

Николай Прохоренок  
Владимир Дронов

**PRO**

**ПРОФЕССИОНАЛЬНОЕ  
ПРОГРАММИРОВАНИЕ**

# Python 3 и PyQt 6

## Разработка приложений

Типы данных Python

Объектно-ориентированное  
программирование

Работа с файлами и каталогами

Взаимодействие с Windows

Создание оконных программ

Работа с базами данных

Мультимедиа

Запись звука, видео и фото

Печать и экспорт в формат PDF

Работающий пример:  
приложение «Судoku»



Материалы  
на [www.bhv.ru](http://www.bhv.ru)



УДК 004.43  
ББК 32.973.26-018.1  
П84

**Прохоренок, Н. А.**

П84 Python 3 и PyQt 6. Разработка приложений / Н. А. Прохоренок,  
В. А. Дронов. — СПб.: БХВ-Петербург, 2023. — 832 с.: ил. —  
(Профессиональное программирование)

ISBN 978-5-9775-1706-5

Описан язык Python 3: типы данных, операторы, условия ветвления и выбора, циклы, регулярные выражения, функции, классы, работа с файлами и каталогами, взаимодействие с механизмами Windows, часто используемые модули стандартной библиотеки. Особое внимание уделено библиотеке PyQt, позволяющей создавать приложения с графическим интерфейсом. Описаны средства для создания и вывода окон, основных компонентов (кнопок, полей, списков, таблиц, меню, панелей инструментов и др.). Рассмотрена обработка событий и сигналов, разработка многопоточных программ, работа с базами данных, вывод графики, воспроизведение мультимедиа, запись аудио, видео и фото, печать документов, экспорт их в формат Adobe PDF и сохранения настроек программ. Дан пример полнофункционального приложения для создания и решения головоломок судоку. На сайте издательства размещен электронный архив со всеми примерами из книги.

*Для программистов*

УДК 004.43  
ББК 32.973.26-018.1

**Группа подготовки издания:**

Руководитель проекта	<i>Евгений Рыбаков</i>
Зав. редакцией	<i>Людмила Гауль</i>
Редактор	<i>Григорий Добин</i>
Компьютерная верстка	<i>Ольги Сергиенко</i>
Дизайн серии	<i>Марины Дамбиевой</i>
Оформление обложки	<i>Зои Канторович</i>

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

ISBN 978-5-9775-1706-5

© ООО "БХВ", 2023  
© Оформление. ООО "БХВ-Петербург", 2023

# Оглавление

<b>Предисловие .....</b>	<b>15</b>
Python.....	15
PyQt.....	16
Использованное ПО.....	16
Типографские соглашения.....	16
 <b>ЧАСТЬ I. ОСНОВЫ ЯЗЫКА PYTHON.....</b>	 <b>19</b>
 <b>Глава 1. Первые шаги .....</b>	 <b>21</b>
1.1. Установка Python .....	21
1.2. Интерактивный режим Python. Утилита IDLE .....	24
1.3. Введение в Python-программирование .....	25
1.4. Принципы написания Python-программ .....	27
1.4.1. Комментарии и строки документирования .....	30
1.4.2. Кодировки, поддерживаемые Python .....	31
1.4.3. Подготовка Python-программ для выполнения в UNIX .....	31
1.5. Дополнительные возможности IDLE .....	32
1.6. Вывод данных .....	33
1.7. Ввод данных.....	35
1.8. Утилита <i>pip</i> : установка дополнительных библиотек .....	37
1.9. Доступ к документации.....	42
1.10. Компиляция Python-файлов .....	44
 <b>Глава 2. Переменные и типы данных .....</b>	 <b>46</b>
2.1. Переменные.....	46
2.2. Типы данных. Понятие объекта и ссылки .....	47
2.3. Присваивание значений переменным .....	50
2.4. Проверка типа данных.....	52
2.5. Преобразование типов данных .....	53
2.6. Удаление переменных .....	56
 <b>Глава 3. Операторы .....</b>	 <b>57</b>
3.1. Математические операторы.....	57
3.2. Двоичные операторы.....	59

3.3. Операторы для работы с последовательностями .....	60
3.4. Операторы присваивания .....	61
3.5. Пустой оператор .....	63
3.6. Приоритет операторов .....	63
<b>Глава 4. Инструкции ветвления, выбора и циклы .....</b>	<b>65</b>
4.1. Операторы сравнения .....	66
4.2. Инструкция ветвления .....	68
4.3. Инструкция выбора .....	71
4.4. Цикл перебора последовательности .....	77
4.5. Цикл с условием .....	78
4.6. Оператор <i>continue</i> : переход на следующую итерацию цикла .....	79
4.7. Оператор <i>break</i> : прерывание цикла .....	79
4.8. Оператор присваивания в составе инструкции .....	80
<b>Глава 5. Числа .....</b>	<b>82</b>
5.1. Запись чисел .....	82
5.2. Обработка чисел .....	84
5.3. Математические функции .....	87
5.4. Генерирование случайных чисел .....	89
<b>Глава 6. Строки и двоичные данные .....</b>	<b>92</b>
6.1. Создание строк .....	92
6.1.1. Специальные символы .....	94
6.1.2. Необрабатываемые строки .....	95
6.2. Операции над строками .....	96
6.3. Форматирование строк .....	98
6.4. Метод <i>format()</i> .....	104
6.4.1. Форматируемые строки .....	108
6.5. Функции и методы для работы со строками .....	109
6.6. Настройка локали .....	112
6.7. Изменение регистра символов .....	113
6.8. Функции для работы с символами .....	114
6.9. Поиск и замена в строке .....	114
6.10. Проверка содержимого строки .....	118
6.11. Двоичные данные типа <i>bytes</i> .....	121
6.12. Двоичные данные типа <i>bytearray</i> .....	125
6.13. Сериализация и десериализация значений .....	128
6.14. Хеширование значений .....	129
<b>Глава 7. Регулярные выражения .....</b>	<b>131</b>
7.1. Синтаксис регулярных выражений .....	131
7.2. Поиск первого совпадения с шаблоном .....	140
7.3. Поиск всех совпадений с шаблоном .....	145
7.4. Замена в строке .....	146
7.5. Прочие функции и методы .....	148
<b>Глава 8. Списки, кортежи, множества и диапазоны .....</b>	<b>150</b>
8.1. Создание списков .....	150
8.2. Операции над списками .....	152

8.3. Многомерные списки .....	155
8.4. Перебор списков .....	155
8.5. Генераторы списков и выражения-генераторы .....	156
8.6. Функции <i>map()</i> , <i>zip()</i> , <i>filter()</i> и <i>reduce()</i> .....	158
8.7. Добавление и удаление элементов списка .....	161
8.8. Поиск элемента в списке и получение сведений об элементах списка .....	163
8.9. Переворачивание и перемешивание списка .....	164
8.10. Выбор элементов списка случайным образом .....	165
8.11. Сортировка списка .....	166
8.12. Заполнение списка числами .....	168
8.13. Преобразование списка в строку .....	168
8.14. Кортежи .....	169
8.15. Множества, изменяемые и неизменяемые .....	170
8.16. Диапазоны .....	175
8.17. Модуль <i>itertools</i> .....	177
8.17.1. Генерирование неопределенного количества значений .....	177
8.17.2. Генерирование комбинаций значений .....	178
8.17.3. Фильтрация элементов последовательности .....	180
8.17.4. Прочие функции .....	181
<b>Глава 9. Словари</b> .....	<b>185</b>
9.1. Создание словаря .....	185
9.2. Операции над словарями .....	187
9.3. Перебор элементов словаря .....	189
9.4. Методы и функции для работы со словарями .....	190
9.5. Генераторы словарей .....	193
<b>Глава 10. Работа с датой и временем</b> .....	<b>194</b>
10.1. Получение текущих даты и времени .....	194
10.2. Форматирование даты и времени .....	196
10.3. Приостановка выполнения программы .....	198
10.4. Значения даты и времени .....	198
10.4.1. Временные промежутки .....	198
10.4.2. Значения даты .....	201
10.4.3. Значения времени .....	204
10.4.4. Временные отметки .....	207
10.5. Вывод календаря .....	213
10.5.1. Вывод календаря в текстовом виде .....	213
10.5.2. Вывод календаря в формате HTML .....	215
10.5.3. Другие полезные функции .....	217
10.6. Измерение времени выполнения фрагментов кода .....	220
<b>Глава 11. Функции</b> .....	<b>222</b>
11.1. Определение и вызов функции .....	222
11.1.1. Расположение определений функций .....	224
11.1.2. Локальные и глобальные переменные .....	225
11.1.3. Позиционные и именованные параметры .....	228
11.1.4. Необязательные параметры .....	230

