УДК 004.438Python:004.6
ББК 32.973.22
М15

Уэс Маккинни

М15 Python и анализ данных: Первичная обработка данных с применением pandas, NumPy и Jupiter / пер. с англ. А. А. Слинкина. 3-е изд. – М.: МК Пресс, 2023. – 536 с.: ил.

ISBN 978-5-93700-174-0

Перед вами авторитетный справочник по переформатированию, очистке и обработке наборов данных на Python. Третье издание, переработанное с учетом версий Python 3.10 и pandas 1.4, содержит практические примеры, демонстрирующие эффективное решение широкого круга задач анализа данных. По ходу дела вы узнаете о последних версиях pandas, NumPy и Jupyter.

Книга принадлежит перу Уэса Маккинни, создателя библиотеки pandas, и может служить практическим современным руководством по инструментарию науки о данных на Python. Она идеально подойдет как аналитикам, только начинающим осваивать Python, так и программистам на Python, еще незнакомым с наукой о данных и научными приложениями. Файлы данных и прочие материалы к книге находятся в репозитории на GitHub и на сайте издательства *dmkpress.com*.

УДК 004.438Python:004.6
ББК 32.973.22

ISBN (анг.) 978-1-09810-403-0
ISBN (рус.) 978-5-93700-174-0

Оглавление

Об авторе **13**

Об иллюстрации на обложке **14**

Предисловие от издательства **15**

Предисловие **16**

Графические выделения 16
О примерах кода 17
Как с нами связаться 17
Благодарности 18

Глава 1. Предварительные сведения **22**

1.1. О чем эта книга? 22
Какого рода данные? 22
1.2. Почему именно Python? 23
Python как клей 23
Решение проблемы «двух языков» 24
Недостатки Python 24
1.3. Необходимые библиотеки для Python 25
NumPy 25
pandas 25
matplotlib 27
IPython и Jupyter 27
SciPy 28
scikit-learn 28
statsmodels 29
1.4. Установка и настройка 29
Miniconda в Windows 30
GNU/Linux 30
Miniconda в macOS 31
Установка необходимых пакетов 32
Интегрированные среды разработки (IDE) 33
1.5. Сообщество и конференции 33
1.6. Структура книги 34
Примеры кода 35
Данные для примеров 35
Соглашения об импорте 36

Глава 2. Основы языка Python, IPython и Jupyter-блокноты..... 37

2.1. Интерпретатор Python.....38
2.2. Основы IPython 39
Запуск оболочки IPython.....39
Запуск Jupyter-блокнота.....40
Завершение по нажатии клавиши Tab.....43
Интроспекция.....45
2.3. Основы языка Python.....46
Семантика языка 46
Скалярные типы 53
Поток управления.....61
2.4. Заключение 64

Глава 3. Встроенные структуры данных, функции и файлы..... 65

3.1. Структуры данных и последовательности 65
Кортеж 65
Список 68
Словарь.....72
Множество.....76
Встроенные функции последовательностей 78
Списковое, словарное и множественное включения.....80
3.2. Функции.....82
Пространства имен, области видимости и локальные функции 83
Возврат нескольких значений 84
Функции являются объектами.....85
Анонимные (лямбда-) функции 87
Генераторы.....87
Обработка исключений.....90
3.3. Файлы и операционная система.....92
Байты и Unicode в применении к файлам 96
3.4. Заключение 98

Глава 4. Основы NumPy: массивы и векторные вычисления 99

4.1. NumPy ndarray: объект многомерного массива.....101
Создание ndarray 102
Тип данных для ndarray 104
Арифметические операции с массивами NumPy.....107
Индексирование и вырезание.....108
Булево индексирование 113
Прихотливое индексирование.....116
Транспонирование массивов и перестановка осей 117
4.2. Генерирование псевдослучайных чисел.....119
4.3. Универсальные функции: быстрые поэлементные операции над массивами 120
4.4. Программирование на основе массивов.....123
Запись логических условий в виде операций с массивами.....125

Математические и статистические операции ... 126
Методы булевых массивов ... 128
Сортировка ... 128
Устранение дубликатов и другие теоретико-множественные операции ... 130
4.5. Файловый ввод-вывод массивов ... 130
4.6. Линейная алгебра ... 131
4.7. Пример: случайное блуждание ... 133
Моделирование сразу нескольких случайных блужданий ... 135
4.8. Заключение ... 136

Глава 5. Первое знакомство с pandas ... 137

5.1. Введение в структуры данных pandas ... 138
Объект Series ... 138
Объект DataFrame ... 142
Индексные объекты ... 149
5.2. Базовая функциональность ... 151
Переиндексация ... 151
Удаление элементов из оси ... 154
Доступ по индексу, выборка и фильтрация ... 155
Арифметические операции и выравнивание данных ... 165
Применение функций и отображение ... 170
Сортировка и ранжирование ... 172
Индексы по осям с повторяющимися значениями ... 175
5.3. Редукция и вычисление описательных статистик ... 177
Корреляция и ковариация ... 180
Уникальные значения, счетчики значений и членство ... 181
5.4. Заключение ... 185

Глава 6. Чтение и запись данных, форматы файлов ... 186

6.1. Чтение и запись данных в текстовом формате ... 186
Чтение текстовых файлов порциями ... 193
Вывод данных в текстовом формате ... 195
Обработка данных в других форматах с разделителями ... 196
Данные в формате JSON ... 198
XML и HTML: разбор веб-страниц ... 200
6.2. Двоичные форматы данных ... 203
Формат HDF5 ... 205
6.3. Взаимодействие с HTML и Web API ... 208
6.4. Взаимодействие с базами данных ... 209
6.5. Заключение ... 211

Глава 7. Очистка и подготовка данных ... 212

7.1. Обработка отсутствующих данных ... 212
Фильтрация отсутствующих данных ... 214
Восполнение отсутствующих данных ... 216

7.2. Преобразование данных ... 218
Устранение дубликатов ... 218
Преобразование данных с помощью функции или отображения ... 220
Замена значений ... 221
Переименование индексов осей ... 222
Дискретизация и группировка по интервалам ... 223
Обнаружение и фильтрация выбросов ... 226
Перестановки и случайная выборка ... 227
Вычисление индикаторных переменных ... 229
7.3. Расширение типов данных ... 232
7.4. Манипуляции со строками ... 235
Встроенные методы строковых объектов ... 235
Регулярные выражения ... 237
Строковые функции в pandas ... 240
7.5. Категориальные данные ... 243
Для чего это нужно ... 244
Расширенный тип Categorical в pandas ... 245
Вычисления с объектами Categorical ... 248
Категориальные методы ... 250
7.6. Заключение ... 253

Глава 8. Переформатирование данных: соединение, комбинирование и изменение формы ... 254

8.1. Иерархическое индексирование ... 254
Переупорядочение и уровни сортировки ... 257
Сводная статистика по уровню ... 258
Индексирование столбцами DataFrame ... 258
8.2. Комбинирование и слияние наборов данных ... 260
Слияние объектов DataFrame как в базах данных ... 260
Соединение по индексу ... 265
Конкатенация вдоль оси ... 269
Комбинирование перекрывающихся данных ... 274
8.3. Изменение формы и поворот ... 276
Изменение формы с помощью иерархического индексирования ... 276
Поворот из «длинного» в «широкий» формат ... 279
Поворот из «широкого» в «длинный» формат ... 282
8.4. Заключение ... 284

Глава 9. Построение графиков и визуализация ... 285

9.1. Краткое введение в API библиотеки matplotlib ... 286
Рисунки и подграфики ... 287
Цвета, маркеры и стили линий ... 291
Риски, метки и надписи ... 292
Аннотации и рисование в подграфике ... 295
Сохранение графиков в файле ... 297

Конфигурирование matplotlib 298
9.2. Построение графиков с помощью pandas и seaborn 299
Линейные графики 299
Столбчатые диаграммы 302
Гистограммы и графики плотности 308
Диаграммы рассеяния 310
Фасетные сетки и категориальные данные 313
9.3. Другие средства визуализации для Python 315
9.4. Заключение 316

Глава 10. Агрегирование данных и групповые операции 317

10.1. Как представлять себе групповые операции 318
Обход групп 322
Выборка столбца или подмножества столбцов 323
Группировка с помощью словарей и объектов Series 324
Группировка с помощью функций 325
Группировка по уровням индекса 325
10.2. Агрегирование данных 326
Применение функций, зависящих от столбца, и нескольких функций 328
Возврат агрегированных данных без индексов строк 332
10.3. Метод apply: общий принцип разделения–применения–объединения 332
Подавление групповых ключей 334
Квантильный и интервальный анализы 335
Пример: подстановка зависящих от группы значений вместо отсутствующих 337
Пример: случайная выборка и перестановка 339
Пример: групповое взвешенное среднее и корреляция 341
Пример: групповая линейная регрессия 343
10.4. Групповые преобразования и «развернутая» группировка 343
10.5. Сводные таблицы и перекрестная табуляция 347
Перекрестная табуляция: crosstab 350
10.5. Заключение 351

Глава 11. Временные ряды 352

11.1. Типы данных и инструменты, относящиеся к дате и времени 353
Преобразование между строкой и datetime 354
11.2. Основы работы с временными рядами 356
Индексирование, выборка, подмножества 358
Временные ряды с неуникальными индексами 360
11.3. Диапазоны дат, частоты и сдвиг 361
Генерирование диапазонов дат 362
Частоты и смещения дат 364
Сдвиг данных (с опережением и с запаздыванием) 366
11.4. Часовые пояса 369
Локализация и преобразование 369

