

---

# Оглавление

Введение.....	26
Для кого эта книга.....	27
Что нового во втором издании.....	27
Структура книги.....	28
Версии Python.....	31
Условные обозначения.....	31
Использование примеров кода.....	32
От издательства.....	32
Благодарности.....	33
Об авторе.....	34

## Часть I. Основы Python

<b>Глава 1.</b> Python: с чем его едят.....	36
Тайны.....	36
Маленькие программы.....	38
Более объемная программа.....	40
Python в реальном мире.....	44
Python против языка с планеты X.....	45
Почему же Python?.....	48
Когда не стоит использовать Python.....	49
Python 2 против Python 3.....	50
Установка Python.....	51
Запуск Python.....	51
Интерактивный интерпретатор.....	51
Файлы Python.....	52
Что дальше?.....	53

---

Момент просветления.....	53
Читайте далее.....	54
Упражнения.....	54
<b>Глава 2. Данные: типы, значения, переменные и имена.....</b>	<b>55</b>
В Python данные являются объектами.....	55
Типы.....	56
Изменчивость.....	57
Значения-литералы.....	58
Переменные.....	58
Присваивание.....	60
Переменные — это имена, а не локации.....	61
Присваивание нескольким именам.....	64
Переназначение имени.....	64
Копирование.....	64
Выбираем хорошее имя переменной.....	65
Читайте далее.....	66
Упражнения.....	66
<b>Глава 3. Числа.....</b>	<b>67</b>
Булевы значения.....	67
Целые числа.....	68
Числа-литералы.....	68
Операции с целыми числами.....	69
Целые числа и переменные.....	71
Приоритет операций.....	73
Системы счисления.....	74
Преобразования типов.....	76
Насколько объемов тип int.....	78
Числа с плавающей точкой.....	79
Математические функции.....	80
Читайте далее.....	81
Упражнения.....	81
<b>Глава 4. Выбираем с помощью оператора if.....</b>	<b>82</b>
Комментируем с помощью символа #.....	82
Продлеваем строки с помощью символа \.....	83
Сравниваем с помощью операторов if, elif и else.....	84
Что есть истина?.....	87
Выполняем несколько сравнений с помощью оператора in.....	88

---

Новое: I Am the Walrus .....	89
Читайте далее.....	90
Упражнения .....	90
<b>Глава 5. Текстовые строки .....</b>	<b>91</b>
Создаем строки с помощью кавычек .....	91
Создаем строки с помощью функции str().....	94
Создаем escape-последовательности с помощью символа \.....	94
Объединяем строки с использованием символа +.....	96
Размножаем строки с помощью символа * .....	96
Извлекаем символ с помощью символов [ ].....	97
Извлекаем подстроки, используя разделение .....	98
Измеряем длину строки с помощью функции len().....	100
Разделяем строку с помощью функции split().....	100
Объединяем строки с помощью функции join() .....	101
Заменяем символы с использованием функции replace() .....	101
Устраняем символы с помощью функции strip() .....	102
Поиск и выбор .....	103
Регистр .....	104
Выравнивание.....	105
Форматирование.....	105
Старый стиль: % .....	106
Новый стиль: используем символы {} и функцию format().....	108
Самый новый стиль: f-строки .....	110
Что еще можно делать со строками.....	111
Читайте далее.....	111
Упражнения.....	111
<b>Глава 6. Создаем циклы с помощью ключевых слов while и for.....</b>	<b>113</b>
Повторяем действия с помощью цикла while.....	113
Прерываем цикл с помощью оператора break.....	114
Пропускаем итерации, используя оператор continue .....	114
Проверяем, завершился ли цикл раньше, с помощью блока else .....	115
Выполняем итерации с использованием ключевых слов for и in.....	115
Прерываем цикл с помощью оператора break.....	116
Пропускаем итерации, используя оператор continue .....	116
Проверяем, завершился ли цикл раньше, с помощью блока else .....	116
Генерируем числовые последовательности с помощью функции range() .....	117
Прочие итераторы .....	118
Читайте далее.....	118
Упражнения.....	118

---

<b>Глава 7. Кортежи и списки</b> .....	119
Кортежи .....	119
Создаем кортежи с помощью запятых и оператора () .....	120
Создаем кортежи с помощью функции tuple() .....	121
Объединяем кортежи с помощью оператора + .....	121
Размножаем элементы с помощью оператора * .....	122
Сравниваем кортежи .....	122
Итерируем по кортежам с помощью for и in .....	122
Изменяем кортеж .....	122
Списки .....	123
Создаем списки с помощью скобок [] .....	123
Создаем список или преобразуем в список с помощью функции list() .....	123
Создаем список из строки с использованием функции split() .....	124
Получаем элемент с помощью конструкции [смещение] .....	124
Извлекаем элементы с помощью разделения .....	125
Добавляем элемент в конец списка с помощью функции append() .....	126
Добавляем элемент на определенное место с помощью функции insert() .....	126
Размножаем элементы с помощью оператора * .....	127
Объединяем списки с помощью метода extend() или оператора + .....	127
Изменяем элемент с помощью конструкции [смещение] .....	128
Изменяем элементы с помощью разделения .....	128
Удаляем заданный элемент с помощью оператора del .....	129
Удаляем элемент по значению с помощью функции remove() .....	129
Получаем и удаляем заданный элемент с помощью функции pop() .....	129
Удаляем все элементы с помощью функции clear() .....	130
Определяем смещение по значению с помощью функции index() .....	130
Проверяем на наличие элемента в списке с помощью оператора in .....	131
Подсчитываем количество включений значения с помощью функции count() .....	131
Преобразуем список в строку с помощью функции join() .....	131
Меняем порядок элементов с помощью функций sort() или sorted() .....	132
Получаем длину списка с помощью функции len() .....	133
Присваиваем с помощью оператора = .....	133
Копируем списки с помощью функций copy() и list() или путем разделения .....	134
Копируем все с помощью функции deepcopy() .....	134
Сравниваем списки .....	135
Итерируем по спискам с помощью операторов for и in .....	136

