

Николай Прохоренок

Python 3 и PyQt

Разработка приложений

Санкт-Петербург

«БХВ-Петербург»

2012

УДК 681.3.06
ББК 32.973.26-018.2
П84

Прохоренок Н. А.

П84 Python 3 и PyQt. Разработка приложений. — СПб.: БХВ-Петербург, 2012. — 704 с.: ил.

ISBN 978-5-9775-0797-4

Описан базовый синтаксис языка Python: типы данных, операторы, условия, циклы, регулярные выражения, встроенные функции, объектно-ориентированное программирование, работа с файлами и каталогами, часто используемые модули стандартной библиотеки. Приведены основы базы данных SQLite, интерфейс доступа к базе и способы получения данных из Интернета. Особое внимание уделено библиотеке PyQt, позволяющей создавать приложения с графическим интерфейсом на языке Python. Рассмотрены способы обработки сигналов и событий, управление свойствами окна, создание формы с помощью программы Qt Designer, работа многопоточных приложений, а также все основные компоненты (кнопки, текстовые поля, списки, таблицы, меню, панели инструментов и др.) и варианты их размещения внутри окна. На сайте издательства приведены все примеры из книги.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

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

Подписано в печать 01.12.11.

Формат 70×100^{1/16}. Печать офсетная. Усл. печ. л. 56,76.

Тираж 1200 экз. Заказ №

"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0797-4

© Прохоренок Н. А., 2012
© Оформление, издательство "БХВ-Петербург", 2012

Оглавление

Введение	13
ЧАСТЬ I. ОСНОВЫ PYTHON 3	15
Глава 1. Первые шаги	17
1.1. Установка Python	17
1.2. Первая программа на Python	23
1.3. Структура программы	24
1.4. Комментарии	27
1.5. Скрытые возможности IDLE	28
1.6. Вывод результатов работы программы.....	29
1.7. Ввод данных	31
1.8. Доступ к документации	33
Глава 2. Переменные	36
2.1. Именованние переменных.....	36
2.2. Типы данных	38
2.3. Присваивание значения переменным.....	41
2.4. Проверка типа данных.....	43
2.5. Преобразование типов данных	44
2.6. Удаление переменной.....	47
Глава 3. Операторы	48
3.1. Математические операторы	48
3.2. Двоичные операторы	50
3.3. Операторы для работы с последовательностями	51
3.4. Операторы присваивания	52
3.5. Приоритет выполнения операторов	53
Глава 4. Условные операторы и циклы	55
4.1. Операторы сравнения	56
4.2. Оператор ветвления <i>if...else</i>	58
4.3. Цикл <i>for</i>	61

4.4. Функции <i>range()</i> и <i>enumerate()</i>	63
4.5. Цикл <i>while</i>	66
4.6. Оператор <i>continue</i> . Переход на следующую итерацию цикла.....	67
4.7. Оператор <i>break</i> . Прерывание цикла.....	67
Глава 5. Числа	69
5.1. Встроенные функции для работы с числами.....	70
5.2. Модуль <i>math</i> . Математические функции.....	72
5.3. Модуль <i>random</i> . Генерация случайных чисел.....	73
Глава 6. Строки	76
6.1. Создание строки.....	77
6.2. Специальные символы.....	81
6.3. Операции над строками.....	81
6.4. Форматирование строк.....	84
6.5. Метод <i>format()</i>	90
6.6. Функции и методы для работы со строками.....	94
6.7. Настройка локали.....	97
6.8. Изменение регистра символов.....	98
6.9. Функции для работы с символами.....	98
6.10. Поиск и замена в строке.....	99
6.11. Проверка типа содержимого строки.....	102
6.12. Тип данных <i>bytes</i>	105
6.13. Тип данных <i>bytearray</i>	108
6.14. Преобразование объекта в последовательность байтов.....	112
6.15. Шифрование строк.....	112
Глава 7. Регулярные выражения	114
7.1. Синтаксис регулярных выражений.....	114
7.2. Поиск первого совпадения с шаблоном.....	123
7.3. Поиск всех совпадений с шаблоном.....	127
7.4. Замена в строке.....	128
7.5. Прочие функции и методы.....	130
Глава 8. Списки и кортежи	132
8.1. Создание списка.....	132
8.2. Операции над списками.....	136
8.3. Многомерные списки.....	139
8.4. Перебор элементов списка.....	139
8.5. Генераторы списков и выражения-генераторы.....	140
8.6. Функции <i>map()</i> , <i>zip()</i> , <i>filter()</i> и <i>reduce()</i>	141
8.7. Добавление и удаление элементов списка.....	144
8.8. Поиск элемента в списке.....	147
8.9. Переворачивание и перемешивание списка.....	148
8.10. Выбор элементов случайным образом.....	148
8.11. Сортировка списка.....	149
8.12. Заполнение списка числами.....	150
8.13. Преобразование списка в строку.....	151
8.14. Кортежи.....	151

