

Python 3

САМОЕ
НЕОБХОДИМОЕ



Основы языка Python 3
Классы и объекты
Итераторы и перечисления
Обработка исключений
Работа с файлами и каталогами
Основы SQLite
Доступ к данным SQLite и MySQL
Использование ODBC
Pillow и Wand: работа с графикой
Получение данных из Интернета
Сжатие и распаковка данных
Примеры и советы из практики



Материалы
на www.bhv.ru

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. — СПб.: БХВ-Петербург, 2016. — 464 с.: ил. — (Самое необходимое)
ISBN 978-5-9775-3631-8

Описан базовый синтаксис языка Python 3: типы данных, операторы, условия, циклы, регулярные выражения, встроенные функции, классы и объекты, итераторы и перечисления, обработка исключений, часто используемые модули стандартной библиотеки. 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 работа с архивами 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.

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:

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Евгений Рыбаков</i>
Зав. редакцией	<i>Екатерина Капалыгина</i>
Редактор	<i>Григорий Добин</i>
Компьютерная верстка	<i>Ольги Сергиенко</i>
Корректор	<i>Зинаида Дмитриева</i>
Дизайн серии	<i>Инны Тачиной</i>
Оформление обложки	<i>Марины Дамбиевой</i>

Подписано в печать 30.06.15.
Формат 70×100^{1/16}. Печать офсетная. Усл. печ. л. 37,41.
Тираж 1000 экз. Заказ №
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.
Первая Академическая типография "Наука"
199034, Санкт-Петербург, 9 линия, 12/28

ISBN 978-5-9775-3631-8

© Прохоренок Н. А., Дронов В. А., 2016
© Оформление, издательство "БХВ-Петербург", 2016

Оглавление

- Введение 9**
- Глава 1. Первые шаги 11**
 - 1.1. Установка Python 11
 - 1.1.1. Установка нескольких интерпретаторов Python 15
 - 1.1.2. Запуск программы с помощью разных версий Python 17
 - 1.2. Первая программа на Python..... 18
 - 1.3. Структура программы 20
 - 1.4. Комментарии..... 23
 - 1.5. Скрытые возможности IDLE 24
 - 1.6. Вывод результатов работы программы 25
 - 1.7. Ввод данных..... 27
 - 1.8. Доступ к документации 29
- Глава 2. Переменные 32**
 - 2.1. Именованние переменных 32
 - 2.2. Типы данных 34
 - 2.3. Присваивание значения переменным 37
 - 2.4. Проверка типа данных..... 39
 - 2.5. Преобразование типов данных 40
 - 2.6. Удаление переменной 43
- Глава 3. Операторы 44**
 - 3.1. Математические операторы..... 44
 - 3.2. Двоичные операторы..... 46
 - 3.3. Операторы для работы с последовательностями 47
 - 3.4. Операторы присваивания..... 48
 - 3.5. Приоритет выполнения операторов 49
- Глава 4. Условные операторы и циклы 51**
 - 4.1. Операторы сравнения..... 52
 - 4.2. Оператор ветвления *if...else* 54
 - 4.3. Цикл *for* 57
 - 4.4. Функции *range()* и *enumerate()* 59

4.5. Цикл <i>while</i>	62
4.6. Оператор <i>continue</i> . Переход на следующую итерацию цикла	63
4.7. Оператор <i>break</i> . Прерывание цикла	63

Глава 5. Числа..... 65

5.1. Встроенные функции и методы для работы с числами	67
5.2. Модуль <i>math</i> . Математические функции.....	69
5.3. Модуль <i>random</i> . Генерация случайных чисел	70

Глава 6. Строки и двоичные данные 73

6.1. Создание строки.....	74
6.2. Специальные символы	78
6.3. Операции над строками.....	78
6.4. Форматирование строк.....	81
6.5. Метод <i>format()</i>	87
6.6. Функции и методы для работы со строками	91
6.7. Настройка локали	95
6.8. Изменение регистра символов.....	96
6.9. Функции для работы с символами	96
6.10. Поиск и замена в строке.....	97
6.11. Проверка типа содержимого строки	100
6.12. Тип данных <i>bytes</i>	103
6.13. Тип данных <i>bytearray</i>	107
6.14. Преобразование объекта в последовательность байтов	110
6.15. Шифрование строк	111

