'install_requires': ['nose'],
    'packages': ['NAME'],
    'scripts': [],
    'name': 'projectname'
}

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

```
from nose.tools import *
import NAME

def setup():
    print("УСТАНОВКА!")

def teardown():
    print("ЗАВЕРШЕНИЕ!")

def test_basic():
    print("ВЫПОЛНЕНИЕ!")
```

Окончательная структура каталогов

Когда вы закончите формирование структуры каталогов с файлами, она должна выглядеть следующим образом:

```
skeleton/
    NAME/
        __init__.py
    bin/
    docs/
    setup.py
    tests/
```

```
    NAME_tests.py
    __init__.py
```

Теперь вы должны выполнять свои команды из этого каталога. Если команда `ls -R` не отображает такую структуру, вы находитесь в неправильном каталоге. К примеру, многие 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 программисты совершают эту ошибку довольно часто.

Внимание! К моменту отправки книги в типографию я узнал, что проект nose был заброшен и может работать со сбоями. Если при выполнении команды `nosetests` у вас возникают странные ошибки синтаксиса, обратите внимание на текст ошибки. Если в нем упоминается версия `python2.7`, возможно, `nosetests` пытается

запустить на вашем компьютере Python версии 2.7. Решение данной проблемы состоит в том, чтобы запускать nose с помощью команды `python3.6 -m "nose"`. Эта проблема касается операционной системы Linux и macOS. В среде Windows, скорее всего, все будет нормально. Но если ошибка все же возникнет, вы также можете решить ее с помощью команды `python -m "nose"`.

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 лишь внимание, что 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 рутинной работы по подготовке проекта. Каждый раз при создании нового проекта, выполняйте следующие действия:

1. Создайте копию вашего каркаса папок 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.

Обязательный опросник

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 языке Python помочь вам.

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.

После выполнения команды `nosetests` отображается результат только одного теста. Так и должно быть?

Да, именно так. На моем компьютере результат такой же.

УПРАЖНЕНИЕ 47

Автоматизированное 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 многократной ручной проверки кода.

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 нарабатываете опыт в программировании. Вы изучаете книгу, которую держите в руках, чтобы разрабатывать программы. Теперь нужно сделать следующий шаг и написать код, который проверяет вашу программу. Так вы будете иметь четкое представление о том, что вы только что 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* в *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 с его выполнением.

Внимание! Помните о выполнении команды `nosetests` для запуска тестирования. Вы можете использовать команду `python3.6 ex47_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

```
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)
```

Теперь создайте модульный тест.

ex47_tests.py

```
from nose.tools import *
from ex47.game import Room


def test_room():
    gold = Room("GoldRoom",
                """В этой комнате полно золота, которое можно украсть.
                Здесь есть дверь с выходом на север.""")
    assert_equal(gold.name, "GoldRoom")
    assert_equal(gold.paths, {})

def test_room_paths():
    center = Room("Center", "Тестирование центральной комнаты.")
    north = Room("North", "Тестирование северной комнаты.")
    south = Room("South", "Тестирование южной комнаты.")

    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", "Вы можете идти на запад и провалиться в яму.")
    west = Room("Trees", """Здесь есть деревья, и вы можете
                            отправиться на восток.""")
    down = Room("Dungeon", "Здесь темно, и вы можете подняться вверх.")

    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 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:

1. 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.

2. Пишите отдельный тестовый файл для каждого создаваемого вами модуля.

3. Составляйте краткий код тестов (функций) и не волнуйтесь, если они будут несколько неаккуратными. В тестовых файлах, как правило, допустим небольшой беспорядок.

4. Но и в этом случае старайтесь придерживаться аккуратности и удаляйте любой повторяющийся код. Создавайте вспомогательные функции, позволяющие избавиться от дублирующего кода. Вы еще поблагодарите меня, когда внесете изменения в программу, а затем понадобится скорректировать код тестов. Дублирующийся код затруднит изменение тестов.

5. 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
...
```

```
----------------------------------------------------------------------
Ran3  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

1. Поищите в Интернете информацию о команде `nosetests`, а также поищите альтернативные возможности.

2. Поищите в Интернете информацию о «доктестах» Python и проверьте, может быть, они более удобны.

3. Расширьте возможности тестируемой сцены и переделайте код игры. Протестируйте код.

Распространенные вопросы

У меня выводится ошибка синтаксиса при выполнении команды `nosetests`. Почему это происходит?

В этом случае, 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=.` 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.
- 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 = input('> ')
words = stuff.split()
```

Да, это действительно весь код, который нам сейчас нужен, но он реально будет работать очень хорошо.

Лексические кортежи

Раз нам понятно, 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 = ('verb', 'go')
second_word = ('direction', 'north')
third_word = ('direction', 'west')
sentence = [first_word, second_word, third_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, 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 схитрим 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:

Сеанс упражнения 48 в Python

```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> int("ад")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'ад'
```

`ValueError` – это исключение, вызванное функцией `int()`, потому что вы передали `int()` нечисловое значение. Функция `int()` могла бы вернуться 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`:

ex48_convert.py

```
1 def convert_number(s):
2     try:
3         return int(s)
4     except ValueError:
5         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`.

Разрабатывая анализатор, вы должны использовать такой код, чтобы проверить, что введенные данные представляют собой число.

Тактика «сначала тест»

«Сначала тест» — это тактика программирования, когда вы пишете автоматизированный тест, который предполагает работоспособность кода, а затем пишете код, чтобы тест действительно работал. Этот метод хорош для тех случаев, когда вы не можете реализовать код, но представляете, как с ним работать. Например, если вы знаете, как использовать новый класс в другом

модуле, но еще не вполне понимаете, как этот класс реализовать, то сначала напишите тест.

Я предоставлю вам тест, а вы будете использовать его для написания кода, который заставит его работать. Для этого выполните следующие действия:

1. Создайте небольшой фрагмент предоставленного мной теста.
2. Убедитесь в том, что он выполняется, но пройти его не удается. Это будет означать, что он работает так, как задумано.
3. Откройте исходный файл *lexicon.py* и напишите код, который позволит пройти этот тест.
4. Повторяйте эти действия, пока не реализуете все части теста.

Когда вы дойдете до пункта 3, подключите другой метод написания кода:

1. Создайте каркас нужной вам функции или класса.
2. Напишите внутри него комментарии, объясняющие работу функции.
3. Напишите код, который выполняет то, что написано в комментариях.
4. Удалите все комментарии, которые просто дублируют код.

Этот метод написания кода называется псевдокодом, и он отлично работает в том случае, если вы не знаете, как что-то реализовать, но можете описать это своими словами.

Сочетание методов «сначала тест» и псевдокод позволяет свести процесс программирования к простой последовательности действий:

1. Написание фрагмента теста, который не удается пройти.
2. Написание скелета необходимой для теста функции/модуля/класса.
3. Наполнение скелета комментариями, в которых простыми словами объясняется принцип работы кода.
4. Замена комментариев кодом до удачного прохождения теста.
5. Повторение предыдущих шагов.

В данном упражнении вы попробуете этот метод работы на практике, применив предоставленный мною тест к модулю *lexicon.py*.

Что нужно тестировать?

Ниже представлен пример, *tests/lexicon_tests.py*, который вы должны использовать, но пока не набирайте его.

lexicon_tests.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'),
```

```
32                          ('noun', 'princess')])
33
34 def test_numbers():
35     assert_equal(lexicon.scan("1234"), [('number', 1234)])
36     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"),
43     [('error', 'ASDFADFASDF')])
44     result = lexicon.scan("bear IAS princess")
45     assert_equal(result, [('noun', 'bear'),
46                          ('error', 'IAS'),
47                          ('noun', 'princess')])
```

Вам нужно создать новый проект на основе «каркаса» проекта из упражнения 47. Затем вам нужно будет создать тестовый случай (тест-кейс) и файл *lexicon.py*, который в нем будет использоваться. Посмотрите путь к файлу в верхней части тестового случая, чтобы понять, как он импортируется, и выяснить, куда он отправляется.

Затем выполняйте описанные ранее действия, создавая тест небольшими фрагментами. Например, я мог бы сделать следующее:

1. Написать код импорта в верхней части и сделать так, чтобы он работал.
2. Создать пустую версию первого тестового случая, `test_directions`. Убедиться в его работоспособности.
3. Написать первую строку тестового случая `test_directions`. Сделать так, чтобы тест не был пройден.
4. Перейти к файлу *lexicon.py* и создать пустую функцию `scan()`.
5. Запустить тест и удостовериться в том, что функция `scan()`, по крайней мере, запускается, хотя и не дает положительного результата.
6. Дополнить псевдокод комментариями о том, как должна работать функция `scan()`, чтобы пройти `test_directions`.

7. Писать соответствующий комментариям код, пока тест `test_directions` не будет пройден.

8. Вернуться к тесту `test_directions` и дописать оставшиеся строки.