8.15. Модуль <i>itertools</i>	153
8.15.1. Генерация неопределенного количества значений.....	153
8.15.2. Генерация комбинаций значений.....	154
8.15.3. Фильтрация элементов последовательности.....	156
8.15.4. Прочие функции.....	157
Глава 9. Словари и множества.....	159
9.1. Создание словаря.....	159
9.2. Операции над словарями.....	161
9.3. Перебор элементов словаря.....	163
9.4. Методы для работы со словарями.....	164
9.5. Генераторы словарей.....	166
9.6. Множества.....	167
9.7. Генераторы множеств.....	172
Глава 10. Работа с датой и временем.....	173
10.1. Получение текущей даты и времени.....	173
10.2. Форматирование даты и времени.....	175
10.3. "Засыпание" скрипта.....	177
10.4. Модуль <i>datetime</i> . Манипуляции датой и временем.....	178
10.4.1. Класс <i>timedelta</i>	178
10.4.2. Класс <i>date</i>	180
10.4.3. Класс <i>time</i>	183
10.4.4. Класс <i>datetime</i>	184
10.5. Модуль <i>calendar</i> . Вывод календаря.....	189
10.5.1. Методы классов <i>TextCalendar</i> и <i>LocaleTextCalendar</i>	190
10.5.2. Методы классов <i>HTMLCalendar</i> и <i>LocaleHTMLCalendar</i>	191
10.5.3. Другие полезные функции.....	192
10.6. Измерение времени выполнения фрагментов кода.....	195
Глава 11. Пользовательские функции.....	197
11.1. Создание функции и ее вызов.....	197
11.2. Расположение определений функций.....	200
11.3. Необязательные параметры и сопоставление по ключам.....	201
11.4. Переменное число параметров в функции.....	203
11.5. Анонимные функции.....	205
11.6. Функции-генераторы.....	206
11.7. Декораторы функций.....	207
11.8. Рекурсия. Вычисление факториала.....	209
11.9. Глобальные и локальные переменные.....	210
11.10. Вложенные функции.....	213
11.11. Аннотации функций.....	215
Глава 12. Модули и пакеты.....	216
12.1. Инструкция <i>import</i>	216
12.2. Инструкция <i>from</i>	219
12.3. Пути поиска модулей.....	222
12.4. Повторная загрузка модулей.....	223
12.5. Пакеты.....	224

Глава 13. Объектно-ориентированное программирование.....	228
13.1. Определение класса и создание экземпляра класса.....	228
13.2. Методы <code>__init__()</code> и <code>__del__()</code>	231
13.3. Наследование.....	232
13.4. Множественное наследование.....	234
13.5. Специальные методы.....	236
13.6. Перегрузка операторов.....	239
13.7. Статические методы и методы класса.....	242
13.8. Абстрактные методы.....	243
13.9. Ограничение доступа к идентификаторам внутри класса.....	244
13.10. Свойства класса.....	245
13.11. Декораторы классов.....	246
Глава 14. Обработка исключений.....	247
14.1. Инструкция <code>try...except...else...finally</code>	248
14.2. Инструкция <code>with...as</code>	252
14.3. Классы встроенных исключений.....	254
14.4. Пользовательские исключения.....	256
Глава 15. Работа с файлами и каталогами.....	259
15.1. Открытие файла.....	259
15.2. Методы для работы с файлами.....	265
15.3. Доступ к файлам с помощью модуля <code>os</code>	271
15.4. Классы <code>StringIO</code> и <code>BytesIO</code>	273
15.5. Права доступа к файлам и каталогам.....	277
15.6. Функции для манипулирования файлами.....	278
15.7. Преобразование пути к файлу или каталогу.....	281
15.8. Перенаправление ввода/вывода.....	283
15.9. Сохранение объектов в файл.....	286
15.10. Функции для работы с каталогами.....	289
Глава 16. Основы SQLite.....	293
16.1. Создание базы данных.....	293
16.2. Создание таблицы.....	295
16.3. Вставка записей.....	301
16.4. Обновление и удаление записей.....	303
16.5. Изменение свойств таблицы.....	304
16.6. Выбор записей.....	305
16.7. Выбор записей из нескольких таблиц.....	308
16.8. Условия в инструкции <code>WHERE</code>	309
16.9. Индексы.....	312
16.10. Вложенные запросы.....	314
16.11. Транзакции.....	315
16.12. Удаление таблицы и базы данных.....	317
Глава 17. Доступ к базе данных SQLite из Python.....	318
17.1. Создание и открытие базы данных.....	319
17.2. Выполнение запроса.....	319
17.3. Обработка результата запроса.....	324

17.4. Управление транзакциями.....	327
17.5. Создание пользовательской сортировки.....	328
17.6. Поиск без учета регистра символов	330
17.7. Создание агрегатных функций.....	331
17.8. Преобразование типов данных	332
17.9. Сохранение в таблице даты и времени	336
17.10. Обработка исключений	337

Глава 18. Взаимодействие с Интернетом 340

18.1. Разбор URL-адреса	340
18.2. Кодирование и декодирование строки запроса	343
18.3. Преобразование относительной ссылки в абсолютную	347
18.4. Разбор HTML-эквивалентов	347
18.5. Обмен данными по протоколу HTTP	348
18.6. Обмен данными с помощью модуля <i>urllib.request</i>	354
18.7. Определение кодировки	357

