УДК 004.438 Python
ББК 32.973.26-018.1
П84

Прохоренок, Н. А.

П84 Python 3. Самое необходимое / Н. А. Прохоренок, В. А. Дронов. — 2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2018. — 608 с.: ил. — (Самое необходимое)

ISBN 978-5-9775-3994-4

Описан базовый синтаксис языка Python 3: типы данных, операторы, условия, циклы, регулярные выражения, встроенные функции, объектно-ориентированное программирование, обработка исключений, часто используемые модули стандартной библиотеки и установка дополнительных модулей. Даны основы SQLite, описан интерфейс доступа к базам данных SQLite и MySQL, в том числе посредством ODBC. Рассмотрена работа с изображениями с помощью библиотек Pillow и Wand, получение данных из Интернета и использование архивов различных форматов. Книга содержит более двухсот практических примеров, помогающих начать программировать на языке Python самостоятельно. Весь материал тщательно подобран, хорошо структурирован и компактно изложен, что позволяет использовать книгу как удобный справочник.

Во втором издании описана актуальная версия Python — 3.6.4, добавлены описания утилиты pip, работы с данными в формате JSON, библиотеки Tkinter и разработки оконных приложений с ее помощью, реализации параллельного программирования и использования потоков для выполнения программного кода.

Электронное приложение-архив, доступное на сайте издательства, содержит листинги описанных в книге примеров.

Для программистов

УДК 004.438 Python
ББК 32.973.26-018.1

Группа подготовки издания:

Руководитель проекта	*Евгений Рыбаков*
Зав. редакцией	*Екатерина Капалыгина*
Компьютерная верстка	*Ольги Сергиенко*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Марины Дамбиевой*

"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

ISBN 978-5-9775-3994-4

Оглавление

Введение

Добро пожаловать в мир Python!

Python — это интерпретируемый, объектно-ориентированный, тьюринг-полный язык программирования высокого уровня, предназначенный для решения самого широкого круга задач. С его помощью можно обрабатывать числовую и текстовую информацию, создавать изображения, работать с базами данных, разрабатывать веб-сайты и приложения с графическим интерфейсом. Python — язык *кроссплатформенный*, он позволяет создавать программы, которые будут работать во всех операционных системах. В этой книге мы рассмотрим базовые возможности Python версии 3.6.4 применительно к операционной системе Windows.

Согласно официальной версии, название языка произошло вовсе не от змеи. Создатель языка Гвидо ван Россум (Guido van Rossum) назвал свое творение в честь британского комедийного телешоу BBC «Летающий цирк Монти Пайтона» (Monty Python's Flying Circus). Поэтому правильное произношение названия этого замечательного языка — Пайтон.

Программа на языке Python представляет собой обычный текстовый файл с расширением py (консольная программа) или pyw (программа с графическим интерфейсом). Все инструкции из этого файла выполняются интерпретатором построчно. Для ускорения работы при первом импорте модуля создается промежуточный байт-код, который сохраняется в одноименном файле с расширением pyc. При последующих запусках, если модуль не был изменен, исполняется именно байт-код. Для выполнения низкоуровневых операций и задач, требующих высокой скорости работы, можно написать модуль на языке C или C++, скомпилировать его, а затем подключить к основной программе.

Как уже отмечено, Python относится к категории языков *объектно-ориентированных*. Это означает, что практически все данные в нем являются объектами, даже значения, относящиеся к элементарным типам — наподобие чисел и строк, а также сами типы данных. В переменной всегда сохраняется только ссылка на объект, а не сам объект. Например, можно создать функцию, сохранить ссылку на нее в переменной, а затем вызвать функцию через эту переменную. Такое обстоятельство делает язык Python идеальным инструментом для создания программ, использующих функции обратного вызова, — например, при разработке графического интерфейса. Тот факт, что язык является объектно-ориентированным, отнюдь не означает, что и объектно-ориентированный стиль программирования (ООП) является при его использовании обязательным. На языке Python можно писать программы как в стиле ООП, так и в процедурном стиле, — как того требует конкретная ситуация или как предпочитает программист.

Python — самый стильный язык программирования в мире, он не допускает двоякого написания кода. Так, языку Perl присущи зависимость от контекста и множественность синтаксиса, и часто два программиста, пишущих на Perl, просто не понимают код друг друга. В Python же код можно написать только одним способом. В нем отсутствуют лишние конструкции. Все программисты должны придерживаться стандарта PEP-8, описанного в документе **https://www.python.org/dev/peps/pep-0008/**. Более читаемого кода нет ни в одном другом языке программирования.

Синтаксис языка Python вызывает много нареканий у программистов, знакомых с другими языками программирования. На первый взгляд может показаться, что отсутствие ограничительных символов (фигурных скобок или конструкции `begin...end`) для выделения блоков и обязательная вставка пробелов впереди инструкций могут приводить к ошибкам. Однако это только первое и неправильное впечатление. Хороший стиль программирования в любом языке обязывает выделять инструкции внутри блока одинаковым количеством пробелов. В этой ситуации ограничительные символы просто ни к чему. Бытует мнение, что программа будет по-разному смотреться в разных редакторах. Это неверно. Согласно стандарту, для выделения блоков необходимо использовать *четыре пробела*. А четыре пробела в любом редакторе будут смотреться одинаково. Если в другом языке вас не приучили к хорошему стилю программирования, то язык Python быстро это исправит. Если количество пробелов внутри блока окажется разным, то интерпретатор выведет сообщение о фатальной ошибке, и программа будет остановлена. Таким образом, язык Python приучает программистов писать красивый и понятный код.

Поскольку программа на языке Python представляет собой обычный текстовый файл, его можно редактировать с помощью любого текстового редактора, например с помощью Notepad++. Можно использовать и другие, более специализированные программы такого рода: PyScripter, PythonWin, UliPad, Eclipse с установленным модулем PyDev, Netbeans и др. (полный список приемлемых редакторов можно найти на странице **https://wiki.python.org/moin/PythonEditors**). Мы же в процессе изложения материала этой книги будем пользоваться интерактивным интерпретатором IDLE, который входит в состав стандартной поставки Python в Windows, — он идеально подходит для изучения языка Python.

Любопытно, что в состав Python входит библиотека Tkinter, предназначенная для разработки приложений с графическим интерфейсом, — ее возможностей вполне хватит для написания небольших программ и утилит.

Каталоги с примерами Tkinter-приложений, иллюстрирующими возможности этой библиотеки, вы найдете в сопровождающем книгу электронном архиве, который можно загрузить с FTP-сервера издательства «БХВ-Петербург» по ссылке: **ftp://ftp.bhv.ru/9785977539944.zip** или со страницы книги на сайте **www.bhv.ru** (см. *приложение*). В этом же архиве находится и файл Listings.doc, содержащий все приведенные в книге листинги.

Сообщения обо всех замеченных ошибках и опечатках, равно как и возникающие в процессе чтения книги вопросы, авторы просят присылать на адрес издательства «БХВ-Петербург»: **mail@bhv.ru**.

Желаем приятного чтения и надеемся, что эта книга выведет вас на верный путь в мире профессионального программирования. И не забывайте, что книги по программированию нужно не только читать, — весьма желательно выполнять все имеющиеся в них примеры, а также экспериментировать, что-нибудь в этих примерах изменяя.

ГЛАВА 1

Первые шаги

Прежде чем мы начнем рассматривать синтаксис языка, необходимо сделать два замечания. Во-первых, как уже было отмечено во *введении*, не забывайте, что книги по программированию нужно не только читать, — весьма желательно выполнять все имеющиеся в них примеры, а также экспериментировать, что-нибудь в этих примерах изменяя. Поэтому, если вы удобно устроились на диване и настроились просто читать, у вас практически нет шансов изучить язык. Чем больше вы будете делать самостоятельно, тем большему научитесь.

Ну что, приступим к изучению языка? Python достоин того, чтобы его знал каждый программист!

> ***Внимание!***
>
> Начиная с версии 3.5, Python более не поддерживает Windows XP. В связи с этим в книге не будут описываться моменты, касающиеся его применения под этой версией операционной системы.

1.1. Установка Python

Вначале необходимо установить на компьютер *интерпретатор* Python (его также называют *исполняющей средой*).

1. Для загрузки дистрибутива заходим на страницу **https://www.python.org/downloads/** и в списке доступных версий щелкаем на гиперссылке **Python 3.6.4** (эта версия является самой актуальной из стабильных версий на момент подготовки книги). На открывшейся странице находим раздел **Files** и щелкаем на гиперссылке **Windows x86 executable installer** (32-разрядная редакция интерпретатора) или **Windows x86-64 executable installer** (его 64-разрядная редакция) — в зависимости от версии вашей операционной системы. В результате на наш компьютер будет загружен файл python-3.6.4.exe или python-3.6.4-amd64.exe соответственно. Затем запускаем загруженный файл двойным щелчком на нем.
2. В открывшемся окне (рис. 1.1) проверяем, установлен ли флажок **Install launcher for all users (recommended)** (Установить исполняющую среду для всех пользователей), устанавливаем флажок **Add Python 3.6 to PATH** (Добавить Python 3.6 в список путей переменной PATH) и нажимаем кнопку **Customize installation** (Настроить установку).
3. В следующем диалоговом окне (рис. 1.2) нам предлагается выбрать устанавливаемые компоненты. Оставляем установленными все флажки, представляющие эти компоненты, и нажимаем кнопку **Next**.