Операции над объектами Timestamp с учетом часового пояса ... 371
Операции над датами из разных часовых поясов ... 372
11.5. Периоды и арифметика периодов ... 373
Преобразование частоты периода ... 374
Квартальная частота периода ... 376
Преобразование временных меток в периоды и обратно ... 377
Создание PeriodIndex из массивов ... 379
11.6. Передискретизация и преобразование частоты ... 380
Понижающая передискретизация ... 382
Повышающая передискретизация и интерполяция ... 384
Передискретизация периодов ... 386
Групповая передискретизация по времени ... 387
11.7. Скользящие оконные функции ... 389
Экспоненциально взвешенные функции ... 392
Бинарные скользящие оконные функции ... 394
Скользящие оконные функции, определенные пользователем ... 395
11.8. Заключение ... 396

Глава 12. Введение в библиотеки моделирования на Python ... 397

12.1. Интерфейс между pandas и кодом модели ... 397
12.2. Описание моделей с помощью Patsy ... 400
Преобразование данных в формулах Patsy ... 402
Категориальные данные и Patsy ... 404
12.3. Введение в statsmodels ... 406
Оценивание линейных моделей ... 407
Оценивание процессов с временными рядами ... 409
12.4. Введение в scikit-learn ... 410
12.5. Заключение ... 414

Глава 13. Примеры анализа данных ... 415

13.1. Набор данных Bitly с сайта 1.usa.gov ... 415
Подсчет часовых поясов на чистом Python ... 416
Подсчет часовых поясов с помощью pandas ... 418
13.2. Набор данных MovieLens 1M ... 424
Измерение несогласия в оценках ... 428
13.3. Имена, которые давали детям в США за период с 1880 по 2010 год ... 432
Анализ тенденций в выборе имен ... 437
13.4. База данных о продуктах питания министерства сельского хозяйства США ... 446
13.5. База данных Федеральной избирательной комиссии ... 451
Статистика пожертвований по роду занятий и месту работы ... 454
Распределение суммы пожертвований по интервалам ... 457
Статистика пожертвований по штатам ... 459
13.6. Заключение ... 460

Приложение A. Дополнительные сведения о библиотеке NumPy 461

A.1. Внутреннее устройство объекта ndarray 461
Иерархия типов данных в NumPy 462
A.2. Дополнительные манипуляции с массивами 463
Изменение формы массива 464
Упорядочение элементов массива в C и в Fortran 465
Конкатенация и разбиение массива 466
Эквиваленты прихотливого индексирования: функции take и put 470
A.3. Укладывание 471
Укладывание по другим осям 474
Установка элементов массива с помощью укладывания 476
A.4. Дополнительные способы использования универсальных функций 477
Методы экземпляра u-функций 477
Написание новых u-функций на Python 479
A.5. Структурные массивы и массивы записей 480
Вложенные типы данных и многомерные поля 481
Зачем нужны структурные массивы? 482
A.6. Еще о сортировке 482
Косвенная сортировка: методы argsort и lexsort 483
Альтернативные алгоритмы сортировки 485
Частичная сортировка массивов 485
Метод numpy.searchsorted: поиск элементов в отсортированном массиве 486
A.7. Написание быстрых функций для NumPy с помощью Numba 487
Создание пользовательских объектов numpy.ufunc с помощью Numba 489
A.8. Дополнительные сведения о вводе-выводе массивов 489
Файлы, отображенные на память 489
HDF5 и другие варианты хранения массива 491
A.9. Замечания о производительности 491
Важность непрерывной памяти 491

Приложение B. Еще о системе IPython 494

B.1. Комбинации клавиш 494
B.2. О магических командах 495
Команда %run 497
Исполнение кода из буфера обмена 498
B.3. История команд 499
Поиск в истории команд и повторное выполнение 500
Входные и выходные переменные 500
B.4. Взаимодействие с операционной системой 501
Команды оболочки и псевдонимы 502
Система закладок на каталоги 503

B.5. Средства разработки программ 504
Интерактивный отладчик 504
Хронометраж программы: %time и %timeit 508
Простейшее профилирование: %prun и %run -p 510
Построчное профилирование функции 512
B.6. Советы по продуктивной разработке кода с использованием IPython 514
Перезагрузка зависимостей модуля 514
Советы по проектированию программ 515
B.7. Дополнительные возможности IPython 516
Профили и конфигурирование 516
B.8. Заключение 517

Предметный указатель 518

Об авторе

Уэс Маккинни – разработчик программного обеспечения и предприниматель из Нэшвилла. Получив степень бакалавра математики в МТИ в 2007 году, он поступил на работу в компанию AQR Capital Management в Гринвиче, где занимался финансовой математикой. Неудовлетворенный малопригодными средствами анализа данных, Уэс изучил язык Python и приступил к созданию того, что в будущем стало проектом pandas. Сейчас он активный член сообщества обработки данных на Python и агитирует за использование Python в анализе данных, финансовых задачах и математической статистике.

Впоследствии Уэс стал сооснователем и генеральным директором компании DataPad, технологические активы и коллектив которой в 2014 году приобрела компания Cloudera. С тех пор он занимается технологиями больших данных и является членом комитетов по управлению проектами Apache Arrow и Apache Parquet, курируемых фондом Apache Software Foundation. В 2018 году он основал компанию Ursa Labs, некоммерческую организацию, ориентированную на разработку проекта Apache Arrow в партнерстве с компаниями RStudio и Two Sigma Investments. В 2021 году вошел в число учредителей технологического стартапа Voltron Data, где в настоящее время занимает пост технического директора.

Предисловие

Первое издание этой книги вышло в 2012 году, когда Python-библиотеки для анализа данных с открытым исходным кодом (в частности, pandas) были еще внове и быстро развивались. Когда в 2016–2017 годах приспело время написать второе издание, мне пришлось не только привести текст в соответствие с версией Python (в первом издании использовалась версия Python 2.7), но и учесть многочисленные изменения, внесенные в pandas за прошедшие пять лет. Теперь, в 2022 году, изменений в Python оказалось меньше (сейчас текущей является версия 3.10, а на подходе 3.11), зато развитие pandas продолжилось.

В третьем издании я ставил целью привести материал в соответствие с текущими версиями Python, NumPy, pandas и другими проектами, не слишком увлекаясь обсуждением новых проектов на Python, появившихся за последние пять лет. Поскольку книга стала важным ресурсом для многих университетских курсов и практикующих профессионалов, я постараюсь избегать тем, которые рискуют устареть через год-другой. Это позволит сохранить актуальность печатных экземпляров в 2023–2024 годах и, возможно, в более длительной перспективе.

В третьем издании добавилось новшество – открытый доступ к онлайновой версии, размещенной на моем сайте *https://wesmckinney.com/book*. Это будет удобно для владельцев печатных и цифровых версий книги. Я собираюсь поддерживать текст в более-менее актуальном состоянии, поэтому если вы встретите в печатной версии что-то не работающее, как должно, всегда можно будет справить с последними обновлениями.

Графические выделения

В книге применяются следующие графические выделения.

Курсив
: Новые термины, URL-адреса, адреса электронной почты, имена и расширения имен файлов.

`Моноширинный`
: Листинги программ, а также элементы кода в основном тексте: имена переменных и функций, базы данных, типы данных, переменные окружения, предложения и ключевые слова языка.

`Моноширинный полужирный`
: Команды или иной текст, который должен быть введен пользователем буквально.

`Моноширинный курсив`
: Текст, вместо которого следует подставить значения, заданные пользователем или определяемые контекстом.

Так обозначается совет или рекомендация.

Так обозначается замечание общего характера.

Так обозначается предупреждение или предостережение.

О ПРИМЕРАХ КОДА

Файлы данных и прочие материалы, организованные по главам, можно найти в репозитории книги на GitHub по адресу *http://github.com/wesm/pydata-book* или на его зеркале на Gitee (для тех, у кого нет доступа к GitHub) по адресу *https://gitee.com/wesmckinn/pydata-book*.

Эта книга призвана помогать вам в работе. Поэтому вы можете использовать приведенный в ней код в собственных программах и в документации. Спрашивать у нас разрешение необязательно, если только вы не собираетесь воспроизводить значительную часть кода. Например, никто не возбраняет включить в свою программу несколько фрагментов кода из книги. Однако для продажи или распространения примеров из книг издательства O'Reilly на компакт-диске разрешение требуется. Цитировать книгу и примеры в ответах на вопросы можно без ограничений. Но для включения значительных объемов кода в документацию по собственному продукту нужно получить разрешение.

Мы высоко ценим, хотя и не требуем, ссылки на наши издания. В ссылке обычно указываются название книги, имя автора, издательство и ISBN, например: «Python for Data Analysis by Wes McKinney (O'Reilly). Copyright 2022 Wes McKinney, 78-1-098-10403-0».

Если вы полагаете, что планируемое использование кода выходит за рамки изложенной выше лицензии, пожалуйста, обратитесь к нам по адресу *permissions@oreilly.com*.

Глава 1

Предварительные сведения

1.1. О чем эта книга?

Эта книга посвящена вопросам преобразования, обработки, очистки данных и вычислениям на языке Python. Моя цель – предложить руководство по тем частям языка программирования Python и экосистемы его библиотек и инструментов, относящихся к обработке данных, которые помогут вам стать хорошим аналитиком данных. Хотя в названии книги фигурируют слова «анализ данных», основной упор сделан на программировании на Python, библиотеках и инструментах, а не на методологии анализа данных как таковой. Речь идет о программировании на Python, необходимом *для* анализа данных.

Спустя некоторое время после выхода этой книги в 2012 году термин «наука о данных» (data science) стали употреблять для всего на свете: от простой описательной статистики до более сложного статистического анализа и машинного обучения. Открытая экосистема для анализа данных (или науки о данных) на Python с тех пор также значительно расширилась. Сейчас имеется много книг, посвященных специально более продвинутым методологиям. Льщу себя надеждой, что эта книга подготовит вас к изучению ресурсов, ориентированных на конкретные области применения.

Возможно, кто-то сочтет, что книга в большей степени посвящена «манипулированию данными», а не «анализу данных». Мы используем также термины «первичная обработка данных» (data wrangling) и «подготовка данных» (data munging) в качестве синонимов «манипулированию данными».