Глава 7. Регулярные выражения 113

7.1. Синтаксис регулярных выражений	113
7.2. Поиск первого совпадения с шаблоном.....	122
7.3. Поиск всех совпадений с шаблоном	127
7.4. Замена в строке	129
7.5. Прочие функции и методы.....	131

Глава 8. Списки, кортежи, множества и диапазоны 132

8.1. Создание списка.....	133
8.2. Операции над списками	136
8.3. Многомерные списки.....	139
8.4. Перебор элементов списка.....	140
8.5. Генераторы списков и выражения-генераторы.....	141
8.6. Функции <i>map()</i> , <i>zip()</i> , <i>filter()</i> и <i>reduce()</i>	142
8.7. Добавление и удаление элементов списка.....	145
8.8. Поиск элемента в списке и получение сведений о значениях, входящих в список	147
8.9. Переворачивание и перемешивание списка	149
8.10. Выбор элементов случайным образом.....	149
8.11. Сортировка списка.....	150
8.12. Заполнение списка числами.....	151
8.13. Преобразование списка в строку	152
8.14. Кортежи	152
8.15. Множества.....	154

8.16. Диапазоны	159
8.17. Модуль <i>itertools</i>	161
8.17.1. Генерация неопределенного количества значений.....	161
8.17.2. Генерация комбинаций значений.....	162
8.17.3. Фильтрация элементов последовательности.....	163
8.17.4. Прочие функции	164
Глава 9. Словари	167
9.1. Создание словаря.....	167
9.2. Операции над словарями.....	170
9.3. Перебор элементов словаря	171
9.4. Методы для работы со словарями.....	172
9.5. Генераторы словарей.....	175
Глава 10. Работа с датой и временем.....	176
10.1. Получение текущих даты и времени.....	176
10.2. Форматирование даты и времени.....	178
10.3. «Засыпание» скрипта.....	180
10.4. Модуль <i>datetime</i> . Манипуляции датой и временем.....	181
10.4.1. Класс <i>timedelta</i>	181
10.4.2. Класс <i>date</i>	183
10.4.3. Класс <i>time</i>	187
10.4.4. Класс <i>datetime</i>	189
10.5. Модуль <i>calendar</i> . Вывод календаря	193
10.5.1. Методы классов <i>TextCalendar</i> и <i>LocaleTextCalendar</i>	195
10.5.2. Методы классов <i>HTMLCalendar</i> и <i>LocaleHTMLCalendar</i>	196
10.5.3. Другие полезные функции	197
10.6. Измерение времени выполнения фрагментов кода	200
Глава 11. Пользовательские функции	203
11.1. Определение функции и ее вызов	203
11.2. Расположение определений функций	206
11.3. Необязательные параметры и сопоставление по ключам	207
11.4. Переменное число параметров в функции	210
11.5. Анонимные функции	212
11.6. Функции-генераторы	213
11.7. Декораторы функций.....	214
11.8. Рекурсия. Вычисление факториала.....	216
11.9. Глобальные и локальные переменные	217
11.10. Вложенные функции	220
11.11. Аннотации функций	222
Глава 12. Модули и пакеты.....	223
12.1. Инструкция <i>import</i>	223
12.2. Инструкция <i>from</i>	227
12.3. Пути поиска модулей	229
12.4. Повторная загрузка модулей	230
12.5. Пакеты	231