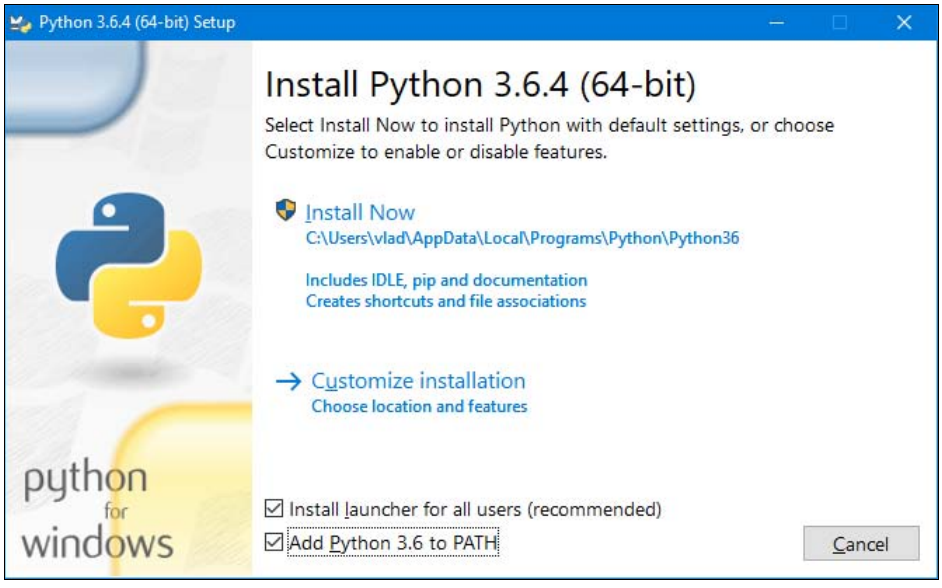

Рис. 1.1. Установка Python 3.6: шаг 1

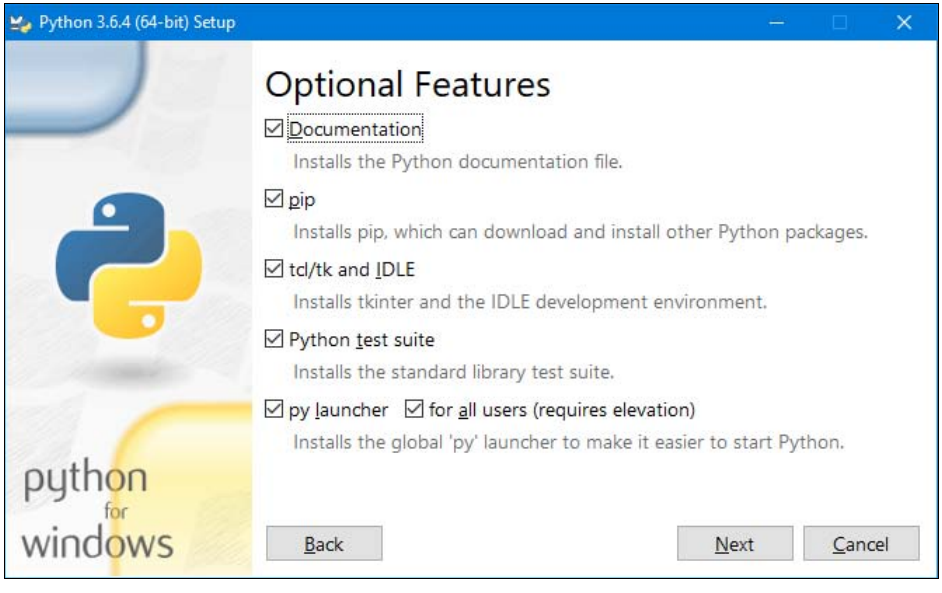

Рис. 1.2. Установка Python 3.6: шаг 2

4. На следующем шаге (рис. 1.3) мы зададим некоторые дополнительные настройки и выберем путь установки. Проверим, установлены ли флажки **Associate files with Python (requires the py launcher)** (Ассоциировать файлы с Python), **Create shortcuts for installed applications** (Создать ярлыки для установленных приложений) и **Add Python to environment variables** (Добавить Python в переменные окружения), и установим флажки **Install for all users** (Установить для всех пользователей) и **Precompile standard library** (Предварительно откомпилировать стандартную библиотеку).

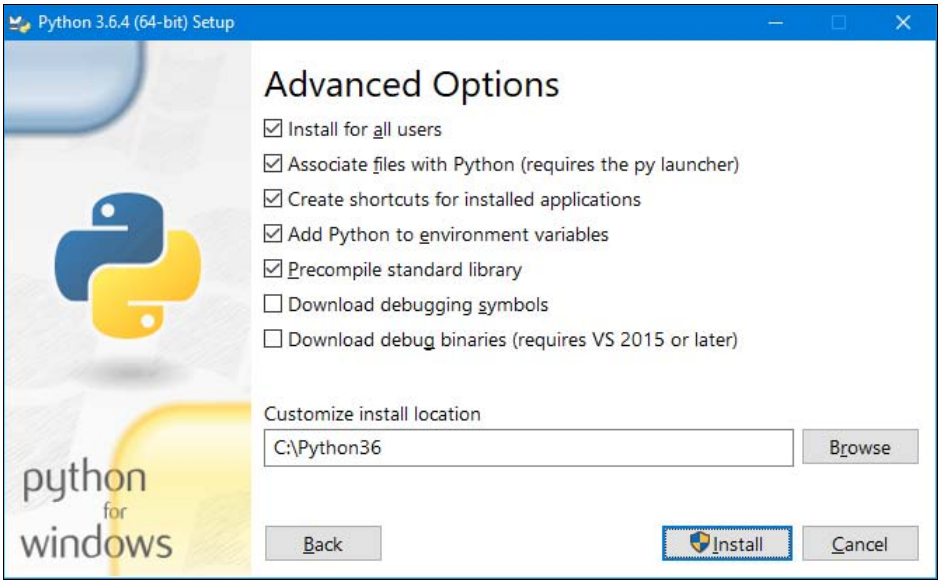

Рис. 1.3. Установка Python 3.6: шаг 3

Внимание!

Некоторые параметры при установке Python приходится задавать по нескольку раз на разных шагах. Вероятно, это недоработка разработчиков инсталлятора.

Теперь уточним путь, по которому будет установлен Python. Изначально нам предлагается установить интерпретатор по пути C:\Program Files\Python36. Можно сделать и так, но тогда при установке любой дополнительной библиотеки понадобится запускать командную строку с повышенными правами, иначе библиотека не установится.

Авторы книги рекомендуют установить Python по пути C:\Python36, то есть непосредственно в корень диска (см. рис. 1.3). В этом случае мы избежим проблем при установке дополнительных библиотек.

Задав все необходимые параметры, нажимаем кнопку **Install** и положительно отвечаем на появившееся на экране предупреждение UAC.

5. После завершения установки откроется окно, изображенное на рис. 1.4. Нажимаем в нем кнопку **Close** для выхода из программы установки.

В результате установки исходные файлы интерпретатора будут скопированы в каталог C:\Python36. В этом каталоге вы найдете два исполняемых файла: python.exe и pythonw.exe. Файл python.exe предназначен для выполнения консольных приложений. Именно эта программа запускается при двойном щелчке на файле с расширением py. Файл pythonw.exe служит для запуска оконных приложений (при двойном щелчке на файле с расширением pyw) — в этом случае окно консоли выводиться не будет.

Итак, если выполнить двойной щелчок на файле python.exe, то интерактивная оболочка запустится в окне консоли (рис. 1.5). Символы `>>>` в этом окне означают приглашение для ввода инструкций языка Python. Если после этих символов ввести, например, `2 + 2` и нажать клавишу <Enter>, то на следующей строке сразу будет выведен результат выполнения, а затем опять приглашение для ввода новой инструкции. Таким образом, это окно можно использовать в качестве калькулятора, а также для изучения языка.