11.1.5. Произвольное количество параметров .....	231
11.1.6. Распаковка последовательностей и отображений .....	233
11.6.7. Функция как значение. Функции обратного вызова.....	233
11.2. Анонимные функции .....	234
11.3. Функции-генераторы .....	236
11.4. Декораторы функций.....	237
11.5. Рекурсия .....	239
11.6. Вложенные функции .....	240
11.7. Аннотации функций .....	242
<b>Глава 12. Модули, пакеты и импорт .....</b>	<b>243</b>
12.1. Импорт модуля целиком .....	243
12.2. Импорт отдельных идентификаторов .....	246
12.2.1. Указание идентификаторов, доступных для импорта .....	248
12.2.2. Управление доступом к идентификаторам .....	248
12.3. Пути поиска модулей .....	249
12.4. Перегрузка модулей .....	251
12.5. Пакеты .....	252
<b>Глава 13. Объекты и классы.....</b>	<b>256</b>
13.1. Определение классов, создание объектов и работа с ними.....	256
13.2. Атрибуты класса .....	259
13.3. Конструкторы и деструкторы .....	260
13.4. Наследование .....	261
13.4.1. Множественное наследование.....	263
13.4.1.1. Примеси и их использование .....	265
13.5. Специальные методы.....	266
13.6. Перегрузка операторов.....	269
13.7. Статические методы и методы класса .....	271
13.8. Абстрактные методы .....	272
13.9. Закрытые атрибуты и методы .....	273
13.10. Свойства .....	274
13.11. Декораторы классов .....	276
<b>Глава 14. Исключения и их обработка.....</b>	<b>278</b>
14.1. Обработчики исключений.....	279
14.2. Обработчики контекстов.....	283
14.3. Классы встроенных исключений.....	285
14.4. Генерирование исключений.....	287
14.5. Пользовательские исключения .....	289
14.6. Проверочная инструкция .....	290
<b>Глава 15. Итераторы, контейнеры и перечисления.....</b>	<b>292</b>
15.1. Итераторы .....	292
15.2. Контейнеры .....	293
15.2.1. Контейнеры-последовательности.....	293
15.2.2. Контейнеры-отображения.....	295
15.3. Перечисления .....	296

<b>Глава 16. Работа с файлами и каталогами.....</b>	<b>302</b>
16.1. Открытие файлов.....	302
16.1.1. Указание путей к файлам и каталогам.....	305
16.1.2. Текущий рабочий каталог.....	306
16.2. Чтение и запись данных: объектные инструменты.....	307
16.3. Чтение и запись данных: низкоуровневые инструменты.....	313
16.4. Файлы в памяти.....	315
16.5. Задание прав доступа к файлам и каталогам.....	319
16.6. Работа с файлами.....	321
16.7. Работа с путями.....	325
16.8. Перенаправление ввода/вывода.....	326
16.9. Сохранение объектов в файлах.....	328
16.10. Работа с каталогами.....	331
16.10.1. Функция <i>scandir()</i> .....	335
16.11. Исключения, генерируемые файловыми операциями.....	337
<b>Глава 17. Работа с механизмами Windows .....</b>	<b>338</b>
17.1. Работа с реестром.....	338
17.1.1. Открытие и закрытие ветвей реестра.....	339
17.1.2. Чтение и запись данных реестра.....	340
17.1.3. Перебор элементов и вложенных ветвей реестра.....	343
17.2. Получение путей к системным каталогам.....	344
17.3. Создание ярлыков.....	345
<b>ЧАСТЬ II. БИБЛИОТЕКА PYQT 6.....</b>	<b>347</b>
<b>Глава 18. Введение в PyQt 6.....</b>	<b>349</b>
18.1. Установка PyQt 6.....	349
18.2. Первая оконная программа.....	349
18.3. Структура PyQt-программы.....	350
18.4. ООП-стиль создания окна.....	352
18.5. Создание окон с помощью программы Qt Designer.....	356
18.5.1. Создание окон.....	356
18.5.2. Использование UI-файла в программе.....	359
18.5.3. Преобразование UI-файла в модуль Python.....	361
18.6. Модули PyQt 6.....	362
18.7. Управление циклом обработки событий.....	363
18.8. Многопоточные программы.....	365
18.8.1. Потоки.....	365
18.8.2. Управление потоками.....	368
18.8.3. Очереди.....	372
18.8.4. Блокировщики и автоблокировщики.....	376
18.9. Вывод заставки.....	379
<b>Глава 19. Окна .....</b>	<b>382</b>
19.1. Создание и вывод окон.....	382
19.1.1. Типы окон.....	383
19.2. Размеры окон и управление ими.....	384

19.3. Местоположение окна и управление им .....	387
19.4. Классы, задающие координаты и размеры .....	390
19.4.1. Класс <i>QPoint</i> : координаты точки .....	390
19.4.2. Класс <i>QSize</i> : размеры прямоугольной области .....	391
19.4.3. Класс <i>QRect</i> : координаты и размеры прямоугольной области .....	393
19.5. Разворачивание и сворачивание окон .....	399
19.6. Управление прозрачностью окна .....	401
19.7. Модальные окна .....	401
19.8. Смена значка окна .....	403
19.9. Изменение цвета фона окна .....	404
19.10. Вывод изображения в качестве фона .....	405
19.11. Окна произвольной формы .....	406
19.12. Всплывающие и расширенные подсказки .....	408
19.13. Программное закрытие окна .....	409
19.14. Использование таблиц стилей CSS для оформления окон .....	409
<b>Глава 20. Обработка сигналов и событий .....</b>	<b>414</b>
20.1. Назначение обработчиков сигналов .....	414
20.1.1. Слоты .....	417
20.1.2. Передача данных в обработчик сигнала .....	418
20.2. Блокировка и удаление обработчиков сигналов .....	419
20.3. Генерирование сигналов .....	421
20.4. Пользовательские сигналы .....	421
20.5. Использование таймеров .....	423
20.6. Обработка всех событий .....	426
20.7. События окна .....	429
20.7.1. Изменение состояния окна .....	429
20.7.2. Изменение местоположения и размеров окна .....	430
20.7.3. Перерисовка окна или его части .....	431
20.7.4. Предотвращение закрытия окна .....	431
20.8. События клавиатуры .....	432
20.8.1. Управление фокусом ввода .....	432
20.8.2. Назначение клавиш быстрого доступа .....	435
20.8.3. Нажатие и отпускание клавиш .....	437
20.9. События мыши .....	439
20.9.1. Нажатие и отпускание кнопок мыши .....	439
20.9.2. Перемещение курсора мыши .....	440
20.9.3. Наведение и увод курсора мыши .....	441
20.9.4. Прокрутка колесика мыши .....	441
20.9.5. Изменение курсора мыши .....	442
20.10. Операция перетаскивания (drag & drop) .....	443
20.10.1. Запуск перетаскивания .....	443
20.10.1.1. Задание перетаскиваемых данных .....	445
20.10.2. Обработка перетаскивания и сброса .....	446
20.11. Работа с буфером обмена .....	448
20.12. Фильтрация событий .....	449
20.13. Генерирование событий .....	450
20.14. Пользовательские события .....	450