Итерируем по нескольким последовательностям с помощью функции <code>zip()</code> .....	137
Создаем список с помощью списковых включений .....	138
Списки списков .....	140
Кортежи или списки? .....	141
Включений кортежей не существует .....	141
Читайте далее .....	142
Упражнения .....	142
<b>Глава 8. Словари и множества</b> .....	144
Словари .....	144
Создаем словарь с помощью <code>{}</code> .....	144
Создаем словарь с помощью функции <code>dict()</code> .....	145
Преобразуем с помощью функции <code>dict()</code> .....	146
Добавляем или изменяем элемент с помощью конструкции [ключ] .....	146
Получаем элемент словаря с помощью конструкции [ключ] или функции <code>get()</code> .....	148
Получаем все ключи с помощью функции <code>keys()</code> .....	148
Получаем все значения с помощью функции <code>values()</code> .....	149
Получаем все пары «ключ — значение» с помощью функции <code>items()</code> .....	149
Получаем длину словаря с помощью функции <code>len()</code> .....	149
Объединяем словари с помощью конструкции <code>{**a, **b}</code> .....	149
Объединяем словари с помощью функции <code>update()</code> .....	150
Удаляем элементы по их ключу с помощью оператора <code>del</code> .....	151
Получаем элемент по ключу и удаляем его с помощью функции <code>pop()</code> .....	151
Удаляем все элементы с помощью функции <code>clear()</code> .....	151
Проверяем на наличие ключа с помощью оператора <code>in</code> .....	152
Присваиваем значения с помощью оператора <code>=</code> .....	152
Копируем значения с помощью функции <code>copy()</code> .....	152
Копируем все с помощью функции <code>deepcopy()</code> .....	153
Сравниваем словари .....	154
Итерируем по словарям с помощью <code>for</code> и <code>in</code> .....	154
Включения словарей .....	155
Множества .....	156
Создаем множество с помощью функции <code>set()</code> .....	157
Преобразуем другие типы данных с помощью функции <code>set()</code> .....	157
Получаем длину множества с помощью функции <code>len()</code> .....	158
Добавляем элемент с помощью функции <code>add()</code> .....	158
Удаляем элемент с помощью функции <code>remove()</code> .....	158

Итерируем по множествам с помощью <code>for</code> и <code>in</code> .....	158
Проверяем на наличие значения с помощью оператора <code>in</code> .....	158
Комбинации и операторы .....	159
Включение множества.....	162
Создаем неизменяемое множество с помощью функции <code>frozenset()</code> .....	162
Структуры данных, которые мы уже рассмотрели.....	163
Создание крупных структур данных.....	164
Читайте далее.....	164
Упражнения.....	165
<b>Глава 9. Функции</b> .....	<b>166</b>
Определяем функцию с помощью ключевого слова <code>def</code> .....	166
Вызываем функцию с помощью скобок.....	167
Аргументы и параметры.....	167
None — это полезно.....	169
Позиционные аргументы .....	170
Аргументы — ключевые слова.....	171
Указываем значение параметра по умолчанию .....	171
Получаем/разбиваем аргументы — ключевые слова с помощью символа <code>*</code> .....	172
Получаем/разбиваем аргументы — ключевые слова с помощью символов <code>**</code> .....	174
Аргументы, передаваемые только по ключевым словам.....	175
Изменяемые и неизменяемые аргументы .....	176
Строки документации.....	176
Функции — это объекты первого класса.....	177
Внутренние функции .....	179
Анонимные функции: лямбда-выражения .....	181
Генераторы .....	182
Функции-генераторы .....	182
Включения генераторов.....	183
Декораторы.....	183
Пространства имен и область определения .....	186
Использование символов <code>_</code> и <code>__</code> в именах.....	188
Рекурсия .....	188
Асинхронные функции.....	190
Исключения .....	190
Обрабатываем ошибки с помощью операторов <code>try</code> и <code>except</code> .....	191
Создаем собственные исключения .....	192
Читайте далее.....	193
Упражнения.....	193

---

<b>Глава 10. Ой-ой-ой: объекты и классы</b> .....	194
Что такое объекты .....	194
Простые объекты .....	195
Определяем класс с помощью ключевого слова <code>class</code> .....	195
Атрибуты.....	196
Методы .....	197
Инициализация .....	197
Наследование.....	198
Наследование от родительского класса .....	199
Переопределение методов.....	200
Добавление метода .....	201
Получаем помощь от своего родителя с использованием метода <code>super()</code> .....	202
Множественное наследование.....	203
Примеси .....	205
В защиту <code>self</code> .....	205
Доступ к атрибутам.....	206
Прямой доступ .....	206
Геттеры и сеттеры.....	206
Свойства для доступа к атрибутам .....	207
Свойства для вычисляемых значений .....	209
Искажение имен для безопасности .....	209
Атрибуты классов и объектов.....	210
Типы методов .....	211
Методы объектов .....	211
Методы классов .....	212
Статические методы.....	212
Утиная типизация .....	213
Магические методы .....	215
Агрегирование и композиция .....	218
Когда использовать объекты, а когда — что-то другое .....	218
Именованные кортежи .....	219
Классы данных .....	221
<code>attrs</code> .....	222
Читайте далее.....	222
Упражнения.....	222
<b>Глава 11. Модули, пакеты и программы</b> .....	224
Модули и оператор <code>import</code> .....	224
Импортируем модуль .....	224