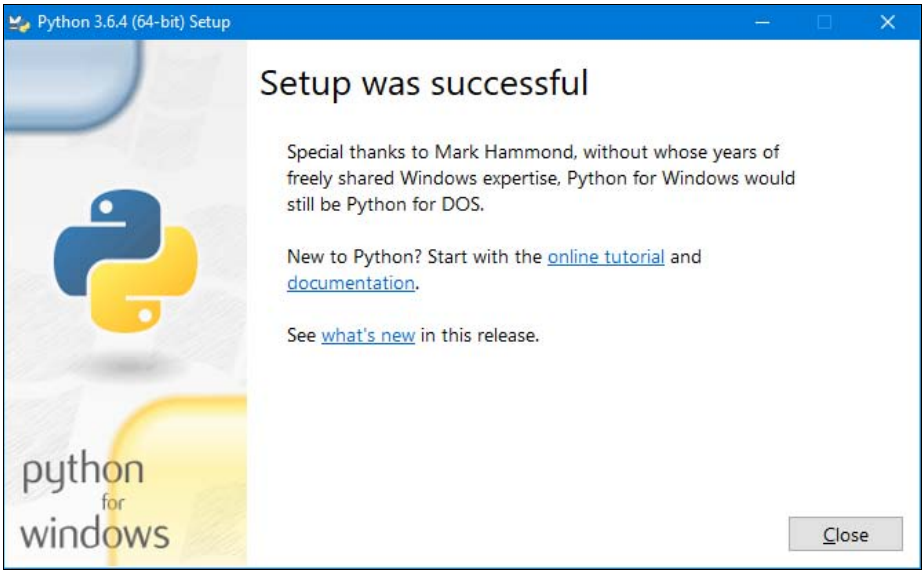

Рис. 1.4. Установка Python 3.6: шаг 4

Открыть это же окно можно, выбрав пункт **Python 3.6 (32-bit)** или **Python 3.6 (64-bit)** в меню **Пуск | Программы (Все программы) | Python 3.6**.

Однако для изучения языка, а также для создания и редактирования файлов с программами лучше пользоваться редактором IDLE, который входит в состав установленных компонентов. Для запуска этого редактора в меню **Пуск | Программы (Все программы) | Python 3.6**

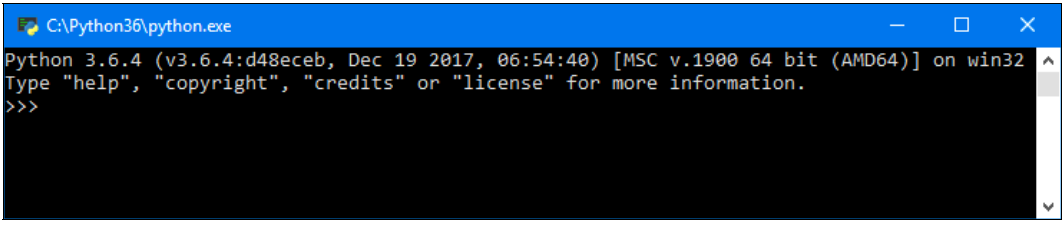

Рис. 1.5. Интерактивная оболочка Python 3.6

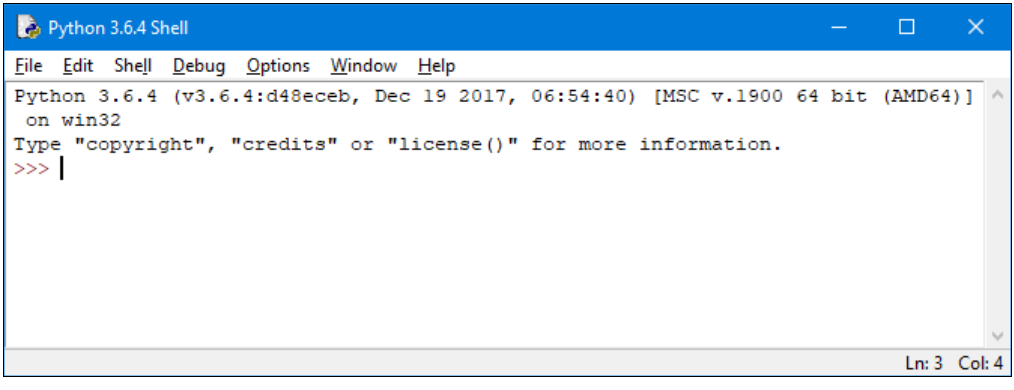

Рис. 1.6. Окно **Python 3.6.4 Shell** редактора IDLE

выбираем пункт **IDLE (Python 3.6 32-bit)** или **IDLE (Python 3.6 64-bit)**. В результате откроется окно **Python Shell** (рис. 1.6), которое выполняет все функции интерактивной оболочки, но дополнительно производит подсветку синтаксиса, выводит подсказки и др. Именно этим редактором мы будем пользоваться в процессе изучения материала книги. Более подробно редактор IDLE мы рассмотрим немного позже.

1.1.1. Установка нескольких интерпретаторов Python

Версии языка Python выпускаются с завидной регулярностью, но, к сожалению, сторонние разработчики не успевают за таким темпом и не столь часто обновляют свои модули. Поэтому иногда приходится при наличии версии Python 3 использовать на практике также и версию Python 2. Как же быть, если установлена версия 3.6, а необходимо запустить модуль для версии 2.7? В этом случае удалять версию 3.6 с компьютера не нужно. Все программы установки позволяют выбрать устанавливаемые компоненты. Существует также возможность задать ассоциацию запускаемой версии с файловым расширением — так вот эту возможность необходимо отключить при установке.

В качестве примера мы дополнительно установим на компьютер версию 2.7.14.2717, использовав альтернативный дистрибутив от компании ActiveState. Для этого переходим на страницу **https://www.activestate.com/activepython/downloads** и скачиваем дистрибутив. Установку программы производим в каталог по умолчанию (C:\Python27).

> ***Внимание!***
>
> При установке Python 2.7 в окне **Choose Setup Type** (рис. 1.7) необходимо нажать кнопку **Custom**, а в окне **Choose Setup Options** (рис. 1.8) — сбросить флажки **Add Python to the PATH environment variable** и **Create Python file extension associations**. Не забудьте это сделать, иначе Python 3.6.4 перестанет быть текущей версией.

После установки интерпретатора ActivePython в контекстное меню добавится пункт **Edit with Pythonwin**. С помощью этого пункта запускается редактор PythonWin, который можно использовать вместо IDLE.

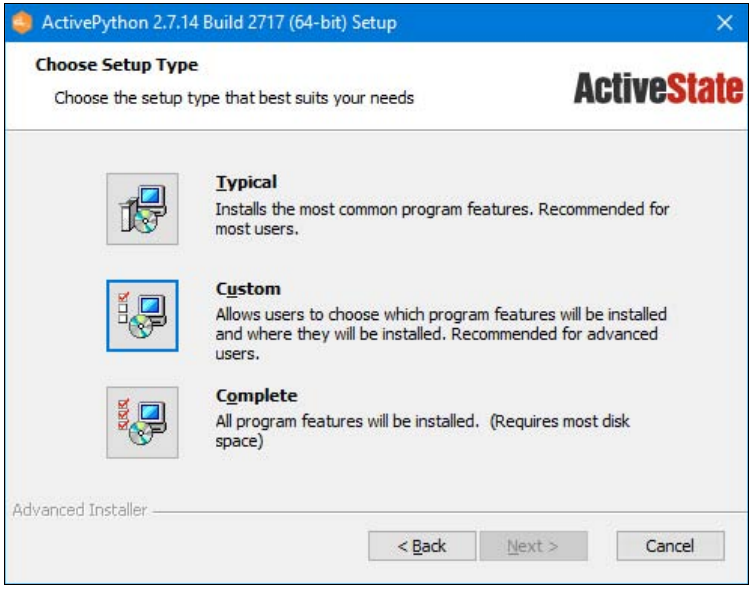

Рис. 1.7. Установка Python 2.7: окно **Choose Setup Type**

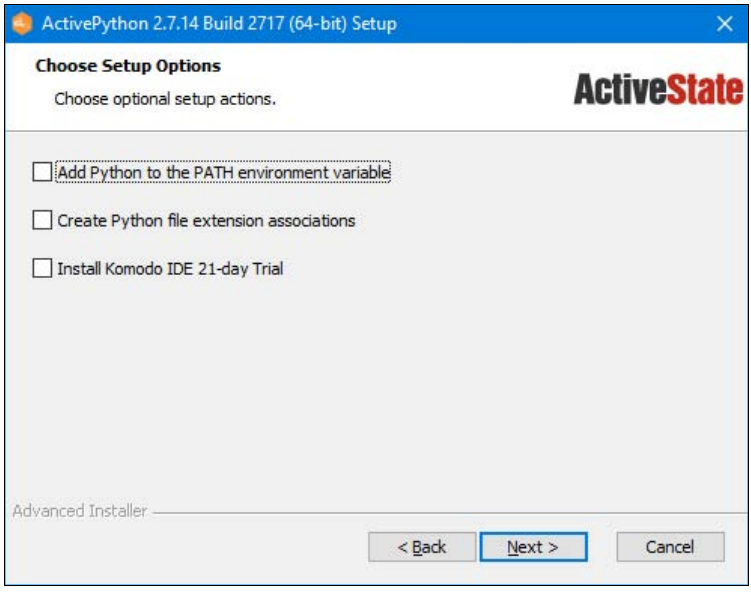

Рис. 1.8. Установка Python 2.7: окно **Choose Setup Options**

В состав ActivePython, кроме PythonWin, входит также редактор IDLE. Однако в меню **Пуск** нет пункта, с помощью которого можно его запустить. Чтобы это исправить, создадим файл IDLE27.cmd со следующим содержимым:

```
@echo off
start C:\Python27\pythonw.exe C:\Python27\Lib\idlelib\idle.pyw
```

С помощью двойного щелчка на этом файле можно будет запускать редактор IDLE для версии Python 2.7.