9. Вернуться к функции `scan()` в файле *lexicon.py* и доработать ее так, чтобы новый тест был пройден.

10. По окончании у вас будет первый рабочий тест, и вы сможете приступить к следующему.

Написание кода небольшими фрагментами позволяет разделить огромную задачу на маленькие, легко выполнимые этапы. Это все равно, что подниматься в гору с остановками через каждые пару сотен метров.

Практические задания

1. Улучшите модульный тест, чтобы охватить большую 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?

Вы, наверное, 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:

Сеанс упражнения 48 в Python

```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from ex48 import lexicon
>>> lexicon.scan("go north")
[('verb', 'go'), ('direction', 'north')]
>>> lexicon.scan("kill the princess")
[('verb', 'kill'), ('stop', 'the'), ('noun', 'princess')]
>>> lexicon.scan("eat the bear")
[('verb', 'eat'), ('stop', 'the'), ('noun', 'bear')]
```

Это также будет работать и с предложениями длиннее, например, `lexicon.scan("open the door and smack the bear in the nose")`.

Теперь давайте превратим это в класс предложения с именем `Sentence`. 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 предложения, включающий подлежащее, сказуемое и дополнение.

Соответствия и считывание

Следует учитывать пять необходимых вещей:

1. Способ циклической обработки списка кортежей. Это несложно.

2. Способ поиска «соответствий» кортежей разного типа на основе схемы «подлежащее сказуемое дополнение».

3. Способ «считывания» подходящего кортежа, чтобы мы могли делать определенный выбор.

4. Способ «пропуска» ненужных элементов, таких, как стоп-слова.

5. Объект предложения, в который следует помещать результат.

Мы поместим код этих функций в модуль `ex48.parser` 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 его.

Строение предложений

Прежде чем приступить к написанию кода, необходимо понимать принципы построения предложений. В нашем синтаксическом анализаторе мы создадим объект `Sentence` с тремя атрибутами:

`Sentence.subject` Это подлежащее любого предложения, по умолчанию это может быть «игрок», поскольку под фразой «бежать на север» подразумевается «игрок бежит на север». Это будет имя существительное.

`Sentence.verb` Это сказуемое. В предложении «бежать на север» сказуемым является слово «бежать». Это будет глагол.

`Sentence.object` Дополнение – это другое существительное, которое является объектом действия. В нашей игре мы выделяем направления, которые также будут являться дополнениями. В предложении «бежать на север», слово «север» будет дополнением. В предложении «ударить медведя», дополнением будет слово «медведя».

Затем наш синтаксический анализатор должен использовать описанные функции и, учитывая отсканированное предложение, преобразовывать его в списки объектов `Sentence` так, чтобы они соответствовали вводу.

Пара слов об исключениях

Вы уже немного знаете об исключениях, но не умеете генерировать их. Следующий 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.

Код синтаксического анализатора

Если вам нужны дополнительные сложности, прервитесь и попытайтесь написать код на основе моего описания. Если возникают трудности, вы можете подсмотреть, как поступил я, но попытка самостоятельно написать код синтаксического анализатора – отличная практика. Далее я разберу код, чтобы вы могли ввести его и сохранить в файл *ex48/parser.py*. Мы начнем с исключения, которое необходимо для синтаксического анализа ошибки:

parser.py

```
1 class ParserError(Exception):
2     pass
```

Так вы создаете класс исключений `ParserError`, который затем можете вызвать. Также нам понадобится объект `Sentence`, который мы создадим:

parser.py

```
1 class Sentence(object):
2
3     def __init__(self, subject, verb, obj):
4         # помните, мы брали кортежи ('noun','princess') и конвертировали их
5         self.subject = subject[1]
6         self.verb = verb[1]
7         self.object = obj[1]
```

Пока ничего сложного в этом коде нет. Вы просто создаете обычные классы.

В описании задачи шла речь о том, что нам необходима функция, которая будет просматривать список слов и возвращать их тип:

parser.py

```
def peek(word_list):
    if word_list:
        word = word_list[0]
        return word[0]
    else:
        return None
```

Нам нужна эта функция, чтобы принимать решение, с каким предложением мы имеем дело, основываясь на том, каким будет следующее слово. Затем мы вызываем другую функцию, чтобы взять это слово и продолжить с ним работать.

Чтобы работать со словом, мы используем функцию `match()`, которая подтверждает, что данное слово правильного типа, берет его из списка и возвращает назад:

parser.py

```
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, это довольно просто, но убедитесь, что вы понимаете этот код. Также важно, чтобы вы понимали, почему я это делаю именно так. Мне нужно взглянуть на слова в списке, чтобы определить тип предложения, а затем мне нужно сравнить эти слова для создания объекта `Sentence`.

И последнее что мне необходимо, это способ пропускать слова, которые не нужны для объекта `Sentence`. Эти слова обозначены как «стоп-слова», (тип `'stop'`), например «the», «and» и «a».

parser.py

```
1 def skip(word_list, word_type):
2     while peek(word_list) == word_type:
3         match(word_list, word_type)
```

Учитывайте, что функция `skip()` пропускает не одно слово, а все слова данного типа, которые сможет отыскать. Получается, если кто-то печатает «scream at the bear», вы получите слова «scream» и «bear».

Это наш базовый набор функций для синтаксического анализатора, и с ним мы можем анализировать практически любой текст. Однако наш анализатор очень простой, поэтому нужно добавить еще пару функций.

Сначала разберемся с анализом глаголов.

parser.py

```
1 def parse_verb(word_list):
2     skip(word_list, 'stop')
3
4     if peek(word_list) == 'verb':
5         return match(word_list, 'verb')
6     else:
7         raise ParserError("Expected a verb next.")
```

Мы пропускаем все стоп-слова, а затем проверяем, является ли следующее слово глаголом. Если нет, получаем `ParserError`. Если да, то сравниваем его и берем его из списка. Схожая функция для дополнений.

parser.py

```
1 def parse_object(word_list):
2     skip(word_list, 'stop')
3     next_word = peek(word_list)
4
5     if next_word == 'noun':
```

```
        return match(word_list, 'noun')
    elif next_word == 'direction':
        return match(word_list, 'direction')
    else:
        raise ParserError("Expected a noun or direction next.")
```

Опять, пропускаем стоп-слова, заглядываем вперед и, основываясь на данной информации, определяем, верно ли предложение. Однако, в функции `parse_object`, нам нужно обрабатывать слова «существительное» и «направление» как возможные дополнения. С подлежащим так же, но поскольку мы собираемся применять существительное «игрок», нам необходимо использовать функцию `peek()`.

parser.py

```
def parse_subject(word_list):
    skip(word_list, 'stop')
    next_word = peek(word_list)

    if next_word == 'noun':
        return match(word_list, 'noun')
    elif next_word == 'verb':
        return ('noun', 'player')
    else:
        raise ParserError("Expected a verb next.")
```

Теперь остается последняя и очень простая функция `parse_sentence()`.

parser.py

```
def parse_sentence(word_list):
    subj = parse_subject(word_list)
    verb = parse_verb(word_list)
    obj = parse_object(word_list)

    return Sentence(subj, verb, obj)
```

Эксперименты с синтаксическим анализатором

Чтобы увидеть, как анализатор работает, вы можете поиграть в игру:

Сеанс упражнения 49 в Python

```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from ex48.parser import *
>>> x = parse_sentence([('verb', 'run'), ('direction', 'north')])
>>> x.subject
'player'
>>> x.verb
'run'
>>> x.object
'north'
>>> x = parse_sentence([('noun', 'bear'), ('verb', 'eat'), ('stop',
'the'),
... ('noun', 'honey')])
>>> x.subject
'bear'
>>> x.verb
'eat'
>>> x.object
'honey'
```

Попробуйте сопоставить предложения в правильные пары в предложении. Например, как бы вы сказали: «медведь бежит на юг»?

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, что 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 это не требуется. Поверьте, это очень важный навык.

Практические задания

1. Измените код методов `parse_` и попытайтесь поместить их в класс вместо использования только как функций. 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.

5. Попробуйте написать аналогичный синтаксический анализатор для другого языка, например, русского.

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 возникают сложности, вернитесь к упражнению 46 и выполните его еще раз.

Установка фреймворка Flask

Прежде чем приступить к созданию веб-приложения, вам понадобится установить веб-фреймворк под названием *flask*. Термин «фреймворк» обычно означает «некий пакет, который упрощает для меня какие-то действия». В мире веб-приложений разработчики создают веб-фреймворки для избегания сложных проблем, с которыми они могут столкнуться при создании своих собственных сайтов. Такие общие решения распространяются в виде пакетов, которые можно скачать для использования при разработке ваших собственных проектов.

В нашем упражнении мы будем использовать фреймворк flask. Сначала изучите фреймворк flask, а затем переходите к другому, когда наберетесь опыта (или просто продолжайте использовать flask, так как его возможностей вполне достаточно).

Используя pip, установите фреймворк flask:

```
$ sudo pip install flask
[sudo] password for zedshaw:
Downloading/unpacking flask
  Running setup.py egg_info for package flask

Installing collected packages: flask
  Running setup.py install for flask

Successfully installed flask
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 flask`. Если у вас не получается установить данный фреймворк, вернитесь к упражнению 46 и проверьте, все ли вы делаете правильно.