Импортируем модуль с другим именем.....	226
Импортируем только самое необходимое.....	226
Пакеты.....	227
Путь поиска модуля.....	228
Относительный и абсолютный импорт.....	229
Пакеты пространств имен.....	229
Модули против объектов.....	230
Достоинства стандартной библиотеки Python.....	231
Обрабатываем отсутствующие ключи с помощью функций <code>setdefault()</code> и <code>defaultdict()</code> .....	231
Подсчитываем элементы с помощью функции <code>Counter()</code> .....	233
Упорядочиваем по ключу с помощью <code>OrderedDict()</code> .....	235
Стек + очередь == <code>deque</code> .....	235
Итерируем по структурам кода с помощью модуля <code>itertools</code> .....	236
Красиво выводим данные на экран с помощью функции <code>pprint()</code> .....	238
Работаем со случайными числами.....	238
Нужно больше кода.....	239
Читайте далее.....	240
Упражнения.....	240

## Часть II. Python на практике

<b>Глава 12.</b> Обрабатываем данные.....	242
Текстовые строки: Unicode.....	243
Строки формата Unicode в Python 3.....	244
Кодирование и декодирование с помощью кодировки UTF-8.....	246
Кодирование.....	247
Декодирование.....	249
Сущности HTML.....	250
Нормализация.....	251
Подробная информация.....	252
Текстовые строки: регулярные выражения.....	253
Ищем точное начальное совпадение с помощью функции <code>match()</code> .....	254
Ищем первое совпадение с помощью функции <code>search()</code> .....	255
Ищем все совпадения, используя функцию <code>findall()</code> .....	255
Разбиваем совпадения с помощью функции <code>split()</code> .....	256
Заменяем совпадения с помощью функции <code>sub()</code> .....	256
Шаблоны: специальные символы.....	256
Шаблоны: использование спецификаторов.....	258
Шаблоны: указываем способ вывода совпадения.....	261



---

Бинарные данные .....	261
bytes и bytearray.....	262
Преобразуем бинарные данные с помощью модуля struct.....	263
Другие инструменты для работы с бинарными данными.....	266
Преобразуем байты/строки с помощью модуля binascii .....	267
Битовые операторы.....	267
Аналогия с ювелирными изделиями .....	268
Читайте далее.....	268
Упражнения.....	268
<b>Глава 13. Календари и часы.....</b>	<b>271</b>
Високосный год.....	272
Модуль datetime.....	273
Модуль time.....	275
Читаем и записываем дату и время.....	277
Все преобразования .....	281
Альтернативные модули .....	281
Читайте далее.....	282
Упражнения.....	282
<b>Глава 14. Файлы и каталоги.....</b>	<b>283</b>
Ввод информации в файлы и ее вывод из них .....	283
Создаем или открываем файлы с помощью функции open() .....	284
Записываем в текстовый файл с помощью функции print().....	284
Записываем в текстовый файл с помощью функции write() .....	285
Считываем данные из текстового файла, используя функции read(), readline() и readlines() .....	286
Записываем данные в бинарный файл с помощью функции write() .....	288
Читаем бинарные файлы с помощью функции read().....	289
Закрываем файлы автоматически с помощью ключевого слова with.....	289
Меняем позицию с помощью функции seek().....	289
Отображение в памяти .....	291
Операции с файлами.....	292
Проверяем существование файла с помощью функции exists().....	292
Проверяем тип с помощью функции isfile().....	292
Копируем файлы, используя функцию copy().....	293
Изменяем имена файлов с помощью функции rename().....	293
Создаем ссылки с помощью функции link() или symlink().....	293
Изменяем разрешения с помощью функции chmod().....	294
Изменение владельца файла с помощью функции chown().....	294
Удаляем файл с помощью функции remove().....	294

---

Каталоги.....	295
Создаем каталог с помощью функции <code>mkdir()</code> .....	295
Удаляем каталог, используя функцию <code>rmdir()</code> .....	295
Выводим на экран содержимое каталога с помощью функции <code>listdir()</code> .....	295
Изменяем текущий каталог с помощью функции <code>chdir()</code> .....	296
Перечисляем совпадающие файлы, используя функцию <code>glob()</code> .....	296
Pathname.....	297
Получаем путь с помощью функции <code>abspath()</code> .....	298
Получаем символическую ссылку с помощью функции <code>realpath()</code> .....	298
Построение пути с помощью функции <code>os.path.join()</code> .....	298
Модуль <code>rathlib</code> .....	298
BytesIO и StringIO.....	299
Читайте далее.....	301
Упражнения.....	301
<b>Глава 15. Данные во времени: процессы и конкурентность</b> .....	<b>302</b>
Программы и процессы.....	302
Создаем процесс с помощью модуля <code>subprocess</code> .....	303
Создаем процесс с помощью модуля <code>multiprocessing</code> .....	304
Убиваем процесс, используя функцию <code>terminate()</code> .....	305
Получаем системную информацию с помощью модуля <code>os</code> .....	306
Получаем информацию о процессах с помощью модуля <code>psutil</code> .....	306
Автоматизация команд.....	307
Invoke.....	307
Другие вспомогательные методы для команд.....	308
Конкурентность.....	308
Очереди.....	309
Процессы.....	310
Потоки.....	311
Concurrent.futures.....	314
Зеленые потоки и <code>gevent</code> .....	317
twisted.....	320
asyncio.....	321
Redis.....	321
Помимо очередей.....	325
Читайте далее.....	326
Упражнения.....	326
<b>Глава 16. Данные в коробке: надежные хранилища</b> .....	<b>327</b>
Плоские текстовые файлы.....	327
Текстовые файлы, дополненные пробелами.....	328