Глава 13. Объектно-ориентированное программирование	235
13.1. Определение класса и создание экземпляра класса.....	235
13.2. Методы <code>__init__()</code> и <code>__del__()</code>	239
13.3. Наследование	239
13.4. Множественное наследование	241
13.4.1. Примеси и их использование.....	243
13.5. Специальные методы.....	244
13.6. Перегрузка операторов.....	247
13.7. Статические методы и методы класса	249
13.8. Абстрактные методы	250
13.9. Ограничение доступа к идентификаторам внутри класса	252
13.10. Свойства класса	253
13.11. Декораторы классов	255
Глава 14. Обработка исключений.....	256
14.1. Инструкция <code>try...except...else...finally</code>	257
14.2. Инструкция <code>with...as</code>	261
14.3. Классы встроенных исключений.....	263
14.4. Пользовательские исключения.....	265
Глава 15. Итераторы, контейнеры и перечисления	269
15.1. Итераторы	270
15.2. Контейнеры	271
15.2.1. Контейнеры-последовательности.....	271
15.2.2. Контейнеры-словари	273
15.3. Перечисления	274
Глава 16. Работа с файлами и каталогами.....	279
16.1. Открытие файла	279
16.2. Методы для работы с файлами.....	286
16.3. Доступ к файлам с помощью модуля <code>os</code>	292
16.4. Классы <code>StringIO</code> и <code>BytesIO</code>	294
16.5. Права доступа к файлам и каталогам.....	298
16.6. Функции для манипулирования файлами.....	300
16.7. Преобразование пути к файлу или каталогу.....	303
16.8. Перенаправление ввода/вывода.....	305
16.9. Сохранение объектов в файл	308
16.10. Функции для работы с каталогами.....	312
16.11. Исключения, возбуждаемые файловыми операциями	315
Глава 17. Основы SQLite	317
17.1. Создание базы данных	317
17.2. Создание таблицы.....	319
17.3. Вставка записей	325
17.4. Обновление и удаление записей.....	328
17.5. Изменение структуры таблицы.....	328
17.6. Выбор записей	329
17.7. Выбор записей из нескольких таблиц.....	332

17.8. Условия в инструкциях <i>WHERE</i> и <i>HAVING</i>	334
17.9. Индексы	337
17.10. Вложенные запросы	339
17.11. Транзакции	340
17.12. Удаление таблицы и базы данных	343

Глава 18. Доступ к базе данных SQLite из Python 344

18.1. Создание и открытие базы данных	345
18.2. Выполнение запросов	346
18.3. Обработка результата запроса	350
18.4. Управление транзакциями	354
18.5. Создание пользовательской сортировки	356
18.6. Поиск без учета регистра символов	357
18.7. Создание агрегатных функций	358
18.8. Преобразование типов данных	359
18.9. Сохранение в таблице даты и времени	363
18.10. Обработка исключений	364
18.11. Трассировка выполняемых запросов	367

Глава 19. Доступ к базам данных MySQL 368

19.1. Библиотека <i>MySQLClient</i>	369
19.1.1. Подключение к базе данных	369
19.1.2. Выполнение запросов	372
19.1.3. Обработка результата запроса	375
19.2. Библиотека <i>PyODBC</i>	378
19.2.1. Подключение к базе данных	379
19.2.2. Выполнение запросов	380
19.2.3. Обработка результата запроса	382

Глава 20. Библиотека *Pillow*. Работа с изображениями 386

20.1. Загрузка готового изображения	386
20.2. Создание нового изображения	388
20.3. Получение информации об изображении	389
20.4. Манипулирование изображением	390
20.5. Рисование линий и фигур	394
20.6. Библиотека <i>Wand</i>	396
20.7. Вывод текста	402
20.8. Создание скриншотов	406

Глава 21. Взаимодействие с Интернетом 407

21.1. Разбор URL-адреса	407
21.2. Кодирование и декодирование строки запроса	410
21.3. Преобразование относительного URL-адреса в абсолютный	414
21.4. Разбор HTML-эквивалентов	414
21.5. Обмен данными по протоколу HTTP	416
21.6. Обмен данными с помощью модуля <i>urllib.request</i>	421
21.7. Определение кодировки	424

Глава 22. Сжатие данных	426
22.1. Сжатие и распаковка по алгоритму GZIP	426
22.2. Сжатие и распаковка по алгоритму BZIP2	428
22.3. Сжатие и распаковка по алгоритму LZMA.....	430
22.4. Работа с архивами ZIP.....	433
22.5. Работа с архивами TAR.....	436
Заключение.....	441
Приложение. Описание электронного архива.....	443
Предметный указатель	445



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. Во-вторых, помните, что прочитать такую книгу один раз недостаточно. Начальные главы книги вы должны выучить наизусть! Сколько на это уйдет времени, зависит от ваших способностей и желания. Чем больше вы будете делать самостоятельно, тем большему научитесь. Ну что, приступим к изучению языка? Python достоин того, чтобы его знал каждый программист!