<b>Глава 21. Размещение компонентов в окнах. Контейнеры .....</b>	<b>451</b>
21.1. Абсолютное позиционирование .....	451
21.2. Контейнеры-стопки .....	452
21.3. Контейнер-сетка .....	455
21.4. Контейнер-форма .....	457
21.5. Стеки .....	460
21.6. Управление размерами компонентов .....	461
21.7. Группа .....	462
21.8. Панель с рамкой .....	464
21.9. Панель с вкладками .....	465
21.10. Аккордеон .....	469
21.11. Панель с изменяемыми областями .....	471
21.12. Прокручиваемая панель .....	473
<b>Глава 22. Основные компоненты .....</b>	<b>475</b>
22.1. Надпись .....	475
22.2. Кнопка .....	477
22.3. Переключатель .....	479
22.4. Флажок .....	480
22.5. Поле ввода .....	480
22.5.1. Основные методы и сигналы .....	481
22.5.2. Ввод данных по маске .....	484
22.5.3. Контроль ввода с помощью валидаторов .....	485
22.6. Область редактирования .....	486
22.6.1. Основные методы и сигналы .....	486
22.6.2. Задание параметров области редактирования .....	488
22.6.3. Указание параметров текста и фона .....	490
22.6.4. Класс <i>QTextDocument</i> .....	491
22.6.5. Класс <i>QTextCursor</i> .....	494
22.7. Текстовый браузер .....	497
22.8. Поля для ввода целых и вещественных чисел .....	499
22.9. Поля для ввода даты и времени .....	501
22.10. Календарь .....	504
22.11. Семисегментный индикатор .....	506
22.12. Индикатор процесса .....	507
22.13. Шкала с ползунком .....	508
22.14. Круговая шкала с ползунком .....	510
22.15. Полоса прокрутки .....	511
22.16. Веб-браузер .....	511
<b>Глава 23. Списки и таблицы .....</b>	<b>516</b>
23.1. Раскрывающийся список .....	516
23.1.1. Добавление, изменение и удаление элементов .....	516
23.1.2. Изменение параметров списка .....	517
23.1.3. Поиск элементов .....	518
23.1.4. Сигналы .....	519
23.2. Список для выбора шрифта .....	519
23.3. Роли элементов .....	520

23.4. Модели.....	521
23.4.1. Доступ к данным внутри модели .....	521
23.4.2. Класс <i>QStringListModel</i> .....	523
22.4.3. Класс <i>QStandardItemModel</i> .....	524
23.4.4. Класс <i>QStandardItem</i> .....	528
23.5. Представления .....	531
23.5.1. Класс <i>QAbstractItemView</i> .....	532
23.5.2. Простой список.....	535
23.5.3. Таблица.....	537
23.5.4. Иерархический список .....	539
23.5.5. Управление заголовками строк и столбцов.....	541
23.6. Управление выделением элементов.....	544
22.7. Промежуточные модели.....	546
23.8. Использование делегатов.....	547
<b>Глава 24. Работа с базами данных .....</b>	<b>551</b>
24.1. Соединение с базой данных .....	551
24.2. Получение сведений о структуре таблиц.....	554
24.2.1. Получение сведений о таблицах.....	554
24.2.2. Получение сведений о полях таблиц .....	555
24.2.3. Получение сведений о ключевом индексе.....	556
24.2.4. Получение сведений об ошибке .....	556
24.3. Выполнение SQL-запросов и получение их результатов .....	557
24.3.1. Выполнение запросов .....	557
24.3.2. Обработка результатов выполнения запросов .....	560
24.3.3. Очистка запроса.....	562
24.3.4. Получение служебных сведений о запросе .....	563
24.4. Модели, связанные с данными .....	563
24.4.1. Модель, связанная с SQL-запросом .....	563
24.4.2. Модель, связанная с таблицей.....	565
24.4.3. Модель, поддерживающая межтабличные связи.....	570
24.4.4. Использование связанных делегатов .....	573
<b>Глава 25. Работа с графикой .....</b>	<b>575</b>
25.1. Вспомогательные классы .....	575
25.1.1. Класс <i>QColor</i> : цвет .....	575
25.1.2. Класс <i>QPen</i> : перо.....	579
25.1.3. Класс <i>QBrush</i> : кисть .....	580
25.1.4. Класс <i>QLine</i> : линия.....	581
24.1.5. Класс <i>QPolygon</i> : многоугольник .....	582
25.1.6. Класс <i>QFont</i> : шрифт.....	584
25.2. Класс <i>QPainter</i> .....	586
25.2.1. Рисование линий и фигур .....	587
25.2.2. Вывод текста.....	589
25.2.3. Вывод изображений .....	590
25.2.4. Преобразование систем координат .....	592
25.2.5. Сохранение команд рисования в файл.....	593
25.3. Работа с растровыми изображениями.....	594
25.3.1. Класс <i>QPixmap</i> .....	594
25.3.2. Класс <i>QBitmap</i> .....	597

25.3.3. Класс <i>QImage</i> .....	598
25.3.4. Класс <i>QIcon</i> .....	601
<b>Глава 26. Графическая сцена.....</b>	<b>603</b>
26.1. Графическая сцена.....	603
26.1.1. Настройка графической сцены .....	603
26.1.2. Добавление и удаление графических объектов .....	604
26.1.3. Добавление компонентов на сцену .....	605
26.1.4. Поиск графических объектов .....	605
26.1.5. Управление фокусом ввода .....	607
26.1.6. Управление выделением объектов.....	607
26.1.7. Прочие методы и сигналы .....	608
26.2. Графическое представление .....	609
26.2.1. Настройка графического представления .....	609
26.2.2. Преобразования между координатами представления и сцены .....	611
26.2.3. Поиск объектов.....	612
26.2.4. Преобразование системы координат .....	612
26.2.5. Прочие методы .....	613
26.3. Графические объекты.....	614
26.3.1. Класс <i>QGraphicsItem</i> : базовый класс для графических объектов .....	614
26.3.1.1. Настройка графического объекта.....	614
26.3.1.2. Выполнение преобразований.....	616
26.3.1.3. Прочие методы .....	617
26.3.2. Готовые графические объекты .....	618
26.3.2.1. Линия .....	618
26.3.2.2. Класс <i>QAbstractGraphicsShapeItem</i> .....	618
26.3.2.3. Прямоугольник .....	619
26.3.2.4. Многоугольник .....	619
26.3.2.5. Эллипс .....	619
26.3.2.6. Изображение .....	620
26.3.2.7. Простой текст.....	621
26.3.2.8. Форматированный текст .....	621
26.4. Группировка объектов.....	622
26.5. Эффекты .....	623
26.5.1. Класс <i>QGraphicsEffect</i> .....	623
26.5.2. Тень.....	624
26.5.3. Размытие .....	625
26.5.4. Изменение цвета .....	625
26.5.5. Изменение прозрачности .....	625
26.6. Обработка событий.....	626
26.6.1. События клавиатуры .....	626
26.6.2. События мыши .....	627
26.6.3. Обработка перетаскивания и сброса.....	630
26.6.4. Фильтрация событий.....	631
26.6.5. Обработка изменения состояния объекта.....	631
<b>Глава 27. Диалоговые окна .....</b>	<b>634</b>
27.1. Пользовательские диалоговые окна .....	634
27.2. Класс <i>QDialogButtonBox</i> .....	636