---

Структурированные текстовые файлы.....	328
CSV .....	328
XML.....	331
Примечание о безопасности XML .....	333
HTML.....	333
JSON .....	334
YAML.....	337
Tablib.....	338
Pandas .....	338
Конфигурационные файлы .....	340
Бинарные файлы .....	341
Электронные таблицы .....	341
HDF5.....	341
TileDB.....	342
Реляционные базы данных.....	342
SQL .....	343
DB-API .....	345
SQLite.....	345
MySQL.....	347
PostgreSQL.....	347
SQLAlchemy.....	348
Другие пакеты для работы с базами данных .....	354
Хранилища данных NoSQL .....	354
Семейство dbm.....	354
Memcached.....	355
Redis.....	356
Документоориентированные базы данных.....	363
Базы данных временных рядов.....	364
Графовые базы данных.....	365
Другие серверы NoSQL .....	365
Полнотекстовые базы данных.....	366
Читайте далее.....	366
Упражнения.....	366
<b>Глава 17. Данные в пространстве: сети .....</b>	<b>368</b>
TCP/IP.....	368
Сокеты .....	370
scapy.....	374
Netcat.....	374

---

Паттерны для работы с сетями.....	375
Паттерн «Запрос — ответ» .....	375
ZeroMQ.....	375
Другие инструменты обмена сообщениями .....	380
Паттерн «Публикация — подписка».....	380
Redis.....	380
ZeroMQ.....	382
Другие инструменты «Публикации — подписки» .....	383
Интернет-сервисы.....	384
Доменная система имен.....	384
Модули Python для работы с электронной почтой .....	385
Другие протоколы .....	385
Веб-сервисы и API .....	385
Сериализация данных.....	386
Сериализация с помощью pickle.....	387
Другие форматы сериализации.....	388
Удаленные вызовы процедур.....	388
XML RPC .....	389
JSON RPC.....	390
MessagePack RPC.....	391
Zerorpc.....	392
gRPC.....	393
Twirp .....	393
Инструменты удаленного управления.....	394
Работаем с большими объемами данных .....	394
Hadoop .....	394
Spark .....	395
Disco.....	395
Dask.....	395
Работаем в облаках .....	396
Amazon Web Services.....	397
Google.....	397
Microsoft Azure.....	397
OpenStack .....	398
Docker.....	398
Kubernetes.....	398
Читайте далее.....	398
Упражнения.....	399

---

<b>Глава 18.</b> Распутываем Всемирную паутину .....	400
Веб-клиенты .....	401
Тестируем с помощью telnet.....	402
Тестируем с помощью curl .....	403
Тестируем с использованием httpie.....	404
Тестируем с помощью httpbin .....	405
Стандартные веб-библиотеки Python.....	405
За пределами стандартной библиотеки: requests.....	407
Веб-серверы.....	408
Простейший веб-сервер Python.....	409
Web Server Gateway Interface (WSGI) .....	410
ASGI .....	411
apache.....	411
NGINX .....	412
Другие веб-серверы Python.....	413
Фреймворки для работы веб-серверами .....	413
Bottle .....	414
Flask .....	416
Django.....	420
Другие фреймворки .....	421
Фреймворки для работы с базами данных .....	421
Веб-сервисы и автоматизация.....	422
Модуль webbrowser.....	422
Модуль webview .....	423
REST API .....	424
Поиск и выборка данных .....	424
Scrapy .....	425
BeautifulSoup .....	425
Requests-HTML.....	426
Давайте посмотрим фильм.....	426
Читайте далее.....	429
Упражнения.....	429
<b>Глава 19.</b> Быть питонщиком .....	431
О программировании .....	431
Ищем код на Python .....	432
Установка пакетов .....	432
pip .....	433
virtualenv .....	434

pipenv.....	434
Менеджер пакетов.....	434
Установка из исходного кода.....	435
Интегрированные среды разработки.....	435
IDLE.....	435
PyCharm.....	435
IPython.....	436
Jupyter Notebook.....	438
JupyterLab.....	438
Именованное и документирование.....	438
Добавление подсказок типов.....	440
Тестирование кода.....	440
Программы pylint, pyflakes, flake8 или PEP-8.....	441
Пакет unittest.....	443
Пакет doctest.....	447
Пакет nose.....	448
Другие фреймворки для тестирования.....	449
Постоянная интеграция.....	449
Отладка кода.....	450
Функция print().....	450
Отладка с помощью декораторов.....	451
Отладчик pdb.....	452
Функция breakpoint().....	458
Записываем в журнал сообщения об ошибках.....	458
Оптимизация кода.....	460
Измеряем время.....	461
Алгоритмы и структуры данных.....	464
Cython, NumPy и расширения C.....	465
PyPy.....	465
Numba.....	466
Управление исходным кодом.....	467
Mercurial.....	467
Git.....	467
Распространение ваших программ.....	470
Клонируйте эту книгу.....	470
Как узнать больше.....	470
Книги.....	471
Сайты.....	471
Группы.....	472