Какого рода данные?

Говоря «данные», я имею в виду прежде всего *структурированные данные*; это намеренно расплывчатый термин, охватывающий различные часто встречающиеся виды данных, как то:

- табличные данные, когда данные в разных столбцах могут иметь разный тип (строки, числа, даты или еще что-то). Сюда относятся данные, которые обычно хранятся в реляционных базах или в файлах с запятой в качестве разделителя;
- многомерные списки (матрицы);

- данные, представленные в виде нескольких таблиц, связанных между собой по ключевым столбцам (то, что в SQL называется первичными и внешними ключами);
- равноотстоящие и неравноотстоящие временные ряды.

Этот список далеко не полный. Значительную часть наборов данных можно преобразовать к структурированному виду, более подходящему для анализа и моделирования, хотя сразу не всегда очевидно, как это сделать. В тех случаях, когда это не удается, иногда есть возможность извлечь из набора данных структурированное множество признаков. Например, подборку новостных статей можно преобразовать в таблицу частот слов, к которой затем применить анализ эмоциональной окраски.

Большинству пользователей электронных таблиц типа Microsoft Excel, пожалуй, самого широко распространенного средства анализа данных, такие виды данных хорошо знакомы.

1.2. Почему именно Python?

Для многих людей (и меня в том числе) Python – язык, в который нельзя не влюбиться. С момента своего появления в 1991 году Python стал одним из самых популярных динамических языков программирования наряду с Perl, Ruby и другими. Относительно недавно Python и Ruby приобрели особую популярность как средства создания веб-сайтов в многочисленных каркасах, например Rails (Ruby) и Django (Python). Такие языки часто называют *скриптовыми*, потому что они используются для быстрого написания небольших программ – *скриптов*. Лично мне термин «скриптовый язык» не нравится, поскольку он наводит на мысль, будто для создания ответственного программного обеспечения язык не годится. Из всех интерпретируемых языков Python выделяется большим и активным сообществом научных расчетов и анализа данных. За последние 20 лет Python перешел из разряда ультрасовременного языка научных расчетов, которым пользуются на свой страх и риск, в один из самых важных языков, применяемых в науке о данных, машинном обучении и разработке ПО общего назначения в академических учреждениях и промышленности.

В области анализа данных и интерактивных научно-исследовательских расчетов с визуализацией результатов Python неизбежно приходится сравнивать со многими предметно-ориентированными языками программирования и инструментами – с открытым исходным кодом и коммерческими, такими как R, MATLAB, SAS, Stata и другими. Сравнительно недавнее появление улучшенных библиотек для Python (прежде всего pandas) сделало его серьезным конкурентом в решении задач манипулирования данными. В сочетании с достоинствами Python как универсального языка программирования это делает его отличным выбором для создания приложений обработки данных.

Python как клей

Своим успехом в области научных расчетов Python отчасти обязан простоте интеграции с кодом на C, C++ и FORTRAN. Во многих современных вычислительных средах применяется общий набор унаследованных библиотек, напи-

санных на FORTRAN и C, содержащих реализации алгоритмов линейной алгебры, оптимизации, интегрирования, быстрого преобразования Фурье и других. Поэтому многочисленные компании и национальные лаборатории используют Python как «клей» для объединения написанных за много лет программ.

Большинство программ содержат небольшие участки кода, на выполнение которых уходит большая часть времени, и большие куски «склеивающего кода», который выполняется нечасто. Во многих случаях время выполнения склеивающего кода несущественно, реальную отдачу дает оптимизация узких мест, которые иногда имеет смысл переписать на низкоуровневом языке типа C.

Решение проблемы «двух языков»

Во многих организациях принято для научных исследований, создания опытных образцов и проверки новых идей использовать предметно-ориентированные языки типа MATLAB или R, а затем переносить удачные разработки в производственную систему, написанную на Java, C# или C++. Но все чаще люди приходят к выводу, что Python подходит не только для стадий исследования и создания прототипа, но и для построения самих производственных систем. Я полагаю, что компании все чаще будут выбирать этот путь, потому что использование одного и того же набора программных средств учеными и технологами несет несомненные выгоды организации.

За последнее десятилетие появились новые подходы к решению проблемы «двух языков», в частности язык программирования Julia. Во многих случаях для получения максимальной пользы от Python *необходимо* программировать на языке более низкого уровня типа C или C++ и создавать интерфейс между таким кодом и Python. Вместе с тем технология «своевременных» (JIT) компиляторов, предлагаемая такими библиотеками, как Numba, позволила добиваться великолепной производительности многих численных алгоритмов, не покидая среду программирования на Python.

Недостатки Python

Python – великолепная среда для создания приложений для научных расчетов и большинства систем общего назначения, но тем не менее существуют задачи, для которых Python не очень подходит.

Поскольку Python – интерпретируемый язык программирования, в общем случае написанный на нем код работает значительно медленнее, чем эквивалентный код на компилируемом языке типа Java или C++. Но поскольку *время программиста* обычно стоит гораздо дороже *времени процессора*, многих такой компромисс устраивает. Однако в приложениях, где задержка должна быть очень мала (например, в торговых системах с большим количеством транзакций), время, потраченное на программирование на низкоуровневом и не обеспечивающем максимальную продуктивность языке типа C++, во имя достижения максимальной производительности, будет потрачено не зря.

Python – не идеальный язык для программирования многопоточных приложений с высокой степенью параллелизма, особенно при наличии многих потоков, активно использующих процессор. Проблема связана с наличием *глобальной блокировки интерпретатора* (GIL) – механизма, который не дает

интерпретатору исполнять более одной команды байт-кода Python в каждый момент времени. Объяснение технических причин существования GIL выходит за рамки этой книги, но на данный момент представляется, что GIL вряд ли скоро исчезнет. И хотя во многих приложениях обработки больших объектов данных для обеспечения приемлемого времени приходится организовывать кластер машин, встречаются все же ситуации, когда более желательна однопроцессная многопоточная система.

Я не хочу сказать, что Python вообще непригоден для исполнения многопоточного параллельного кода. Написанные на C или C++ расширения Python, пользующиеся платформенной многопоточностью, могут исполнять код параллельно и не ограничены механизмом GIL, при условии что им не нужно регулярно взаимодействовать с Python-объектами.

1.3. Необходимые библиотеки для Python

Для читателей, плохо знакомых с экосистемой Python и используемыми в книге библиотеками, я приведу краткий обзор библиотек.

NumPy

NumPy (*https://numpy.org/*), сокращение от «Numerical Python», – основной пакет для выполнения научных расчетов на Python. Большая часть этой книги базируется на NumPy и построенных поверх него библиотеках. В числе прочего он предоставляет:

- быстрый и эффективный объект многомерного массива *ndarray*;
- функции для выполнения вычислений над элементами одного массива или математических операций с несколькими массивами;
- средства для чтения и записи на диски наборов данных, представленных в виде массивов;
- операции линейной алгебры, преобразование Фурье и генератор случайных чисел;
- зрелый C API, позволяющий обращаться к структурам данных и вычислительным средствам NumPy из расширений Python и кода на C или C++.

Помимо быстрых средств работы с массивами, одной из основных целей NumPy в части анализа данных является организация контейнера для передачи данных между алгоритмами. Как средство хранения и манипуляции данными массивы NumPy куда эффективнее встроенных в Python структур данных. Кроме того, библиотеки, написанные на низкоуровневом языке типа C или Fortran, могут работать с данными, хранящимися в массиве NumPy, вообще без копирования в другое представление. Таким образом, многие средства вычислений, ориентированные на Python, либо используют массивы NumPy в качестве основной структуры данных, либо каким-то иным способом организуют интеграцию с NumPy.

pandas

Библиотека pandas (*https://pandas.pydata.org/*) предоставляет структуры данных и функции, призванные сделать работу со структурированными данны-

ми простым, быстрым и выразительным делом. С момента своего появления в 2010 году она способствовала превращению Python в мощную и продуктивную среду анализа данных. Основные объекты pandas, используемые в этой книге, – `DataFrame` – двумерная таблица, в которой строки и столбцы имеют метки, и `Series` – объект одномерного массива с метками.

Библиотека pandas сочетает высокую производительность средств работы с массивами, присущую NumPy, с гибкими возможностями манипулирования данными, свойственными электронным таблицам и реляционным базам данных (например, на основе SQL). Она предоставляет развитые средства индексирования, позволяющие без труда изменять форму наборов данных, формировать продольные и поперечные срезы, выполнять агрегирование и выбирать подмножества. Поскольку манипулирование данными, их подготовка и очистка играют огромное значение в анализе данных, в этой книге библиотека pandas будет одним из основных инструментов.

Если кому интересно, я начал разрабатывать pandas в начале 2008 года, когда работал в компании AQR Capital Management, занимающейся управлением инвестициями. Тогда я сформулировал специфический набор требований, которому не удовлетворял ни один отдельно взятый инструмент, имеющийся в моем распоряжении:

- структуры данных с помеченными осями, которые поддерживали бы автоматическое или явное выравнивание данных, – это предотвратило бы типичные ошибки, возникающие при работе с невыровненными данными и данными из разных источников, которые по-разному индексированы;
- встроенная функциональность временных рядов;
- одни и те же структуры данных должны поддерживать как временные ряды, так и данные других видов;
- арифметические операции и упрощения должны сохранять метаданные;
- гибкая обработка отсутствующих данных;
- поддержка слияния и других реляционных операций, имеющихся в популярных базах данных (например, на основе SQL).

Я хотел, чтобы вся эта функциональность была в одном месте и предпочтительно реализована на языке, хорошо приспособленном для разработки ПО общего назначения. Python выглядел хорошим кандидатом на эту роль, но в то время в нем не было подходящих встроенных структур данных и средств. Поскольку изначально библиотека pandas создавалась для решения финансовых задач и задач бизнес-аналитики, в ней особенно глубоко проработаны средства работы с временными рядами, ориентированные на обработку данных с временными метками, которые порождаются бизнес-процессами.

