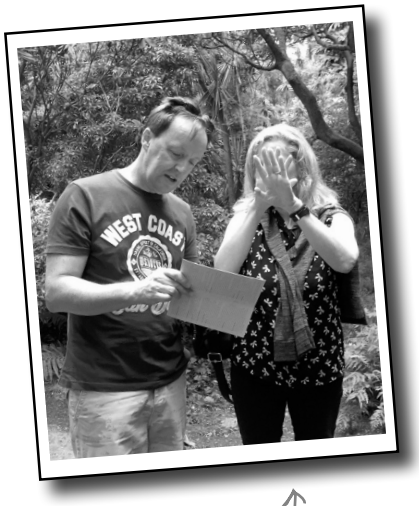
Я посвящ ю эту книгу всем бескорыстным
член м сообществ Python, которые
помог ют этому языку идти в ногу
со временем.

И всем тем, кто сдел л изучение Python
и связ нных с ним технологий н столько
сложным, что нужн *подобная* книг ,
чтобы спр виться с ними.

Пол Бэрри: «Изучаем программирование на Python», 2-е издание

На прогулке Пол остановился, чтобы обсудить правильное произношение слова «tuple» со своей терпеливой женой.



Обычная реакция Дейдры ☺

Пол Бэрри живет и р бот ет в *Карлоу* (*Ирландия*), м леньком городке с н селением около 35 тысяч человек в 80 км н юго-з п д от Дублин .

Пол имеет степень *бакалавра наук в области информационных систем* и степень *магистра в области вычислений*. Он т кже з кончил спир нтуру и получил свидетельство н пр во *преподавания и обучения*.

Пол р бот ет в Технологическом институте К рлоу с 1995 и чит ет лекции с 1997 год . Прежде чем н ч ть препод в тельскую деятельность, Пол десять лет посвятил ИТ-индустрии, р бот л в Ирл ндии и К н де, больш я ч сть его р боты был связ н с медицинскими учреждениями. Пол жен т н Дейдре, у них трое детей (двое сейч с уч тся в колледже).

Язык прогр ммиров ния Python (и связ нные с ним технологии) сост вляют основу послевузовских курсов Пол с 2007 учебного год .

Пол является втором (или со втором) еще четырех книг: двух о Python и двух о Perl. В прошлом он подготовил довольно много ст тей для *Linux Journal Magazine*, в котором является пишущим ред ктором.

Пол вырос в Белф сте, Северн я Ирл ндия, и это во многом объясняет некоторые его взгляды и з б вный кцент (впрочем, если вы тоже «с север », тогд взгляды Пол и его кцент пок жутся в м вполне норм льными).

Вы можете н йти Пол в *Твиттере (@barrypj)*. У него есть т кже своя дом шняя стр ничк *http://paulbarry.itcarlow.ie*.

Оглавление (Краткое)

Содержание (Конкретное)

Введение

Ваш мозг и Python. Вы пытаетесь чему-то *научиться*, а мозг делает вам одолжение и *сопротивляется* изо всех сил. Он думает: «Лучше оставить место для запоминания действительно важных вещей. Вдруг нам встретится голодный тигр или захочется покататься голышом на сноуборде. Я должен помнить об опасности». Как же нам *обмануть* ваш мозг, чтобы он считал программирование на Python важным для выживания?

Основы

1 Начнем поскорее

Начнем программировать на Python как можно скорее.

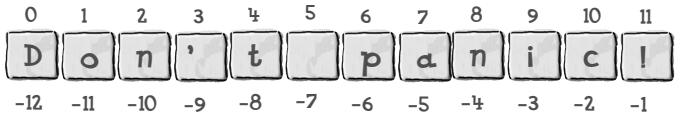
В этой главе мы ознакомимся с основами программирования на Python и сделаем это в характерном для нас стиле: с места в карьер. Через несколько страниц вы запустите свою первую программу. К концу главы вы сможете не только запускать типичные программы, но также понимать их код (и это еще не все!). Попутно вы познакомитесь с некоторыми особенностями языка **Python**. Итак, не будем больше тратить время. Переверните страницу — и вперед!

2 Списки

Работа с упорядоченными данными

Все программы обрабатывают данные, и программы на Python — не исключение.

На самом деле *данные повсюду*. Ведь в основном программирование — это работа с данными: *получение* данных, *обработка* данных, *интерпретация* данных. Чтобы работать с данными более эффективно, нужен какой-то контейнер, куда их можно *сложить*. Python предоставляет удобные структуры данных *широкого применения*: **списки**, **словари**, **кортежи** и **множества**. В этой главе мы бегло рассмотрим все четыре, а затем углубимся в изучение **списков** (остальные три структуры подробнее рассмотрены в следующей главе). Мы уже затрагивали эту тему ранее, поскольку все, с чем нам приходится сталкиваться при программировании на Python, так или иначе относится к работе с данными.