ЧАСТЬ II. СОЗДАНИЕ ОКОННЫХ ПРИЛОЖЕНИЙ..... 359

Глава 19. Знакомство с PyQt 361

19.1. Установка PyQt	361
19.2. Первая программа.....	364
19.3. Структура программы	365
19.4. ООП-стиль создания окна	367
19.5. Создание окна с помощью программы Qt Designer.....	371
19.5.1. Создание формы	371
19.5.2. Загрузка ui-файла в программе.....	373
19.5.3. Преобразование ui-файла в ru-файл	375
19.6. Модули PyQt.....	377
19.7. Типы данных в PyQt	377
19.8. Управление основным циклом приложения.....	379
19.9. Многопоточные приложения.....	381
19.9.1. Класс <i>QThread</i> . Создание потока	381
19.9.2. Управление циклом внутри потока.....	384
19.9.3. Модуль <i>queue</i> . Создание очереди заданий	388
19.9.4. Классы <i>QMutex</i> и <i>QMutexLocker</i>	391
19.10. Вывод заставки.....	395
19.11. Доступ к документации	397

Глава 20. Управление окном приложения..... 398

20.1. Создание и отображение окна	398
20.2. Указание типа окна	399
20.3. Изменение и получение размеров окна.....	401
20.4. Местоположение окна на экране	403
20.5. Указание координат и размеров	406
20.5.1. Класс <i>QPoint</i> . Координаты точки.....	407
20.5.2. Класс <i>QSize</i> . Размеры прямоугольной области.....	408
20.5.3. Класс <i>QRect</i> . Координаты и размеры прямоугольной области	410
20.6. Разворачивание и сворачивание окна	415
20.7. Управление прозрачностью окна	416

20.8. Модальные окна	417
20.9. Смена иконки в заголовке окна	419
20.10. Изменение цвета фона окна	420
20.11. Использование изображения в качестве фона	421
20.12. Создание окна произвольной формы	422
20.13. Всплывающие подсказки	423
20.14. Закрытие окна из программы	424
Глава 21. Обработка сигналов и событий.....	426
21.1. Назначение обработчиков сигналов	426
21.2. Блокировка и удаление обработчика	430
21.3. Генерация сигнала из программы	433
21.4. Новый стиль назначения и удаления обработчиков	435
21.5. Передача данных в обработчик	438
21.6. Использование таймеров	439
21.7. Перехват всех событий	442
21.8. События окна	445
21.8.1. Изменение состояния окна	445
21.8.2. Изменение положения окна и его размеров	446
21.8.3. Перерисовка окна или его части	447
21.8.4. Предотвращение закрытия окна	448
21.9. События клавиатуры	449
21.9.1. Установка фокуса ввода	449
21.9.2. Назначение клавиш быстрого доступа	451
21.9.3. Нажатие и отпускание клавиши на клавиатуре	454
21.10. События мыши	455
21.10.1. Нажатие и отпускание кнопки мыши	455
21.10.2. Перемещение указателя	456
21.10.3. Наведение и выведение указателя	457
21.10.4. Прокрутка колесика мыши	457
21.10.5. Изменение внешнего вида указателя мыши	458
21.11. Технология drag & drop	460
21.11.1. Запуск перетаскивания	460
21.11.2. Класс <i>QMimeData</i>	462
21.11.3. Обработка сброса	463
21.12. Работа с буфером обмена	465
21.13. Фильтрация событий	465
21.14. Искусственные события	466
Глава 22. Размещение нескольких компонентов в окне	468
22.1. Абсолютное позиционирование	468
22.2. Горизонтальное и вертикальное выравнивание	469
22.3. Выравнивание по сетке	472
22.4. Выравнивание компонентов формы	473
22.5. Классы <i>QStackedLayout</i> и <i>QStackedWidget</i>	475
22.6. Класс <i>QSizePolicy</i>	477
22.7. Объединение компонентов в группу	478
22.8. Панель с рамкой	479
22.9. Панель с вкладками	480

22.10. Компонент "аккордеон"	484
22.11. Панели с изменяемым размером	485
22.12. Область с полосами прокрутки.....	487
Глава 23. Основные компоненты	489
23.1. Надпись.....	489
23.2. Командная кнопка.....	492
23.3. Переключатель.....	494
23.4. Флажок.....	494
23.5. Однострочное текстовое поле.....	495
23.5.1. Основные методы и сигналы.....	495
23.5.2. Ввод данных по маске.....	498
23.5.3. Контроль ввода.....	499
23.6. Многострочное текстовое поле	500
23.6.1. Основные методы и сигналы.....	500
23.6.2. Изменение настроек поля	502
23.6.3. Изменение характеристик текста и фона.....	504
23.6.4. Класс <i>QTextDocument</i>	505
23.6.5. Класс <i>QTextCursor</i>	508
23.7. Текстовый браузер.....	511
23.8. Поля для ввода целых и вещественных чисел.....	512
23.9. Поля для ввода даты и времени	514
23.10. Календарь	516
23.11. Электронный индикатор.....	517
23.12. Индикатор хода процесса.....	518
23.13. Шкала с ползунком	519
23.14. Класс <i>QDial</i>	521
23.15. Полоса прокрутки	522
Глава 24. Списки и таблицы	523
24.1. Раскрывающийся список.....	523
24.1.1. Добавление, изменение и удаление элементов	523
24.1.2. Изменение настроек	524
24.1.3. Поиск элемента внутри списка.....	526
24.1.4. Сигналы.....	526
24.2. Список для выбора шрифта	526
24.3. Роли элементов	527
24.4. Модели.....	528
24.4.1. Доступ к данным внутри модели	528
24.4.2. Класс <i>QStringListModel</i>	529
24.4.3. Класс <i>QStandardItemModel</i>	530
24.4.4. Класс <i>QStandardItem</i>	533
24.5. Представления.....	536
24.5.1. Класс <i>QAbstractItemView</i>	536
24.5.2. Простой список.....	539
24.5.3. Таблица.....	540
24.5.4. Иерархический список	543
24.5.5. Управление заголовками строк и столбцов.....	545
24.6. Управление выделением элементов	547
24.7. Промежуточные модели	549