1.1. Установка Python

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

1. Для загрузки дистрибутива переходим на страницу <https://www.python.org/downloads/> и в списке доступных версий щелкаем на гиперссылке **Python 3.4.3** (эта версия является самой актуальной из стабильных версий на момент подготовки книги). На открывшейся странице находим раздел **Files** и щелкаем на гиперссылке **Windows x86 MSI installer** (32-разрядная версия интерпретатора) или **Windows x86-64 MSI installer** (его 64-разрядная версия). В результате на наш компьютер будет загружен файл `python-3.4.3.msi` или `python-3.4.3.amd64.msi` соответственно. Затем запускаем загруженный файл двойным щелчком на нем.
2. В открывшемся окне (рис. 1.1) устанавливаем переключатель **Install for all users** (Установить для всех пользователей) и нажимаем кнопку **Next**.
3. На следующем шаге (рис. 1.2) нам предлагается выбрать каталог для установки. Оставляем каталог по умолчанию (`C:\Python34\`) и нажимаем кнопку **Next**.
4. В следующем диалоговом окне (рис. 1.3) выбираем компоненты, которые необходимо установить. По умолчанию устанавливаются все компоненты и прописывается ассоциация с файловыми расширениями `py`, `pyw` и др. В этом случае запускать Python-программы можно будет с помощью двойного щелчка мышью на значке файла. Оставляем выбранными все компоненты и нажимаем кнопку **Next**.

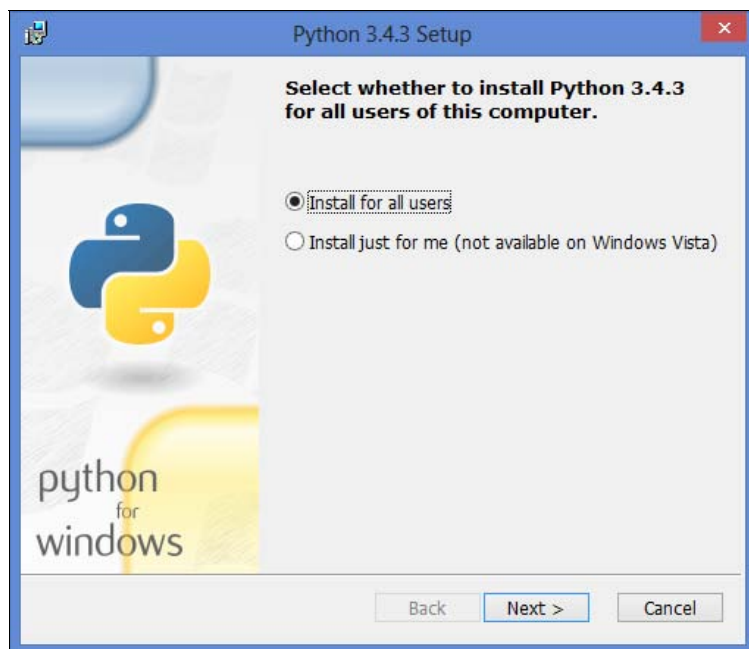


Рис. 1.1. Установка Python. Шаг 1

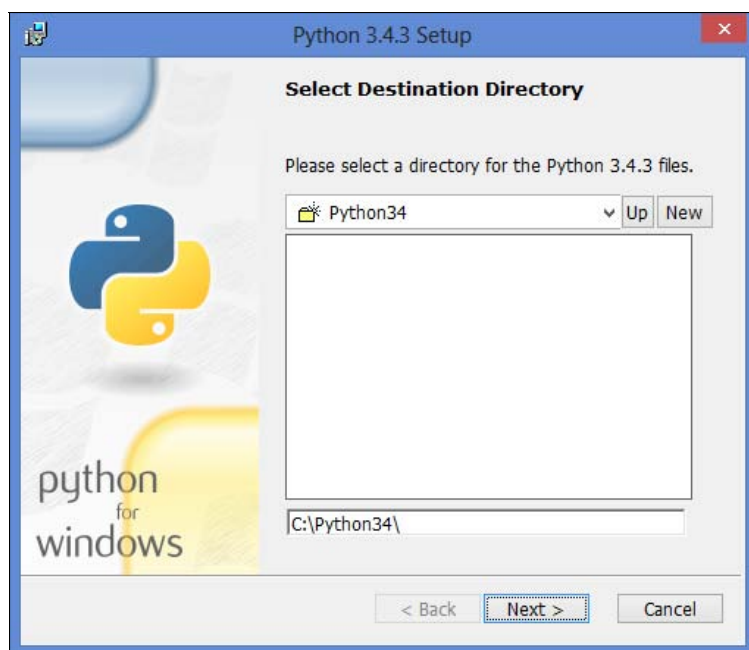


Рис. 1.2. Установка Python. Шаг 2



Рис. 1.3. Установка Python. Шаг 3

ПРИМЕЧАНИЕ

Пользователям Windows Vista и более поздних версий этой системы следует положительно ответить на появившееся на экране предупреждение системы UAC (Контроль учетных записей). Если этого не сделать, Python установлен не будет.