Ну, а запуск IDLE для версии Python 3.6 будет по-прежнему осуществляться так же, как и предлагалось ранее, — выбором в меню **Пуск | Программы (Все программы) | Python 3.6** пункта **IDLE (Python 3.6 32-bit)** или **IDLE (Python 3.6 64-bit)**.

1.1.2. Запуск программы с помощью разных версий Python

Теперь рассмотрим запуск программы с помощью разных версий Python. Как уже отмечалось, по умолчанию при двойном щелчке на значке файла запускается Python 3.6. Чтобы запустить Python-программу с помощью другой версии этого языка, щелкаем правой кнопкой мыши на значке файла с программой и в контекстном меню выбираем пункт **Открыть с помощью**. В результате на экране откроется небольшое окно выбора альтернативной программы для запуска файла. Сразу же сбросим флажок **Всегда использовать это приложение для открытия .py файлов** (подпись у этого флажка различна в разных версиях Windows) и щелкнем на гиперссылке **Еще приложения** — в окне появится список установленных на вашем компьютере программ, но нужного нам приложения Python 2.7 в нем не будет. Поэтому щелкнем на ссылке **Найти другое приложение на этом компьютере**, находящейся под списком. На экране откроется стандартное диалоговое окно открытия файла, в котором мы выберем программу python.exe, python2.exe или python2.7.exe из каталога C:\Python27.

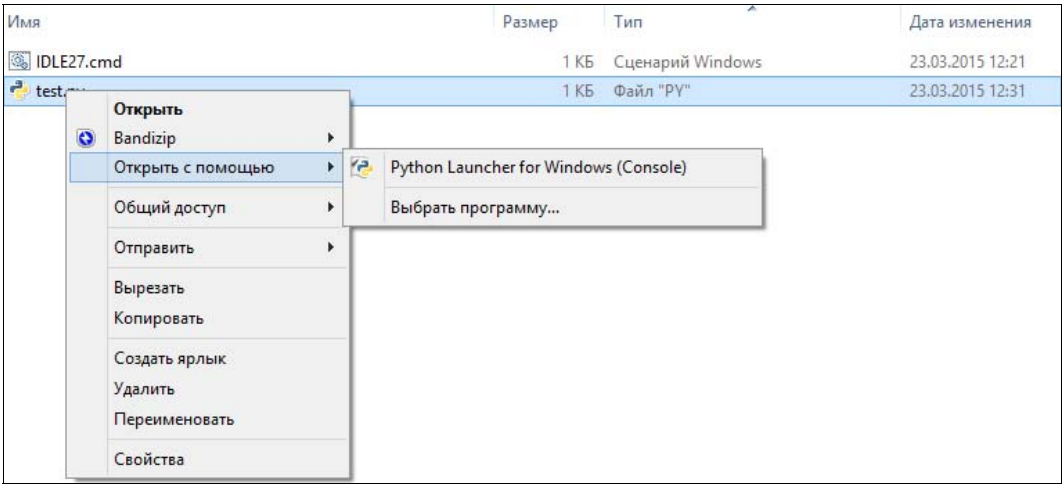

Рис. 1.9. Варианты запуска программы разными версиями Python

В Windows Vista, 7, 8 и 8.1 выбранная нами программа появится в подменю, открывающемся при выборе пункта **Открыть с помощью** (рис. 1.9), — здесь Python 2.7 представлен как **Python Launcher for Windows (Console)**. А в Windows 10 она будет присутствовать в списке, что выводится в диалоговом окне выбора альтернативной программы (рис. 1.10).

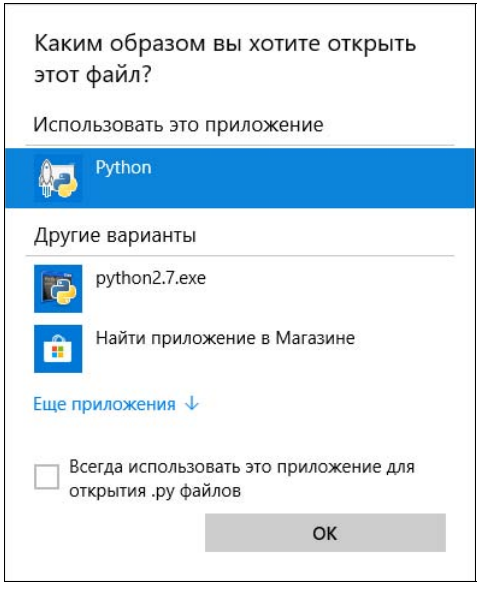

Рис. 1.10. Windows 10: диалоговое окно выбора альтернативной программы для запуска файла

Для проверки установки создайте файл test.py с помощью любого текстового редактора, например Блокнота. Содержимое файла приведено в листинге 1.1.

Листинг 1.1. Проверка установки

```
import sys
print (tuple(sys.version_info))
```

```
try:
    raw_input()         # Python 2
except NameError:
    input()             # Python 3
```

Затем запустите программу с помощью двойного щелчка на значке файла. Если результат выполнения: `(3, 6, 4, 'final', 0)`, то установка прошла нормально, а если `(2, 7, 14, 'final', 0)`, то вы не сбросили флажки **Add Python to the PATH environment variable** и **Create Python file extension associations** в окне **Choose Setup Options** (см. рис. 1.8).

Для изучения материала этой книги по умолчанию должна запускаться версия Python 3.6.

1.2. Первая программа на Python

Изучение языков программирования принято начинать с программы, выводящей надпись «Привет, мир!» Не будем нарушать традицию и продемонстрируем, как это будет выглядеть на Python (листинг 1.2).

Листинг 1.2. Первая программа на Python

```
# Выводим надпись с помощью функции print()
print("Привет, мир!")
```

Для запуска программы в меню **Пуск | Программы (Все программы) | Python 3.6** выбираем пункт **IDLE (Python 3.6 32-bit)** или **IDLE (Python 3.6 64-bit)**. В результате откроется окно **Python Shell**, в котором символы >>> означают приглашение ввести команду (см. рис. 1.6). Вводим сначала первую строку из листинга 1.2, а затем вторую. После ввода каждой строки нажимаем клавишу <Enter>. На следующей строке сразу отобразится результат, а далее — приглашение для ввода новой команды. Последовательность выполнения нашей программы такова:

```
>>> # Выводим надпись с помощью функции print()
>>> print("Привет, мир!")
Привет, мир!
>>>
```

> ***Примечание***
>
> Символы >>> вводить не нужно, они вставляются автоматически.

Для создания файла с программой в меню **File** выбираем пункт **New File** или нажимаем комбинацию клавиш <Ctrl>+<N>. В открывшемся окне набираем код из листинга 1.2, а затем сохраняем его под именем hello_world.py, выбрав пункт меню **File | Save** (комбинация клавиш <Ctrl>+<S>). При этом редактор сохранит файл в кодировке UTF-8 без BOM (Byte Order Mark, метка порядка байтов). Именно UTF-8 является кодировкой по умолчанию в Python 3. Если файл содержит инструкции в другой кодировке, то необходимо в первой или второй строке программы указать кодировку с помощью инструкции:

```
# -*- coding: <Кодировка> -*-
```

Например, для кодировки Windows-1251 инструкция будет выглядеть так:

```
# -*- coding: cp1251 -*-
```

Редактор IDLE учитывает указанную кодировку и автоматически производит перекодирование при сохранении файла. При использовании других редакторов следует проконтролировать соответствие указанной кодировки и реальной кодировки файла. Если кодировки не совпадают, то данные будут преобразованы некорректно, или во время преобразования произойдет ошибка.

Запустить программу на выполнение можно, выбрав пункт меню **Run | Run Module** или нажав клавишу <F5>. Результат выполнения программы будет отображен в окне **Python Shell**.

Запустить программу также можно с помощью двойного щелчка мыши на значке файла. В этом случае результат выполнения будет отображен в консоли Windows. Следует учитывать, что после вывода результата окно консоли сразу закрывается. Чтобы предотвратить закрытие окна, необходимо добавить в программу вызов функции `input()`, которая станет ожидать нажатия клавиши <Enter> и не позволит окну сразу закрыться. С учетом сказанного наша программа будет выглядеть так, как показано в листинге 1.3.

Листинг 1.3. Программа для запуска с помощью двойного щелчка мыши

```
# -*- coding: utf-8 -*-
print("Привет, мир!")          # Выводим строку
input()                        # Ожидаем нажатия клавиши <Enter>
```

> ***Примечание***
>
> Если до выполнения функции `input()` возникнет ошибка, то сообщение о ней будет выведено в консоль, но сама консоль после этого сразу закроется, и вы не сможете прочитать сообщение об ошибке. Попав в подобную ситуацию, запустите программу из командной строки или с помощью редактора IDLE, и вы сможете прочитать сообщение об ошибке.