Глава 25. Работа с графикой	551
25.1. Вспомогательные классы	551
25.1.1. Класс <i>QColor</i> . Цвет.....	551
25.1.2. Класс <i>QPen</i> . Перо	555
25.1.3. Класс <i>QBrush</i> . Кисть.....	556
25.1.4. Класс <i>QLine</i> . Линия	557
25.1.5. Класс <i>QPolygon</i> . Многоугольник	558
25.1.6. Класс <i>QFont</i> . Шрифт	560
25.2. Класс <i>QPainter</i>	562
25.2.1. Рисование линий и фигур	562
25.2.2. Вывод текста	565
25.2.3. Вывод изображения.....	566
25.2.4. Преобразование систем координат	567
25.2.5. Сохранение команд рисования в файл.....	568
25.3. Работа с изображениями	569
25.3.1. Класс <i>QPixmap</i>	570
25.3.2. Класс <i>QBitmap</i>	572
25.3.3. Класс <i>QImage</i>	573
25.3.4. Класс <i>QIcon</i>	576
Глава 26. Графическая сцена	578
26.1. Класс <i>QGraphicsScene</i> . Сцена	578
26.1.1. Настройка параметров сцены	579
26.1.2. Добавление и удаление графических объектов	579
26.1.3. Добавление компонентов на сцену	580
26.1.4. Поиск объектов.....	581
26.1.5. Управление фокусом ввода	582
26.1.6. Управление выделением объектов.....	582
26.1.7. Прочие методы и сигналы	583
26.2. Класс <i>QGraphicsView</i> . Представление	584
26.2.1. Настройка параметров представления.....	584
26.2.2. Преобразования между координатами представления и сцены	586
26.2.3. Поиск объектов.....	586
26.2.4. Трансформация систем координат.....	587
26.2.5. Прочие методы	587
26.3. Класс <i>QGraphicsItem</i> . Базовый класс для графических объектов.....	588
26.3.1. Настройка параметров объекта	588
26.3.2. Трансформация объекта.....	590
26.3.3. Прочие методы	591
26.4. Графические объекты	592
26.4.1. Линия	592
26.4.2. Класс <i>QAbstractGraphicsShapeItem</i>	593
26.4.3. Прямоугольник	593
26.4.4. Многоугольник	593
26.4.5. Эллипс	594
26.4.6. Изображение	594
26.4.7. Простой текст	595
26.4.8. Форматированный текст	595

26.5. Группировка объектов.....	597
26.6. Эффекты	597
26.6.1. Класс <i>QGraphicsEffect</i>	597
26.6.2. Тень.....	598
26.6.3. Размытие	599
26.6.4. Изменение цвета	599
26.6.5. Изменение прозрачности	600
26.7. Обработка событий.....	600
26.7.1. События клавиатуры	600
26.7.2. События мыши	601
26.7.3. Обработка перетаскивания и сброса.....	603
26.7.4. Фильтрация событий.....	605
26.7.5. Обработка изменения состояния объекта.....	605
Глава 27. Диалоговые окна.....	607
27.1. Пользовательские диалоговые окна	607
27.2. Класс <i>QDialogButtonBox</i>	610
27.3. Класс <i>QMessageBox</i>	612
27.3.1. Основные методы и сигналы.....	614
27.3.2. Окно для вывода обычного сообщения	616
27.3.3. Окно запроса подтверждения.....	616
27.3.4. Окно для вывода предупреждающего сообщения.....	617
27.3.5. Окно для вывода критического сообщения.....	617
27.3.6. Окно "О программе"	618
27.3.7. Окно "About Qt"	618
27.4. Класс <i>QInputDialog</i>	618
27.4.1. Основные методы и сигналы.....	619
27.4.2. Окно для ввода строки	621
27.4.3. Окно для ввода целого числа.....	621
27.4.4. Окно для ввода вещественного числа.....	622
27.4.5. Окно для выбора пункта из списка	623
27.5. Класс <i>QFileDialog</i>	623
27.5.1. Основные методы и сигналы.....	624
27.5.2. Окно для выбора каталога	626
27.5.3. Окна для открытия файла	627
27.5.4. Окна для сохранения файла.....	628
27.6. Окно для выбора цвета	629
27.7. Окно для выбора шрифта.....	630
27.8. Окно для вывода сообщения об ошибке.....	631
27.9. Окно с индикатором хода процесса	632
27.10. Создание многостраничного мастера.....	633
27.10.1. Класс <i>QWizard</i>	633
27.10.2. Класс <i>QWizardPage</i>	637
Глава 28. Создание SDI- и MDI-приложений	640
28.1. Создание главного окна приложения.....	640
28.2. Меню.....	644
28.2.1. Класс <i>QMenuBar</i>	645
28.2.2. Класс <i>QMenu</i>	646