Значительную часть 2011 и 2012 годов я потратил на расширение возможностей pandas, в чем мне помогали бывшие коллеги по компании AQR Адам Клейн (Adam Klein) и Чань Ше (Chang She). В 2013 году я отошел от ежедневной работы по развитию проекта, и pandas перешла в полную собственность сообщества, в которое входит свыше двух тысяч программистов со всего мира.

Пользователям языка статистических расчетов R название DataFrame покажется знакомым, потому что оно выбрано по аналогии с объектом `data.frame` в R. В отличие от Python, фреймы данных уже встроены в язык R и его стандартную

библиотеку. Поэтому многие средства, присутствующие в pandas, либо являются частью ядра R, либо предоставляются дополнительными пакетами.

Само название pandas образовано как от *panel data* (панельные данные), применяемого в эконометрике термина для обозначения многомерных структурированных наборов данных, так и от фразы *Python data analysis*.

matplotlib

Библиотека matplotlib (*https://matplotlib.org/*) – самый популярный в Python инструмент для создания графиков и других способов визуализации двумерных данных. Первоначально она была написана Джоном Д. Хантером (John D. Hunter), а теперь сопровождается большой группой разработчиков. Она отлично подходит для создания графиков, пригодных для публикации. Хотя программистам на Python доступны и другие библиотеки визуализации, matplotlib используется чаще всего и потому хорошо интегрирована с другими частями экосистемы. На мой взгляд, если вам нужно средство визуализации, то это самый безопасный выбор.

IPython и Jupyter

Проект IPython (*http://ipython.org/*) начал в 2001 году Фернандо Перес (Fernando Perez) как побочный проект, имеющий целью создать более удобный интерактивный интерпретатор Python. За прошедшие с тех пор 16 лет он стал одним из самых важных элементов современного инструментария Python. Хотя IPython сам по себе не содержит никаких средств вычислений или анализа данных, он изначально спроектирован, имея в виду обеспечение максимальной продуктивности интерактивных вычислений и разработки ПО. Он поощряет цикл *выполнить–исследовать* вместо привычного цикла *редактировать–компилировать–выполнить*, свойственного многим другим языкам программирования. Кроме того, он позволяет легко обращаться к оболочке и файловой системе операционной системы. Поскольку написание кода анализа данных часто подразумевает исследование методом проб и ошибок и опробование разных подходов, то благодаря IPython работу удается выполнить быстрее.

В 2014 году Фернандо и команда разработки IPython анонсировали проект Jupyter (*https://jupyter.org/*) – широкую инициативу проектирования языковонезависимых средств интерактивных вычислений. Веб-блокнот IPython превратился в Jupyter-блокнот, который ныне поддерживает более 40 языков программирования. Систему IPython теперь можно использовать как *ядро* (языковой режим) для совместной работы Python и Jupyter.

Сам IPython стал компонентом более широкого проекта Jupyter с открытым исходным кодом, предоставляющего продуктивную среду для интерактивных исследовательских вычислений. В своем самом старом и простом «режиме» это улучшенная оболочка Python, имеющая целью ускорить написание, тестирование и отладку кода на Python. Систему IPython можно использовать также через Jupyter-блокнот.

Jupyter-блокноты позволяют создавать контент на языках разметки Markdown и HTML, т. е. готовить комбинированные документы, содержащие код и текст.

Лично я при работе с Python использую в основном IPython и Jupyter – для выполнения, отладки и тестирования кода.

В сопроводительных материалах к книге (*https://github.com/wesm/pydata-book*) вы найдете Jupyter-блокноты, содержащие примеры кода к каждой главе. Если у вас нет доступа к GitHub, попробуйте зеркало на сайте Gitee (*https://gitee.com/wesmckinn/pydata-book*).

SciPy

SciPy – собрание пакетов, предназначенных для решения различных стандартных вычислительных задач. Вот несколько из них.

`scipy.integrate`
: Подпрограммы численного интегрирования и решения дифференциальных уравнений.

`scipy.linalg`
: Подпрограммы линейной алгебры и разложения матриц, дополняющие те, что включены в `numpy.linalg`.

`scipy.optimize`
: Алгоритмы оптимизации функций (нахождения минимумов) и поиска корней.

`scipy.signal`
: Средства обработки сигналов.

`scipy.sparse`
: Алгоритмы работы с разреженными матрицами и решения разреженных систем линейных уравнений.

`scipy.special`
: Обертка вокруг SPECFUN, написанной на Fortran-библиотеке, содержащей реализации многих стандартных математических функций, в том числе гамма-функции.

`scipy.stats`
: Стандартные непрерывные и дискретные распределения вероятностей (функции плотности вероятности, формирования выборки, функции непрерывного распределения вероятности), различные статистические критерии и дополнительные описательные статистики.

Совместно NumPy и SciPy образуют достаточно полную замену значительной части системы MATLAB и многочисленных дополнений к ней.

scikit-learn

Проект scikit-learn (*https://scikit-learn.org/stable/*), запущенный в 2007 году, с самого начала стал основным инструментарием машинного обучения для программистов на Python. На момент написания книги сообщество насчитывает более 2000 разработчиков. В нем имеются подмодули для следующих моделей.

- Классификация: метод опорных векторов, метод ближайших соседей, случайные леса, логистическая регрессия и т. д.
- Регрессия: Lasso, гребневая регрессия и т. д.

- Кластеризация: метод *k*-средних, спектральная кластеризация и т. д.
- Понижение размерности: метод главных компонент, отбор признаков, матричная факторизация и т. д.
- Выбор модели: поиск на сетке, перекрестный контроль, метрики.
- Предобработка: выделение признаков, нормировка.

Наряду с pandas, statsmodels и IPython библиотека scikit-learn сыграла важнейшую роль для превращения Python в продуктивный язык программирования для науки о данных. Я не смогу включить в эту книгу полное руководство по scikit-learn, но все же приведу краткое введение в некоторые используемые в ней модели и объясню, как их применять совместно с другими средствами.

statsmodels

Пакет для статистического анализа statsmodels (*http://www.statsmodels.org/stable/index.html*) начал разрабатываться по инициативе профессора статистики из Стэнфордского университета Джонатана Тэйлора (Jonathan Taylor), который реализовал ряд моделей регрессионного анализа, популярных в языке программирования R. Скиппер Сиболд (Skipper Seabold) и Джозеф Перктольд (Josef Perktold) формально создали новый проект statsmodels в 2010 году, и с тех пор он набрал критическую массу заинтересованных пользователей и соразработчиков. Натаниэль Смит (Nathaniel Smith) разработал проект Patsy, который предоставляет средства для задания формул и моделей для statsmodels по образцу системы формул в R.

По сравнению со scikit-learn, пакет statsmodels содержит алгоритмы классической (прежде всего частотной) статистики и эконометрики. Он включает следующие подмодули:

- регрессионные модели: линейная регрессия, обобщенные линейные модели, робастные линейные модели, линейные модели со смешанными эффектами и т. д.;
- дисперсионный анализ (ANOVA);
- анализ временных рядов: AR, ARMA, ARIMA, VAR и другие модели;
- непараметрические методы: ядерная оценка плотности, ядерная регрессия;
- визуализация результатов статистического моделирования.

Пакет statsmodels в большей степени ориентирован на статистический вывод, он дает оценки неопределенности и *p*-значения параметров. Напротив, scikit-learn ориентирован главным образом на предсказание.

Как и для scikit-learn, я приведу краткое введение в statsmodels и объясню, как им пользоваться в сочетании с NumPy и pandas.

1.4. Установка и настройка

Поскольку Python используется в самых разных приложениях, не существует единственно верной процедуры установки Python и необходимых дополнительных пакетов. У многих читателей, скорее всего, нет среды, подходящей для научных применений Python и проработки этой книги, поэтому я приведу подробные инструкции для разных операционных систем. Я буду использовать Miniconda, минимальный дистрибутив менеджера пакетов conda, а вмес-

те с ним conda-forge (*https://conda-forge.org/*), поддерживаемый сообществом дистрибутив, основанный на conda. В этой книге всюду используется версия Python 3.10, но если вам доведется читать ее в будущем, то ничто не помешает установить более новую версию.

Если по какой-то причине эти инструкции устареют к моменту чтения, загляните на сайт книги (*https://wesmckinney.com/book/*), где я постараюсь выкладывать актуальные инструкции по установке.

Miniconda в Windows

Чтобы начать работу в Windows, скачайте последний из имеющихся установщиков Miniconda (в настоящее время для версии Python 3.9) по адресу *https://conda.io*. Я рекомендую следовать инструкциям по установке для Windows на сайте conda, которые могли измениться за время, прошедшее с момента выхода книги. На большинстве компьютеров установлена 64-разрядная версия, но если на вашей машине она не заработает, можете установить 32-разрядную.

В ответ на вопрос, хотите ли вы установить программу только для себя или для всех пользователей системы, решите, что вам больше подходит. Установки для себя достаточно, чтобы выполнять приведенные в книге примеры. Кроме того, установщик спросит, нужно ли добавлять путь к Miniconda в системную переменную окружения PATH. Если вы ответите утвердительно (я обычно так и делаю), то данная установка Miniconda может переопределить другие установленные версии Python. Если нет, то для запуска Miniconda нужно будет использовать добавленную в меню **Пуск** ссылку. Соответствующий пункт в меню **Пуск** может называться как-то вроде «Anaconda3 (64-bit)».