27.3. Класс <i>QMessageBox</i> .....	639
27.3.1. Основные методы и сигналы .....	640
27.3.2. Окно информационного сообщения .....	642
27.3.3. Окно подтверждения .....	643
27.3.4. Окно предупреждающего сообщения .....	644
27.3.5. Окно критического сообщения .....	644
27.3.6. Окно сведений о программе .....	645
27.3.7. Окно сведений о фреймворке Qt .....	645
27.4. Класс <i>QInputDialog</i> .....	646
27.4.1. Основные методы и сигналы .....	647
27.4.2. Окно для ввода строки .....	649
27.4.3. Окно для ввода целого числа .....	649
27.4.4. Окно для ввода вещественного числа .....	650
27.4.5. Окно для выбора пункта из списка .....	651
27.4.6. Окно для ввода большого текста .....	651
27.5. Класс <i>QFileDialog</i> .....	652
27.5.1. Основные методы и сигналы .....	653
27.5.2. Окно для выбора каталога .....	655
27.5.3. Окно для открытия файлов .....	656
27.5.4. Окно для сохранения файла .....	658
27.6. Окно для выбора цвета .....	659
27.7. Окно для выбора шрифта .....	660
27.8. Окно для вывода сообщения об ошибке .....	661
27.9. Окно с индикатором хода процесса .....	662
27.10. Создание многостраничного мастера .....	663
27.10.1. Класс <i>QWizard</i> .....	663
27.10.2. Класс <i>QWizardPage</i> .....	667
<b>Глава 28. Создание SDI- и MDI-программ .....</b>	<b>670</b>
28.1. Главное окно программы .....	670
28.2. Меню и действия .....	675
28.2.1. Класс <i>QMenuBar</i> .....	675
28.2.2. Класс <i>QMenu</i> .....	676
28.2.3. Контекстное меню компонента .....	679
28.2.4. Класс <i>QAction</i> .....	680
28.2.5. Объединение действий-переключателей в группу .....	683
28.3. Панели инструментов .....	684
28.3.1. Класс <i>QToolBar</i> .....	685
28.3.2. Класс <i>QToolButton</i> .....	686
28.4. Прикрепляемые панели .....	688
28.5. Строка состояния .....	689
28.6. MDI-программы .....	690
28.6.1. Класс <i>QMdiArea</i> .....	690
28.6.2. Класс <i>QMdiSubWindow</i> .....	693
28.7. Добавление значка программы в область уведомлений .....	694
<b>Глава 29. Мультимедиа .....</b>	<b>696</b>
29.1. Воспроизведение мультимедиа .....	696
29.1.1. Мультимедийный проигрыватель .....	696
29.1.2. Звуковой выход. Воспроизведение звука .....	699

29.1.3. Метаданные мультимедийного источника .....	704
29.1.4. Видеопанель. Воспроизведение видео .....	707
29.2. Запись мультимедиа .....	709
29.2.1. Транспорт.....	709
29.2.2. Звуковой вход .....	710
29.2.3. Кодировщик звука и видео .....	710
29.2.4. Указание форматов кодирования. Запись звука .....	712
29.2.5. Камера. Запись видео .....	716
29.2.6. Кодировщик статичных изображений. Захват фото.....	720
29.3. Воспроизведение звуковых эффектов.....	722
<b>Глава 30. Печать документов .....</b>	<b>726</b>
30.1. Основные средства печати.....	726
30.1.1. Класс <i>QPrinter</i> .....	726
30.1.2. Вывод на печать.....	729
30.1.3. Служебные классы .....	735
30.1.3.1. Класс <i>QPageSize</i> .....	735
30.1.3.2. Класс <i>QPageLayout</i> .....	737
30.2. Задание параметров принтера и страницы .....	739
30.2.1. Класс <i>QPrintDialog</i> .....	739
30.2.2. Класс <i>QPageSetupDialog</i> .....	740
30.3. Предварительный просмотр .....	742
30.3.1. Класс <i>QPrintPreviewDialog</i> .....	742
30.3.2. Класс <i>QPrintPreviewWidget</i> .....	745
30.4. Класс <i>QPrinterInfo</i> : получение сведений об устройстве печати .....	747
30.5. Класс <i>QPdfWriter</i> : экспорт в формат PDF .....	749
<b>Глава 31. Сохранение настроек программ .....</b>	<b>752</b>
31.1. Создание объекта класса <i>QSettings</i> .....	752
31.2. Запись и чтение данных .....	753
31.2.1. Базовые средства записи и чтения данных.....	753
31.2.2. Группировка сохраняемых значений. Ключи .....	755
31.2.3. Запись списков.....	757
31.3. Вспомогательные методы класса <i>QSettings</i> .....	759
31.4. Где хранятся настройки?.....	759
<b>Глава 32. Программа «Судоку».....</b>	<b>761</b>
32.1. Правила судоку .....	761
32.2. Описание программы «Судоку» .....	762
32.3. Разработка программы .....	764
32.3.1. Подготовительные действия.....	764
32.3.2. Класс <i>MyLabel</i> : ячейка поля судоку .....	764
32.3.3. Класс <i>Widget</i> : поле судоку .....	768
32.3.3.1. Конструктор класса <i>Widget</i> .....	769
32.3.3.2. Прочие методы класса <i>Widget</i> .....	771
32.3.4. Класс <i>MainWindow</i> : основное окно программы.....	775
32.3.4.1. Конструктор класса <i>MainWindow</i> .....	776
32.3.4.2. Остальные методы класса <i>MainWindow</i> .....	779
32.3.5. Главный модуль.....	779

32.3.6. Копирование и вставка головоломок.....	780
32.3.6.1. Форматы данных .....	780
32.3.6.2. Реализация копирования и вставки в классе <i>Widget</i> .....	780
32.3.6.3. Реализация копирования и вставки в классе <i>MainWindow</i> .....	783
32.3.7. Сохранение и загрузка данных.....	787
32.3.8. Печать и предварительный просмотр.....	789
32.3.8.1. Реализация печати в классе <i>Widget</i> .....	790
32.3.8.2. Класс <i>PreviewDialog</i> : диалоговое окно предварительного просмотра.....	791
32.3.8.3. Реализация печати в классе <i>MainWindow</i> .....	794
<b>Заключение.....</b>	<b>796</b>
<b>Приложение. Описание электронного архива.....</b>	<b>797</b>
<b>Предметный указатель .....</b>	<b>798</b>



# ГЛАВА 1

## Первые шаги

Прежде чем мы начнем знакомство с языком Python, хочется отметить, что книги по программированию нужно не только читать, — весьма желательно выполнять все имеющиеся в них примеры, а также экспериментировать, что-нибудь в этих примерах изменяя. Поэтому, если вы удобно устроились на диване и настроились просто читать, у вас практически нет шансов изучить язык. Чем больше вы будете делать самостоятельно, тем большему научитесь.

### **ВНИМАНИЕ!**

Текущая версия Python поддерживает только Microsoft Windows версий 8.1, 10 и 11. Более старые версии этой операционной системы не поддерживаются.

## 1.1. Установка Python

Вначале необходимо установить на компьютер *интерпретатор* (или *исполняющую среду*) Python — программный пакет, который переводит программы, написанные на этом языке, в представление, «понятное» центральному процессору компьютера. Без интерпретатора ни одну Python-программу запустить не получится.

1. Для загрузки дистрибутива исполняющей среды заходим на страницу <https://www.python.org/downloads/> и в списке доступных версий щелкаем на гиперссылке **Python 3.10.1** (эта версия является наиболее актуальной на момент подготовки книги). На открывшейся странице находим раздел **Files** и щелкаем на гиперссылке **Windows installer (32-bit)** (32-разрядная редакция интерпретатора) или **Windows installer (64-bit)** (его 64-разрядная редакция) — в зависимости от редакции нашей операционной системы. В результате на наш компьютер будет загружен файл `python-3.10.1.exe` или `python-3.10.1-amd64.exe` соответственно. Запускаем загруженный файл двойным щелчком на нем.
2. В открывшемся окне (рис. 1.1) проверяем, установлен ли флажок **Install launcher for all users (recommended)** (Установить утилиту launcher для всех пользователей), устанавливаем флажок **Add Python 3.10 to PATH** (Добавить Python 3.10 в список путей из переменной `PATH`) и нажимаем кнопку **Customize installation** (Настроить установку).

Утилита launcher, поставляемая в составе дистрибутива, предназначена для запуска программ под определенной версией Python, если на компьютере установлены несколько разных версий этого языка. К сожалению, если ее не установить, не получится ассоциировать файлы Python-программ с исполняющей средой Python, в результате чего запускать программы щелчком мыши станет невозможно.



Рис. 1.1. Установка Python: шаг 1

3. В следующем диалоговом окне (рис. 1.2) предлагается выбрать устанавливаемые компоненты. Оставляем установленными все флажки, представляющие эти компоненты, и нажимаем кнопку **Next**.