Создание простого проекта

Для начала создадим простенькое веб-приложение «Привет, мир!» и каталог для этого проекта с помощью фреймворка flask. Сначала подготовим каталог проекта:

```
$ cd projects
$ mkdir gothonweb
$ cd gothonweb
$ mkdir bin
$ mkdir gothonweb
$ mkdir tests
$ mkdir docs
$ mkdir bin/templates
$ touch gothonweb/__init__.py
$ touch tests/__init__.py
```

Последние две команды в Windows выглядят иначе:

```
$ new-item -type file gothonweb/__init__.py
$ new-item -type file tests/__init__.py
```

Вы возьмете игру из упражнения 43 и превратите ее в веб-приложение. Но сначала нужно создать базовое приложение flask. Напишите следующий код в файле *bin/app.py*:

ex50.py

```
1 from flask import Flask
2 app = Flask(__name__)
3
4 @app.route('/')
5 def hello_world():
6     return 'Привет, мир!'
```

```
7
8 if __name__ == "__main__":
9     app.run()
```

Затем запустите приложение примерно так:

```
(lpthw) $ python3.6 app.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Наконец, запустите веб-браузер и перейдите по адресу **http://localhost:5000/**. Вы должны увидеть следующее.

Во-первых, в вашем браузере появится надпись «Привет, мир!». Во-вторых, в оболочке командной строки появится следующий вывод:

```
(lpthw) $ python3.6 app.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [07/Feb/2018 10:01:56] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [07/Feb/2018 10:01:56] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [07/Feb/2018 10:01:56] "GET /favicon.ico HTTP/1.1" 404 -
```

Это так называемые журнальные сообщения, выводимые flask, чтобы вы могли убедиться в работе сервера 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 flask, а затем реструктурирую для понимания его установки.

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`). Для подключения используется порт 5000.

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 `/`, то flask находит класс `index` и загружает его для обработки запроса.

4. Теперь, когда flask обнаружил класс `index`, вызывается метод `index.GET` для экземпляра этого класса, чтобы фактически обработать запрос. Соответствующая функция выполняется, и возвращается строка, которую flask должен передать в браузер.

5. И, наконец, flask обрабатывает запрос и отправляет ответ браузеру, в котором вы его и видите.

Убедитесь, что вы действительно поняли принцип работы. Составьте схему работы, как данные из вашего браузера передаются flask, затем методу `index.GET` и обратно в браузер.

Создание базовых шаблонов

Вы разобрали приложение flask и наверняка заметили, что Hello World! не самая привлекательная HTML-страница. Это веб-приложение, и поэтому оно нуждается в надлежащем HTML-ответе. Чтобы достичь этого, вы создадите простой шаблон, форматирующий текст крупным шрифтом зеленого цвета.

Первый шаг заключается в создании файла *templates/index.html* со следующим кодом:

index.html

```
<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!
    {% endif %}

    </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 flask 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, вам нужно добавить код, сообщающий flask, где расположен шаблон и как визуализировать его в браузере. Измените код в файле *bin/app.py* следующим образом:

app.py

```
 1 from flask import Flask
 2 from flask import render_template
 3
 4 app = Flask(__name__)
 5
 6 @app.route("/")
 7 def index():
 8     greeting = "Привет, мир!"
 9     return render_template("index.html", greeting=greeting)
10
11 if __name__ == "__main__":
12     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_python3.zip**.

После того, как внесены все изменения, перезагрузите веб-страницу в браузере, и вы увидите новое 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 функцию, `render_template()`.

2. Функция `render_template()` «знает», что нужно загружать файлы *.html* из каталога *templates/*, поскольку этот параметр настроен по умолчанию во flask.

3. Далее по коду, когда браузер встречает строку `def index`, вместо простого возврата строки `greeting`, как это было ранее, вы вызываете метод `render_template` и передаете ему приветствие в качестве переменной.

4. Метод `render_template` переходит в каталог *templates/*, ищет страницу с именем *index.html*, а затем обрабатывает ее (даже если вы явно не указали каталог *templates*).

5. В файле `templates/index.html` находится, на первый взгляд, обычный HTML-код, который на самом деле является кодом, помещенным между двумя типами маркеров. Один из них, `{% %}`, определяет части исполняемого кода (инструкции `if`, циклы `for` и т. д.). Второй, `{{ }}`, определяет места переменных, которые нужно конвертировать в текст и поместить в вывод HTML. Выполняемый код, помещенный вместо маркеров `{% %}`, не отображается в браузере.

Чтобы узнать больше о шаблонах, прочитайте справочные сведения о Jinja2.

Чтобы погрузиться глубже в это упражнение, измените значение переменной `greeting` и соответствующий HTML-код. Кроме того, создайте еще один шаблон с именем *templates/foo.html* и попробуйте применить его.

Работа над ошибками

Во-первых, удалите строку 8, в которой вы присваиваете значение переменной `greeting`; затем нажмите кнопку Обновить (Refresh) в браузере. Затем, нажав сочетание клавиш **Ctrl+C**, завершите работу flask, и начните заново. После его запуска, обновите страницу в своем браузере. Вы должны увидеть сообщение Internal Server Error (внутренняя ошибка сервера). Вернитесь к оболочке командной строки, и в строке `[VENV]` вы увидите путь к вашей директории с виртуальным окружением.

```
(lpthw) $ python3.6 app.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
[2017-02-22 14:35:54,256] ERROR in app: Exception on / [GET]
Traceback (most recent call last):
File "[VENV]/site-packages/flask/app.py",
line 1982, in wsgi_app
response = self.full_dispatch_request()
File "[VENV]/site-packages/flask/app.py",
line 1614, in full_dispatch_request
rv = self.handle_user_exception(e)
File "[VENV]/site-packages/flask/app.py",
line 1517, in handle_user_exception
reraise(exc_type, exc_value, tb)
File "[VENV]/site-packages/flask/_compat.py",
line 33, in reraise
raise value
File "[VENV]/site-packages/flask/app.py",
line 1612, in full_dispatch_request
rv = self.dispatch_request()
File "[VENV]/site-packages/flask/app.py",
line 1598, in dispatch_request
return self.view_functions[rule.endpoint](**req.view_args)
File "app.py", line 8, in index
return render_template("index.html", greeting=greeting)
```

```
NameError: name 'greeting' is not defined
127.0.0.1 - - [22/Feb/2017 14:35:54] "GET / HTTP/1.1" 500 -
```

Код работает неплохо, но вы также можете запустить flask в режиме отладки. Так вы получите исчерпывающую информацию об ошибках. Проблема с режимом отладки связана с небезопасностью запуска в Интернете, поэтому вы должны использовать следующие команды:

```
(lpthw) $ export FLASK_DEBUG=1
(lpthw) $ python3.6 app.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger pin code: 222-752-342
```

Затем обновите страницу в браузере, чтобы увидеть подробную информацию и активную консоль, которые вы сможете использовать во время отладки приложения.

Внимание! Активная консоль отладки и развернутый вывод flask делают режим отладки потенциально опасным для использования в Интернете. При помощи этой информации хакеры могут удаленно полностью управлять вашим компьютером. Если вы разместили свое веб-приложение в Интернете, никогда не запускайте режим отладки. Я вообще усложнил бы доступ к параметру `FLASK_DEBUG`. Очень заманчиво использовать режим отладки, чтобы сэкономить время при разработке, но в случае, если хакеры проникнут на ваш сервер и воспользуются данной уязвимостью, вы пожалеете о том, что поступили таким необдуманным образом.

Практические задания

1. Прочитайте документацию на сайте **flask.pocoo.org/docs/0.12/** по фреймворку, аналогичному flask.

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:5000.

В качестве альтернативы попробуйте подключиться по адресу **http://127.0.0.1:5000/**. Убедитесь, что указываете правильный номер порта.

В чем разница между flask и web.py?

Ее нет. Я просто «заблокировал» конкретную версию *web.py*, чтобы использовать ее для обучения с помощью этой книги, и назвал ее *flask*. 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 `python3.6 bin/app.py`, вы находитесь в неправильном каталоге.

Почему мы присваиваем переменной идентичное значение, `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 5000 на моем компьютере.

Вероятно, у вас установлена антивирусная программа, которая использует этот порт. Попробуйте другой порт.

После установки flask 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 flask и установить его заново. Если это не сработает, убедитесь, что вы используете правильную версию Python.

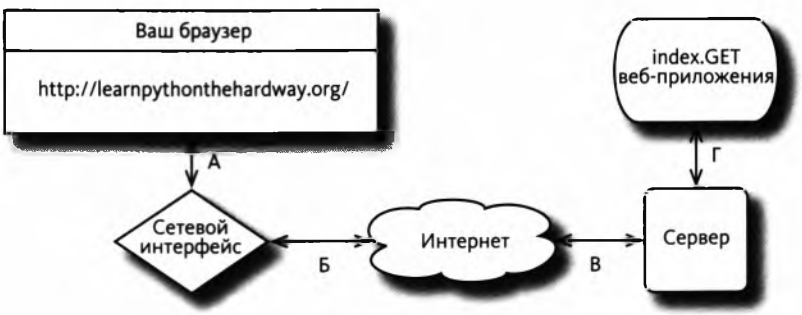
Получение ввода из браузера

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 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://test.com/** 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://test.com/**), 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://test.com/**, который указывает, куда браузер должен обращаться. Первая часть (**http**) указывает на используемый протокол (в данном случае, HyperText Transfer Protocol – Протокол передачи гипертекста). Адрес **ftp://ibiblio.org** – пример протокола FTP (File Transfer Protocol – Протокол передачи файлов). Вторая часть (**test.com**) 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 включать путь наподобие **/book** следующим образом: **http://test.com/book**. Путь указывает на файл или какой-либо другой ресурс на сервере для извлечения с помощью запроса. Существуют другие части адреса, но основные мы указали.

Соединение (подключение). После того, как браузер «узнал», что вы хотите использовать такой-то протокол (**http**), такой-то сервер (**test.com**) 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 ПК и сервером с сайтом **test.com**. 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://test.com:80** и получить тот же результат, за исключением явного использования порта 80 (который установлен по умолчанию).

Запрос. Браузер осуществляет подключение с помощью указанного вами адреса. Подключившись, 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 то, 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 значительная 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 назад в браузер в качестве ответа на запрос. В случае с файлами, сервер просто считывает их с жесткого диска и отправляет в браузер, при этом упаковывая контент с использованием специального «заголовка», по которому браузер определяет тип получаемых данных. В случае вашего приложения, вы также будете передавать данные, включая заголовок, но будете 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, которое вы сделали в упражнении 50, на соответствующие части. Если у вас это получается, вы понимаете, как устроена Всемирная паутина.

Принцип работы веб-формы

Лучший способ научиться работать с веб-формами – это написать код, который позволяет принимать данные, вводимые в элементы формы, а затем изучить, как с этими данными работать. Откройте файл *bin/app.py* и напишите в нем следующий код:

app.py

```
from flask import Flask
from flask import render_template
from flask import request

app = Flask(__name__)

@app.route("/hello")
def index():
    name = request.args.get('name', 'Некто')

    if name:
        greeting = f"Привет, {name}"
    else:
        greeting = "Привет, мир!"

    return render_template("index.html", greeting=greeting)

if __name__ == "__main__":
    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:5000/hello**. Вы должны увидеть текст «Я просто хочу сказать: *Привет, Некто*». Затем измените URL-адрес в вашем браузере следующим образом: **http://localhost:5000/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 `request.args`, позволяющий получить данные от браузера. Это простой словарь, содержащий данные в виде пар «ключ=значение».

2. Затем я конструирую элемент `greeting` с учетом указанного имени, 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:5000/hello?name=Петька&greet=Здравствуй**. Затем подкорректируйте код в файле *app.py*, внедрив `form.name` и `form.greet` следующим образом:

```
greet = request.args.get('greet', 'Привет,')

greeting = f"{greet}, {name}"
```

После этого, попробуйте перейти по указанному URL-адресу.

Также вы можете не указывать параметры приветствия и имени в URL-адресе. Если в браузере просто перейти по адресу **http://localhost:5000/hello**, вы увидите, что используются значения по умолчанию – «Некто» в качестве имени и «Привет,» в качестве приветствия.

Создание HTML-форм

Передача параметров в URL-адресе – вполне работающий 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>Простая веб-форма</title>
    </head>
    <body>
        <h1>Заполните эту форму</h1>
        <form action="/hello" method="POST">
            Приветствие: <input type="text" name="greet">
            <br/>
            Имя: <input type="text" name="name">
            <br/>
            <input type="submit">
        </form>

    </body>
</html>
```

В файл *bin/app.py* внесите следующие изменения:

app.py

```
1 from flask import Flask
2 from flask import render_template
3 from flask import request
4
5 app = Flask(__name__)
```

```
6
7 @app.route("/hello", methods=['POST', 'GET'])
8 def index():
9     greeting = "Привет, мир!"
10
11     if request.method == "POST":
12         name = request.form['name']
13         greet = request.form['greet']
14         greeting = f"{greet}, {name}"
15         return render_template("index.html", greeting=greeting)
16     else:
17         return render_template("hello_form.html")
18
19
20 if __name__ == "__main__":
21     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 «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:5000/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. Ваш запрос, как обычно, передается функции `index()`, за исключением того, что теперь используется инструкция `if`, проверяющая код `request.method` на использование метода POST или GET. Так браузер сообщает сценарию *app.py*, что запрос передается либо при отправке формы, либо параметрами URL.

2. Используется метод POST, форма обрабатывается так, как если бы она была заполнена и передана с возвращением соответствующего приветствия.

3. Если используется другой метод, возвращается файл *hello_form.html* с формой для заполнения пользователем.

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. Написание каждый раз страницы с нуля быстро надоедает. 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

```
{% extends "layout.html" %}

{% block content %}

{% if greeting %}
    Я просто хочу сказать:
    <em style="color: green; font-size: 2em;">{{ greeting }}</em>.
```

```
{% else %}
    <em>Привет</em>, мир!
{% endif %}

{% endblock %}
```

А файл *templates/hello_form.html* измените следующим образом:

hello_form_laid_out.html

```
{% extends "layout.html" %}

{% block content %}
<h1>Заполните эту форму</h1>

<form action="/hello" method="POST">
    Приветствие: <input type="text" name="greet">
    <br/>
    Имя: <input type="text" name="name">
    <br/>
    <input type="submit">
</form>

{% endblock %}
```

Все, что мы сделали, – удалили шаблонный контент из верхней и нижней части страниц, который отображается на каждой из них. Мы поместим его в единый файл *templates/layout.html*, который теперь будет участвовать в работе веб-приложения.

После того как вы внесли эти изменения, создайте новый файл *templates/layout.html* со следующим кодом:

layout.html

```
<html>
<head>
    <meta charset="utf-8">
    <title>Готоны с планеты Перкаль 25</title>
</head>
<body>
```

```
{% block content %}

{% endblock %}

</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. Другие HTML-шаблоны будут добавлены в раздел `{% block content %}`. Во flask указаны инструкции, как использовать этот *layout.html* в качестве макета, потому вы просто помещаете строку `{% extends "layout.html"%}` в начало кода ваших 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 все 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.

Создайте новый файл с именем *tests/app_tests.py* и напишите в нем следующий код:

app_tests.py

```
1 from nose.tools import *
2 from app import app
3
4 app.config['NTCNBHJDFYBT'] = True
5 web = app.test_client()
6
7 def test_index():
```

```
 8      rv = web.get('/', follow_redirects=True)
 9      assert_equal(rv.status_code, 404)
10
11      rv = web.get('/hello', follow_redirects=True)
12      assert_equal(rv.status_code, 200)
13      assert_in(b"Заполните эту форму", rv.data)
14
15      data = {'name': 'Михаил', 'greet': 'Привет, '}
16      rv = web.post('/hello', follow_redirects=True, data=data)
17      assert_in(b"Михаил", rv.data)
18      assert_in(b"Привет, ", rv.data)
```

Теперь 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 *app.py*, а затем запуск его вручную. Фреймворк flask содержит очень простой API для обработки запросов, который выглядит следующим образом:

```
data = {'name': 'Михаил', 'greet': 'Привет, '}
rv = web.post('/hello', follow_redirects=True, data=data)
```

Это означает, что вы можете передать запрос POST с помощью метода `post()`, а затем предоставить ему данные формы в виде словаря. Все остальное работает так же, как и тестирование запросов с помощью функции `web.get()`.

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 глубоко разобраться в работе последней версии приложения и, особенно, как работает автоматизированное тестирование. Разберитесь, как я

импортировал приложение *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 набросать 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 безумное занятие, но поищите документацию RFC по HTTP (она описывает, как работает протокол 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 настроить веб-сервер, например Apache, Nginx или thttpd. Попробуйте выложить на сервер пару *.html* и *.css* файлов. Не беспокойтесь, если у вас ничего не получится. Веб-серверы – та еще штука.

5. Сделайте перерыв и попробуйте разработать несколько различных веб-приложений.

Ломаем код

Отличный шанс поломать код нашего веб-приложения. Вы должны проанализировать следующее:

1. Насколько опасно использовать параметр `FLASK_DEBUG`? Будьте осторожны, не допускайте глобальных сбоев.

2. Допустим, в вашей форме не используются параметры по умолчанию. К чему это может привести?

3. Вы проверяете метод POST, а затем «еще какой-нибудь». Вы можете воспользоваться консольным инструментом curl для генерации запросов разного типа. Что получится в результате?

УПРАЖНЕНИЕ 52

Игра для Всемирной паутины

Вот мы и добрались до конца книги, и в этом упражнении я собираюсь действительно бросить вам вызов. Когда вы закончите, то станете достаточно компетентным начинающим программистом на языке Python. Вам по-прежнему понадобится прочитать еще несколько книг и разработать много 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, но «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 распределить его на несколько небольших фрагментов и постепенно выполнять их, чтобы в итоге все упражнение было сделано.

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». Данный термин программисты используют для описания процесса использования старого кода с целью его изменения, чтобы внедрить новые возможности или очистить его. Вы занимались этим, не представляя себе, что это вторая природа разработки программного обеспечения.

На данном этапе вы возьмете идеи удоботестируемой 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/planisphere.py*, а файл *tests/ex47_tests.py* в *tests/planisphere_tests.py*, после чего выполните команду `nosetests`, чтобы убедиться, что все работает.

Слово *planisphere* выступает в роли синонима слова *map* и позволяет избежать конфликта со встроенной функцией `map()`. Да здравствует тезаурус!

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 работу с файлами *planisphere.py* и *planisphere_tests.py*.

Разметьте базовую структуру карты с помощью класса `Room`, как показано ниже:

planisphere.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("Центральный коридор",
"""
Готоны с планеты Перкаль 25 захватили ваш корабль и уничтожили всю
команду. Ты - единственный, кто остался в живых. Тебе нужно выкрасть
нейтронную бомбу в оружейной лаборатории, заложить ее в топливном
отсеке и покинуть корабль в спасательной капсуле прежде, чем он
взорвется.

Ты бежишь по центральному коридору в оружейную лабораторию, когда
```

```
24 перед тобой появляется готон с красной чешуйчатой кожей, гнильми
25 зубами и в костюме клоуна. Он с ненавистью смотрит на тебя и,
26 перегородив дорогу в лабораторию, вытаскивает бластер, чтобы
27 отправить тебя к праотцам.
28 """)
29
30
31 laser_weapon_armory = Room("Оружейная лаборатория",
32 """
33 К счастью, ты знаком с культурой готонов и знаешь, что может их
34 рассмешить. Ты рассказываешь бородатый анекдот: Неоколонии,
35 изоморфно релятивные к мультиполосным гиперболическим параболоидам,
36 теоретически катаральны. Готон замирает, старается
37 сдержать смех, а затем начинает безудержно хохотать. Пока он
38 смеется, ты достаешь бластер и стреляешь готону в голову. Он падает,
39 а ты перепрыгиваешь через него и бежишь в оружейную лабораторию.
40
41 Ты вбегаешь в оружейную лабораторию и начинаешь обыскивать комнату,
42 спрятались ли тут другие готоны.  Стоит мертвая тишина. Ты бежишь в
43 дальний угол комнаты и находишь нейтронную бомбу в защитном
44 контейнере. На лицевой стороне контейнера расположена панель с
45 кнопками, и тебе надо ввести правильный код, чтобы достать бомбу.
46 Если ты 10 раз введешь неправильный код, контейнер заблокируется
47 и ты не сможешь достать бомбу. Учти, что код состоит из 3 цифр.
48 """)
49
50
51 the_bridge = Room("Топливный отсек",
52 """
53 Контейнер открывается со щелчком и выпускает сизый газ. Ты
54 вытаскиваешь нейтронную бомбу и бежишь в топливный отсек, чтобы
55 установить бомбу в нужном месте, активировать ее и успеть смотаться
56 с корабля.
57
58 Ты вбегаешь в топливный отсек с нейтронной бомбой и видишь пятерых
59 готонов, безуспешно пытающихся управлять кораблем. Один уродливее
60 другого, и все в клоунских костюмах, как и готон, убитый тобой. Они
61 не достают оружие, так как видят бомбу в твоих руках и не хотят,
62 чтобы ты взорвал ее. Преимущество явно на твоей стороне.
63 """)
64
65
66 escape_pod = Room("Спасательная капсула",
67 """
```

```
Ты указываешь бластером на бомбу в своих руках. Готоны поднимают
лапы вверх и в страхе потеют. Ты осторожно, не отворачиваясь,
подходишь к двери и аккуратно устанавливаешь бомбу, держа готонов на
мушке. Ты запрыгиваешь в шлюз и закрываешь дверь ударом по кнопке, а
затем бластером расплавляешь замок, чтобы готоны не смогли открыть её.
Теперь тебе нужно залезть в спасательную капсулу и удрать с корабля
к чертям собачьим.

Ты мчишься по отсеку со спасательными капсулами. Похоже, готонов на
корабле больше нет, потому что никто тебе не мешает. Некоторые из
капсул могут быть повреждены и взорвутся во время полета. Всего
капсул пять, и у тебя нет времени, чтобы осматривать каждую из них
на отсутствие повреждений. Задумавшись на секунду, ты решаешь сесть
в капсулу под номером... Капсулу под каким номером ты выбираешь?
""")


the_end_winner = Room("Конец",
"""
Ты запрыгиваешь в капсулу номер 2 и нажимаешь кнопку отстыковки.
Капсула вылетает в космическое пространство, а затем отправляется к
планете неподалеку. Ты смотришь в иллюминатор и видишь, как ваш
корабль взрывается. Его осколки повреждают топливный отсек корабля
готонов, и тот тоже разлетается в клочья. Победа за вами!
""")


the_end_loser = Room("Конец",
"""
Ты запрыгиваешь в капсулу со случайным номером и нажимаешь кнопку
отстыковки. Капсула вылетает в космическое пространство, а затем
взрывается с яркой вспышкой, разбрасывая осколки. Ты умираешь.
"""
)

escape_pod.add_paths({
    '2': the_end_winner,
    '*': the_end_loser
})

generic_death = Room("death", "Вы мертвы.")

the_bridge.add_paths({
    'throw the bomb': generic_death,
```

```
112     'slowly place the bomb': escape_pod
113 })
114
115 laser_weapon_armory.add_paths({
116     '0132': the_bridge,
117     '*': generic_death
118 })
119
120 central_corridor.add_paths({
121     'shoot!': generic_death,
122     'dodge!': generic_death,
123     'tell a joke': laser_weapon_armory
124 })
125
126 START = 'central_corridor'
127
128 def load_room(name):
129     """
130     Здесь есть потенциальная проблема безопасности.
131     Кто может задать имя? Может ли это негативно повлиять на переменную?
132     """
133     return globals().get(name)
134
135 def name_room(room):
136     """
137     Аналогичная проблема безопасности по поводу сцены?
138     Можете предложить лучшее решение вместо глобального поиска?
139     """
140     for key, value in globals().items():
141         if value == room:
142             return key
```

Обратите внимание, что есть несколько проблем, связанных с нашим классом `Room` и этой картой:

1. 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.

2. В исходной игре выполнялся код, генерирующий случайный верный код к контейнеру 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.

3. Я создал код `generic_death`, выполняемый при любом неправильном решении в игре, который вы должны закончить самостоятельно. Вам нужно будет просмотреть код исходной игры и добавить все исходы игрового процесса, а также все их проверить.

4. 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/planisphere_tests.py*, необходимый для начала:

planisphere_tests.py

```
from nose.tools import *
from gothonweb.planisphere import *

def test_room():
    gold = Room("GoldRoom",
                """В этой комнате полно золота, которое можно украсть.
                Здесь есть дверь с выходом на север.""")
    assert_equal(gold.name, "GoldRoom")
    assert_equal(gold.paths, {})

def test_room_paths():
    center = Room("Center", "Тестирование центральной комнаты.")
    north = Room("North", "Тестирование северной комнаты.")
    south = Room("South", "Тестирование южной комнаты.")

    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", "Вы можете идти на запад и провалиться в яму.")
    west = Room("Trees", """Здесь есть деревья, и вы можете отправиться
                            на восток.""")
    down = Room("Dungeon", "Здесь темно, и вы можете подняться вверх.")

    start.add_paths({'west': west, 'down': down})
    west.add_paths({'east': start})
    down.add_paths({'up': start})
```

```
29
30      assert_equal(start.go('west'), west)
31      assert_equal(start.go('west').go('east'), start)
32      assert_equal(start.go('down').go('up'), start)
33
34  def test_gothon_game_map():
35      start_room = load_room(START)
36      assert_equal(start_room.go('shoot!'), generic_death)
37      assert_equal(start_room.go('dodge!'), generic_death)
38
39      room = start_room.go('tell a joke')
40      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 карту. Он должен включать в себя 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.

Разработка движка

Итак, у вас теперь есть работающая карта игры и хороший модульный тест для нее. Теперь вам нужно разработать небольшой игровой движок, который будет 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 персонаж не погибнет.

Для достижения обозначенной 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