В языке Python 3 строки по умолчанию хранятся в кодировке Unicode. При выводе кодировка Unicode автоматически преобразуется в кодировку терминала. Поэтому русские буквы отображаются корректно, хотя в окне консоли в Windows по умолчанию используется кодировка cp866, а файл с программой у нас в кодировке UTF-8.

Чтобы отредактировать уже созданный файл, запустим IDLE, выполним команду меню **File | Open** (комбинация клавиш <Ctrl>+<O>) и выберем нужный файл, который будет открыт в другом окне.

> ***Напоминание***
>
> Поскольку программа на языке Python представляет собой обычный текстовый файл, сохраненный с расширением *py* или *pyw*, его можно редактировать с помощью других программ — например, Notepad++. Можно также воспользоваться специализированными редакторами — скажем, PyScripter.

Когда интерпретатор Python начинает выполнение программы, хранящейся в файле, он сначала компилирует ее в особое внутреннее представление, — это делается с целью увеличить производительность кода. Файл с откомпилированным кодом хранится в каталоге __pycache__, вложенном в каталог, где хранится сам файл программы, а его имя имеет следующий вид:

<имя файла с исходным неоткомпилированным кодом>.cpython-<первые две цифры номера версии Python>.pyc

Так, при запуске на исполнение файла hello_world.py будет создан файл откомпилированного кода с именем hello_world.cpython-36.pyc.

При последующем запуске на выполнение того же файла будет исполняться именно откомпилированный код. Если же мы исправим исходный код, программа его автоматически перекомпилирует. При необходимости мы можем удалить файлы с откомпилированным кодом или даже сам каталог __pycache__ — впоследствии интерпретатор сформирует их заново.

1.3. Структура программы

Как вы уже знаете, программа на языке Python представляет собой обычный текстовый файл с инструкциями. Каждая инструкция располагается на отдельной строке. Если инструкция не является вложенной, она должна начинаться с начала строки, иначе будет выведено сообщение об ошибке:

```
>>>  import sys


SyntaxError: unexpected indent
>>>
```

В этом случае перед инструкцией `import` расположен один лишний пробел, который привел к выводу сообщения об ошибке.

Если программа предназначена для исполнения в операционной системе UNIX, то в первой строке необходимо указать путь к интерпретатору Python:

```
#!/usr/bin/python
```

В некоторых операционных системах путь к интерпретатору выглядит по-другому:

```
#!/usr/local/bin/python
```

Иногда можно не указывать точный путь к интерпретатору, а передать название языка программе `env`:

```
#!/usr/bin/env python
```

В этом случае программа `env` произведет поиск интерпретатора Python в соответствии с настройками путей поиска.

Помимо указания пути к интерпретатору Python, необходимо, чтобы в правах доступа к файлу был установлен бит на выполнение. Кроме того, следует помнить, что перевод строки в операционной системе Windows состоит из последовательности символов `\r` (перевод каретки) и `\n` (перевод строки). В операционной системе UNIX перевод строки осуществляется только одним символом `\n`. Если загрузить файл программы по протоколу FTP в бинарном режиме, то символ `\r` вызовет фатальную ошибку. По этой причине файлы по протоколу FTP следует загружать только в текстовом режиме (режим ASCII). В этом режиме символ `\r` будет удален автоматически.

После загрузки файла следует установить права на выполнение. Для исполнения скриптов на Python устанавливаем права в 755 (`-rwxr-xr-x`).

Во второй строке (для ОС Windows — в первой строке) следует указать кодировку. Если кодировка не указана, то предполагается, что файл сохранен в кодировке UTF-8. Для кодировки Windows-1251 строка будет выглядеть так:

```
# -*- coding: cp1251 -*-
```

Как уже отмечалось в предыдущем разделе, редактор IDLE учитывает указанную кодировку и автоматически производит перекодирование при сохранении файла. Получить полный список поддерживаемых кодировок и их псевдонимы позволяет код, приведенный в листинге 1.4.

Листинг 1.4. Вывод списка поддерживаемых кодировок

```
# -*- coding: utf-8 -*-
import encodings.aliases
arr = encodings.aliases.aliases
keys = list( arr.keys() )
keys.sort()
for key in keys:
    print("%s => %s" % (key, arr[key]))
```

Во многих языках программирования (например, в PHP, Perl и др.) каждая инструкция должна завершаться точкой с запятой. В языке Python в конце инструкции также можно поставить точку с запятой, но это не обязательно. Более того, в отличие от языка JavaScript, где рекомендуется завершать инструкции точкой с запятой, в языке Python точку с запятой ставить *не рекомендуется*. Концом инструкции является конец строки. Тем не менее, если необходимо разместить несколько инструкций на одной строке, точку с запятой *следует указать*:

```
>>> x = 5; y = 10; z = x + y # Три инструкции на одной строке
>>> print(z)
15
```

Еще одной отличительной особенностью языка Python является отсутствие ограничительных символов для выделения инструкций внутри блока. Например, в языке PHP инструкции внутри цикла `while` выделяются фигурными скобками:

```
$i = 1;
while ($i < 11) {
   echo $i . "\n";
   $i++;
}
echo "Конец программы";
```

В языке Python тот же код будет выглядеть по-иному (листинг 1.5).

Листинг 1.5. Выделение инструкций внутри блока

```
i = 1
while i < 11:
    print(i)
    i += 1
print("Конец программы")
```

Обратите внимание, что перед всеми инструкциями внутри блока расположено одинаковое количество пробелов. Именно так в языке Python выделяются *блоки*. Инструкции, перед которыми расположено одинаковое количество пробелов, являются *телом блока*. В нашем примере две инструкции выполняются десять раз. Концом блока является инструкция, перед которой расположено меньшее количество пробелов. В нашем случае это функция

`print()`, которая выводит строку `"Конец программы"`. Если количество пробелов внутри блока окажется разным, то интерпретатор выведет сообщение о фатальной ошибке, и программа будет остановлена. Так язык Python приучает программистов писать красивый и понятный код.

> ***Примечание***
>
> В языке Python принято использовать четыре пробела для выделения инструкций внутри блока.

Если блок состоит из одной инструкции, то допустимо разместить ее на одной строке с основной инструкцией. Например, код:

```
for i in range(1, 11):
    print(i)
print("Конец программы")
```

можно записать так:

```
for i in range(1, 11): print(i)
print("Конец программы")
```

Если инструкция является слишком длинной, то ее можно перенести на следующую строку, например, так:

- в конце строки поставить символ \, после которого должен следовать перевод строки. Другие символы (в том числе и комментарии) недопустимы. Пример:

```
x = 15 + 20 \
    + 30
print(x)
```

- поместить выражение внутри круглых скобок. Этот способ лучше, т. к. внутри круглых скобок можно разместить любое выражение. Пример:

```
x = (15 + 20                # Это комментарий
    + 30)
print(x)
```

- определение списка и словаря можно разместить на нескольких строках, т. к. при этом используются квадратные и фигурные скобки соответственно. Пример определения списка:

```
arr = [15, 20,              # Это комментарий
       30]
print(arr)
```

Пример определения словаря:

```
arr = {"x": 15, "y": 20,    # Это комментарий
       "z": 30}
print(arr)
```

1.4. Комментарии

Комментарии предназначены для вставки пояснений в текст программы — интерпретатор полностью их игнорирует. Внутри комментария может располагаться любой текст, включая инструкции, которые выполнять не следует.

Совет

Помните — комментарии нужны программисту, а не интерпретатору Python. Вставка комментариев в код позволит через некоторое время быстро вспомнить предназначение фрагмента кода.

В языке Python присутствует только *однострочный комментарий*. Он начинается с символа #:

```
# Это комментарий
```

Однострочный комментарий может начинаться не только с начала строки, но и располагаться после инструкции. Например, в следующем примере комментарий расположен после инструкции, предписывающей вывести надпись "Привет, мир!":

```
print("Привет, мир!") # Выводим надпись с помощью функции print()
```

Если же символ комментария разместить перед инструкцией, то она выполнена не будет:

```
# print("Привет, мир!") Эта инструкция выполнена не будет
```

Если символ # расположен внутри кавычек или апострофов, то он не является символом комментария:

```
print("# Это НЕ комментарий")
```

Так как в языке Python нет многострочного комментария, то комментируемый фрагмент часто размещают внутри утроенных кавычек (или утроенных апострофов):

```
"""
Эта инструкция выполнена не будет
print("Привет, мир!")
"""
```

Следует заметить, что этот фрагмент кода не игнорируется интерпретатором, поскольку он не является комментарием. В результате выполнения такого фрагмента будет создан объект строкового типа. Тем не менее, инструкции внутри утроенных кавычек выполняться не станут, поскольку интерпретатор сочтет их простым текстом. Такие строки являются *строками документирования*, а не комментариями.