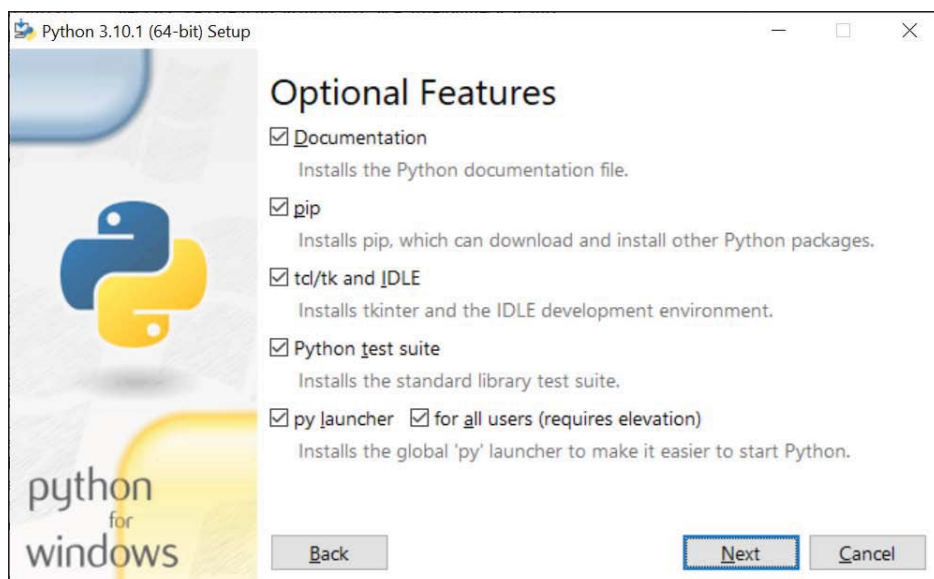


Рис. 1.2. Установка Python: шаг 2

4. На следующем шаге (рис. 1.3) задаем дополнительные настройки и выбираем путь установки. Проверяем, установлены ли флажки **Associate files with Python (requires the py launcher)** (Ассоциировать Python-файлы с исполняющей средой), **Create shortcuts for**



**installed applications** (Создать ярлыки для установленных приложений), **Add Python to environment variables** (Добавить Python в переменные окружения), устанавливаем флажки **Install for all users** (Установить для всех пользователей) и **Precompile standard library** (Предварительно откомпилировать стандартную библиотеку).

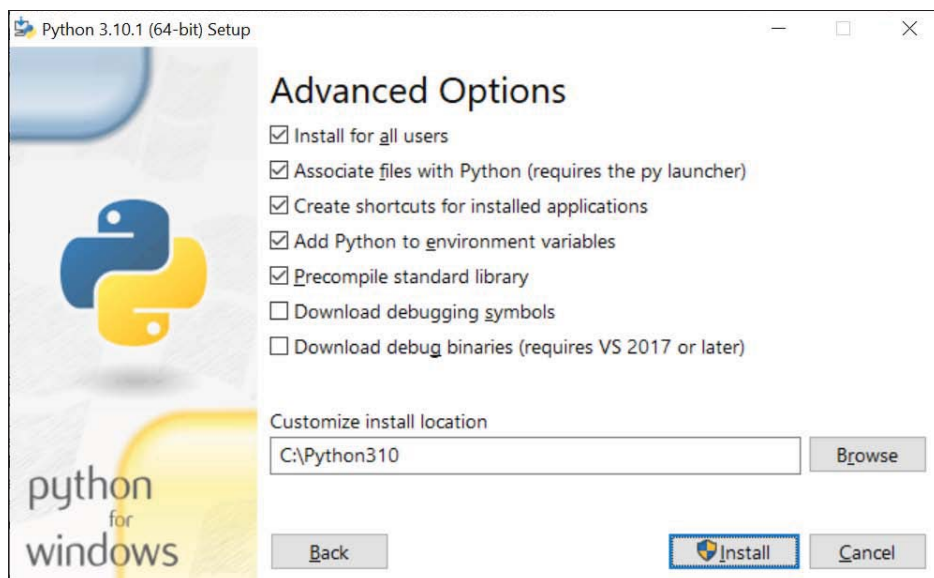


Рис. 1.3. Установка Python: шаг 3

Ассоциирование Python-файлов с исполняющей средой позволит запускать эти файлы щелчками на них. Добавление пути к интерпретатору Python в список путей из переменной PATH даст возможность запускать этот интерпретатор набором в консоли (это сокращенное название командной строки Windows, применяемое далее в книге) команды `python`. Предварительная компиляция стандартной библиотеки ускорит запуск Python-программ.

Обратите внимание на указание пути, по которому будет установлен Python. Изначально предлагается установить интерпретатор по пути `c:\Program Files (x86)\Python310` или `c:\Program Files\Python310`. Однако в этом случае могут возникнуть проблемы при использовании дополнительных библиотек, расширяющих функциональность интерпретатора.

Поэтому авторы книги рекомендуют установить Python по пути `c:\Python310` — т. е. непосредственно в корень диска (см. рис. 1.3). В этом случае никаких проблем при использовании дополнительных библиотек не возникнет.

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

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

В результате на компьютер будут установлены две редакции интерпретатора Python:

- ♦ `python.exe` — предназначена для исполнения консольных программ и задействуется при щелчке мышью на файле с расширением `py`.

С помощью этой редакции можно выполнять и оконные программы, однако в этом случае на экране появится окно консоли, что может обескуражить пользователя;

- ♦ **pythonw.exe** — служит для исполнения оконных программ и задействуется при щелчке мышью на файле с расширением **pyw**. Консоль при этом не выводится.

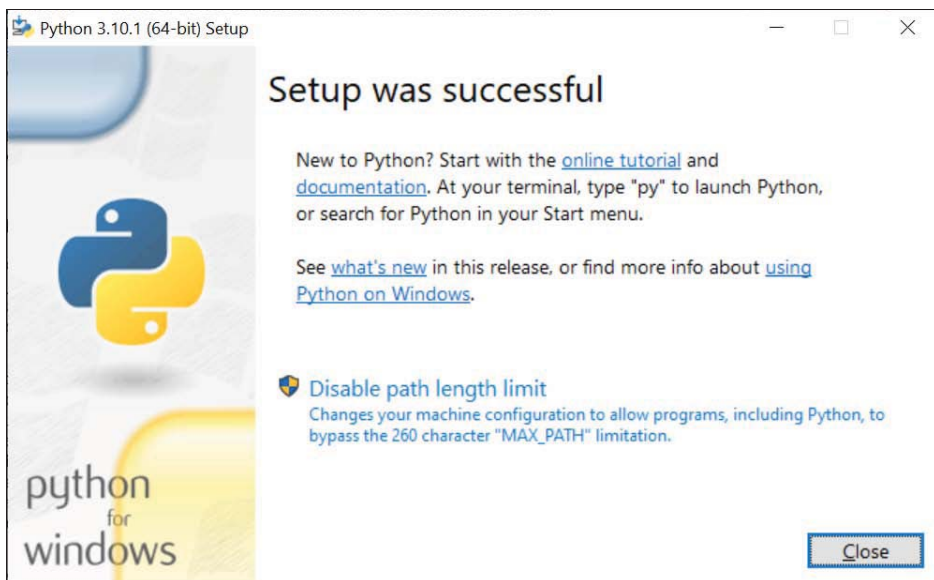


Рис. 1.4. Установка Python: шаг 4

## 1.2. Интерактивный режим Python. Утилита IDLE

Изучать Python удобнее всего, вводя инструкции этого языка вручную и тотчас получая результаты их выполнения. Для этого интерпретатор Python должен работать в *интерактивном режиме*.

Запустить интерпретатор Python в интерактивном режиме можно одним из следующих способов:

- ♦ выбрав в меню **Пуск | Python 3.10** пункт **Python 3.10 (32-bit)** или **Python 3.10 (64-bit)**;
- ♦ набрав в консоли **python** и нажав клавишу **<Enter>**;
- ♦ выполнив щелчок мышью (одинарный или двойной — в зависимости от настроек операционной системы) на файле **python.exe** в каталоге **c:\Python310**.