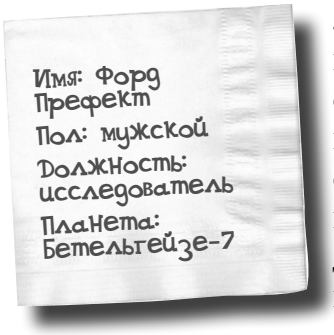


3 Структурированные данные

Работа со структурированными данными

Списки в Python очень удобны, но они не панацея.

Когда имеются *действительно* структурированные данные (для хранения которых список оказывается не лучшим выбором), спасение приходит от встроенных **словарей** Python. Словари «из коробки» позволяют хранить и обрабатывать данные, которые можно представить в виде *пар ключ/значение*. В этой главе мы изучим словари, а также **множества** и **кортежи**. Как и **списки** (которые мы изучили в главе 2), словари, множества и кортежи предоставляют встроенные инструменты для работы с данными, что делает Python еще более удобным языком.

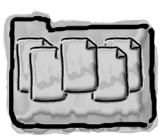


Повторное использование

4 Функции и модули

Повторное использование кода — ключ к построению стабильных систем.

В случае Python все повторное использование начинается и заканчивается **функциями**. Возьмите несколько строк кода, дайте им имя — и у вас готовая функция (которую можно использовать повторно). Возьмите коллекцию функций и сохраните их в файле — у вас готовый **модуль** (который можно использовать повторно). Это правда, когда говорят, что *делиться приятно*, и в конце главы вы уже сможете создавать код для **многократного** и **совместного** использования, благодаря пониманию того, как работают функции и модули Python.



модуль

5 Построение веб-приложения

Возвращение в реальный мир

Вы уже знаете Python достаточно, чтобы быть опасными.

Четыре главы книги освоены, и сейчас вы в состоянии продуктивно использовать Python во многих областях применения (хотя многое еще предстоит узнать). Вместо того чтобы исследовать эти области, в этой и последующих главах мы изучим разработку веб-приложений — в этом Python особенно силен. Попутно вы еще больше узнаете о Python. Однако вначале позвольте кратко подытожить то, что вы уже знаете.

6 Хранение и обработка данных

Где хранить данные

Рано или поздно появляется необходимость обеспечить надежное хранение данных.

И когда придет время **сохранить данные**, Python вам поможет. В этой главе вы узнаете о хранении и извлечении данных из *текстовых файлов*, которые — как механизм хранения — могут показаться слишком простыми, но тем не менее используются во многих проблемных областях. Кроме сохранения и извлечения данных из файлов, вы научитесь некоторым премудростям при работе с данными. «Серьезный материал» (хранение информации в базе данных) мы припасли для следующей главы, однако с файлами тоже придется потрудиться.

Form Data	Remote_addr	User_agent	Results
ImmutableMultiDict([('phrase', 'hitch-hiker'), ('letters', 'aeiou')])	127.0.0.1	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.36	{'e', 'i'}

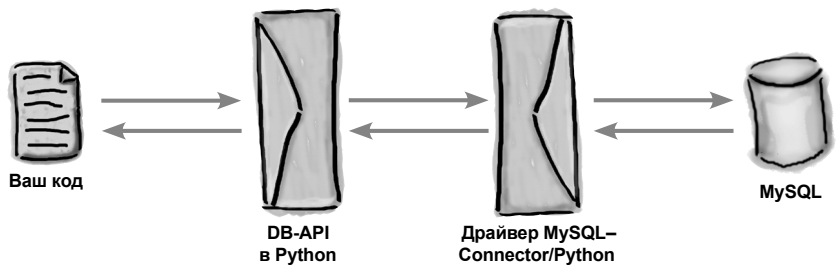
Использование базы данных

7

Используем DB-API в Python

Хранить информацию в реляционной базе данных очень удобно.

В этой главе вы узнаете, как организовать взаимодействие с популярной базой данных (БД) **MySQL**, используя универсальный прикладной программный интерфейс, который называется **DB-API**. Интерфейс DB-API (входящий в состав стандартной библиотеки Python) позволяет писать код, не зависящий от конкретной базы данных… если база данных понимает SQL. Хотя мы будем использовать MySQL, ничто не помешает вам использовать код DB-API с вашей любимой реляционной базой данных, какой бы она ни была. Давайте посмотрим, как пользоваться реляционной базой данных в Python. В этой главе не так много нового в плане изучения Python, но использование Python для общения с БД — **это важно**, поэтому стоит поучиться.



8 Немного о Классах

Абстракция поведения и состояния

Классы позволяют связать поведение кода и состояние вместе.