1.5. Дополнительные возможности IDLE

Поскольку в процессе изучения материала этой книги в качестве редактора мы будем использовать IDLE, рассмотрим дополнительные возможности этой среды разработки.

Как вы уже знаете, в окне **Python Shell** символы >>> означают приглашение ввести команду. Введя команду, нажимаем клавишу <Enter> — на следующей строке сразу отобразится результат (при условии, что инструкция возвращает значение), а далее — приглашение для ввода новой команды. При вводе многострочной команды после нажатия клавиши <Enter> редактор автоматически вставит отступ и будет ожидать дальнейшего ввода. Чтобы сообщить редактору о конце ввода команды, необходимо дважды нажать клавишу <Enter>:

```
>>> for n in range(1, 3):
        print(n)

1
2
>>>
```

В предыдущем разделе мы выводили строку "Привет, мир!" с помощью функции print(). В окне **Python Shell** это делать не обязательно — мы можем просто ввести строку и нажать клавишу <Enter> для получения результата:

```
>>> "Привет, мир!"
'Привет, мир!'
>>>
```

Обратите внимание на то, что строки выводятся в апострофах. Этого не произойдет, если выводить строку с помощью функции print():

```
>>> print("Привет, мир!")
Привет, мир!
>>>
```

Учитывая возможность получить результат сразу после ввода команды, окно **Python Shell** можно использовать для изучения команд, а также в качестве многофункционального калькулятора:

```
>>> 12 * 32 + 54
438
>>>
```

Результат вычисления последней инструкции сохраняется в переменной _ (одно подчеркивание). Это позволяет производить дальнейшие расчеты без ввода предыдущего результата. Вместо него достаточно ввести символ подчеркивания:

```
>>> 125 * 3            # Умножение
375
>>> _ + 50             # Сложение. Эквивалентно 375 + 50
425
>>> _ / 5              # Деление. Эквивалентно 425 / 5
85.0
>>>
```

При вводе команды можно воспользоваться комбинацией клавиш <Ctrl>+<Пробел>. В результате будет отображен список, из которого можно выбрать нужный идентификатор. Если при открытом списке вводить буквы, то показываться будут идентификаторы, начинающиеся с этих букв. Выбирать идентификатор необходимо с помощью клавиш <↑> и <↓>. После выбора не следует нажимать клавишу <Enter>, иначе это приведет к выполнению инструкции, — просто вводите инструкцию дальше, а список закроется. Такой же список будет автоматически появляться (с некоторой задержкой) при обращении к атрибутам объекта или модуля после ввода точки. Для автоматического завершения идентификатора после ввода его первых букв можно воспользоваться комбинацией клавиш <Alt>+</>. При каждом последующем нажатии этой комбинации будет вставляться следующий идентификатор. Эти две комбинации клавиш очень удобны, если вы забыли, как пишется слово, или хотите, чтобы редактор закончил его за вас.

При необходимости повторно выполнить ранее введенную инструкцию, ее приходится набирать заново. Можно, конечно, скопировать инструкцию, а затем вставить, но, как вы можете сами убедиться, в контекстном меню нет пунктов **Copy** (Копировать) и **Paste** (Вставить), — они расположены в меню **Edit**, а постоянно выбирать пункты из этого меню очень неудобно. Одним из решений проблемы является использование комбинации «горячих» клавиш <Ctrl>+<C> (Копировать) и <Ctrl>+<V> (Вставить). Комбинации эти стандартны

для Windows, и вы наверняка их уже использовали ранее. Но, опять-таки, прежде чем скопировать инструкцию, ее предварительно необходимо выделить. Редактор IDLE избавляет нас от таких лишних действий и предоставляет комбинацию клавиш <Alt>+<N> — для вставки первой введенной инструкции, а также комбинацию <Alt>+<P> — для вставки последней инструкции. Каждое последующее нажатие этих комбинаций будет вставлять следующую (или предыдущую) инструкцию. Для еще более быстрого повторного ввода инструкции следует предварительно ввести ее первые буквы — в таком случае перебираться будут только инструкции, начинающиеся с этих букв.

1.6. Вывод результатов работы программы

Вывести результаты работы программы можно с помощью функции `print()`. Функция имеет следующий формат:

```
print([<Объекты>][, sep=' '][, end='\n'][, file=sys.stdout][, flush=False])
```

Функция `print()` преобразует объект в строку и посылает ее в стандартный вывод `stdout`. С помощью параметра `file` можно перенаправить вывод в другое место, например в файл. При этом, если параметр `flush` имеет значение `True`, выводимые значения будут принудительно записаны в файл. Перенаправление вывода мы подробно рассмотрим при изучении файлов.

После вывода строки автоматически добавляется символ перевода строки:

```
print("Строка 1")
print("Строка 2")
```

Результат:

```
Строка 1
Строка 2
```

Если необходимо вывести результат на той же строке, то в функции `print()` данные указываются через запятую в первом параметре:

```
print("Строка 1", "Строка 2")
```

Результат:

```
Строка 1 Строка 2
```

Как видно из примера, между выводимыми строками автоматически вставляется пробел. С помощью параметра `sep` можно указать другой символ. Например, выведем строки без пробела между ними:

```
print("Строка1", "Строка2", sep="")
```

Результат:

```
Строка 1Строка 2
```

После вывода объектов в конце добавляется символ перевода строки. Если необходимо произвести дальнейший вывод на той же строке, то в параметре `end` следует указать другой символ:

```
print("Строка 1", "Строка 2", end=" ")
print("Строка 3")
# Выведет: Строка 1 Строка 2 Строка 3
```

Если, наоборот, необходимо вставить символ перевода строки, то функция `print()` указывается без параметров:

```
for n in range(1, 5):
    print(n, end=" ")
print()
print("Это текст на новой строке")
```

Результат выполнения:

```
1 2 3 4
Это текст на новой строке
```

Здесь мы использовали цикл `for`, который позволяет последовательно перебирать элементы. На каждой итерации цикла переменной `n` присваивается новое число, которое мы выводим с помощью функции `print()`, расположенной на следующей строке.

Обратите внимание, что перед функцией мы добавили четыре пробела. Как уже отмечалось ранее, таким образом в языке Python выделяются *блоки*. При этом инструкции, перед которыми расположено одинаковое количество пробелов, представляют собой *тело цикла*. Все эти инструкции выполняются определенное количество раз. Концом блока является инструкция, перед которой расположено меньшее количество пробелов. В нашем случае это функция `print()` без параметров, которая вставляет символ перевода строки.

Если необходимо вывести большой блок текста, его следует разместить между утроенными кавычками или утроенными апострофами. В этом случае текст сохраняет свое форматирование:

```
print("""Строка 1
Строка 2
Строка 3""")
```

В результате выполнения этого примера мы получим три строки:

```
Строка 1
Строка 2
Строка 3
```

Для вывода результатов работы программы вместо функции `print()` можно использовать метод `write()` объекта `sys.stdout`:

```
import sys                      # Подключаем модуль sys
sys.stdout.write("Строка")      # Выводим строку
```

В первой строке с помощью оператора `import` мы подключаем модуль `sys`, в котором объявлен объект `stdout`. Далее с помощью метода `write()` этого объекта выводим строку. Следует заметить, что метод не вставляет символ перевода строки, поэтому при необходимости следует добавить его самим с помощью символа `\n`:

```
import sys
sys.stdout.write("Строка 1\n")
sys.stdout.write("Строка 2")
```

1.7. Ввод данных

Для ввода данных в Python 3 предназначена функция `input()`, которая получает данные со стандартного ввода `stdin`. Функция имеет следующий формат:

```
[<Значение> = ] input([<Сообщение>])
```

Для примера переделаем нашу первую программу так, чтобы она здоровалась не со всем миром, а только с нами (листинг 1.6).

Листинг 1.6. Пример использования функции `input()`

```
# -*- coding: utf-8 -*-
name = input("Введите ваше имя: ")
print("Привет,", name)
input("Нажмите <Enter> для закрытия окна")
```

Чтобы окно сразу не закрылось, в конце программы указан еще один вызов функции `input()`. В этом случае окно не закроется, пока не будет нажата клавиша <Enter>.

Вводим код и сохраняем файл, например, под именем test2.py, а затем запускаем программу на выполнение с помощью двойного щелчка на значке этого файла. Откроется черное окно, в котором мы увидим надпись: **Введите ваше имя:**. Вводим свое имя, например `Николай`, и нажимаем клавишу <Enter>. В результате будет выведено приветствие: **Привет, Николай**.

При использовании функции `input()` следует учитывать, что при достижении конца файла или при нажатии комбинации клавиш <Ctrl>+<Z>, а затем клавиши <Enter>, генерируется исключение `EOFError`. Если не предусмотреть обработку исключения, то программа аварийно завершится. Обработать исключение можно следующим образом:

```
try:
    s = input("Введите данные: ")
    print(s)
except EOFError:
    print("Обработали исключение EOFError")
```

Тогда, если внутри блока `try` возникнет исключение `EOFError`, управление будет передано в блок `except`. После исполнения инструкций в блоке `except` программа нормально продолжит работу.