28.2.3. Контекстное меню	648
28.2.4. Класс <i>QAction</i>	649
28.2.5. Объединение переключателей в группу	652
28.3. Панели инструментов	653
28.3.1. Класс <i>QToolBar</i>	653
28.3.2. Класс <i>QToolButton</i>	655
28.4. Прикрепляемые панели	656
28.5. Управление строкой состояния	658
28.6. MDI-приложения	659
28.6.1. Класс <i>QMdiArea</i>	659
28.6.2. Класс <i>QMdiSubWindow</i>	662
28.7. Добавление иконки приложения в область уведомлений	663
Заключение	665
Приложение. Описание электронного архива	666
Предметный указатель	667

Введение

Добро пожаловать в мир Python и PyQt!

Python — это интерпретируемый объектно-ориентированный язык программирования высокого уровня, предназначенный для самого широкого круга задач. С его помощью можно обрабатывать различные данные, создавать изображения, работать с базами данных, разрабатывать Web-сайты и приложения с графическим интерфейсом. Python является кросс-платформенным языком, позволяющим создавать программы, которые будут работать во всех операционных системах. В этой книге мы рассмотрим базовые возможности Python 3.2 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 "Monty Python's Flying Circus". Поэтому более правильно будет "Пайтон". Тем не менее многие считают, что для русского человека более привычно называть язык "Питон". Так или иначе, в этой книге мы будем придерживаться традиционного написания слова Python на английском языке.

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 и сохраняется в 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, скомпилировать его, а затем подключить к основной программе.

Python является объектно-ориентированным языком. Это означает, что практически все данные являются объектами, даже 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. Данное 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 ООП-стиль программирования является обязательным. На языке Python можно писать программы как в ООП-стиле, так и в процедурном стиле.

Python — самый стильный язык программирования в мире, не допускающий двоякого написания кода. Например, в языке Perl существует зависимость от контекста и множественность синтаксиса. Часто два программиста, пишущих на Perl, просто не понимают код друг

друга. В Python код можно написать только одним способом, т. к. в нем отсутствуют лишние конструкции. Все программисты должны придерживаться стандарта, описанного в документе <http://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. На первый взгляд может показаться, что отсутствие ограничительных символов (фигурных скобок или конструкции `begin...end`) для выделения блоков и обязательная вставка пробелов впереди инструкций может 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 являются лишними. Бытует мнение, что программа будет по-разному смотреться в разных редакторах. Это неверно. Согласно стандарту для выделения блоков необходимо использовать *четыре пробела*. 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 будет разным, то интерпретатор выведет сообщение о фатальной ошибке, и программа будет остановлена. Таким образом, язык Python приучает программистов писать красивый и понятный код.

Так как программа на языке Python представляет собой обычный текстовый файл, его можно редактировать с помощью любого текстового редактора, например с помощью Notepad++. Однако лучше воспользоваться специализированными редакторами, которые не только подсвечивают код, но и выводят различные подсказки и позволяют отладить программу. Таких редакторов очень много, например PyScripter, PythonWin, UliPad, Eclipse + PyDev, Netbeans и др. Полный список редакторов расположен на странице <http://wiki.python.org/moin/PythonEditors>. В этой книге мы будем пользоваться редактором IDLE, который входит в состав стандартной библиотеки Python в Windows.

Во второй части книги мы рассмотрим библиотеку PyQt, позволяющую создавать кросс-платформенные приложения с графическим интерфейсом. Библиотека очень проста в использовании и идеально подходит для разработки оконных приложений практически любой сложности. В состав библиотеки входит программа Qt Designer, с помощью которой можно размещать компоненты на форме путем перетаскивания мышью. При сохранении формы Qt Designer создает XML-файл, который можно загрузить внутри программы или автоматически преобразовать в код на языке Python.

Желаю приятного прочтения и надеюсь, что эта книга станет верным спутником в вашей повседневной жизни.



ЧАСТЬ I

Основы Python 3

- Глава 1. Первые шаги
- Глава 2. Переменные
- Глава 3. Операторы
- Глава 4. Условные операторы и циклы
- Глава 5. Числа
- Глава 6. Строки
- Глава 7. Регулярные выражения
- Глава 8. Списки и кортежи
- Глава 9. Словари и множества
- Глава 10. Работа с датой и временем
- Глава 11. Пользовательские функции
- Глава 12. Модули и пакеты
- Глава 13. Объектно-ориентированное программирование
- Глава 14. Обработка исключений
- Глава 15. Работа с файлами и каталогами
- Глава 16. Основы SQLite
- Глава 17. Доступ к базе данных SQLite из Python
- Глава 18. Взаимодействие с Интернетом



ГЛАВА 1

Первые шаги

Прежде чем мы начнем рассматривать синтаксис языка, необходимо сделать два замечания. Во-первых, не забывайте, что книги по программированию нужно не только читать, но и выполнять все примеры, а также экспериментировать, изменяя что-нибудь в примерах. Поэтому, если вы удобно устроились на диване и настроились просто читать, у вас практически нет шансов изучить язык. Во-вторых, помните, что прочитать эту книгу один раз недостаточно. Первую часть книги вы должны знать наизусть! Сколько на это уйдет времени, зависит от ваших способностей и желаний. 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.