---

Конференции.....	472
Вакансии, связанные с Python .....	472
Читайте далее.....	473
Упражнения.....	473
<b>Глава 20. Пи-Арт.....</b>	<b>474</b>
Двумерная графика.....	474
Стандартная библиотека .....	474
PIL и Pillow .....	475
ImageMagick.....	478
Трёхмерная графика .....	478
Трёхмерная анимация .....	479
Графические пользовательские интерфейсы (GUI).....	479
Диаграммы, графики и визуализация .....	481
Matplotlib.....	481
Seaborn .....	483
Vokeh .....	485
Игры.....	485
Аудио и музыка .....	486
Читайте далее.....	486
Упражнения.....	486
<b>Глава 21. За работой .....</b>	<b>487</b>
The Microsoft Office Suite .....	487
Выполняем бизнес-задачи.....	488
Обработка бизнес-данных.....	489
Извлечение, преобразование и загрузка.....	489
Валидация данных.....	493
Дополнительные источники информации.....	493
Пакеты для работы с бизнес-данными с открытым исходным кодом .....	494
Python в области финансов.....	494
Безопасность бизнес-данных .....	495
Карты .....	495
Форматы .....	496
Нарисуем карту на основе шейп-файла .....	496
Geopandas.....	498
Другие пакеты для работы с картами .....	500
Приложения и данные .....	501
Читайте далее.....	502
Упражнения.....	502

<b>Глава 22. Python в науке</b> .....	503
Математика и статистика в стандартной библиотеке .....	503
Математические функции .....	503
Работа с комплексными числами.....	505
Рассчитываем точное значение чисел с плавающей точкой с помощью модуля decimal .....	506
Выполняем вычисления для рациональных чисел с помощью модуля fractions .....	507
Используем Packed Sequences с помощью модуля array.....	507
Обрабатываем простую статистику с помощью модуля statistics .....	508
Перемножение матриц .....	508
Python для науки.....	508
NumPy.....	508
Создаем массив с помощью функции array() .....	509
Создаем массив с помощью функции arange().....	510
Создаем массив с помощью функций zeros(), ones() и random().....	511
Изменяем форму массива с помощью метода reshape() .....	512
Получаем элемент с помощью конструкции [] .....	513
Математика массивов.....	514
Линейная алгебра.....	514
Библиотека SciPy .....	515
Библиотека SciKit .....	516
Pandas.....	516
Python и научные области.....	517
Читайте далее.....	518
Упражнения.....	518

## Приложения

<b>Приложение А. Аппаратное и программное обеспечение для начинающих программистов</b> .....	520
Аппаратное обеспечение .....	520
Компьютеры пещерных людей.....	520
Электричество.....	521
Изобретения .....	521
Идеальный компьютер.....	522
Процессор.....	522
Память и кэш.....	522
Хранение .....	522
Ввод данных.....	523



---

Вывод данных.....	523
Относительное время доступа.....	523
Программное обеспечение .....	524
Вначале был бит .....	524
Машинный язык.....	524
Ассемблер .....	525
Высокоуровневые языки.....	525
Операционные системы.....	526
Виртуальные машины .....	527
Контейнеры.....	527
Распределенные вычисления и сети .....	527
Облако .....	528
Kubernetes.....	528
<b>Приложение Б. Установка Python 3 .....</b>	<b>530</b>
Проверьте свою версию Python.....	530
Установка стандартной версии Python .....	531
macOS.....	532
Windows.....	534
Linux или Unix .....	535
Установка менеджера пакетов pip .....	535
Установка virtualenv .....	535
Другие способы работы с пакетами .....	536
Устанавливаем Anaconda .....	536
<b>Приложение В. Нечто совершенно иное: async .....</b>	<b>538</b>
Сопрограммы и циклы событий.....	538
async против.....	542
Асинхронные фреймворки и серверы .....	542
<b>Приложение Г. Ответы к упражнениям .....</b>	<b>544</b>
1. Python: с чем его едят .....	544
2. Типы данных, значения, переменные и имена.....	545
3. Числа .....	545
4. Выбираем с помощью if.....	546
5. Текстовые строки .....	547
6. Создаем циклы с помощью ключевых слов while и for.....	551
7. Кортежи и списки .....	552
8. Словари и множества.....	556
9. Функции.....	559
10. Ой-ой-ой: объекты и классы .....	560

11. Модули, пакеты и программы .....	564
12. Обрабатываем данные .....	566
13. Календари и часы .....	571
14. Файлы и каталоги .....	572
15. Данные во времени: процессы и конкурентность.....	573
16. Данные в коробке: устойчивые хранилища .....	574
17. Данные в пространстве: сети.....	577
18. Распутываем Всемирную паутину.....	584
19. Быть питонщиком.....	585
20. Пи-Арт .....	585
21. За работой.....	586
22. Python в науке .....	586
<b>Приложение Д. Вспомогательные таблицы .....</b>	<b>587</b>
Приоритет операторов.....	587
Строковые методы .....	588
Изменение регистра .....	588
Поиск.....	588
Изменение .....	588
Форматирование .....	589
Тип строки .....	589
Атрибуты модуля string .....	589
Эпилог.....	591

---

# Кортежи и списки

Человек отличается от низших приматов страстью к составлению списков.