Я буду предполагать, что вы не добавляли Miniconda в системную переменную PATH. Чтобы убедиться, что все настроено правильно, щелкните по пункту «Anaconda Prompt (Miniconda3)» под пунктом «Anaconda3 (64-bit)» в меню **Пуск**. Затем попробуйте запустить интерпретатор Python, набрав команду `python`. Должно появиться сообщение вида

```
(base) C:\Users\Wes>python
Python 3.9 [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Чтобы выйти из оболочки Python, введите команду `exit()` и нажмите **Enter**.

GNU/Linux

Детали установки в Linux варьируются в зависимости от дистрибутива, я опишу процедуру, работающую в дистрибутивах Debian, Ubuntu, CentOS и Fedora. Установка в основных чертах производится так же, как для macOS, отличается только порядок установки Miniconda. Большая часть читателей, вероятно, захочет скачать предлагаемый по умолчанию 64-разрядный файл установщика, предназначенный для архитектуры x86 (но вполне возможно, что в будущем у большинства пользователей будут машины под управлением Linux с архитектурой aarch64). Установщик представляет собой скрипт оболочки, запускаемый из терминала. Имя соответствующего файла имеет вид *Miniconda3-latest-Linux-x86_64.sh*. Для установки нужно выполнить такую команду:

```
$ bash Miniconda3-latest-Linux-x86_64.sh
```

В некоторых дистрибутивах Linux менеджеры пакетов, например apt, располагают всеми необходимыми Python-пакетами. Ниже описывается установка с помощью Miniconda, поскольку она одинакова во всех дистрибутивах и упрощает обновление пакетов до последней версии.

Вам будет предложено указать место установки файлов Miniconda. Я рекомендую устанавливать их в свой домашний каталог, например */home/$USER/miniconda* (вместо `$USER` подставьте свое имя пользователя).

Установщик спросит, хотите ли вы модифицировать скрипты оболочки, так чтобы Miniconda активировалась автоматически. Я рекомендую так и поступить (выберите «yes»), поскольку это удобнее.

По завершении установки запустите новый процесс терминала и убедитесь, что выбирается новая версия Miniconda:

```
(base) $ python
Python 3.9 | (main) [GCC 10.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Чтобы выйти из оболочки, нажмите **Ctrl**-**D** или введите команду `exit()` и нажмите **Enter**.

Miniconda в macOS

Скачайте установщик Miniconda для macOS, для компьютеров на базе систем серии Apple Silicon, выпущенных начиная с 2020 года, он должен называться как-то вроде *Miniconda3-latest-MacOSX-arm64.sh*, а для компьютеров с процессором Intel, выпущенных до 2020 года, – *Miniconda3-latest-MacOSX-x86_64.sh*. Откройте приложение **Терминал** и выполните установщик (скорее всего, находится в вашем каталоге `Downloads`) с помощью `bash`:

```
$ bash $HOME/Downloads/Miniconda3-latest-MacOSX-arm64.sh
```

В процессе работы установщик по умолчанию конфигурирует Miniconda в вашем профиле оболочки, подразумеваемом по умолчанию. Скорее всего, это файл */Users/$USER/.zshrc*. Я рекомендую не возражать против этого действия. Если же вы категорически не хотите разрешать установщику модификацию своей среды, то почитайте документацию Miniconda, прежде чем продолжать.

Для проверки работоспособности попробуйте запустить Python из системной оболочки (для получения командной строки откройте приложение **Терминал**):

```
$ python
Python 3.9 (main) [Clang 12.0.1 ] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Чтобы выйти из оболочки, нажмите **Ctrl**-**D** или введите команду `exit()` и нажмите **Enter**.

Установка необходимых пакетов

Теперь, когда менеджер Miniconda установлен, самое время установить основные пакеты, которые понадобятся нам в этой книге. Первый шаг – сделать conda-forge подразумеваемым по умолчанию каналом получения пакетов. Для этого выполните в оболочке следующие команды:

```
(base) $ conda config --add channels conda-forge
(base) $ conda config --set channel_priority strict
```

Далее создадим новую «среду» conda для Python 3.10 командой `conda create`:

```
(base) $ conda create -y -n pydata-book python=3.10
```

По завершении установки активируйте среду командой `conda activate`:

```
(base) $ conda activate pydata-book
(pydata-book) $
```

Команду `conda activate` нужно использовать при каждом открытии нового терминала. В любой момент можно получить информацию об активной среде conda, выполнив в терминале команду `conda info`.

Далее установим необходимые пакеты (вместе с их зависимостями) командой `conda install`:

```
(pydata-book) $ conda install -y pandas jupyter matplotlib
```

Мы будем использовать и некоторые другие пакеты, но установить их можно позже. Есть два способа установки пакетов: командами `conda install` и `pip install`. При работе с Miniconda предпочтительнее использовать `conda install`, но некоторые пакеты через conda недоступны, поэтому если `conda install $package_name` не работает, попробуйте `$package_name`.

Если вы хотите сразу установить все пакеты, которые нам понадобятся по ходу дела, то можете выполнить такую команду:

```
conda install lxml beautifulsoup4 html5lib openpyxl \
              requests sqlalchemy seaborn scipy statsmodels \
              patsy scikit-learn pyarrow pytables numba
```

В Windows в качестве символа продолжения строки используйте знак крышки ^ вместо знака \, применяемого в Linux и macOS.

Для обновления пакетов служит команда `conda update`:

```
conda update package_name
```

pip также поддерживает обновление, нужно только задать флаг `--upgrade`:

```
pip install --upgrade package_name
```

В этой книге вам не раз представится возможность попробовать эти команды в деле.

Если для установки пакетов вы используете и conda, и pip, то не следует пытаться обновлять пакеты conda с помощью pip (и наоборот), поскольку это может привести к повреждению среды. Я рекомендую сначала всегда пробовать команду `conda install` и прибегать к pip, только если установить пакет не получается.

Интегрированные среды разработки (IDE)

Когда меня спрашивают, какой средой разработки я пользуюсь, я почти всегда отвечаю: «IPython плюс текстовый редактор». Обычно я пишу программу и итеративно тестирую и отлаживаю ее по частям в IPython или Jupyter-блокнотах. Полезно также иметь возможность интерактивно экспериментировать с данными и визуально проверять, что в результате определенных манипуляций получается ожидаемый результат. Библиотеки pandas и NumPy спроектированы с учетом простоты использования в оболочке.

Но некоторые пользователи предпочитают разрабатывать программы в полноценной IDE, а не в сравнительно примитивном текстовом редакторе типа Emacs или Vim. Вот некоторые доступные варианты:

- PyDev (бесплатная) – IDE, построенная на платформе Eclipse;
- PyCharm от компании JetBrains (на основе подписки для коммерческих компаний, бесплатна для разработчиков ПО с открытым исходным кодом);
- Python Tools для Visual Studio (для работающих в Windows);
- Spyder (бесплатная) – IDE, которая в настоящий момент поставляется в составе Anaconda;
- Komodo IDE (коммерческая).

Благодаря популярности Python большинство текстовых редакторов, в частности VS Code и Sublime Text 2, обзавелись прекрасной поддержкой для него.

1.5. Сообщество и конференции

Помимо поиска в интернете, существуют полезные списки рассылки, посвященные использованию Python в научных расчетах и для обработки данных. Их участники быстро отвечают на вопросы. Вот некоторые из таких ресурсов:

- pydata: группа Google по вопросам, относящимся к использованию Python для анализа данных и pandas;
- pystatsmodels: вопросы, касающиеся statsmodels и pandas;
- numpy-discussion: вопросы, касающиеся NumPy;
- список рассылки по scikit-learn (*scikit-learn@python.org*) и машинному обучению на Python вообще;
- scipy-user: общие вопросы использования SciPy и Python для научных расчетов.

Я сознательно не публикую URL-адреса, потому что они часто меняются. Поиск в интернете вам в помощь.

Ежегодно в разных странах проводят конференции для программистов на Python. Если вы захотите пообщаться с другими программистами на Python, которые разделяют ваши интересы, то имеет смысл посетить какую-нибудь

(если есть такая возможность). Многие конференции оказывают финансовую поддержку тем, кто не может позволить себе вступительный взнос или транспортные расходы. Приведу неполный перечень конференций.

- PyCon and EuroPython: две самые крупные, проходящие соответственно в Северной Америке и в Европе.
- SciPy и EuroSciPy: конференции, ориентированные на научные применения Python, проходящие соответственно в Северной Америке и в Европе.
- PyData: мировая серия региональных конференций, посвященных науке о данных и анализу данных.
- Международные и региональные конференции PyCon (полный список см. на сайте *http://pycon.org*).

1.6. Структура книги

Если вы раньше никогда не программировали на Python, то имеет смысл потратить время на знакомство с главами 2 и 3, где я поместил очень краткое руководство по языковым средствам Python, а также по оболочке IPython и Jupyter-блокнотам. Эти знания необходимы для чтения книги. Если у вас уже есть опыт работы с Python, то можете вообще пропустить эти главы или просмотреть их по диагонали.

Далее дается краткое введение в основные возможности NumPy, а более подробное изложение NumPy имеется в приложении A. Затем мы познакомимся с pandas и посвятим оставшуюся часть книги анализу данных с применением pandas, NumPy и matplotlib (для визуализации). Я старался построить изложение по возможности поступательно, хотя иногда главы немного пересекаются, и есть несколько случаев, когда используются еще не описанные концепции.

У разных читателей могут быть разные цели, но, вообще говоря, можно предложить такую классификацию задач.

Взаимодействие с внешним миром
: Чтение и запись в файлы и хранилища данных различных форматов.

Подготовка
: Очистка, переформатирование, комбинирование, нормализация, изменение формы, получение продольных и поперечных срезов, трансформация данных для анализа.

Преобразование
: Применение математических и статистических операций к группам наборов данных для получения новых наборов (например, агрегирование большой таблицы по некоторым переменным).

Моделирование и вычисления
: Связывание данных со статистическими моделями, алгоритмами машинного обучения и иными вычислительными средствами.

Презентация
: Создание интерактивных или статических графических визуализаций или текстовых сводных отчетов.

Примеры кода

Примеры кода в большинстве случаев показаны так, как выглядят в оболочке IPython или Jupyter-блокнотах: ввод и вывод.