В результате на экране появится окно, напоминающее таковое у командной строки Windows (рис. 1.5). Символами **>>>** в этом окне помечается приглашение для ввода инструкций. Если после этих символов ввести, например, **2 + 2** и нажать клавишу **<Enter>**, то на следующей строке появится результат сложения — **4**, а затем — приглашение для ввода новой инструкции. Таким образом, интерактивный режим можно использовать и в качестве очень мощного калькулятора.

Однако значительно удобнее учить язык и выполнять вычисления, работая с утилитой IDLE, которая входит в комплект поставки Python. Эта утилита представляет собой своего рода обертку вокруг интерпретатора, работающего в интерактивном режиме, которая

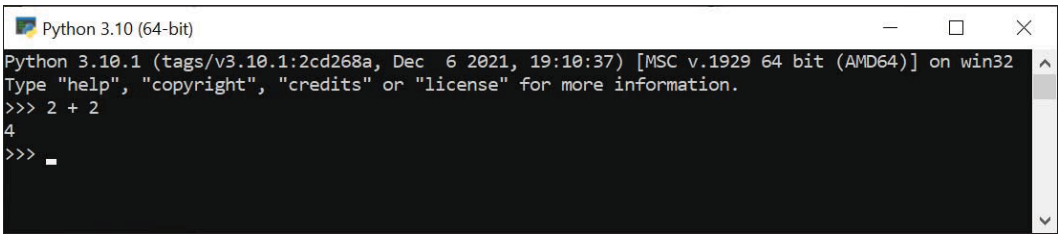


Рис. 1.5. Интерпретатор Python, запущенный в интерактивном режиме

добавляет интерпретатору расширенную функциональность (в частности, синтаксическую подсветку программного кода и вывод подсказок).

Для запуска IDLE достаточно выбрать в меню **Пуск | Python 3.10** пункт **IDLE (Python 3.10 32-bit)** или **IDLE (Python 3.10 64-bit)**. На экране появится окно **IDLE Shell** (рис. 1.6), в котором выводится интерактивный интерпретатор Python.

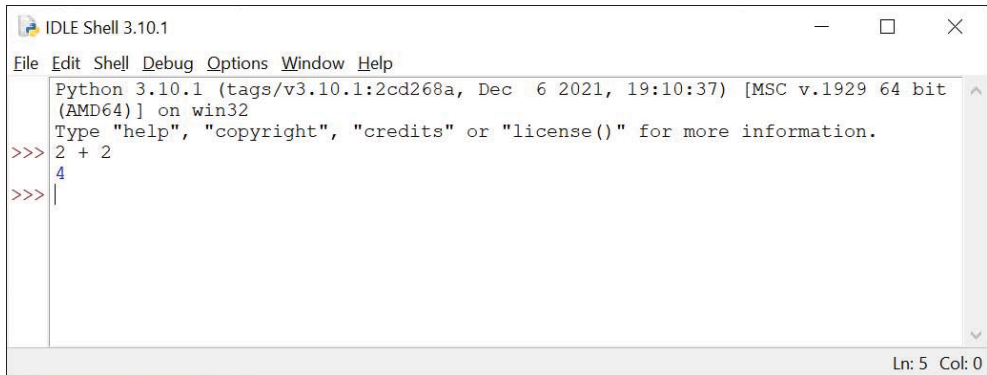


Рис. 1.6. Окно IDLE Shell утилиты IDLE

Отметим, что в окне **IDLE Shell** приглашение `>>>` отображается на свободной полосе, расположенной левее области редактирования, в которой вводятся инструкции Python и выводятся результаты. Поэтому при копировании кода в буфер обмена комбинацией клавиш `<Ctrl>+<C>` приглашение в него не попадает.

Кроме того, IDLE позволяет работать с Python-программами, хранящимися в файлах. Содержимое таких файлов выводится в отдельных окнах. Более подробно функциональные возможности этой утилиты будут рассмотрены позже.

### **ВНИМАНИЕ!**

В дальнейшем для написания и выполнения Python-программ мы будем использовать утилиту IDLE.

## 1.3. Введение в Python-программирование

Изучение языков программирования принято начинать с программы, выводящей надпись «Привет, мир!» Не будем нарушать традиции и продемонстрируем, как это будет выглядеть на Python (листинг 1.1).

**Листинг 1.1. Первая программа на Python**

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

Эта программа состоит из двух *инструкций* (или *выражений*) — предписаний, указывающих Python выполнить какое-либо законченное действие и записываемых на отдельных строках. Первая инструкция представляет собой комментарий — пометку, написанную самим программистом с целью объяснить себе или коллегам, что делает тот или иной код. Вторая инструкция выводит в консоли строку «Привет, мир!»

Вывод значения во второй инструкции выполняется посредством функции `print()`. *Функцией* называется языковая конструкция, выполняющая над переданными ей значениями (*параметрами*) какое-либо относительно сложное действие (в нашем случае — вывод в консоль). Параметры записываются внутри круглых скобок и отделяются друг от друга запятыми. В листинге 1.1 функции `print()` передается всего один параметр — выводимое значение.

Запускаем интерпретатор Python в интерактивном режиме (как это сделать, было описано в *разд. 1.2*). Правее приглашения `>>>` вводим сначала первую строку из листинга 1.1, а затем вторую. После ввода каждой строки нажимаем клавишу `<Enter>`. В третьей строке сразу отобразится результат, а далее — приглашение для ввода новой команды. Мы увидим следующее (напоминаем, что символы `>>>` — это приглашение, их вводить не нужно):

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

**ВНИМАНИЕ!**

В дальнейшем в подобных фрагментах кода инструкции, вводимые вручную, будут помечаться расположенными левее символами приглашения `>>>`, а результаты, выведенные интерпретатором, показываться без этих символов.

Для создания файла с Python-программой в меню **File** окна **IDLE Shell** выбираем пункт **New File** или нажимаем комбинацию клавиш `<Ctrl>+<N>`. В открывшемся окне набираем код из листинга 1.1, а затем сохраняем его в каком-либо каталоге в файле с именем `hello_world.py`, выбрав пункт меню **File | Save** (или нажав комбинацию клавиш `<Ctrl>+<S>`).

Кстати, файл с Python-кодом в терминологии этого языка называется *модулем*.

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

Запустить Python-программу вне среды IDLE можно двумя способами:

- ◆ набрав имя хранящего ее файла вместе с расширением (например: `hello_world.py`) в консоли и нажав клавишу `<Enter>`.

Результат выполнения будет выведен там же, в консоли;

- ◆ выполнив щелчок мышью (двойной или одинарный — в зависимости от настроек системы) на значке файла с этой программой.

В этом случае после вывода результата окно консоли сразу закроется. Чтобы предотвратить его закрытие, необходимо добавить в конец кода программы вызов функции

`input()`, которая станет ожидать нажатия клавиши <Enter> и не позволит окну сразу закрыться. С учетом сказанного наша первая программа будет выглядеть так, как показано в листинге 1.2.

**Листинг 1.2. Программа для запуска с помощью щелчка мыши**

```
print("Привет, мир!")      # Выводим строку
input()                   # Ожидаем нажатия клавиши <Enter>
```

Отметим, что функция `input()` в листинге 1.2 вызывается без параметров. В таком случае после имени функции ставятся пустые круглые скобки.

**ПРИМЕЧАНИЕ**

Если до выполнения функции `input()` в коде возникнет ошибка, то сообщение о ней будет выведено в консоли, но последняя после этого сразу закроется, и вы не сможете прочитать сообщение об ошибке. Попад в подобную ситуацию, запустите программу из консоли или утилиты IDLE, и вы сможете прочитать это сообщение.

Чтобы открыть Python-файл для редактирования, запустим IDLE, выберем пункт меню **File | Open** (комбинация клавиш <Ctrl>+<O>) и выберем нужный файл в появившемся на экране диалоговом окне открытия файла. Файл будет открыт в новом окне утилиты IDLE.