*Гарри Аллен Смит*

В предыдущих главах мы говорили о базовых типах данных Python, таких как булевы значения, целочисленные значения, числа с плавающей точкой и строки. Если представлять их как атомы, то структуры данных, которые мы рассмотрим в этой главе, можно назвать молекулами. Так и есть: мы объединим базовые типы в более сложные структуры, которые вы будете использовать каждый день. Большая часть работы программиста состоит из «разрезания» данных и «склеивания» их в конкретные формы, поэтому сейчас вы узнаете, как пользоваться ножовками и клеевыми пистолетами.

Большинство языков программирования могут представлять последовательность в виде объектов, проиндексированных по их позиции, выраженной целым числом: первый, второй и далее до последнего. Вы уже знакомы со *строками* — последовательностями символов.

В Python есть еще две структуры-последовательности: *кортежи* и *списки*. Они могут содержать ноль и более элементов. В отличие от строк в кортежах и списках допускаются элементы разных типов: по факту каждый элемент может быть *любым* объектом Python. Это позволяет создавать структуры любой сложности и глубины.

Почему же в Python имеются как списки, так и кортежи? Кортежи *неизменяемы*. Когда вы включаете в кортеж элемент (всего один раз), он «запекается» и больше не изменяется. Списки же можно *изменять* — добавлять и удалять элементы в любой удобный момент. Я покажу вам множество примеров использования обоих типов, сделав акцент на списках.

## Кортежи

Давайте сразу же рассмотрим один очевидный вопрос. Вы могли слышать два возможных варианта произношения слова *tuple* (кортеж). Какой же из них является правильным? Гвидо ван Россум, создатель языка Python, написал в Twitter

(<http://bit.ly/tupletweet>): «Я произношу слово tuple как too-pull по понедельникам, средам и пятницам и как tub-pull — по вторникам, четвергам и субботам. В воскресенье я вообще об этом не говорю :)».

## Создаем кортежи с помощью запятых и оператора ()

Синтаксис создания кортежей несколько необычен, что вы и увидите в следующих примерах.

Начнем с создания пустого кортежа с помощью оператора ():

```
>>> empty_tuple = ()
>>> empty_tuple
()
```

Чтобы создать кортеж, содержащий один элемент и более, после каждого элемента надо ставить запятую. Это вариант для кортежей с одним элементом:

```
>>> one_marx = 'Groucho',
>>> one_marx
('Groucho',)
```

Вы можете поместить элемент в круглые скобки и получить такой же кортеж:

```
>>> one_marx = ('Groucho',)
>>> one_marx
('Groucho',)
```

Однако следует иметь в виду: если в круглые скобки вы поместите один объект и опустите при этом запятую, в результате вы получите не кортеж, а тот же самый объект (в этом примере строку 'Groucho'):

```
>>> one_marx = ('Groucho')
>>> one_marx
'Groucho'
>>> type(one_marx)
<class 'str'>
```

Если в вашем кортеже более одного элемента, ставьте запятую после каждого из них, кроме последнего:

```
>>> marx_tuple = 'Groucho', 'Chico', 'Harpo'
>>> marx_tuple
('Groucho', 'Chico', 'Harpo')
```

При отображении кортежа Python выводит на экран скобки. Как правило, они не нужны для определения кортежа, но с ними более безопасно, так как они делают кортеж более заметным:

```
>>> marx_tuple = ('Groucho', 'Chico', 'Harpo')
>>> marx_tuple
('Groucho', 'Chico', 'Harpo')
```

В тех случаях, когда у запятой могут быть и другие варианты использования, также рекомендуется ставить круглые скобки. В следующем примере вы можете создать кортеж с одним элементом и присвоить ему значение, поставив в конце запятую, но не можете передать эту конструкцию как аргумент функции:

```
>>> one_marx = 'Groucho',
>>> type(one_marx)
<class 'tuple'>
>>> type('Groucho',)
<class 'str'>
>>> type(('Groucho',))
<class 'tuple'>
```

Кортежи позволяют присваивать значение нескольким переменным одновременно:

```
>>> marx_tuple = ('Groucho', 'Chico', 'Harpo')
>>> a, b, c = marx_tuple
>>> a
'Groucho'
>>> b
'Chico'
>>> c
'Harpo'
```

Иногда это называется *распаковкой кортежа*.

Вы можете использовать кортежи для обмена значениями с помощью одного выражения, не применяя временную переменную:

```
>>> password = 'swordfish'
>>> icecream = 'tuttifrutti'
>>> password, icecream = icecream, password
>>> password
'tuttifrutti'
>>> icecream
'swordfish'
>>>
```

## Создаем кортежи с помощью функции tuple()

Функция преобразования tuple() создает кортежи из других объектов:

```
>>> marx_list = ['Groucho', 'Chico', 'Harpo']
>>> tuple(marx_list)
('Groucho', 'Chico', 'Harpo')
```

## Объединяем кортежи с помощью оператора +

Это похоже на объединение строк:

```
>>> ('Groucho',) + ('Chico', 'Harpo')
('Groucho', 'Chico', 'Harpo')
```

## Размножаем элементы с помощью оператора \*

Принцип похож на многократное использование оператора +:

```
>>> ('yada',) * 3
('yada', 'yada', 'yada')
```

## Сравниваем кортежи

Сравнение кортежей похоже на сравнение списков:

```
>>> a = (7, 2)
>>> b = (7, 2, 9)
>>> a == b
False
>>> a <= b
True
>>> a < b
True
```

## Итерируем по кортежам с помощью for и in

Итерирование по кортежам выполняется так же, как и итерирование по другим типам:

```
>>> words = ('fresh', 'out', 'of', 'ideas')
>>> for word in words:
...     print(word)
...
fresh
out
of
ideas
```