1. Для загрузки дистрибутива переходим на страницу <http://python.org/download/> и скачиваем файл `python-3.2.msi`. Затем запускаем программу установки с помощью двойного щелчка на значке файла.
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) предлагается выбрать каталог для установки. Оставляем каталог по умолчанию (`C:\Python32\`) 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 программы можно с помощью двойного щелчка мышью на значке файла. Оставляем выбранными все компоненты и нажимаем кнопку **Next**.
5. После завершения установки будет выведено окно, изображенное на рис. 1.4. Нажимаем кнопку **Finish** для выхода из программы установки.

В результате установки исходные файлы интерпретатора будут скопированы в папку `C:\Python32`. В этой папке расположены два исполняемых файла: `python.exe` и `pythonw.exe`. Файл `python.exe` предназначен для выполнения консольных приложений. Именно эта программа запускается при двойном щелчке на значке файла с расширением `py`. Файл `pythonw.exe` используется для запуска оконных приложений. В этом случае окно консоли выводиться не будет. Эта программа запускается при двойном щелчке на значке файла с расширением `pyw`.



Рис. 1.1. Установка Python. Шаг 1

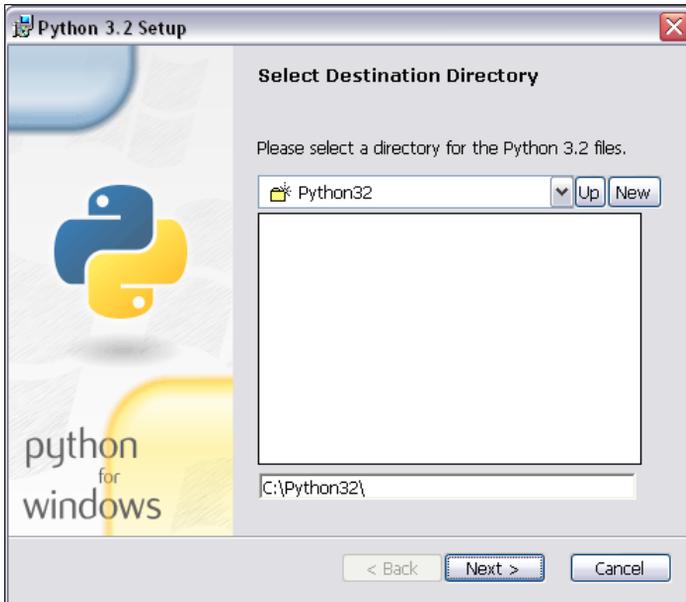


Рис. 1.2. Установка Python. Шаг 2

Если сделать двойной щелчок на файле `python.exe`, то запустится интерактивная оболочка 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 (command line)** в меню **Пуск | Программы | Python 3.2**.



Рис. 1.3. Установка Python. Шаг 3



Рис. 1.4. Установка Python. Шаг 4

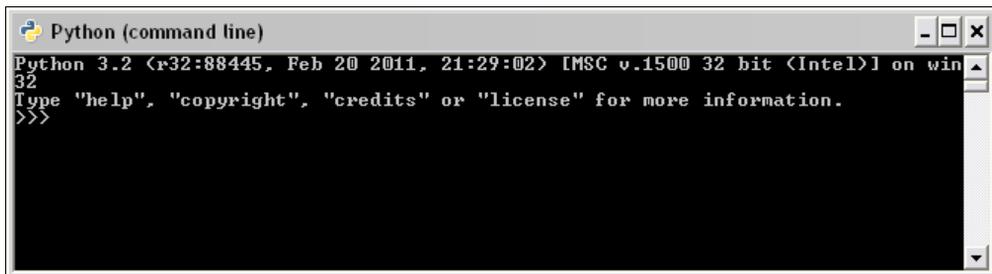


Рис. 1.5. Интерактивная оболочка

Вместо интерактивной оболочки для изучения языка, а также создания и редактирования файлов с программой, лучше воспользоваться редактором IDLE, который входит в состав установленных компонентов. Для запуска редактора в меню **Пуск | Программы | Python 3.2** выбираем пункт **IDLE (Python GUI)**. 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 на протяжении всей книги. Более подробно редактор IDLE мы рассмотрим немного позже.