## 1.4. Принципы написания Python-программ

- ◆ Программа, написанная на Python, представляет собой обычный текстовый файл с расширением `py` (будет исполнен обычным интерпретатором `python.exe`) или `pyw` (будет исполнен «оконным» интерпретатором `pythonw.exe`).

Для написания Python-программ, как уже ранее отмечалось, можно использовать, помимо IDLE, любой подходящий текстовый редактор: стандартный Блокнот, Notepad++ (<https://notepad-plus-plus.org/>), Visual Studio Code (<https://code.visualstudio.com/>) и др. Также существует ряд специализированных текстовых редакторов, предназначенных именно для Python-программистов, в частности PyCharm<sup>1</sup> (<https://www.jetbrains.com/pycharm/>).

- ◆ Инструкции, записанные в коде программы, выполняются последовательно, в порядке сверху вниз.
- ◆ Каждая инструкция Python должна располагаться на отдельной строке. Признаком конца инструкции является перевод строки. Пример программы из двух выражений приведен в листинге 1.3.

**Листинг 1.3. Программа, выполняющая арифметические вычисления**

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

Первое выражение складывает числа 15, 20 и 30 и заносит сумму в переменную с именем `x` (*переменная* — ячейка памяти, имеющая уникальное имя, способная хранить ка-

---

<sup>1</sup> Доступны как базовая бесплатная, так и расширенная платная редакции этого редактора.

кое-либо значение — например, число, и создаваемая при присваивании ей значения). Второе выражение выводит содержимое этой переменной — число 65 — в консоли.

В первом выражении используются два оператора. *Оператором* называется языковая конструкция, выполняющая над переданными ему значениями какое-либо элементарное действие (например, сложение чисел или присваивание результата указанной переменной). Оператор сложения обозначается привычным символом +, а оператор присваивания — символом =.

В Windows перевод строки формируется комбинацией символов \r (перевод каретки) и \n (перевод строки), в UNIX — одним символом \n.

Если загрузить файл Python-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 (ASCII) режиме — тогда символ \r будет удален автоматически.

- ◆ Если строка с инструкцией получилась слишком длинной, инструкцию можно разделить на несколько строк одним из следующих способов:

- поставить в месте разрыва строк символ \, сразу после которого вставить перевод строки (как обычно, нажатием клавиши <Enter>):

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

Как правило, вторая и все последующие строки, содержащие длинное выражение, набираются с отступами слева — для улучшения читаемости кода. Следует только формировать эти отступы *исключительно пробелами* и делать *одинаковой длины* (содержащими одинаковое количество пробелов).

При вводе подобного рода многострочной инструкции в интерпретаторе, работающем в интерактивном режиме, строки с продолжением инструкции помечаются расположенным слева приглашением в виде трех точек (. . ., а не >>>, как обычно). Кроме того, интерпретатор сам выводит последующие строки с отступами. Чтобы завершить ввод многострочной инструкции, следует, введя ее последнюю строку, нажать клавишу <Enter> (при этом после многострочной инструкции будет создана пустая строка с приглашением из трех точек). Пример:

```
>>> x = 15 + 20 + \      # Вводим первую строку многострочной инструкции
...     30              # Вводим вторую, последнюю, строку
...                   # Завершаем ввод многострочной инструкции, нажав <Enter>
>>> print(x)           # Вводим следующую инструкцию
```

- заключить выражение в круглые скобки, внутри которых вставить нужное количество переводов строки:

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

- определение списка и словаря (описываются в главах 8 и 9), при оформлении которых используются квадратные и фигурные скобки соответственно, также можно разбить на несколько строк:

пример определения списка:

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

пример определения словаря:

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

- ♦ Короткие инструкции можно записать в одной строке, разделив их символами точки с запятой (;):

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

После точки с запятой не возбраняется ставить пробел — для улучшения читаемости кода.

- ♦ *Блоки* (наборы из произвольного количества инструкций, входящие в состав более сложных инструкций) формируются *отступами слева*. Такие отступы должны формироваться *исключительно пробелами* и быть *одинаковой длины* (т. е. содержать одинаковое количество пробелов). Символы табуляции в отступах не допускаются и при выполнении программы вызывают возникновение ошибки.

Листинг 1.4 показывает код программы, выводящей последовательно числа от 1 до 10, которые разделяются тремя дефисами.

#### Листинг 1.4. Пример блока

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

Первая строка содержит сложную инструкцию — цикл (описывается в *главе 4*), который выполняется 10 раз. При каждом исполнении он заносит в переменную *i* очередное число в диапазоне от 1 до 10 и выполняет блок, входящий в его состав. Этот блок состоит из двух инструкций, записанных во второй и третьих строках: первая инструкция выводит в консоли число из переменной *i*, а вторая — строку из трех дефисов. Обе инструкции, входящие в блок, имеют одинаковый отступ слева, содержащий 4 пробела.

Числа, перебираемые циклом (и, соответственно, количество выполнений, или *итераций*, цикла), указываются в функции `range()` (первая инструкция в листинге 1.4). Первый параметр задает начальное число перебираемого диапазона, второй — конечное число плюс 1.

#### ПРИМЕЧАНИЕ

4 пробела — общепринятая величина отступа в Python.

При вводе инструкции, содержащей блок, в интерпретаторе, который работает в интерактивном режиме, интерпретатор предваряет выражения, входящие в блок, приглашением в виде трех точек (...) и сам вставляет отступы. Чтобы завершить ввод блока, следует, занеся последнее входящее в него выражение, нажать клавишу <Enter> (при этом в набранном коде появится пустая строка с приглашением в виде трех точек). Пример:



```
>>> for i in range(1, 11):
...     print(i)
...     print("---")
...
>>> # Следующая инструкция
```

- ◆ Если блок содержит одну короткую инструкцию, и сам блок, и сложную инструкцию, в состав которой он входит, можно записать в одну строку. При этом интерпретатор посчитает, что ввод блока продолжится после нажатия клавиши <Enter>, и выведет приглашение в виде трех точек. Чтобы завершить ввод инструкции, следует снова нажать <Enter>. Пример программы, выводящей в консоли числа от 1 до 10:

```
>>> for i in range(1, 11): print(i)
...
... Вывод пропущен ...
```

### 1.4.1. Комментарии и строки документирования

*Комментарий* — это произвольное пояснение, вставленное в код программы, предназначенное исключительно программисту и полностью игнорируемое интерпретатором. Внутри комментария может располагаться любой текст.

Комментарий в языке Python начинается с символа # и заканчивается концом строки:

```
# Выводим числа от 1 до 10
for i in range(1, 11): print(i)
```

Комментарий может располагаться после собственно инструкции:

```
print("Все, числа закончились") # Сообщаем об окончании чисел
```

Если символ # разместить перед инструкцией, то она не будет выполнена (*закомментированная инструкция*):

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

Если требуется разместить комментарий из нескольких строк, перед каждой из них придется ставить символ #:

```
# Это наша первая программа!
# Она выводит числа от 1 до 10
# Да, не впечатляет, но для начала неплохо...
for i in range(1, 11): print(i)
```

*Строки документирования* Python обычно применяются для написания инструкций к программам и модулям (подробности — в главе 6). Однако их можно использовать и для комментирования кода.

Строка документирования заключается в утроенные кавычки или апострофы:

```
"""
Это наша первая программа!
Она выводит числа от 1 до 10
Да, не впечатляет, но для начала неплохо...
"""
for i in range(1, 11): print(i)
```



## 1.4.2. Кодировки, поддерживаемые Python

Код Python-программы, написанный в IDLE, по умолчанию сохраняется в кодировке UTF-8 без BOM<sup>1</sup>.

Если программу следует сохранить в какой-либо другой кодировке (что может пригодиться, например, при переписывании старого Python-кода), в первой строке ее кода следует указать кодировку с помощью инструкции формата:

```
# -*- coding: <Обозначение кодировки> -*-
```

Например, кодировка Windows-1251 указывается инструкцией:

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

Встретив такую инструкцию в коде программы, IDLE при сохранении файла самостоятельно переведет программу в кодировку с заданным обозначением. При использовании других редакторов следует перевести программу в указанную кодировку вручную.