- После завершения установки откроется окно, изображенное на рис. 1.4. Нажимаем кнопку **Finish** для выхода из программы установки.

В результате установки файлы интерпретатора будут скопированы в папку `C:\Python34`. В этой папке 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 на языке 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.

Открыть такое же окно можно с помощью пункта **Python 3.4 (command line - 32 bit)** или **Python 3.4 (command line - 64 bit)** в меню **Пуск | Программы (Все программы) | Python 3.4**.

Вместо такой интерактивной оболочки для изучения языка, а также для создания и редактирования файлов с программой лучше воспользоваться редактором IDLE, который входит



Рис. 1.4. Установка Python. Шаг 4

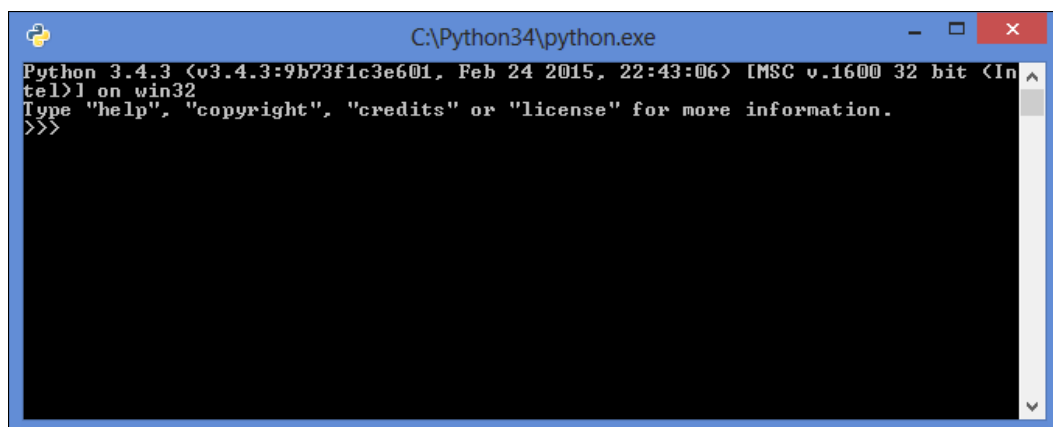
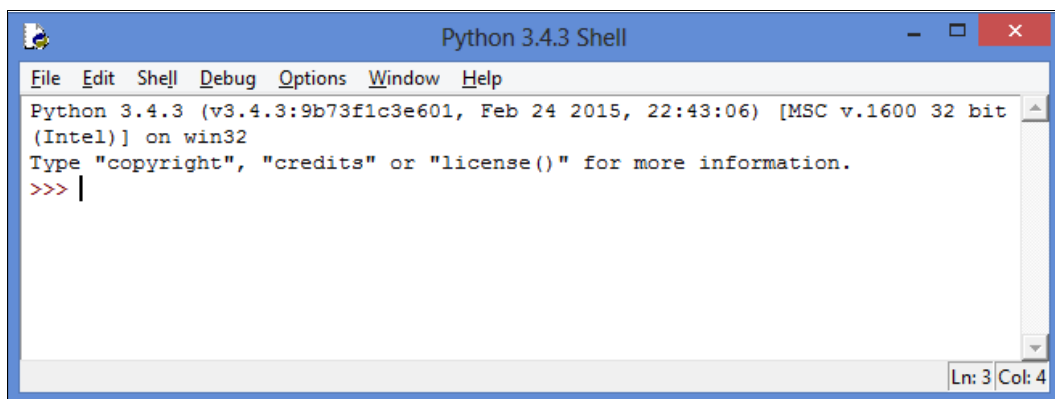


Рис. 1.5. Интерактивная оболочка

в состав установленных компонентов. Для запуска редактора в меню **Пуск | Программы (Все программы) | Python 3.4** выбираем пункт **IDLE (Python 3.4 GUI - 32 bit)** или **IDLE (Python 3.4 GUI - 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.