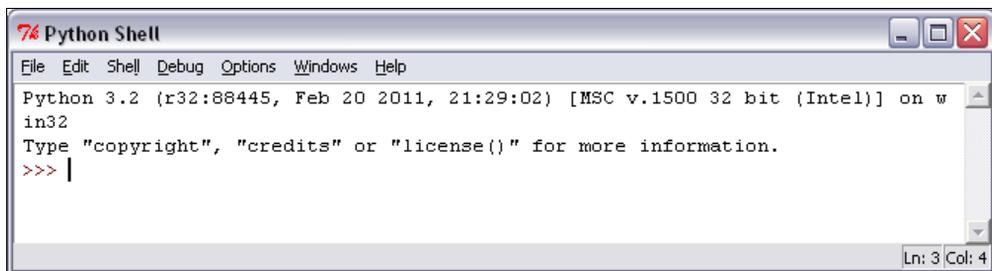


Рис. 1.6. Окно Python Shell редактора IDLE

Версии языка Python выпускаются с завидной регулярностью, но, к сожалению, сторонние разработчики не успевают за такой скоростью и не так часто обновляют свои модули. Поэтому приходится при наличии версии Python 3 использовать на практике версию Python 2. Как же быть, если установлена версия 3.2, а необходимо запустить модуль для версии 2.7 или 2.5? В этом случае удалять версию 3.2 с компьютера не нужно. Все программы установки позволяют выбрать устанавливаемые компоненты. Так, например, существует возможность задать ассоциацию с файловым расширением. И вот этот компонент необходимо просто отключить при установке.

В качестве примера мы дополнительно установим на компьютер версию 3.1, но вместо программы установки с сайта <http://python.org/> выберем альтернативный дистрибутив от компании ActiveState. Переходим на страницу <http://www.activestate.com/activepython/downloads/> и скачиваем дистрибутив. Последовательность запуска нескольких программ установки от компании ActiveState имеет значение, т. к. в контекстное меню добавляется пункт **Edit with Pythonwin**. С помощью этого пункта запускается редактор PythonWin, который можно использовать вместо IDLE. Так вот из контекстного меню будет открываться версия PythonWin, которая была установлена последней. Установку программы производим в каталог по умолчанию (C:\Python31). Обратите особое 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 текущей версией будет не Python 3.2.

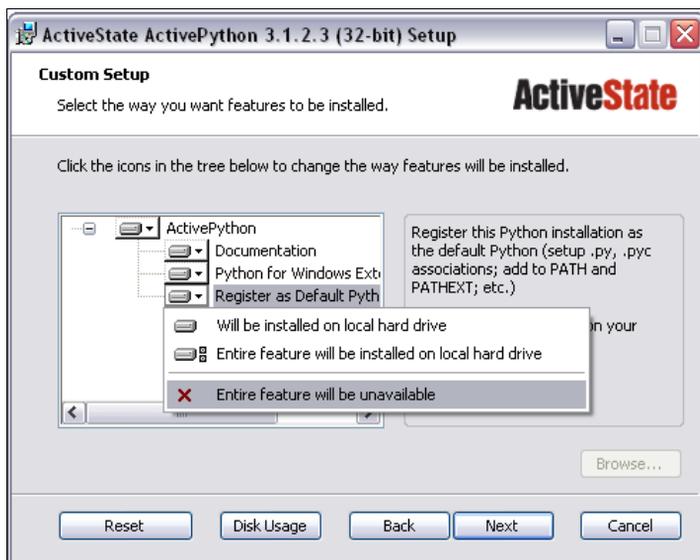


Рис. 1.7. Окно Custom Setup

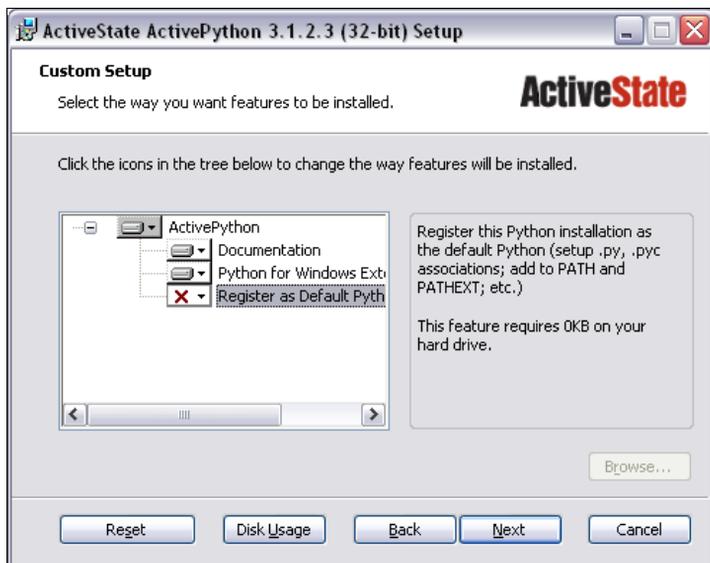


Рис. 1.8. Компонент Register as Default Python отключен

В состав ActivePython кроме редактора PythonWin входит также редактор IDLE. Однако ни в одном меню нет пункта, с помощью которого можно запустить редактор IDLE. Чтобы это исправить, создадим файл IDLE31.bat. Содержимое файла:

```
@echo off
start C:\Python31\pythonw.exe C:\Python31\Lib\idlelib\idle.pyw
```

С помощью двойного щелчка на этом файле можно запускать IDLE для версии Python 3.1. Чтобы запустить редактор для версии Python 3.2, в меню Пуск | Программы | Python 3.2 выбираем пункт **IDLE (Python GUI)**.