Получить полный список поддерживаемых Python кодировок и их обозначения позволяет программа, приведенная в листинге 1.5.

### Листинг 1.5. Вывод списка поддерживаемых кодировок

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

## 1.4.3. Подготовка Python-программ для выполнения в UNIX

Если программа предназначена для исполнения в операционных системах семейства UNIX, в первой строке кода программы необходимо указать путь к интерпретатору Python:

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

В некоторых UNIX-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.

Если программа, исполняемая в UNIX, сохранена в кодировке, отличной от UTF-8, обозначение кодировки указывается во второй строке ее кода:

```
#!/usr/bin/python
# -*- coding: cp1251 -*-
```

---

<sup>1</sup> BOM (Byte Order Mark) — метка порядка байтов. Указывает порядок, в котором записываются байты, кодирующие символы в UTF-8.

Также следует разрешить выполнять Python-программу, указав у ее файла права 755 (`-rwxr-xr-x`).

## 1.5. Дополнительные возможности IDLE

Поскольку далее для работы мы будем использовать утилиту IDLE, рассмотрим предлагаемые ею дополнительные возможности.

Обычно после ввода инструкции и нажатия клавиши <Enter> введенная инструкция выполняется, и на следующей строке выводится ее результат (при условии, что инструкция возвращает результат). После чего появляется приглашение для ввода новой инструкции.

При вводе сложной инструкции, содержащей блок, после нажатия клавиши <Enter> интерпретатор, работающий в интерактивном режиме, автоматически создаст отступ и будет ожидать ввода блока. Чтобы сообщить интерпретатору об окончании ввода инструкции, необходимо нажать <Enter> дважды:

```
>>> for n in range(1, 3):
...     print(n)
...
1
2
>>>
```

Если ввести какое-либо значение — например, строку или число, и нажать <Enter>, это значение появится в следующей строке:

```
>>> "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!
>>>
```

Как говорилось в *разд. 1.2*, окно **IDLE Shell** можно использовать для изучения языка, а также в качестве многофункционального калькулятора (здесь `*` — это оператор умножения):

```
>>> 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 (здесь `/` — оператор деления):

```
>>> 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 нужную языковую конструкцию. Если при открытом списке начать вводить буквы, то показываться будут языковые конструкции, начинающиеся с этих букв. Выбирать конструкцию 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 языковой конструкции после ввода первых букв можно воспользоваться комбинацией клавиш <Alt>+</>. При каждом последующем нажатии этой комбинации будет вставляться следующая конструкция. Эти две комбинации клавиш очень удобны, если вы забыли, как пишется слово, или хотите, чтобы редактор закончил его за вас.

Иногда возникает необходимость выполнить ранее введенную инструкцию повторно. Для таких случаев IDLE предоставляет комбинации клавиш <Alt>+<N> (вставка первой введенной инструкции) и <Alt>+<P> (вставка 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 данных

Вывести заданные значения можно с помощью функции `print()`:

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

Указанные значения при необходимости преобразуются в строки и посылаются в стандартный поток вывода `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 записаны в файл принудительно. Перенаправление вывода мы подробно рассмотрим при изучении файлов.

Если выполняется вывод одного значения, автоматически добавляется символ перевода строки:

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

Результат:

```
Строка 1
Строка 2
```

Можно вывести несколько значений в одну строку, указав их в вызове функции `print()` отдельными параметрами, разделенными запятыми:

```
print("Строка 1", "Строка 2")
```

Результат:

```
Строка 1 Строка 2
```

Как видно из примера, между выводимыми значениями автоматически вставляется пробел. С помощью параметра `sep` можно указать другой разделяющий символ. Например, выведем строки без пробела между ними, указав в качестве разделителя пустую строку:

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

Результат:

```
Строка 1Строка 2
```

Ряд функций, встроенных в Python, поддерживают так называемые *именованные параметры*. В число таких параметров и входит `sep`. Обратим внимание, как задается его значение в приведенном примере.

После вывода нескольких значений в одном вызове функции `print()` в конце добавляется символ перевода строки. Если необходимо произвести дальнейший вывод на той же строке, то в именованном параметре `end` следует указать другой символ:

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

Вызов функции `print()` без параметров выводит пустую строку:

```
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, выполняющийся четыре раза. На каждой итерации он присваивает переменной `n` число от 1 до 4 и выполняет блок, содержащий вызов функции `print()`, которая выводит число из переменной `n`. Не забываем вставить в выражение, входящее в блок, отступ слева.

Как только цикл выполнится четыре раза, будут исполнены два следующих за ним выражения. В них отступы слева указывать не следует, поскольку эти выражения не должны входить в состав блока. Первое выражение выведет пустую строку, второе — надпись «Это текст на новой строке».

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

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

В результате выполнения этого примера мы получим три строки:

```
Строка 1
Строка 2
Строка 3
```

Вместо функции `print()` можно использовать метод `write()` объекта `stdout` из модуля `sys` (методы очень похожи на функции и в подробностях, вместе с объектами, будут рассмотрены позже):

```
import sys                                # Подключаем модуль sys, чтобы использовать
                                         # содержащийся в нем объект stdout
sys.stdout.write("Строка")
```

Как мы помним из *разд. 1.3*, модуль — это просто файл с Python-кодом. Однако модуль `sys` поставляется в составе Python и входит в *стандартную библиотеку* (набор модулей, содержащих полезные функции, объекты и др.) этого языка. Подключив этот модуль, мы можем использовать созданные в нем функции и объекты.

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

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

Метод `write()` возвращает *результат* — значение, полученное в процессе выполненных методом вычислений. Таковым результатом является количество символов в выведенной строке. Его можно присвоить какой-либо переменной и использовать в дальнейшем:

```
import sys
cnt = sys.stdout.write("Привет, Python\n")
print("Символов в выведенной строке: ", cnt)
```

Результат:

```
Привет, Python
Символов в выведенной строке: 15
```

## 1.7. Ввод данных

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

```
input([<Сообщение>])
```

Введенное значение она возвращает в качестве результата. Этот результат следует присвоить какой-либо переменной посредством оператора `=`.

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

**Листинг 1.6. Пример использования функции `input()`**

```
name = input("Введите ваше имя: ")
print("Привет, ", name)
input("Нажмите <Enter> для закрытия окна")
```

В первой инструкции значение, введенное пользователем и полученное функцией `input()`, присваивается переменной `name`.

Сохраняем программу в файле `test.py` и запускаем на выполнение с помощью двойного щелчка. Откроется черное окно, в котором мы увидим надпись: **Введите ваше имя:**. Введем свое имя — например, Николай, и нажимаем клавишу `<Enter>`. В результате будет выведено приветствие: **Привет, Николай**.

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

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

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

Передать данные можно в консоли, указав их после имени файла программы. Такие данные доступны через список `argv` из модуля `sys`. Первый элемент списка `argv` будет содержать имя файла запущенной программы, а последующие элементы — переданные данные. Для примера создадим файл `test2.py` в каталоге `C:\book`. Содержимое файла приведено в листинге 1.7.

#### Листинг 1.7. Получение данных из консоли

```
import sys
arr = sys.argv[1:]
for n in arr:
    print(n)
```

Теперь запустим программу на выполнение из консоли, передав ей данные. Запустим консоль, для чего выберем в меню **Пуск** пункт **Выполнить**, в открывшемся окне наберем команду `cmd` и нажмем кнопку **ОК**. Откроется черное окно с приглашением для ввода команд. Перейдем в каталог `C:\book`, набрав команду:

```
cd C:\book
```

В консоли должно появиться приглашение:

```
C:\book>
```

Для запуска нашей программы вводим команду:

```
test2.py uhud opk987
```

В этой команде мы передаем имя файла (`test2.py`) и некоторые данные (`-uNik` и `-p123`). Результат выполнения программы будет выглядеть так:

```
test2.py
uhud
opk987
```