```
In [5]: КОД
Out[5]: РЕЗУЛЬТАТ
```

Это означает, что вы должны ввести код в блоке `In` в своей рабочей среде и выполнить его, нажав клавишу **Enter** (или **Shift-Enter** в Jupyter). Результат должен быть таким, как показано в блоке `Out`.

Я изменил параметры вывода на консоль, подразумеваемые по умолчанию в NumPy и pandas, ради краткости и удобочитаемости. Так, числовые данные печатаются с большей точностью. Чтобы при выполнении примеров результаты выглядели так же, как в книге, выполните следующий Python-код перед запуском примеров:

```
import numpy as np
import pandas as pd
pd.options.display.max_columns = 20
pd.options.display.max_rows = 20
pd.options.display.max_colwidth = 80
np.set_printoptions(precision=4, suppress=True)
```

Данные для примеров

Наборы данных для примеров из каждой главы находятся в репозитории на сайте GitHub: *https://github.com/wesm/pydata-book* (или его зеркале на Gitee по адресу *https://gitee.com/wesmckinn/pydata-book*, если у вас нет доступа к GitHub). Вы можете получить их либо с помощью командной утилиты системы управления версиями Git, либо скачав zip-файл репозитория с сайта. Если возникнут проблемы, заходите на мой сайт (*https://wesmckinney.com/book*), где выложены актуальные инструкции по получению материалов к книге.

Скачав zip-файл с примерами наборов данных, вы должны будете распаковать его в какой-нибудь каталог и перейти туда в терминале, прежде чем выполнять примеры:

```
$ pwd
/home/wesm/book-materials
$ ls
appa.ipynb ch05.ipynb ch09.ipynb ch13.ipynb README.md
ch02.ipynb ch06.ipynb ch10.ipynb COPYING requirements.txt
ch03.ipynb ch07.ipynb ch11.ipynb datasets
ch04.ipynb ch08.ipynb ch12.ipynb examples
```

Я стремился, чтобы в репозиторий попало все необходимое для воспроизведения примеров, но мог где-то ошибиться или что-то пропустить. В таком случае пишите мне на адрес *book@wesmckinney.com*. Самый лучший способ сообщить об ошибках, найденных в книге, – описать их на странице опечаток на сайте издательства O'Reilly (*https://www.oreilly.com/catalog/errata.csp?isbn=0636920519829*).

Соглашения об импорте

В сообществе Python принят ряд соглашений об именовании наиболее употребительных модулей:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import statsmodels as sm
```

Это означает, что `np.arange` – ссылка на функцию `arange` в пакете NumPy. Так делается, потому что импорт всех имен из большого пакета, каким является NumPy (`from numpy import *`), считается среди разработчиков на Python дурным тоном.

Глава 2

Основы языка Python, IPython и Jupyter-блокноты

В 2011 и 2012 годах, когда я писал первое издание этой книги, ресурсов для изучения анализа данных с применением Python было гораздо меньше. Тут мы имеем что-то похожее на проблему яйца и курицы: многие библиотеки, наличие которых мы сейчас считаем само собой разумеющимся, в т. ч. pandas, scikit-learn и statsmodels, тогда были еще относительно незрелыми. Ныне, в 2022 году, количество литературы по науке о данных, анализу данных и машинному обучению неуклонно растет, дополняя прежние работы по научным расчетам, рассчитанные на специалистов по информатике, физике и другим дисциплинам. Есть также замечательные книги о самом языке программирования Python и о том, как стать эффективным программистом.

Поскольку эта книга задумана как введение в работу с данными на Python, я считаю полезным дать замкнутый обзор некоторых наиболее важных особенностей встроенных в Python структур данных и библиотек с точки зрения манипулирования данными. Поэтому в этой и следующей главах приводится лишь информация, необходимая для чтения книги.

Книга в основном посвящена инструментам табличного анализа и подготовки данных для работы с наборами данных, помещающимися на персональном компьютере. Чтобы применить эти инструменты, зачастую необходимо сначала преобразовать беспорядочные данные низкого качества в более удобную табличную (или *структурную*) форму. К счастью, Python – идеальный язык для этой цели. Чем свободнее вы владеете языком и встроенными в него типами данных, тем проще будет подготовить новый набор данных для анализа.

Некоторые описанные в книге инструменты лучше изучать в интерактивном сеансе IPython или Jupyter. После того как вы научитесь запускать IPython и Jupyter, я рекомендую проработать примеры, экспериментируя и пробуя разные подходы. Как и в любом окружении, ориентированном на работу с клавиатурой, полезно запомнить наиболее употребительные команды на подсознательном уровне.

Некоторые базовые понятия Python, например классы и объектно-ориентированное программирование, в этой главе не рассматриваются, хотя их полезно включить в арсенал средств для анализа данных. Для желающих углубить свои знания я рекомендую дополнить эту главу официальным пособием по Python (*https://docs.python.org/3/*) и, возможно, одной из многих замечательных книг по программированию на Python вообще. Начать можно, например, с таких книг:

- Python Cookbook, Third Edition, by David Beazley and Brian K. Jones (O'Reilly);
- Fluent Python by Luciano Ramalho (O'Reilly)[1];
- Effective Python by Brett Slatkin (Pearson)[2].

2.1. Интерпретатор Python

Python – *интерпретируемый* язык. Интерпретатор Python исполняет программу по одному предложению за раз. Стандартный интерактивный интерпретатор Python запускается из командной строки командой `python`:

```
$ python
Python 3.10.4 | packaged by conda-forge | (main, Mar 24 2022, 17:38:57)
[GCC 10.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 5
>>> print(a)
5
```

Строка `>>>` – это приглашение к вводу выражения. Для выхода из интерпретатора Python нужно либо ввести команду `exit()`, либо нажать `Ctrl-D` (только в Linux и macOS).

Для выполнения Python-программы нужно просто набрать команду `python`, указав в качестве первого аргумента имя файла с расширением `.py`. Допустим, вы создали файл `hello_world.py` с таким содержимым:

```
print("Hello world")
```

Чтобы выполнить его, достаточно ввести следующую команду (файл *hello_world.py* должен находиться в текущем каталоге):

```
$ python hello_world.py
Hello world
```

Многие программисты выполняют свой код на Python именно таким образом, но в мире научных приложений и анализа данных принято использовать IPython, улучшенный и дополненный интерпретатор Python, или веб-блокноты Jupyter, первоначально разработанные как часть проекта IPython. Введение в IPython и Jupyter будет дано в этой главе, а углубленное описание возможностей IPython – в приложении 3. С помощью команды `%run` IPython исполняет код в указанном файле в том же процессе, что позволяет интерактивно изучать результаты по завершении выполнения.

---

[1] *Лусиану Рамальо.* Python. К вершинам мастерства. 2-е изд. ДМК Пресс, 2022.

[2] *Бретт Слаткин.* Секреты Python. Вильямс, 2017.

```
$ ipython
Python 3.10.4 | packaged by conda-forge | (main, Mar 24 2022, 17:38:57)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.31.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: %run hello_world.py
Hello world

In [2]:
```

По умолчанию приглашение IPython содержит не стандартную строку `>>>`, а строку вида `In [2]:`, включающую порядковый номер предложения.

2.2. Основы IPython

В этом разделе мы научимся запускать оболочку IPython и Jupyter-блокнот, а также познакомимся с некоторыми важнейшими понятиями.

Запуск оболочки IPython

IPython можно запустить из командной строки, как и стандартный интерпретатор Python, только для этого служит команда `ipython`:

```
$ ipython
Python 3.10.4 | packaged by conda-forge | (main, Mar 24 2022, 17:38:57)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.31.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: a = 5

In [2]: a
Out[2]: 5
```

Чтобы выполнить произвольное предложение Python, нужно ввести его и нажать клавишу **Enter**. Если ввести только имя переменной, то IPython выведет строковое представление объекта:

```
In [5]: import numpy as np

In [6]: data = [np.random.standard_normal() for i in range(7)]

In [7]: data
Out[7]:
[-0.20470765948471295,
 0.47894333805754824,
 -0.5194387150567381,
 -0.55573030434749,
 1.9657805725027142,
 1.3934058329729904,
 0.09290787674371767]
```

Здесь первые две строки содержат код на Python; во второй строке создается переменная **`data`**, ссылающаяся на только что созданный словарь Python. В последней строке значение **`data`** выводится на консоль.

Многие объекты Python форматируются для удобства чтения; такая *красивая печать* отличается от обычного представления методом `print`. Тот же сло-

варь `data`, напечатанный в стандартном интерпретаторе Python, выглядел бы куда менее презентабельно:

```
>>> import numpy as np
>>> data = [np.random.standard_normal() for i in range(7)]
>>> print(data)
>>> data
[-0.5767699931966723, -0.1010317773535111, -1.7841005313329152,
-1.524392126408841, 0.22191374220117385, -1.9835710588082562,
-1.6081963964963528]
```

IPython предоставляет также средства для исполнения произвольных блоков кода (путем копирования и вставки) и целых Python-скриптов. Эти вопросы будут рассмотрены чуть ниже.

Запуск Jupyter-блокнота

Одним из основных компонентов Jupyter-проекта является *блокнот* – интерактивный документ, содержащий код, текст (простой или размеченный), визуализации и другие результаты выполнения кода. Jupyter-блокнот взаимодействует с *ядрами* – реализациями протокола интерактивных вычислений на различных языках программирования. В ядре Jupyter для Python в качестве основы используется IPython.