Передать данные можно в командной строке, указав их после имени файла программы. Такие данные доступны через список `argv` модуля `sys`. Первый элемент списка `argv` будет содержать название файла запущенной программы, а последующие элементы — переданные данные. Для примера создадим файл test3.py в каталоге C:\book. Содержимое файла приведено в листинге 1.7.

Листинг 1.7. Получение данных из командной строки

```
# -*- coding: utf-8 -*-
import sys
arr = sys.argv[:]
for n in arr:
    print(n)
```

Теперь запустим программу на выполнение из командной строки и передадим ей данные. Для этого вызовем командную строку: выберем в меню **Пуск** пункт **Выполнить**, в открывшемся окне наберем команду `cmd` и нажмем кнопку **OK** — откроется черное окно командной строки с приглашением для ввода команд. Перейдем в каталог C:\book, набрав команду:

```
cd C:\book
```

В командной строке должно появиться приглашение:

```
C:\book>
```

Для запуска нашей программы вводим команду:

```
C:\Python36\python.exe test3.py -uNik -p123
```

В этой команде мы передаем имя файла (`test3.py`) и некоторые данные (`-uNik` и `-p123`). Результат выполнения программы будет выглядеть так:

```
test3.py
-uNik
-p123
```

1.8. Доступ к документации

При установке Python на компьютер помимо собственно интерпретатора копируется документация по этому языку в формате CHM. Чтобы открыть ее, в меню **Пуск | Программы (Все программы) | Python 3.6** нужно выбрать пункт **Python 3.6 Manuals (32-bit)** или **Python 3.6 Manuals (64-bit)**.

Если в меню **Пуск | Программы (Все программы) | Python 3.6** выбрать пункт **Python 3.6 Module Docs (32-bit)** или **Python 3.6 Module Docs (64-bit)**, запустится сервер документов pydoc (рис. 1.11). Он представляет собой написанную на самом Python программу веб-сервера, выводящую результаты своей работы в веб-браузере.

Сразу после запуска pydoc откроется веб-браузер, в котором будет выведен список всех стандартных модулей, поставляющихся в составе Python. Щелкнув на названии модуля,

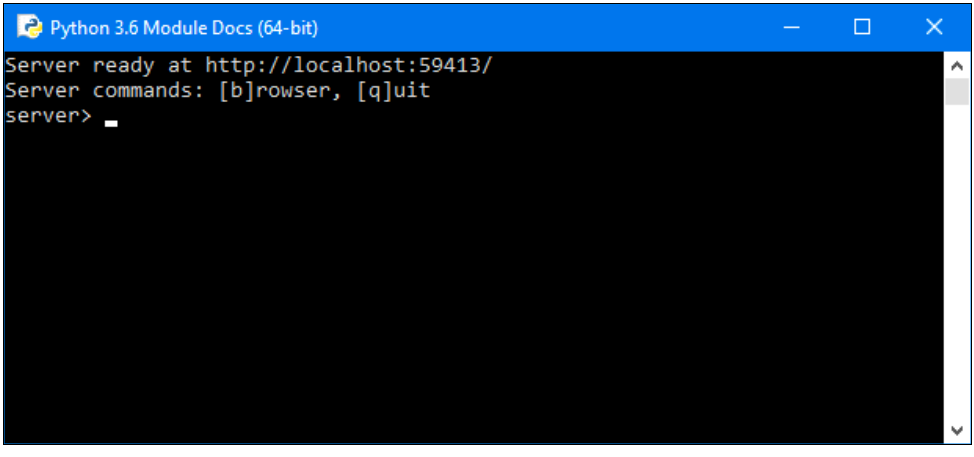

Рис. 1.11. Окно pydoc

представляющем собой гиперссылку, мы откроем страницу с описанием всех классов, функций и констант, объявленных в этом модуле.

Чтобы завершить работу pydoc, следует переключиться в его окно (см. рис. 1.11), ввести в нем команду `q` (от quit, выйти) и нажать клавишу <Enter> — окно при этом автоматически закроется. А введенная там команда `b` (от browser, браузер) повторно выведет в браузере страницу со списком модулей.

В окне **Python Shell** также можно отобразить документацию. Для этого предназначена функция `help()`. В качестве примера отобразим документацию по встроенной функции `input()`:

```
>>> help(input)
```

Результат выполнения:

```
Help on built-in function input in module builtins:

input(prompt=None, /)
    Read a string from standard input.  The trailing newline is stripped.

    The prompt string, if given, is printed to standard output without a
    trailing newline before reading input.

    If the user hits EOF (*nix: Ctrl-D, Windows: Ctrl-Z+Return), raise EOFError.
    On *nix systems, readline is used if available.
```

С помощью функции `help()` можно получить документацию не только по конкретной функции, но и по всему модулю сразу. Для этого предварительно необходимо подключить модуль. Например, подключим модуль `builtins`, содержащий определения всех встроенных функций и классов, а затем выведем документацию по этому модулю:

```
>>> import builtins
>>> help(builtins)
```

При рассмотрении комментариев мы говорили, что часто для комментирования большого фрагмента кода используются утроенные кавычки или утроенные апострофы. Такие строки не являются комментариями в полном смысле этого слова. Вместо комментирования фрагмента создается объект строкового типа, который сохраняется в атрибуте `__doc__`. Функция `help()` при составлении документации получает информацию из этого атрибута. Такие строки называются *строками документирования*.

В качестве примера создадим файл test4.py, содержимое которого показано в листинге 1.8.

Листинг 1.8. Тестовый модуль test4.py

```
# -*- coding: utf-8 -*-
""" Это описание нашего модуля """
def func():
    """ Это описание функции"""
    pass
```

Теперь подключим этот модуль и выведем содержимое строк документирования. Все эти действия выполняет код из листинга 1.9.

Листинг 1.9. Вывод строк документирования посредством функции `help()`

```
# -*- coding: utf-8 -*-
import test4                        # Подключаем файл test4.py
help(test4)
input()
```

Запустим эту программу из среды **Python Shell**. (Если запустить ее щелчком мыши, вывод будет выполнен в окне интерактивной оболочки, и результат окажется нечитаемым. Вероятно, это происходит вследствие ошибки в интерпретаторе.) Вот что мы увидим:

```
Help on module test4:

NAME
    test4 - Это описание нашего модуля

FUNCTIONS
    func()
        Это описание функции

FILE
    d:\data\документы\работа\книги\python самое необходимое ii\примеры\1\test4.py
```

Теперь получим содержимое строк документирования с помощью атрибута `__doc__`. Как это делается, показывает код из листинга 1.10.

Листинг 1.10. Вывод строк документирования посредством атрибута `__doc__`

```
# -*- coding: utf-8 -*-
import test4                        # Подключаем файл test4.py
print(test4.__doc__)
print(test4.func.__doc__)
input()
```

Результат выполнения:

```
Это описание нашего модуля
Это описание функции
```

Атрибут `__doc__` можно использовать вместо функции `help()`. В качестве примера получим документацию по функции `input()`:

```
>>> print(input.__doc__)
```

Результат выполнения:

```
Read a string from standard input.  The trailing newline is stripped.

The prompt string, if given, is printed to standard output without a
trailing newline before reading input.

If the user hits EOF (*nix: Ctrl-D, Windows: Ctrl-Z+Return), raise EOFError.
On *nix systems, readline is used if available.
```

Получить список всех идентификаторов внутри модуля позволяет функция `dir()`. Пример ее использования показывает код из листинга 1.11.

Листинг 1.11. Получение списка идентификаторов

```
# -*- coding: utf-8 -*-
import test4                        # Подключаем файл test4.py
print(dir(test4))
input()
```

Результат выполнения:

```
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__',
'__package__', '__spec__', 'func']
```

Теперь получим список всех встроенных идентификаторов:

```
>>> import builtins
>>> print(dir(builtins))
```

Функция `dir()` может не принимать параметров вообще. В этом случае возвращается список идентификаторов текущего модуля:

```
>>> print(dir())
```

Результат выполнения:

```
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__',
'__package__', '__spec__']
```

1.9. Утилита pip: установка дополнительных библиотек

Интерпретатор Python поставляется с богатой стандартной библиотекой, реализующей, в частности, работу с файлами, шифрование, архивирование, обмен данными по сети и пр. Однако такие операции, как обработка графики, взаимодействие с базами данных SQLite, MySQL и многое другое она не поддерживает, и для их выполнения нам придется устанавливать всевозможные дополнительные библиотеки.

В настоящее время процесс установки дополнительных библиотек в Python исключительно прост. Нам достаточно воспользоваться имеющейся в комплекте поставки Python утилитой pip, которая самостоятельно найдет запрошенную нами библиотеку в интернет-хранилище (репозитории) PyPI (Python Package Index, реестр пакетов Python), загрузит дистрибутивный пакет с этой библиотекой, совместимый с установленной версией Python, и установит ее. Если инсталлируемая библиотека требует для работы другие библиотеки, они также будут установлены.