```
from flask import Flask, session, redirect, url_for, escape, request
from flask import render_template
from gothonweb import planisphere

app = Flask(__name__)

@app.route("/")
def index():
    # используется для "настройки" сеанса с начальными значениями
    session['room_name'] = planisphere.START
    return redirect(url_for("game"))

@app.route("/game", methods=['GET', 'POST'])
def game():
    room_name = session.get('room_name')

    if request.method == "GET":
        if room_name:
            room = planisphere.load_room(room_name)
            return render_template("show_room.html", room=room)
        else:
            # почему здесь этот код? Нужен ли он?
            return render_template("you_died.html")
    else:
        action = request.form.get('action')

        if room_name and action:
            room = planisphere.load_room(room_name)
            next_room = room.go(action)

            if not next_room:
                session['room_name'] = planisphere.name_room(room)
            else:
                session['room_name'] = planisphere.name_room(next_room)

        return redirect(url_for("game"))
```

```
37
38
39 # ВЫ ДОЛЖНЫ ИЗМЕНИТЬ ЭТОТ КОД, ЕСЛИ РАЗМЕЩАЕТЕ ЭТОТ СЦЕНАРИЙ
40 # В ИНТЕРНЕТЕ
41 app.secret_key = 'A0Zr98j/3yX R~XHH!jmN]LWX/,?RT'
42
43 if __name__ == "__main__":
44     app.run()
```

Несмотря на наличие незнакомого кода в этом сценарии, удивительно, что целый движок веб-игры помещается в небольшом файле.

Перед запуском файла *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 ее, в окне программы Terminal (Терминал) 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 это 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

```
{% extends "layout.html" %}

{% block content %}

<h1> {{ room.name }} </h1>
<pre>
```

```
{{ room.description }}
</pre>

{% if room.name in ["death", "The End"] %}
    <p><a href="/">Играть с начала?</a></p>
{% else %}
    <p>
    <form action="/game" method="POST">
        - <input type="text" name="action"> <input type="SUBMIT">
    </form>
    </p>
{% endif %}

{% endblock %}
```

Это шаблон для отображения сцены в игровом процессе. Далее вам нужно сообщить пользователям об окончании 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 с начала</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 используются пользовательские сеансы.

2. Удалите файлы из каталога *sessions/* и убедитесь, что игра запустилась заново.

3. Выполните команду `python3.6 app.py` и протестируйте игру.

Кроме того, вы должны уметь обновлять игру и исправлять в ней баги, а также изменять 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 любые, которые вы найдете самостоятельно.

2. Улучшите автоматизированные тесты, чтобы более детально проверять приложения. Разработайте тест, позволяющий проверять работу веб-приложений лучше, чем в браузере.

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 на той или иной сцене в игре.

7. Добавьте любые другие возможности, которые вам захочется 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 значительная 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 документацию flask, касающуюся использования виртуальных сеансов во время тестирования, по адресу **flask.pocoo.org/docs/0.12/testing/#other-testing-tricks**.

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 еще не совсем 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 ней на отлично. Эта книга учит вас получению нужных навыков и отношению, необходимых, чтобы прочитать другие книги по языку Python и чему-то научиться. Обучение даже может показаться более простым.

Я рекомендую вам изучить следующие проекты и попробовать использовать их в своей работе:

- Легкий способ изучить Ruby (**learnrubythehardway.org**). Вы станете еще более продвинутым программистом, поскольку освоите дополнительные языки программирования, в данном случае, Ruby.

- Веб-фреймворк Django (**docs.djangoproject.com/en/1.10/intro/**). 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, 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.

- TensorFlow (**www.tensorflow.org**). Инструменты для машинного обучения и визуализации.

- Requests (**docs.python-requests.org/en/latest/index.html**). Библиотека для управления HTTP-запросами.

- ScraPy (**scrapy.org**). Фреймворк для сбора данных с веб-сайтов.

- Kivy (**kivy.org**). Графический фреймворк для разработки пользовательских интерфейсов для настольных и мобильных платформ.

- Learn C the Hard Way (**c.learncodethehardway.org**) после того, как вы познакомитесь с Python, попробуйте изучить язык C и алгоритмы с помощью еще одной моей книги. Не торопитесь; язык C существенно отличается от Python, но прекрасно подходит для обучения.

Выберите один из представленных выше ресурсов, и поищите учебники и документацию по выбранной теме. По мере обучения, вводите код вручную и выполняйте его. Именно так поступаю я. И все программисты поступают точно так же. 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 вас 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 так, 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 выполняю его.

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 выше процесс также применим ко всем новым технологиям, с учетом, что есть книги по изучаемой теме. Если подходящей книги нет, придерживайтесь указанного выше процесса, используя документацию в Интернете 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. Изучив 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 быть глупым туристом. Глупый турист тот, кто отправляется путешествовать в другую страну и жалуется, что местная еда не такая, как дома. «Почему я не могу купить вкусный гамбургер в этой дурацкой стране!?» Когда вы изучаете новый язык, учитывайте, что какие-то процедуры в нем выполняются не глупым образом, а просто иначе. Примите это, и вы легко сможете изучить его.

После изучения языка, не будьте рабом чужих привычек. Иногда люди, изучающие программирования, 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. На момент написания этой книги, 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 в данный момент моей жизни это занятие мне не кажется интересным. В долгом процессе своего 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 вам делать что-то по-настоящему интересное.

Программирование как интеллектуальная деятельность – это единственный вид творчества, который позволяет создавать интерактивные предметы искусства. Вы можете создавать проекты, с 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 (не имея 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, нужно попытаться направить это на улучшение своей жизни, любым доступным вам способом. Вперед, исследуйте этот интересный, удивительный, новый интеллектуальный маршрут, который за последние 50 лет исследовал чуть ли ни каждый. Попробуйте и вы!

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 здорово!

Приложение. Экспресс-курс по оболочке командной строки

Это приложение представляет собой очень быстрый 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, с помощью команд которой ваш компьютер сможет выполнять определенные задачи. Курс, разумеется, не так подробен и детализирован, как другие мои книги. В нем преследуется цель научить вас использовать компьютер, как это делают программисты. Когда вы дочитаете приложение, вы сможете использовать большинство основных команд, с которыми программисты сталкиваются ежедневно. Вы разберетесь с каталогами и другими концепциями.

Единственный совет, который я собираюсь дать вам, это:

Соберитесь и введите все описываемые команды!

Прошу простить, но это действительно вам нужно 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 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, вы 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 блокнот и ручку.
- Начните читать приложение с самого начала и выполняйте каждое упражнение точно так, как написано в книге.
- Если вы не поняли что-либо, 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 части 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, а затем прикрепить на стену в ванной комнате. Во время душа, закрыв глаза, пытайтесь вспомнить информацию, поглядывая на лист в случае затруднения.

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, но, 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.

Подготовка

Сейчас вы научитесь делать три вещи:

- Работать в оболочке командной строки (консоли, терминале, PowerShell).

- Понимать, что вы только что сделали.

- Выполнять дополнительные операции.

В этом первом упражнении вы научитесь получать доступ к оболочке командной строки и запускать ее так, чтобы можно было выполнить 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 **z**, 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 **T,** щелкните мышью по ней, чтобы вызвать меню. Выберите в меню пункт Параметры Þ Оставить в **Dock** (Options Þ 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 версии, 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 в Интернете 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 реально одаренный друг, пользующийся Linux, не обращайте на него 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 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 вы доверяете, а не от незнакомых хулиганов в Интернете.

Дополнительно

Это упражнение включает объемный дополнительный раздел. Остальные упражнения не так велики, и сейчас вы подготовитесь к остальной части приложения, заучив дополнительные сведения. Поверьте мне, и дальнейшее обучение окажется более быстрым и простым.

Linux/macOS

Используя этот список команд, подготовьте карточки 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 текст на экран
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 ОСТОРОЖНО!

Учите, учите и еще раз учите! Учите, пока описание команды не будет отскакивать от зубов при виде ее имени. Затем обратите процесс, чтобы читать описание команды и сразу вспоминать ее имя. Так вы запомните их все.

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 все приведенное здесь, только частично:

- Вам не нужно вводить символы `$` (Unix) или `>` (Windows). Это лишь демонстрация моего сеанса, чтобы вы могли увидеть результат действия команды.
- Вы вводите команды после символа `$` 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 вы могли наглядно увидеть результат:

Linux/macOS

Сеанс упражнения 2

```
$ pwd
/Users/zedshaw
$
```

Windows

Сеанс упражнения 2 в Windows

```
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 привыкает, что символ $ обозначает приглашение.

Что вы изучили

Содержимое вашей 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, будет тоже выглядеть иначе. Суть заключается 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 точно 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.

Если вы заблудились

Когда вы выполняете все эти команды, вы можете потеряться в дебрях каталогов. Вы можете не знать, где находитесь, или где находится нужный файл, и не иметь ни малейшего представления о том, как продолжить. Чтобы решить эту проблему, я собираюсь научить вас пользоваться командами, способными прояснить ситуацию.