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

1.1.1. Установка нескольких интерпретаторов Python

Версии языка Python выпускаются с завидной регулярностью, но, к сожалению, сторонние разработчики не успевают за такой скоростью и не столь часто обновляют свои модули. Поэтому приходится при наличии версии Python 3 использовать на практике также и версию Python 2. Как же быть, если установлена версия 3.4, а необходимо запустить модуль для версии 2.7? В этом случае удалять версию 3.4 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 при установке просто отключить.

В качестве примера мы дополнительно установим на компьютер версию 2.7.8.10, но вместо программы установки с сайта <https://www.python.org/> выберем альтернативный дистрибутив от компании ActiveState.

Итак, переходим на страницу <http://www.activestate.com/activepython/downloads/> и скачиваем дистрибутив. Последовательность запуска нескольких программ установки от компании ActiveState имеет значение, поскольку в контекстное меню добавляется пункт **Edit with Pythonwin**. С помощью этого пункта запускается редактор PythonWin, который можно использовать вместо IDLE. Соответственно, из контекстного меню будет открываться версия PythonWin, которая была установлена последней. Установку программы производим в каталог по умолчанию (C:\Python27\).

ВНИМАНИЕ!

При установке в окне **Custom Setup** (рис. 1.7) необходимо отключить компонент **Register as Default Python** (рис. 1.8). Не забудьте это сделать, иначе Python 3.4.3 перестанет быть текущей версией.

В состав ActivePython, кроме редактора PythonWin, входит также редактор IDLE. Однако ни в одном меню 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.