## Изменяем кортеж

Этого сделать вы не можете! Как и строки, кортежи неизменяемы. Но ранее вы уже видели на примере строк, что можно *сконкатенировать* (объединить) кортежи и создать таким образом новый кортеж:

```
>>> t1 = ('Fee', 'Fie', 'Foe')
>>> t2 = ('Flop',)
>>> t1 + t2
('Fee', 'Fie', 'Foe', 'Flop')
```

Это означает, что вы можете изменить кортеж следующим образом:

```
>>> t1 = ('Fee', 'Fie', 'Foe')
>>> t2 = ('Flop',)
>>> t1 += t2
>>> t1
('Fee', 'Fie', 'Foe', 'Flop')
```

Это уже не тот же самый кортеж `t1`. Python создал новый кортеж из исходных `t1` и `t2` и присвоил ему имя `t1`. С помощью `id()` вы можете увидеть, когда имя переменной будет указывать на новое значение:

```
>>> t1 = ('Fee', 'Fie', 'Foe')
>>> t2 = ('Flop',)
>>> id(t1)
4365405712
>>> t1 += t2
>>> id(t1)
4364770744
```

## Списки

Списки особенно удобны для хранения в них объектов в определенном порядке, особенно если порядок или содержимое нужно будет изменить. В отличие от строк список изменяем: вы можете добавить новые элементы, перезаписать существующие и удалить ненужные. Одно и то же значение может встречаться в списке несколько раз.

### Создаем списки с помощью скобок []

Список можно создать из нуля и более элементов, разделенных запятыми и заключенных в квадратные скобки:

```
>>> empty_list = [ ]
>>> weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
>>> big_birds = ['emu', 'ostrich', 'cassowary']
>>> first_names = ['Graham', 'John', 'Terry', 'Terry', 'Michael']
>>> leap_years = [2000, 2004, 2008]
>>> randomness = ['Punxsatawney', {"groundhog": "Phil"}, "Feb. 2"]
```

Список `first_names` показывает, что значения не должны быть уникальными.



Если вы хотите размещать в последовательности только уникальные значения и вам неважен их порядок, множество (`set`) может оказаться более удобным вариантом, чем список. В предыдущем примере список `big_birds` вполне мог быть множеством. О множествах вы прочитаете в главе 8.

### Создаем список или преобразуем в список с помощью функции list()

Вы также можете создать пустой список с помощью функции `list()`:

```
>>> another_empty_list = list()
>>> another_empty_list
[]
```

Функция `list()` преобразует другие *итерабельные* типы данных (например, кортежи, строки, множества и словари) в списки. В следующем примере строка преобразуется в список, состоящий из односимвольных строк:

```
>>> list('cat')
['c', 'a', 't']
```

В этом примере кортеж преобразуется в список:

```
>>> a_tuple = ('ready', 'fire', 'aim')
>>> list(a_tuple)
['ready', 'fire', 'aim']
```

## Создаем список из строки с использованием функции `split()`

Как я упоминал в разделе «Разделяем строку с помощью функции `split()`» главы 5, функцию `split()` можно использовать для преобразования строки в список, указав некую строку-разделитель:

```
>>> talk_like_a_pirate_day = '9/19/2019'
>>> talk_like_a_pirate_day.split('/')
['9', '19', '2019']
```

Что, если в оригинальной строке содержится несколько включений строки-разделителя подряд? В этом случае в качестве элемента списка вы получите пустую строку:

```
>>> splitme = 'a/b//c/d///e'
>>> splitme.split('/')
['a', 'b', '', 'c', 'd', '', '', 'e']
```

Если бы вы использовали разделитель `//`, состоящий из двух символов, то получили бы следующий результат:

```
>>> splitme = 'a/b//c/d///e'
>>> splitme.split('//')
>>>
['a/b', 'c/d', '/e']
```

## Получаем элемент с помощью конструкции [смещение]

Как и в случае со строками, вы можете извлечь одно значение из списка, указав его смещение:

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> marxes[0]
'Groucho'
>>> marxes[1]
'Chico'
>>> marxes[2]
'Harpo'
```



Так же и отрицательные индексы отсчитываются с конца строки:

```
>>> marxex[-1]
'Harpo'
>>> marxex[-2]
'Chico'
>>> marxex[-3]
'Groucho'
>>>
```



Смещение должно быть допустимым для этого списка — позицией, которой вы ранее присвоили значение. Если вы укажете позицию, которая находится перед списком или после него, будет сгенерировано исключение (ошибка). Вот что случится, если мы попробуем получить шестого брата Маркс (Marxes) (смещение равно 5, если считать от нуля) или же пятого перед списком:

```
>>> marxex = ['Groucho', 'Chico', 'Harpo']
>>> marxex[5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> marxex[-5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

## Извлекаем элементы с помощью разделения

Можно извлечь из списка подсписок, используя *разделение* (slice):

```
>>> marxex = ['Groucho', 'Chico', 'Harpo']
>>> marxex[0:2]
['Groucho', 'Chico']
```

Такой фрагмент списка тоже является списком.