Каждый раз, когда вы чувствуете, что-то идет не так, то, скорее всего, причина заключается в том, что вы вводили команды, не зная, где находитесь. Решение заключается в том, что вы должны выполнить команду для вывода текущего каталога. Так вы узнаете, где находитесь.

Следующим важным моментом является необходимость вернуться в безопасное место, в 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.

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. Перед выполнением всех упражнений из этого приложения, всегда переходите в домашний каталог!

Linux/macOS

Сеанс упражнения 4

```
$ pwd
$ cd ~
$ mkdir temp
$ mkdir temp/stuff
$ mkdir temp/stuff/things
$ mkdir -p temp/stuff/things/orange/apple/pear/grape
$
```

Windows

Сеанс упражнения 4 в Windows

```
> pwd
> cd ~
> mkdir temp

    Каталог: C:\Users\zed


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        07.02.2018     16:52                tmp


> mkdir temp/stuff

    Каталог: C:\Users\zed\temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        07.02.2018     16:53                stuff


> mkdir temp/stuff/things

    Каталог: C:\Users\zed\temp\stuff


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        07.02.2018     16:53                things


> mkdir temp/stuff/things/orange/apple/pear/grape

    Каталог: C:\Users\zed\temp\stuff\things\orange\apple\pear
```

```
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        07.02.2018     16:55                grape
```

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. 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 он поддерживается 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 "Моя папка"`.

- 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*).

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.

Linux/macOS

Сеанс упражнения 5

```
$ cd temp
$ pwd
```

```
~/temp
$ cd stuff
$ pwd
~/temp/stuff
$ cd things
$ pwd
~/temp/stuff/things
$ cd orange/
$ pwd
~/temp/stuff/things/orange
$ cd apple/
$ pwd
~/temp/stuff/things/orange/apple
$ cd pear/
$ pwd
~/temp/stuff/things/orange/apple/pear
$ cd grape/
$ pwd
~/temp/stuff/things/orange/apple/pear/grape
$ cd ..
$ cd ..
$ pwd
~/temp/stuff/things/orange/apple
$ cd ..
$ cd ..
$ pwd
~/temp/stuff/things
$ cd ../../..
$ pwd
~/
$ cd temp/stuff/things/orange/apple/pear/grape
$ pwd
~/temp/stuff/things/orange/apple/pear/grape
$ cd ../../../../../../../
$ pwd
~/
$
```

Windows

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 orange
> pwd

Path
----
C:\Users\zed\temp\stuff\things\orange

> cd apple
> pwd

Path
----
C:\Users\zed\temp\stuff\things\orange\apple

> cd pear
> pwd

Path
----
C:\Users\zed\temp\stuff\things\orange\apple\pear

> cd grape
> pwd

Path
----
```

```
C:\Users\zed\temp\stuff\things\orange\apple\pear\grape

> cd ..
> cd ..
> cd ..
> pwd

Path
----
C:\Users\zed\temp\stuff\things\orange

> cd ../..
> pwd

Path
----
C:\Users\zed\temp\stuff

> cd ..
> cd ..
> cd temp/stuff/things/orange/apple/pear/grape
> 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 моем 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 с помощью символов `...`

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 совершенно не понимает принцип 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 *apple*.
- С помощью одной команды `cd` вернитесь в каталог *temp*, но не выйдите из него случайно.
- 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 Моя папка, используйте команду `cd "Моя папка"`.

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.

Linux/macOS

Сеанс упражнения 6

```
$ cd temp
$ ls
stuff
$ cd stuff
$ ls
things
$ cd things
$ ls
orange
$ cd orange
$ ls
apple
$ cd apple
$ ls
pear
$ cd pear
$ ls
$ cd grape
$ ls
$ cd ..
$ ls
```

```
grape
$ cd ../../../
$ ls
orange
$ cd ../../
$ ls
stuff
$
```

Windows

Сеанс упражнения 6 в Windows

```
> cd temp
> ls

    Каталог: C:\Users\zed\temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        07.02.2018     16:52                stuff


> cd stuff
> ls


    Каталог: C:\Users\zed\temp\stuff


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        07.02.2018     16:53                things


> cd things
> ls


    Каталог: C:\Users\zed\temp\stuff\things
```

```
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        07.02.2018     16:53                orange


> cd orange
> ls


    Каталог: C:\Users\zed\temp\stuff\things\orange


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        07.02.2018     16:55                apple


> cd apple
> ls


    Каталог: C:\Users\zed\temp\stuff\things\orange\apple


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        07.02.2018     16:55                pear


> cd pear
> ls


    Каталог: C:\Users\zed\temp\stuff\things\orange\apple\pear


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        07.02.2018     16:55                grape


> cd grape
> ls
> cd ..
```

```
> ls


    Каталог: C:\Users\zed\temp\stuff\things\orange\apple\pear


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        07.02.2018     16:55                grape


> cd ..
> ls


    Каталог: C:\Users\zed\temp\stuff\things\orange\apple


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        07.02.2018     16:55                pear


> cd ../../..
> ls


    Каталог: C:\Users\zed\temp\stuff


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        07.02.2018     16:55                things


> cd ..
> ls


    Каталог: C:\Users\zed\temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
```

```
d-----        07.02.2018     16:55                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`.

Удаление каталога (`rmdir`)

В этом упражнении вы научитесь удалять пустые каталоги.

Практикум

Linux/macOS

Сеанс упражнения 7

```
$ cd temp
$ ls
stuff
$ cd stuff/things/orange/apple/pear/grape/
$ cd ..
$ rmdir grape
$ cd ..
$ rmdir pear
$ cd ..
$ ls
apple
$ rmdir apple
$ cd ..
$ ls
orange
$ rmdir orange
$ 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).

Windows

Сеанс упражнения 7 в Windows

```
> cd temp
> ls


    Каталог: C:\Users\zed\temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        07.02.2018     16:55                stuff


> cd stuff/things/orange/apple/pear/grape/
> cd ..
> rmdir grape
> cd ..
> rmdir pear
> cd ..
> rmdir apple
> cd ..
> rmdir orange
> cd ..
> ls


    Каталог: C:\Users\zed\temp\stuff


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        07.02.2018     16:55                things


> rmdir things
> cd ..
```

```
> ls


    Каталог: C:\Users\zed\temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        07.02.2018     16:55                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 их вводите их без ошибок. Каждая ошибка – это результат того, что вы были невнимательны. Если вы стали делать 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 в 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.

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

Linux/macOS

Сеанс упражнения 8

```
$ cd temp
$ mkdir 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
$
```

Windows

Сеанс упражнения 8 в Windows

```
> cd temp
> mkdir i/like/icecream


    Каталог: C:\Users\zed\temp\i\like


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        07.02.2018     16:55                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
>
```

Внимание! В операционной системе Windows, обычно, не нужно указывать параметр `-p` в отличие от Linux. Тем не менее, это нововведение, и, возможно, вы можете столкнуться с устаревшими версиями Windows PowerShell, в которых требуется указывать параметр `-p`.

Что вы изучили

Изучая эти команды, вы вторгаетесь во владения программистов, но команды настолько удобны, что я не могу не 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.
- Помните, что в операционной системе Windows для работы с полными путями не требуется указывать параметр `-p`.

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

Linux/macOS

Сеанс упражнения 9

```
$ cd temp
$ touch iamcool.txt
$ ls
```

```
iamcool.txt
$
```

Windows

Сеанс упражнения 9 в Windows

```
> cd temp
> New-Item iamcool.txt -type file
> ls


    Каталог: C:\Users\zed\temp\


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        07.02.2018     16:55                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 с подтверждением, действительно ли вы хотите удалить каталог.

Копирование файла (cp)

В этом упражнении вы узнаете, как скопировать файл из одного каталога в другой при помощи команды cp.

Практикум

Linux/macOS

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
$
```

Windows

Сеанс упражнения 10 в Windows

```
> cd temp
> cp iamcool.txt neat.txt
> ls


    Каталог: C:\Users\zed\temp
```

```
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a---         07.02.2018     16:55              0 iamcool.txt
-a---         07.02.2018     16:55              0 neat.txt


> cp neat.txt awesome.txt
> ls


    Каталог: C:\Users\zed\temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a---         07.02.2018     16:55              0 awesome.txt
-a---         07.02.2018     16:55              0 iamcool.txt
-a---         07.02.2018     16:55              0 neat.txt


> cp awesome.txt thefourthfile.txt
> ls


    Каталог: C:\Users\zed\temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a---         07.02.2018     16:55              0 awesome.txt
-a---         07.02.2018     16:55              0 iamcool.txt
-a---         07.02.2018     16:55              0 neat.txt
-a---         07.02.2018     16:55              0 thefourthfile.txt


> mkdir something


    Каталог: C:\Users\zed\temp


Mode                 LastWriteTime         Length Name
```

```
----                 -------------         ------ ----
d----         07.02.2018     16:55                something


> cp awesome.txt something/
> ls


    Каталог: C:\Users\zed\temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----         07.02.2018     16:55                something
-a---         07.02.2018     16:55              0 awesome.txt
-a---         07.02.2018     16:55              0 iamcool.txt
-a---         07.02.2018     16:55              0 neat.txt
-a---         07.02.2018     16:55              0 thefourthfile.txt


> ls something


    Каталог: C:\Users\zed\temp\something


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a---         07.02.2018     16:55              0 awesome.txt


> cp -recurse something newplace
> ls newplace


    Каталог: C:\Users\zed\temp\newplace


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a---         07.02.2018     16:55              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. И 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 ее, 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 в Интернете. За редким исключением (вроде `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 данное имя обозначает каталог, и поэтому, если каталог не существует, отображается сообщение об ошибке.