Для запуска Jupyter выполните в терминале команду `jupyter notebook`:

```
$ jupyter notebook
[I 15:20:52.739 NotebookApp] Serving notebooks from local directory:
/home/wesm/code/pydata-book
[I 15:20:52.739 NotebookApp] 0 active kernels
[I 15:20:52.739 NotebookApp] The Jupyter Notebook is running at:
http://localhost:8888/?token=0a77b52fefe52ab83e3c35dff8de121e4bb443a63f2d...
[I 15:20:52.740 NotebookApp] Use Control-C to stop this server and shut down
all kernels (twice to skip confirmation).
Created new window in existing browser session.
To access the notebook, open this file in a browser:
    file:///home/wesm/.local/share/jupyter/runtime/nbserver-185259-open.html
Or copy and paste one of these URLs:
    http://localhost:8888/?token=0a77b52fefe52ab83e3c35dff8de121e4...
 or http://127.0.0.1:8888/?token=0a77b52fefe52ab83e3c35dff8de121e4...
```

На многих платформах Jupyter автоматически открывается в браузере по умолчанию (если только при запуске не был указан флаг `--no-browser`). Если это не так, то можете сами ввести URL при запуске блокнота, в данном случае `http://localhost:8888/?token=0a77b52fefe52ab83e3c35dff8de121e4bb443a63f2d3055`. На рис. 2.1 показано, как выглядит блокнот в браузере Google Chrome.

Многие используют Jupyter в качестве локальной среды вычислений, но его можно также развернуть на сервере и обращаться удаленно. Я не буду здесь вдаваться в эти детали, при необходимости вы сможете найти информацию в интернете.

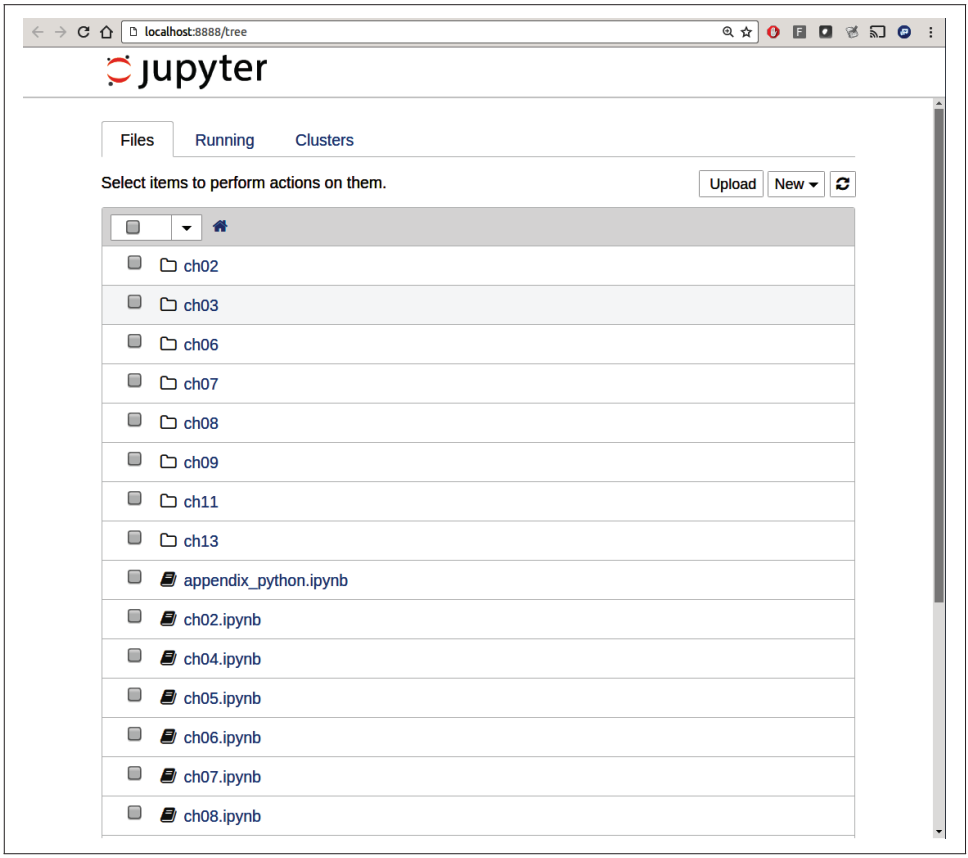

Рис. 2.1. Начальная страница Jupyter-блокнота

Для создания нового блокнота нажмите кнопку **New** и выберите «Python 3». На экране появится окно, показанное на рис. 2.2. Если вы здесь впервые, попробуйте щелкнуть по пустой «ячейке» кода и ввести строку кода на Python. Для выполнения нажмите **Shift**-**Enter**.

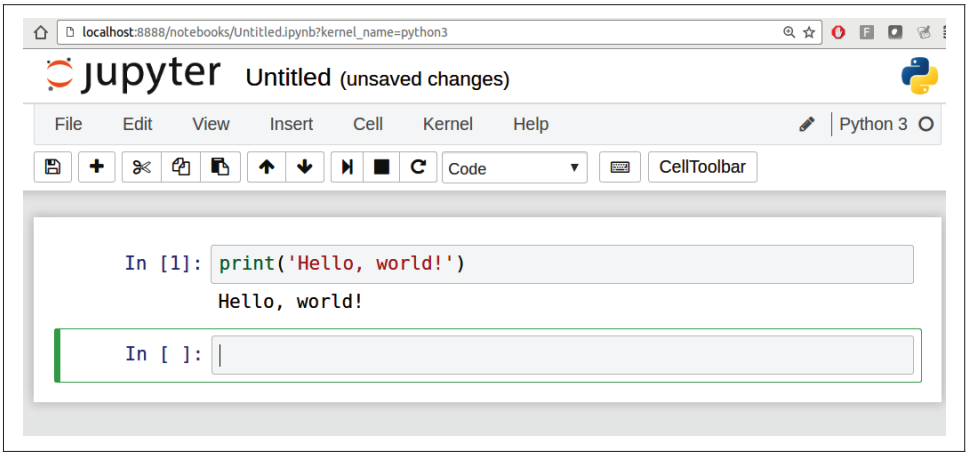

Рис. 2.2. Так выглядит новый Jupyter-блокнот

После сохранения блокнота (команда «Save and Checkpoint» в меню **File**) будет создан файл с расширением *.ipynb*. В нем содержится все, что сейчас находится в блокноте (включая все результаты выполнения кода).

Чтобы загрузить существующий блокнот, поместите файл в тот каталог, из которого был запущен блокнот (или в его подкаталог), и дважды щелкните по имени файла на начальной странице. Можете попробовать проделать это с моими блокнотами, находящимися в репозитории *wesm/pydata-book* на GitHub. См. рис. 2.3.

Когда захотите закрыть блокнот, выберите команду «Close and Halt» из меню **File**. Если вы просто закроете вкладку браузера, то ассоциированный с блокнотом процесс Python продолжит работать в фоновом режиме.

Хотя может показаться, что работа с Jupyter-блокнотами отличается от работы в оболочке IPython, на самом деле почти все описанные в этой главе команды и инструменты можно использовать в обеих средах.

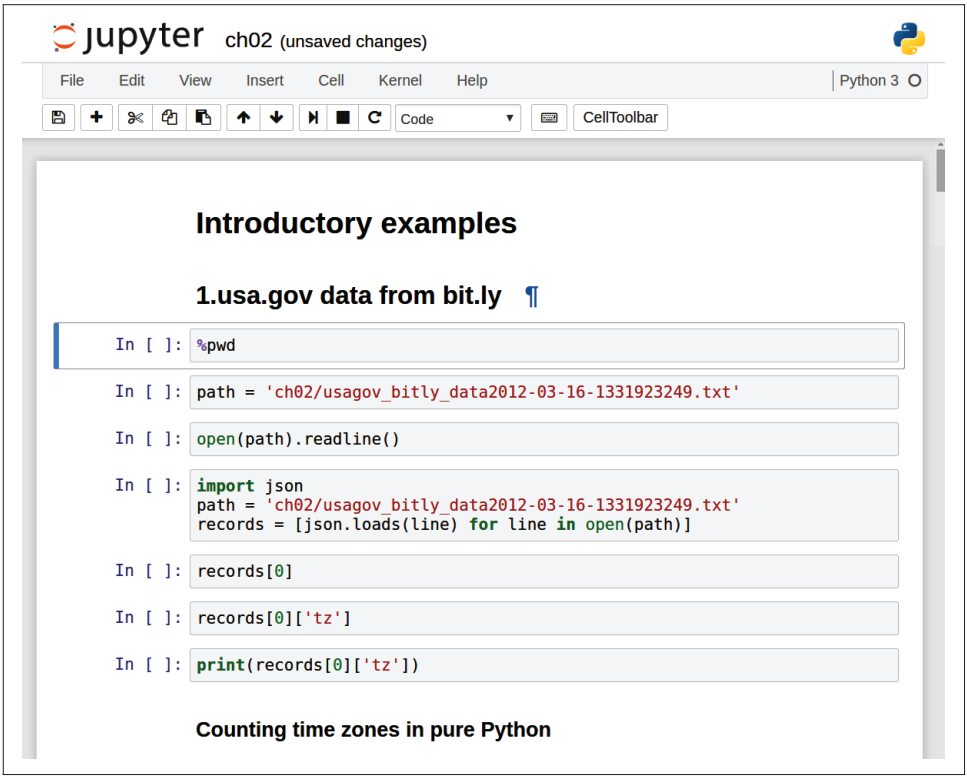

Рис. 2.3. Пример существующего Jupyter-блокнота

Завершение по нажатии клавиши Tab

На первый взгляд, оболочка IPython очень похожа на стандартный интерпретатор Python (вызываемый командой `python`) с мелкими косметическими изменениями. Одно из существенных преимуществ над стандартной оболочкой Python – *завершение по нажатии клавиши* ***Tab***, реализованное в большинстве IDE и других средах интерактивных вычислений. Если во время ввода выражения нажать `<Tab>`, то оболочка произведет поиск в пространстве имен всех переменных (объектов, функций и т. д.), имена которых начинаются с введенной к этому моменту строки:

```
In [1]: an_apple = 27

In [2]: an_example = 42

In [3]: an<Tab>
an_apple and an_example any
```

Обратите внимание, что IPython вывел обе определенные выше переменные, а также ключевое слово Python `and` и встроенную функцию `any`. Естественно, можно также завершать имена методов и атрибутов любого объекта, если предварительно ввести точку:

```
In [3]: b = [1, 2, 3]

In [4]: b.<Tab>
append() count()   insert()  reverse()
clear()  extend()  pop()     sort()
copy()   index()   remove()
```

То же самое относится и к модулям:

```
In [1]: import datetime

In [2]: datetime.<Tab>
date          MAXYEAR    timedelta
datetime      MINYEAR    timezone
datetime_CAPI time       tzinfo
```

Отметим, что IPython по умолчанию скрывает методы и атрибуты, начинающиеся знаком подчеркивания, например магические методы и внутренние «закрытые» методы и атрибуты, чтобы не загромождать экран (и не смущать неопытных пользователей). На них автозавершение также распространяется, нужно только сначала набрать знак подчеркивания. Если вы предпочитаете всегда видеть такие методы при автозавершении, измените соответствующий режим в конфигурационном файле IPython. О том, как это сделать, см. в документации по IPython (*https://ipython.readthedocs.io/en/stable/*).