Как и в случае со строками, при разделении можно пропускать некоторые значения. В следующем примере мы извлечем каждый нечетный элемент:

```
>>> marxex[::2]
['Groucho', 'Harpo']
```

Теперь начнем с последнего элемента и будем смещаться влево на 2:

```
>>> marxex[::-2]
['Harpo', 'Groucho']
```

И наконец, рассмотрим прием инверсии списка:

```
>>> marxex[::-1]
['Harpo', 'Chico', 'Groucho']
```

Ни одно из этих разделений не затронуло сам список `marxes`, поскольку мы не выполняли присваивание. Чтобы изменить порядок элементов в списке, используйте функцию `list.reverse()`:

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> marxes.reverse()
>>> marxes
['Harpo', 'Chico', 'Groucho']
```




---

Функция `reverse()` изменяет список, но не возвращает его значения.

---

Как и в случае со строками, если при разделении указать некорректный индекс, исключение не генерируется. Будет использован ближайший корректный индекс или же возвращено пустое значение:

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> marxes[4:]
[]
>>> marxes[-6:]
['Groucho', 'Chico', 'Harpo']
>>> marxes[-6:-2]
['Groucho']
>>> marxes[-6:-4]
[]
```

## Добавляем элемент в конец списка с помощью функции `append()`

Традиционный способ добавления элементов в список — вызов метода `append()`, который один за одним добавит их в конец списка. В предыдущих примерах мы забыли о `Zeppo`, но ничего страшного не случилось, поскольку список можно изменить. Добавим его прямо сейчас:

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> marxes.append('Zeppo')
>>> marxes
['Groucho', 'Chico', 'Harpo', 'Zeppo']
```

## Добавляем элемент на определенное место с помощью функции `insert()`

Функция `append()` добавляет элементы только в конец списка. Когда вам нужно добавить элемент и поставить его на заданную позицию, используйте функцию `insert()`. Если вы укажете смещение `0`, элемент будет добавлен в начало списка. Если значение смещения выходит за пределы списка, элемент будет добавлен в ко-

нец, как делает и функция `append()`: таким образом, вам не нужно беспокоиться о том, что Python сгенерирует исключение:

```
>>> marxex = ['Groucho', 'Chico', 'Harpo']
>>> marxex.insert(2, 'Gummo')
>>> marxex
['Groucho', 'Chico', 'Harpo', 'Gummo']
>>> marxex.insert(10, 'Zeppo')
>>> marxex
['Groucho', 'Chico', 'Harpo', 'Gummo', 'Zeppo']
```

## Размножаем элементы с помощью оператора \*

В главе 5 вы видели, что можно размножить символы строки с помощью оператора `*`. Точно так же можно сделать и со списками:

```
>>> ["blah"] * 3
['blah', 'blah', 'blah']
```

## Объединяем списки с помощью метода `extend()` или оператора `+`

Можно объединить один список с другим, используя `extend()`. Предположим, что некий добрый человек дал нам новый список братьев Маркс, который называется `others`, и мы хотим добавить его в основной список `marxex`:

```
>>> marxex = ['Groucho', 'Chico', 'Harpo', 'Zeppo']
>>> others = ['Gummo', 'Karl']
>>> marxex.extend(others)
>>> marxex
['Groucho', 'Chico', 'Harpo', 'Zeppo', 'Gummo', 'Karl']
```

Можно также использовать операторы `+` или `+=`:

```
>>> marxex = ['Groucho', 'Chico', 'Harpo', 'Zeppo']
>>> others = ['Gummo', 'Karl']
>>> marxex += others
>>> marxex
['Groucho', 'Chico', 'Harpo', 'Zeppo', 'Gummo', 'Karl']
```

Если бы мы использовали `append()`, список `others` был бы добавлен как *один* из элементов списка, а не дополнил бы своими элементами список `marxex`:

```
>>> marxex = ['Groucho', 'Chico', 'Harpo', 'Zeppo']
>>> others = ['Gummo', 'Karl']
>>> marxex.append(others)
>>> marxex
['Groucho', 'Chico', 'Harpo', 'Zeppo', ['Gummo', 'Karl']]
```

Это еще раз показывает, что список может содержать элементы разных типов. В этом случае — четыре строки и список из двух строк.

## Изменяем элемент с помощью конструкции [смещение]

Так же как значение какого-либо элемента из списка можно получить по смещению, его можно и изменить:

```
>>> marxex = ['Groucho', 'Chico', 'Harpo']
>>> marxex[2] = 'Wanda'
>>> marxex
['Groucho', 'Chico', 'Wanda']
```

Опять же смещение должно быть корректным для заданного списка.

Вы не можете таким способом изменить символ в строке, поскольку строки, в отличие от списков, неизменяемы. В списке можно изменить как количество элементов, так и сами элементы.

## Изменяем элементы с помощью разделения

В предыдущем разделе вы увидели, как получить подсписок с помощью разделения. Помимо этого, с помощью разделения можно присвоить значения под-списку:

```
>>> numbers = [1, 2, 3, 4]
>>> numbers[1:3] = [8, 9]
>>> numbers
[1, 8, 9, 4]
```

То, что находится справа от = и что вы присваиваете списку, может содержать иное количество элементов, нежели список, указанный слева:

```
>>> numbers = [1, 2, 3, 4]
>>> numbers[1:3] = [7, 8, 9]
>>> numbers
[1, 7, 8, 9, 4]
```

```
>>> numbers = [1, 2, 3, 4]
>>> numbers[1:3] = []
>>> numbers
[1, 4]
```

На самом деле то, что находится справа от оператора присваивания, может даже не быть списком. Подойдет любой итерабельный объект, элементы которого можно сделать элементами списка:

```
>>> numbers = [1, 2, 3, 4]
>>> numbers[1:3] = (98, 99, 100)
>>> numbers
[1, 98, 99, 100, 4]
```

```
>>> numbers = [1, 2, 3, 4]
>>> numbers[1:3] = 'wat?'
>>> numbers
[1, 'w', 'a', 't', '?', 4]
```