> ***Примечание***
>
> Все устанавливаемые таким образом дополнительные библиотеки записываются по пути *<Путь, по которому установлен Python>\Lib\site-packages*.

Помимо этого, утилита pip позволит нам просмотреть список уже установленных дополнительных библиотек, получить сведения о выбранной библиотеке и удалить ненужную библиотеку.

Для использования утилиты pip в командной строке следует набрать команду следующего формата:

```
pip <Команда и ее опции> <Универсальные опции>
```

Параметр <Команда и ее опции> указывает, что должна сделать утилита: установить библиотеку, вывести список библиотек, предоставить сведения об указанной библиотеке или удалить ее. Параметр <Универсальные опции> задает дополнительные настройки для самой утилиты и действует на все поддерживаемые ей команды.

Далее приведен сокращенный список поддерживаемых утилитой pip команд вместе с их собственными опциями, а также и универсальных опций, включающий наиболее востребованные из таковых. Полный список всех команд утилиты pip можно получить, воспользовавшись командой `help` и опцией `-h`.

Итак, утилита pip поддерживает следующие наиболее полезные нам команды:

- `install` — установка указанной библиотеки. Формат этой команды таков:

  ```
  pip install [<Опции>] <Название библиотеки>
  ```

  Если в параметре <Опции> не указана ни одна из них (см. далее), утилита просто загрузит и установит библиотеку с названием, заданным в параметре <Название библиотеки>. Если такая библиотека уже установлена, ничего не произойдет. Вот пример установки библиотеки `Pillow`:

  ```
  pip install pillow
  ```

  Доступные опции:

  - `-U` (или `--upgrade`) — обновление библиотеки с заданным названием до актуальной версии, имеющейся в репозитории PyPI. Обновляем библиотеку `Pillow`:

    ```
    pip install -U pillow
    ```

  - `--force-reinstall` — выполняет полную переустановку заданной библиотеки. Обычно используется вместе с опцией `-U`.

  В качестве параметра <Название библиотеки> также можно использовать конструкцию такого формата (кавычки обязательны):

  ```
  "<Название библиотеки><Оператор сравнения><Номер версии>"
  ```

  В качестве параметра <Оператор сравнения> можно использовать следующие символы и их комбинации:

  - < — меньше;
  - > — больше;
  - <= — меньше или равно;
  - >= — больше или равно;
  - == — равно.

  Пример установки библиотеки `Pillow` версии 5.0.0 или более новой:

  ```
  pip install "pillow>=5.0.0"
  ```

- `list` — вывод списка установленных библиотек с указанием их версий в круглых скобках. Формат команды:

  ```
  pip list [<Опции>]
  ```

Пример:

```
pip list
```

У авторов было выведено:

```
Pillow (5.0.0)
pip (9.0.1)
setuptools (38.4.0)
```

Единственная доступная здесь опция: `--format=<Формат вывода>`, задающая формат вывода. В качестве параметра `<Формат вывода>` можно указать `legacy` (вывод обычным списком, как было показано в примере ранее, — это формат вывода по умолчанию) или `columns` (вывод в виде таблицы). Вот пример вывода списка установленных библиотек, оформленного в виде таблицы:

```
pip list --format=columns
```

У авторов было выведено:

```
Package    Version
---------- -------
Pillow     5.0.0
pip        9.0.1
setuptools 38.4.0
```

> ***Замечание***
>
> Как видим, изначально в комплекте поставки Python уже присутствуют две библиотеки такого рода: `pip`, реализующая функциональность одноименной утилиты, и `setuptools`, предоставляющая специфические инструменты, которые помогают устанавливать дополнительные библиотеки.

- `show` — вывод сведений об указанной библиотеке. Формат команды:

```
pip show [<Опции>] <Название библиотеки>
```

Выводятся название библиотеки, ее версия, краткое описание, интернет-адрес «домашнего» сайта, имя разработчика, его адрес электронной почты, название лицензии, по которой распространяется библиотека, путь, по которому она установлена, и список библиотек, требующихся ей для работы (если таковые есть). Для примера посмотрим сведения о `Pillow`:

```
pip show pillow
```

Вывод:

```
Name: Pillow
Version: 5.0.0
Summary: Python Imaging Library (Fork)
Home-page: https://python-pillow.org
Author: Alex Clark (Fork Author)
Author-email: aclark@aclark.net
License: Standard PIL License
Location: c:\python36\lib\site-packages
Requires:
```

Единственная доступная опция: `-f` (или `--files`), которая указывает утилите pip дополнительно вывести список всех файлов, составляющих библиотеку. Вот пример вывода сведений о библиотеке `Pillow`, включая перечень составляющих ее файлов:

```
pip show -f pillow
```

- `uninstall` — удаление указанной библиотеки. Формат команды:

  ```
  pip uninstall [<Опции>] <Название библиотеки>
  ```

  Сначала будет выведен список всех файлов, составляющих удаляемую библиотеку, и вопрос, действительно ли пользователь хочет удалить ее. Чтобы подтвердить удаление, нужно ввести букву `y`, чтобы отменить его — `n`, после чего в любом случае нажать клавишу <Enter>. Вот пример удаления библиотеки `Pillow`:

  ```
  pip uninstall pillow
  ```

  Из всех доступных опций для нас будет полезна только `-y` (или `--yes`), подавляющая вывод вопроса на удаление библиотеки, а также списка составляющих ее файлов. Вот пример удаления библиотеки `Pillow` без вывода запроса:

  ```
  pip uninstall -y pillow
  ```

- `help` — вывод справочных сведений об утилите pip, поддерживаемых ею командах и опциях. Формат команды:

  ```
  pip help [<Команда>]
  ```

  - Если `<Команда>` не указана, будет выведен список всех поддерживаемых утилитой pip команд и универсальных опций:

    ```
    pip help
    ```

    Того же самого результата можно достичь, просто запустив в командной строке утилиту pip без всяких параметров.

  - Если `<Команда>` указана, будет выведена справочная информация об этой команде и всех ее опциях, а также перечень универсальных опций. В качестве примера выведем описание команды `install`:

    ```
    pip help install
    ```

Теперь рассмотрим список поддерживаемых pip универсальных опций:

- `--proxy` — задает прокси-сервер, через который будет выполняться доступ к Интернету. Формат использования:

  ```
  --proxy=[<Имя пользователя>:<Пароль пользователя>@]<Интернет-адрес>:<Номер порта>
  ```

  Пример:

  ```
  pip install pillow --proxy=user123:pAsSwOrD@192.168.1.1:3128
  ```

- `-v` (или `--verbose`) — вывод более подробных сведений о выполняемых утилитой pip действиях. Также может быть указана дважды или трижды, тем самым задавая вывод еще более подробных и самых подробных сведений соответственно:

  ```
  pip show pillow -v
  pip install pillow -v -v -v
  ```

  Дает эффект не со всеми командами pip.

- `-q` (или `--quiet`) — вывод менее подробных сведений о выполняемых утилитой pip действиях. Также может быть указана дважды или трижды, тем самым задавая вывод еще менее подробных и минимальных сведений соответственно:

```
pip show pillow -q
pip install pillow -q -q -q
```

  Дает эффект не со всеми командами pip.

- `-h` (или `--help`) — вывод справочных сведений о заданной команде pip, всех ее опциях и универсальных опциях pip (то есть дает эффект, аналогичный отдаче описанной ранее команды `help` с указанием команды, для которой нужно вывести справку). Для примера выведем сведения о команде `uninstall`:

```
pip uninstall -h
```

ГЛАВА 2

Переменные

Все данные в языке Python представлены *объектами*. Каждый объект имеет тип данных и значение. Для доступа к объекту предназначены *переменные*. При инициализации в переменной сохраняется *ссылка* на объект (адрес объекта в памяти компьютера). Благодаря этой ссылке можно в дальнейшем изменять объект из программы.

2.1. Именование переменных

Каждая переменная должна иметь уникальное имя, состоящее из латинских букв, цифр и знаков подчеркивания, причем имя переменной не может начинаться с цифры. Кроме того, следует избегать указания символа подчеркивания в начале имени, поскольку идентификаторам с таким символом определено специальное назначение. Например, имена, начинающиеся с символа подчеркивания, не импортируются из модуля с помощью инструкции `from module import *`, а имена, включающие по два символа подчеркивания — в начале и в конце, для интерпретатора имеют особый смысл.

В качестве имени переменной нельзя использовать *ключевые слова*. Получить список всех ключевых слов позволяет код, приведенный в листинге 2.1.

Листинг 2.1. Список всех ключевых слов

```
>>> import keyword
>>> keyword.kwlist
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue',
'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if',
'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return',
'try', 'while', 'with', 'yield']
```

Помимо ключевых слов, следует избегать совпадений со встроенными идентификаторами. Дело в том, что, в отличие от ключевых слов, встроенные идентификаторы можно переопределять, но дальнейший результат может стать для вас неожиданным (листинг 2.2).

Листинг 2.2. Ошибочное переопределение встроенных идентификаторов

```
>>> help(abs)
Help on built-in function abs in module builtins:
```