Завершение по нажатии **Tab** работает во многих контекстах, помимо поиска в интерактивном пространстве имен и завершения атрибутов объекта или модуля. Если нажать `<Tab>` при вводе чего-то, похожего на путь к файлу (даже внутри строки Python), то будет произведен поиск в файловой системе.

В сочетании с командой `%run` (см. ниже) эта функция несомненно позволит вам меньше лупить по клавиатуре.

Автозавершение позволяет также сэкономить время при вводе именованных аргументов функции (в том числе и самого знака `=`). См. рис. 2.4.

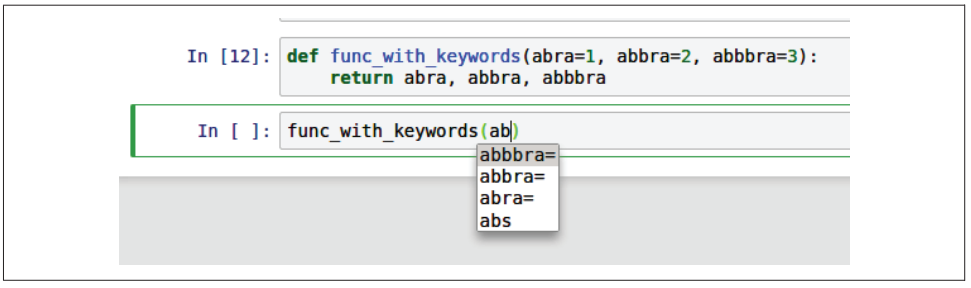

Рис. 2.4. Автозавершение ключевых аргументов функции в Jupyter-блокноте

Ниже мы еще поговорим о функциях.

Интроспекция

Если ввести вопросительный знак (?) до или после имени переменной, то будет напечатана общая информация об объекте:

```
In [1]: b = [1, 2, 3]

In [2]: b?
Type: list
String form: [1, 2, 3]
Length: 3
Docstring:
Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list.
The argument must be an iterable if specified.

In [3]: print?
Docstring:
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file: a file-like object (stream); defaults to the current sys.stdout.
sep:  string inserted between values, default a space.
end:  string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
Type:    builtin_function_or_method
```

Это называется *интроспекцией объекта*. Если объект представляет собой функцию или метод экземпляра, то будет показана строка документации, при условии ее существования. Допустим, мы написали такую функцию (этот код можно ввести в IPython or Jupyter):

```
def add_numbers(a, b):
"""
Сложить два числа

Возвращает
-------
the_sum : типа аргументов
"""
return a + b
```

Тогда при вводе знака ? мы увидим строку документации:

```
In [6]: add_numbers?
Signature: add_numbers(a, b)
Docstring:
Сложить два числа
Возвращает
----------
the_sum : типа аргументов
File:      <ipython-input-9-6a548a216e27>
Type:      function
```

И последнее применение ? – поиск в пространстве имен IPython по аналогии со стандартной командной строкой UNIX или Windows. Если ввести несколько

символов в сочетании с метасимволом *, то будут показаны все имена по указанной маске. Например, вот как можно получить список всех функций в пространстве имен верхнего уровня NumPy, имена которых содержат строку load:

```
In [9]: import numpy as np

In [10]: np.*load*?
np.__loader__
np.load
np.loads
np.loadtxt
```

2.3. Основы языка Python

В этом разделе я приведу обзор наиболее важных концепций программирования на Python и механизмов языка. В следующей главе мы более подробно рассмотрим структуры данных, функции и другие средства Python.

Семантика языка

Язык Python отличается удобочитаемостью, простотой и ясностью. Некоторые даже называют написанный на Python код «исполняемым псевдокодом».

Отступы вместо скобок

В Python для структурирования кода используются пробелы (или знаки табуляции), а не фигурные скобки, как во многих других языках, например R, C++, Java и Perl. Вот как выглядит цикл в алгоритме быстрой сортировки:

```
for x in array:
    if x < pivot:
        less.append(x)
    else:
        greater.append(x)
```

Двоеточием обозначается начало блока кода с отступом, весь последующий код до конца блока должен быть набран с точно таким же отступом.

Нравится вам это или нет, но синтаксически значимые пробелы – факт, с которым программисты на Python должны смириться. Поначалу такой стиль может показаться чужеродным, но со временем вы привыкнете и полюбите его.

Я настоятельно рекомендую использовать 4 пробела в качестве величины отступа по умолчанию и настроить редактор так, чтобы он заменял знаки табуляции четырьмя пробелами. IPython и Jupyter-блокноты автоматически вставляют четыре пробела в новой строке после двоеточия и заменяют знаки табуляции четырьмя пробелами.

Вы уже поняли, что предложения в Python не обязаны завершаться точкой с запятой. Но ее можно использовать, чтобы отделить друг от друга предложения, находящиеся в одной строчке:

```
a = 5; b = 6; c = 7
```

Впрочем, писать несколько предложений в одной строчке не рекомендуется, потому что код из-за этого становится труднее читать.

Всё является объектом

Важная характеристика языка Python – последовательность его *объектной модели*. Все числа, строки, структуры данных, функции, классы, модули и т. д. в интерпретаторе заключены в «ящики», которые называются *объектами Python*. С каждым объектом ассоциирован *тип* (например, *целое число*, *строка* или *функция*) и внутренние данные. На практике это делает язык более гибким, потому что даже функции можно рассматривать как объекты.

Комментарии

Интерпретатор Python игнорирует текст, которому предшествует знак решетки #. Часто этим пользуются, чтобы включить в код комментарии. Иногда желательно исключить какие-то блоки кода, не удаляя их. Самое простое решение – *закомментировать* такой код:

```
results = []
for line in file_handle:
    # пока оставляем пустые строки
    # if len(line) == 0:
    #     continue
    results.append(line.replace("foo", "bar"))
```

Комментарии могут также встречаться после строки исполняемого кода. Некоторые программисты предпочитают располагать комментарий над строкой кода, к которой он относится, но и такой стиль временами бывает полезен:

```
print("Reached this line") # Простое сообщение о состоянии
```

Вызов функции и метода объекта

После имени функции ставятся круглые скобки, внутри которых может быть нуль или более параметров. Возвращенное значение может быть присвоено переменной:

```
result = f(x, y, z)
g()
```

Почти со всеми объектами в Python ассоциированы функции, которые имеют доступ к внутреннему состоянию объекта и называются методами. Синтаксически вызов методов выглядит так:

```
obj.some_method(x, y, z)
```

Функции могут принимать позиционные и именованные аргументы:

```
result = f(a, b, c, d=5, e="foo")
```

Подробнее об этом ниже.

Переменные и передача аргументов

Присваивание значения переменной (или *имени*) в Python приводит к созданию *ссылки* на объект, стоящий в правой части присваивания. Рассмотрим список целых чисел:

```
In [8]: a = [1, 2, 3]
```

Предположим, что мы присвоили значение `a` новой переменной `b`:

```
In [9]: b = a

In [10]: b
Out[10]: [1, 2, 3]
```

В некоторых языках такое присваивание приводит к копированию данных `[1, 2, 3]`. В Python `a` и `b` указывают на один и тот же объект – исходный список `[1, 2, 3]` (это схематически изображено на рис. 2.5). Чтобы убедиться в этом, добавим в список `a` еще один элемент и проверим затем список `b`:

```
In [11]: a.append(4)

In [12]: b
Out[12]: [1, 2, 3, 4]
```

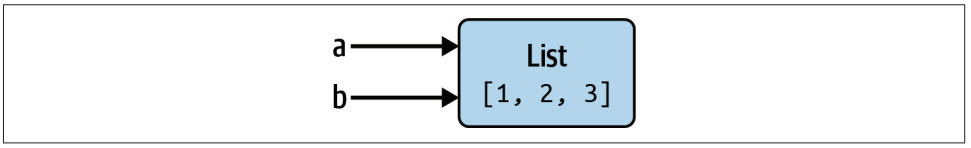

Рис. 2.5. Две ссылки на один объект

Понимать семантику ссылок в Python и знать, когда, как и почему данные копируются, особенно важно при работе с большими наборами данных.

Операцию присваивания называют также *связыванием*, потому что мы связываем имя с объектом. Имена переменных, которым присвоено значение, иногда называют связанными переменными.

Когда объекты передаются функции в качестве аргументов, создаются новые локальные переменные, ссылающиеся на исходные объекты, – копирование не производится. Если новый объект связывается с переменной внутри функции, то переменная с таким же именем в «области видимости» вне этой функции («родительской области видимости») не перезаписывается. Поэтому функция может модифицировать внутреннее содержимое изменяемого аргумента. Пусть имеется такая функция:

```
In [13]: def append_element(some_list, element):
    ....:     some_list.append(element)
```

Тогда:

```
In [14]: data = [1, 2, 3]

In [15]: append_element(data, 4)

In [16]: data
Out[16]: [1, 2, 3, 4]
```