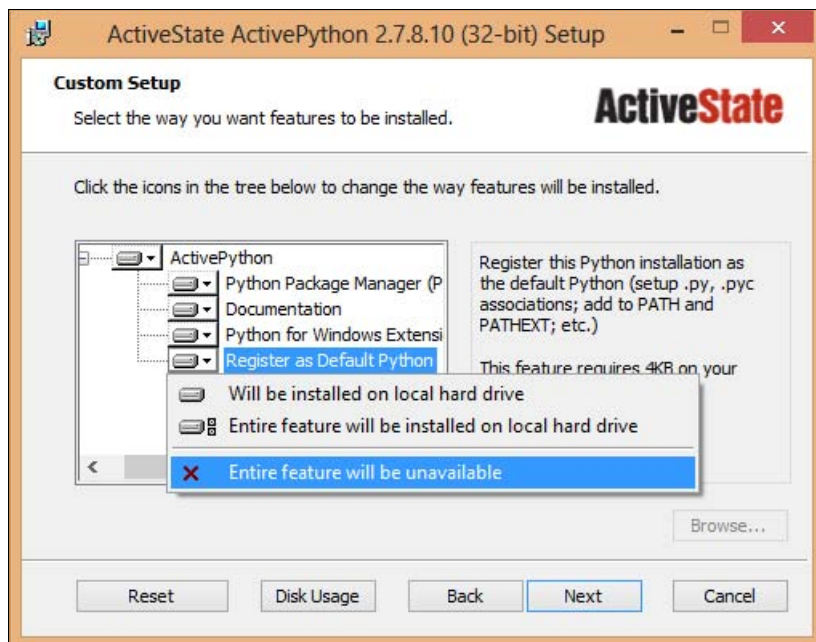


Рис. 1.7. Окно Custom Setup

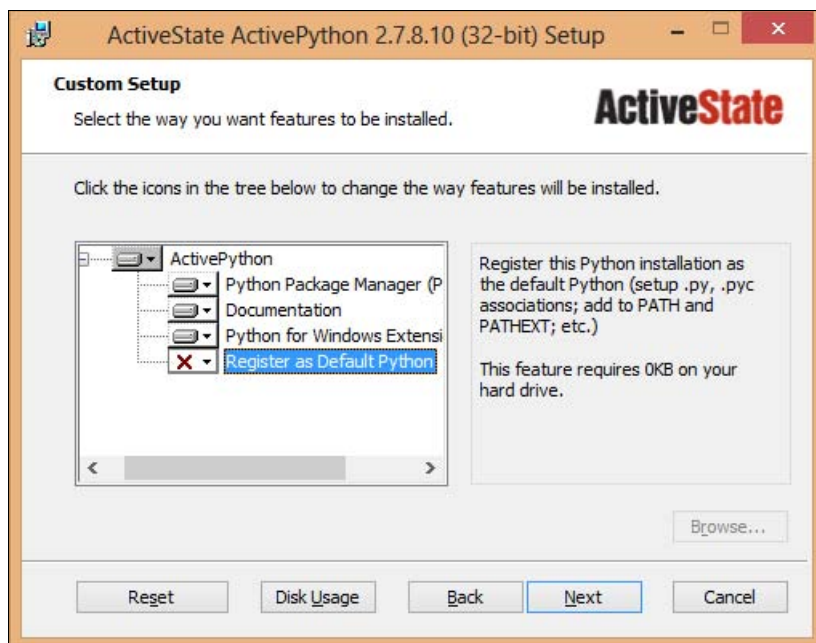


Рис. 1.8. Компонент Register as Default Python отключен

Ну, а запуск IDLE для версии Python 3.4 будет по-прежнему осуществляться так же, как и предлагалось ранее, — выбором в меню **Пуск | Программы (Все программы) | Python 3.4** пункта **IDLE (Python 3.4 GUI - 32 bit)** или **IDLE (Python 3.4 GUI - 64 bit)**.

1.1.2. Запуск программы с помощью разных версий Python

Теперь рассмотрим запуск программы с помощью разных версий Python. По умолчанию при двойном щелчке на значке файла запускается Python 3.4. Чтобы запустить Python-программу с помощью другой версии этого языка, щелкаем правой кнопкой мыши на значке файла с программой и в контекстном меню находим пункт **Открыть с помощью**.

В Windows XP при выборе этого пункта появится подменю, в котором изначально будет присутствовать только программа `python.exe`. Чтобы добавить другую версию, щелкаем на пункте **Выбрать программу**, в открывшемся окне нажимаем кнопку **Обзор** и выбираем программу `python2.7.exe` из папки `C:\Python27`. Выбранная нами программа будет добавлена в подменю, открывающееся при выборе пункта **Открыть с помощью**. Впоследствии для выполнения файла под управлением Python 2.7 мы просто выберем этот пункт.

В Windows Vista и более поздних версиях этой системы при выборе упомянутого пункта изначально не будет открываться никакого подменю. Вместо этого на экране появится небольшое окно выбора альтернативной программы для запуска файла (рис. 1.9). Сразу же сбросим флажок **Использовать это приложение для всех файлов .py** и нажмем ссылку **Дополнительно**. В окне появится список установленных на нашем компьютере программ, но нужного нам приложения `python2.7.exe` в нем не будет. Поэтому щелкнем на ссылке **Найти другое приложение на этом компьютере**, находящейся под списком. На экране появится стандартное диалоговое окно открытия файла, в котором мы выберем программу `python2.7.exe` из папки `C:\Python27`. Теперь эта программа появится в подменю, открывающемся при выборе пункта **Открыть с помощью** (рис. 1.10), — здесь Python 2.7 представлен как **Python Launcher for Windows (Console)**.

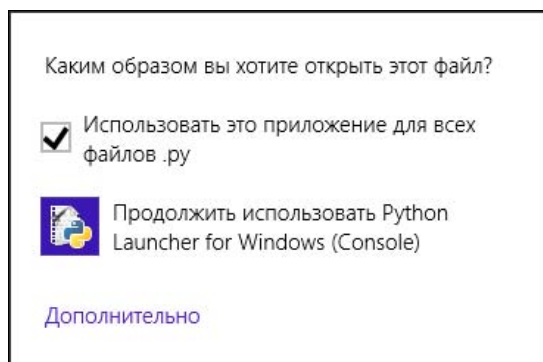


Рис. 1.9. Диалоговое окно выбора альтернативной программы для запуска файла

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, 4, 3, 'final', 0), то установка прошла нормально, а если (2, 7, 8, 'final', 0), то вы не отключили компонент **Register as Default Python**.

Для изучения материала этой книги по умолчанию должна запускаться версия Python 3.4.