В этой главе мы отложим веб-приложение в сторону и будем учиться создавать **классы**. Это умение понадобится вам при создании диспетчера контекста. Классы — настолько полезная штука, что вам в любом случае стоит ознакомиться с ними поближе, поэтому мы посвятили им отдельную главу. Мы не будем рассматривать классы во всех подробностях, а коснемся лишь тех аспектов, которые пригодятся для создания диспетчера контекста, которого ожидает наше веб-приложение. А теперь вперед, посмотрим, что к чему.

countfromby.py - /Users/paul/Desktop/_NewBook/ch07/countfromby.py (3.5.1)

```
class CountFromBy:

    def __init__(self, v: int, i: int) -> None:
        self.val = v
        self.incr = i

    def increase(self) -> None:
        self.val += self.incr
```

Ln: 2 Col: 0

Протокол управления контекстом

9 Подключение к инструкции with

Пора применить все изученное на практике.

В главе 7 мы обсудили использование **реляционных баз данных** в Python, а в главе 8 ознакомились с использованием **классов**. В главе 9 мы объединим обе методики и создадим **диспетчер контекста**, который позволит расширить инструкцию `with` для работы с системами реляционных баз данных. В этой главе вы подключитесь к инструкции `with`, создав новый класс, соответствующий требованиям **протокола управления контекстом** в языке Python.


```
File  Edit  Window  Help  Checking our log DB
$ mysql -u vsearch -p vsearchlogDB
Enter password:

Welcome to MySQL monitor...


mysql> select * from log;
+----+---------------------+-------------------------+---------+-----------+----------------+----------------------+
| id | ts                  | phrase                  | letters | ip        | browser_string | results              |
+----+---------------------+-------------------------+---------+-----------+----------------+----------------------+
|  1 | 2016-03-09 13:40:46 | life, the uni ... ything | aeiou   | 127.0.0.1 | firefox        | {'u', 'e', 'i', 'a'} |
|  2 | 2016-03-09 13:42:07 | hitch-hiker             | aeiou   | 127.0.0.1 | safari         | {'i', 'e'}           |
|  3 | 2016-03-09 13:42:15 | galaxy                  | xyz     | 127.0.0.1 | chrome         | {'y', 'x'}           |
|  4 | 2016-03-09 13:43:07 | hitch-hiker             | xyz     | 127.0.0.1 | firefox        | set()                |
+----+---------------------+-------------------------+---------+-----------+----------------+----------------------+
4 rows in set (0.0 sec)


mysql> quit
Bye
```

10 Декораторы функций

Обертывание функций

Протокол управления контекстом из главы 9 — не единственная возможность улучшить код.

Python также позволяет использовать **декораторы** функций — технологию, с помощью которой можно добавлять код к существующим функциям, *не* изменяя имеющийся код. Если вы увидите в этом сходство с черной магией, не переживайте: ничего подобного. Однако многие программисты на Python считают создание декораторов функций одной из наиболее сложных сторон языка, поэтому пользуются этой возможностью реже, чем должны бы. В этой главе мы покажем, что создавать и использовать декораторы совсем не сложно, несмотря на всю их «продвинутость»

11 Обработка исключений

Что делать, когда что-то идет не так

Каким бы хорошим ни был ваш код, иногда все равно что-то идет не так.

Вы успешно выполнили все примеры в книге и, скорее всего, убедились, что имеющийся код работает. Значит ли это, что его можно считать надежным? По всей видимости, нет. Писать код и надеяться, что ничего плохого не случится, в лучшем случае наивно. В худшем случае — опасно, поскольку что-то непредвиденное порой все же происходит (и будет происходить). При написании кода лучше быть осторожным, а не доверчивыми. Осторожным, чтобы ваш код делал именно то, что вам нужно, и правильно реагировал, если что-то пойдет не так. В этой главе вы увидите, что может пойти не так, и узнаете, что делать, когда (и, чаще всего, прежде чем) такое произойдет.


```
  ...
Exception
      +-- StopIteration
      +-- StopAsyncIteration
      +-- ArithmeticError
      |     +-- FloatingPointError
      |     +-- OverflowError
      |     +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EOFError
            ...
```

Немного о многопоточности

11 3/4 Обработка ожидания

Иногда для выполнения кода может потребоваться много времени.

В одном случае это становится проблемой, в другом — нет. Если код работает «за кулисами» и ему нужно 30 секунд, чтобы сделать какие-то свои дела, ожидание не превращается в проблему. Однако если приложению требуется 30 секунд, чтобы ответить пользователю, это заметят все. Решение проблемы зависит от того, какие действия пытаетесь выполнить вы (и кто именно ожидает). В этой короткой главе мы обсудим некоторые варианты, а затем рассмотрим одно из решений данной проблемы: *как быть, если что-то требует слишком много времени?*