Теперь рассмотрим запуск программы с помощью разных версий Python. По умолчанию при двойном щелчке на значке файла запускается Python 3.2. Чтобы запустить с помощью другой версии, щелкаем правой кнопкой мыши на значке файла с программой и в контекстном меню выбираем пункт **Открыть с помощью**. При первом запуске в списке будет только программа `python.exe`. Чтобы добавить другую версию, щелкаем на пункте **Выбрать программу**, в открывшемся окне нажимаем кнопку **Обзор** и выбираем программу `python31.exe` из папки `C:\Python31`. В результате список в контекстном меню будет выглядеть так, как показано на рис. 1.9. Выбирая нужную версию из списка, можно производить запуск программы с помощью разных версий Python.

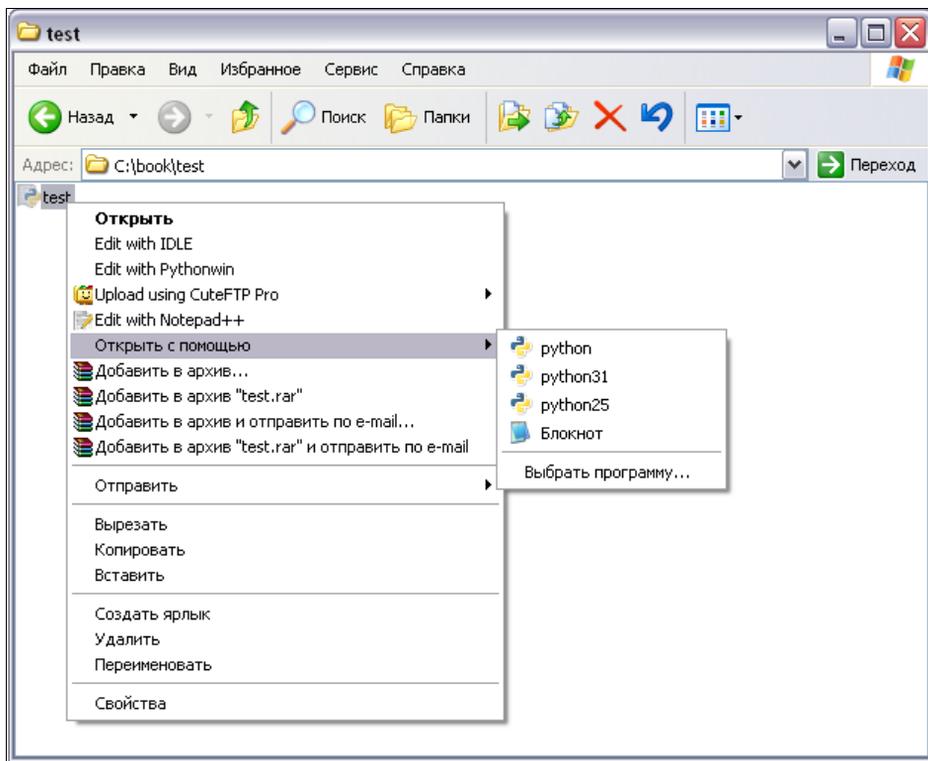


Рис. 1.9. Варианты запуска программы разными версиями Python

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, 2, 0, 'final', 0)`, то установка прошла нормально, а если `(3, 1, 2, 'final', 0)`, то вы не отключили компонент **Register as Default Python**.

Для изучения материала этой книги по умолчанию должна запускаться версия Python 3.2.

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** выбираем пункт **Программы | Python 3.2 | IDLE (Python GUI)**. 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 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 Window**. В открывшемся окне набираем код из листинга 1.2, а затем сохраняем его под именем `test.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 будет ожидать нажатия клавиши <Enter> и не позволит окну сразу закрыться. С учетом сказанного ранее 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 будет закрыта, и вы не сможете прочитать сообщение об ошибке. Если подобная ситуация возникла, то 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, щелкаем правой кнопкой мыши на значке файла и из контекстного меню выбираем пункт **Edit with IDLE**. Так как программа на языке Python представляет собой обычный текстовый файл, сохраненный с расширением `.py`, его можно редактировать с помощью других программ, например Notepad++. Можно также воспользоваться специализированными редакторами, скажем, PyScripter.

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 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 был установлен бит на выполнение в правах доступа к файлу. 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. 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, то 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. Таким образом 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 будет 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 нижеприведенными способами:

- ◆ 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. Это лучший способ, т. к. любое выражение можно разместить внутри круглых скобок. Пример:

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

- ◆ определение списка и словаря можно разместить на нескольких строках, т. к. 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 скрипта, и интерпретатор полностью их игнорирует. Внутри комментария может располагаться любой текст, включая инструкции, которые выполнять не следует. 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 *одноточный комментарий*. Он начинается с символа `#`:

```
# Это комментарий
```

Одноточный комментарий может начинаться не только с начала строки, но и располагаться после инструкции. Например, после инструкции вывести надпись "Привет, мир!":

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

Если 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, т. к. все инструкции будут считаться простым текстом. Такие строки являются строками документирования, а не комментариями.

1.5. Скрытые возможности IDLE

На всем протяжении этой книги в качестве редактора мы будем использовать IDLE. По этой причине 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([<Объекты>][, sep=' '][, end='\n'][, file=sys.stdout])
```

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. 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. 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
```

В этом примере 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. Таким образом в языке Python выделяются *блоки*. Инструкции, перед которыми расположено одинаковое количество пробелов, являются *телом 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.