Перемещение файла (mv)

В этом упражнении вы узнаете, как перемещать файлы из одного каталога в другой при помощи команды mv.

Практикум

Linux/macOS

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
$
```

Windows

Сеанс упражнения 11 в Windows

```
> cd temp
> mv awesome.txt uncool.txt
> ls


    Каталог: C:\Users\zed\temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----         07.02.2018     16:55                newplace
d----         07.02.2018     16:55                something
-a---         07.02.2018     16:55              0 iamcool.txt
-a---         07.02.2018     16:55              0 neat.txt
-a---         07.02.2018     16:55              0 thefourthfile.txt
```

```
-a---         07.02.2018     16:55              0 uncool.txt


> mv newplace oldplace
> ls


    Каталог: C:\Users\zed\temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----         07.02.2018     16:55                oldplace
d----         07.02.2018     16:55                something
-a---         07.02.2018     16:55              0 iamcool.txt
-a---         07.02.2018     16:55              0 neat.txt
-a---         07.02.2018     16:55              0 thefourthfile.txt
-a---         07.02.2018     16:55              0 uncool.txt


> mv oldplace newplace
> ls newplace


    Каталог: C:\Users\zed\temp\newplace


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a---         07.02.2018     16:55              0 awesome.txt


> ls


    Каталог: C:\Users\zed\temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----         07.02.2018     16:55                newplace
d----         07.02.2018     16:55                something
-a---         07.02.2018     16:55              0 iamcool.txt
-a---         07.02.2018     16:55              0 neat.txt
```

```
-a---          07.02.2018     16:55              0 thefourthfile.txt
-a---          07.02.2018     16:55              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.

Просмотр файла (`less, more`)

Для выполнения этого упражнения понадобится выполнить определенную подготовку. Вам также понадобится простой редактор, позволяющий создавать и сохранять текстовые файлы (с расширением *.txt*). Выполните следующие действия:

1. Запустите текстовый редактор и напечатайте любой текст в созданном файле. В операционной системе macOS это может быть программа TextWrangler. В системе Windows – Notepad++, а в Linux – gedit. Любой текстовый редактор подойдет.

2. Сохраните файл на рабочем столе под именем *test.txt*.

3. 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

Linux/macOS

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).

Windows

Сеанс упражнения 12 в Windows

```
> more test.txt
[здесь отображается содержимое файла]
>
```

Примечание. Обратите внимание: 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.
- Теперь выполните упражнение снова, но на этот раз 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 пустые файлы, созданные вами ранее.
- Команда `cp` перезаписывает файлы, если они существуют, поэтому будьте осторожны при копировании файлов.

Вывод содержимого файла (`cat`)

Вновь потребуется небольшая подготовка и создание файлов в одной программе и получение 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

Linux/macOS

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.
$
```

Windows

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`.

- **Unix:** Попробуйте выполнить команду `cat test.txt test2.txt` и посмотрите, что произойдет.
- **Windows:** Попробуйте выполнить команду `cat test.txt, test2.txt` и посмотрите, что произойдет.

Удаление файла (`rm`)

В этом упражнении вы научитесь удалять (стирать) файлы с помощью команды `rm`.

Практикум

Linux/macOS

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
$
```

Windows

Сеанс упражнения 14 в Windows

```
> cd temp
> ls
```

```
    Каталог: C:\Users\zed\temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----         07.02.2018     16:55                newplace
d----         07.02.2018     16:55                something
-a---         07.02.2018     16:55              0 iamcool.txt
-a---         07.02.2018     16:55              0 neat.txt
-a---         07.02.2018     16:55              0 thefourthfile.txt
-a---         07.02.2018     16:55              0 uncool.txt


> rm uncool.txt
> ls

    Каталог: C:\Users\zed\temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----         07.02.2018     16:55                newplace
d----         07.02.2018     16:55                something
-a---         07.02.2018     16:55              0 iamcool.txt
-a---         07.02.2018     16:55              0 neat.txt
-a---         07.02.2018     16:55              0 thefourthfile.txt


> rm iamcool.txt
> rm neat.txt
> rm thefourthfile.txt
> ls

    Каталог: C:\Users\zed\temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----         07.02.2018     16:55                newplace
d----         07.02.2018     16:55                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` для папки с содержимым? Попытка провалилась, поскольку нельзя удалять каталоги с содержимым. 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 во всех предыдущих упражнениях.
- Пометьте в записной книжке, что нужно быть осторожным при рекурсивном удалении файлов.

Выход из оболочки (`exit`)

Практикум

Linux/macOS

Сеанс упражнения 15

```
$ exit
```

Windows

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 вас 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 приведу несколько источников информации, которые стоят того, чтобы их посетить.

Руководства по Unix Bash

Оболочка командной строки, которой 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

Символы

Латиница

A

B

C

D

E

F

G

H

I

L

M

N

Научно-популярное издание

МИРОВОЙ КОМПЬЮТЕРНЫЙ БЕСТСЕЛЛЕР

Зед Шоу

ЛЕГКИЙ СПОСОБ ВЫУЧИТЬ PYTHON 3

Главный редактор *Р. Фасхутдинов*
Руководитель направления *В. Обручев*
Ответственный редактор *Е. Минина*
Младший редактор *Д. Атакишиева*
Художественный редактор *А. Шуклин*

В коллаже на обложке использованы иллюстрации:
Mikhail H, Ozz Design / Shutterstock.com
Используется по лицензии от Shutterstock.com

ООО «Издательство «Эксмо»
123308, Москва, ул. Зорге, д. 1. Тел.: 8 (495) 411-68-86.
Home page: www.eksmo.ru E-mail: info@eksmo.ru
Өндіруші: «ЭКСМО» АҚБ Баспасы, 123308, Мәскеу, Ресей, Зорге көшесі, 1 үй.
Тел.: 8 (495) 411-68-86.
Home page: www.eksmo.ru E-mail: info@eksmo.ru.
Тауар белгісі: «Эксмо»
Интернет-магазин : www.book24.ru

Интернет-магазин : www.book24.kz
Интернет-дүкен : www.book24.kz
Импортёр в Республику Казахстан ТОО «РДЦ-Алматы».
Қазақстан Республикасындағы импорттаушы «РДЦ-Алматы» ЖШС.
Дистрибьютор и представитель по приему претензий на продукцию,
в Республике Казахстан: ТОО «РДЦ-Алматы»
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) 251-59-90/91/92; E-mail: RDC-Almaty@eksmo.kz
Өнімнің жарамдылық мерзімі шектелмеген.
Сертификация туралы ақпарат сайтта: www.eksmo.ru/certification

Сведения о подтверждении соответствия издания согласно законодательству РФ о техническом регулировании можно получить на сайте Издательства «Эксмо» www.eksmo.ru/certification
Өндірген мемлекет: Ресей. Сертификация қарастырылмаған

Подписано в печать 16.04.2019. Формат 70x100$^{1}/_{16}$.
Печать офсетная. Усл. печ. л. 29,81.
Тираж 2000 экз. Заказ № 2582.

Отпечатано в ОАО «Можайский полиграфический комбинат».
143200, г. Можайск, ул. Мира, 93.
www.oaompk.ru, тел.: (495) 745-84-28, (49638) 20-685

12+

EKSMO.RU
новинки издательства

ISBN 978-5-04-093536-9

Вы выучите Python 3!

Хотите получить «черный пояс»
по программированию?
Зед Шоу гарантирует его вам!

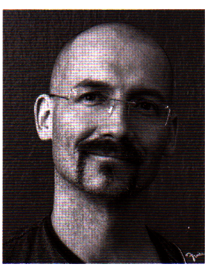
В этой книге вы найдете

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.

Бумажные книги были и остаются отличными «наставниками» по программированию. Магия в том, что их всегда можно закрыть и попробовать воспроизвести пример кода по памяти. А это один из лучших способов быстро начать программировать. Любая книга не подойдет, но на это проверенное издание можно с уверенностью положиться — миллионы последователей авторского подхода Зеда Шоу тому веское доказательство.

Кирилл Жвалов,
сооснователь «Moscow Coding School»

ISBN 978-5-04-093536-9