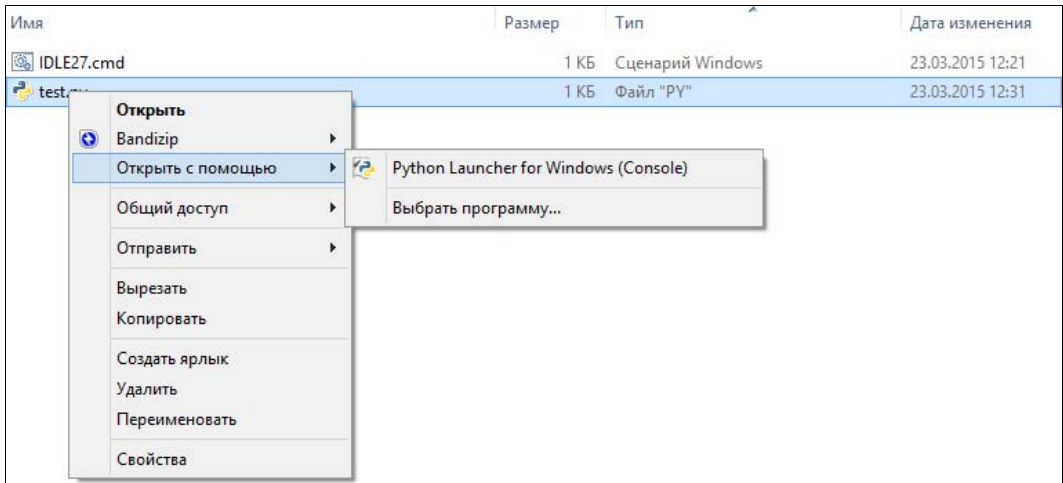


Рис. 1.10. Варианты запуска программы разными версиями Python

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 3.4** выбираем пункт **IDLE (Python 3.4 GUI - 32 bit)** или **IDLE (Python 3.4 GUI - 64 bit)**. В результате откроется окно **Python Shell**, в котором символы `>>>` означают приглашение ввести команду. 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 показана в листинге 1.3.

Листинг 1.3. Последовательность выполнения программы в окне *Python Shell*

```
>>> # 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**. В открывшемся окне набираем код из листинга 1.2, а затем сохраняем его под именем `hello_world.py`, выбрав пункт меню **File | Save As**. При этом редактор сохранит файл в кодировке UTF-8 без BOM (Byte Order Mark, метка порядка байтов). Именно кодировка UTF-8 является кодировкой по умолчанию в Python 3. Если файл содержит инструкции в другой кодировке, то необходимо в первой или второй строке 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 можно также 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 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 1.4.

Листинг 1.4. Программа для запуска с помощью двойного щелчка мыши

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

ПРИМЕЧАНИЕ

Если до 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** и укажем 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 папке `__pycache__`, вложенной в папку, 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 `test4.py` будет создан файл откомпилированного кода с именем `test4.cpython-34.pyc`.

При последующем запуске того же файла на выполнение 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 саму папку `__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, то она должна начинаться с начала строки, иначе будет выведено сообщение об ошибке (листинг 1.5).

Листинг 1.5. Ошибка `SyntaxError`

```
>>> 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 дополнительно 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 программы `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 двух 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 -*-
```

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 1.6.

Листинг 1.6. 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* (листинг 1.7).

Листинг 1.7. Несколько инструкций на одной строке

```
>>> 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-другому (листинг 1.8).

Листинг 1.8. 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 разместить символ `\`. После этого символа должен следовать символ перевода 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 не будет выполнена:

```
# 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 часто комментируемый фрагмент размещают внутри утроенных кавычек (или утроенных апострофов):

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

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

1.5. Скрытые возможности IDLE

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

Как вы уже знаете, в окне **Python Shell** символы `>>>` означают приглашение ввести команду. После ввода команды 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>`. Пример:

```
>>> 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. Например, 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.

Пример:

```
>>> 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. Пример:

```
>>> 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 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**. Постоянно выбирать пункты из этого меню очень неудобно. Одним из решений проблемы является использование комбинации клавиш быстрого доступа `<Ctrl>+<C>` (Копировать) и `<Ctrl>+<V>` (Вставить). Комбинации 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 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 клавиш будет вставлять следующую (или предыдущую) инструкцию. Для еще более быстрого повторного ввода инструкции следует предварительно ввести ее первые буквы. В этом случае перебирать 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([<Объект>][, 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 `False`, 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. Пример:

```
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, то его следует разместить между утроенными кавычками или утроенными апострофами. В этом случае текст сохраняет свое форматирование. Пример:

```
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. Далее с помощью метода `write()` 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 1.9).

Листинг 1.9. Пример использования функции `input()`

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