

Оглавление

Предисловие	16
Примеры кода	17
Благодарности	18
Об авторе	19
От издательства	20

ЧАСТЬ I PYTHON В JUPYTER NOTEBOOK

Глава 1. Введение в Jupyter Notebook	22
Выполнение операторов в Python	23
Блокноты Jupyter	24
Блокноты Google Colab	24
Текстовые ячейки Colab	26
LaTeX	29
Ячейки кода Colab	29
Файлы Colab	29
Управление документами Colab	31
Фрагменты кода Colab	31
Существующие коллекции	32
Системные псевдонимы	33
Магические функции	33
Резюме	34
Вопросы для закрепления	34

Глава 2. Основы Python	35
Основные типы в Python	36
Высокоуровневые и низкоуровневые языки	37
Операторы	38
Выполнение базовых математических операций	46
Классы, объекты и точечная нотация	47
Резюме	47
Вопросы для закрепления	47
Глава 3. Последовательности	48
Общие операции	49
Проверка принадлежности	49
Индексирование	50
Слайсинг	50
Сбор информации	51
Математические операции	52
Списки и кортежи	53
Создание списков и кортежей	53
Добавление и удаление элементов списка	54
Распаковка	56
Сортировка списков	57
Строки	57
Диапазоны	59
Резюме	60
Вопросы для закрепления	60
Глава 4. Прочие структуры данных	61
Словари	62
Создание словарей	62
Доступ, добавление и обновление с помощью ключей	63
Удаление элементов из словарей	64
Представления словаря	65

Проверка наличия ключа в словаре	68
Метод <code>get</code>	69
Допустимые типы ключей	70
Метод <code>hash</code>	71
Множества	72
Операции над множествами	75
Замороженные множества	80
Резюме	81
Вопросы для закрепления	81
Глава 5. Управление выполнением	82
Составные операторы	83
Структура составных операторов	83
Оценка <code>True</code> или <code>False</code>	84
Операторы <code>if</code>	87
Циклы <code>while</code>	91
Циклы <code>for</code>	92
Операторы <code>break</code> и <code>continue</code>	93
Резюме	93
Вопросы для закрепления	94
Глава 6. Функции	95
Объявление функций	96
Управляющий оператор	96
Строки документации	96
Параметры	98
Операторы возврата	104
Область видимости в функциях	105
Декораторы	105
Анонимные функции	110
Резюме	110
Вопросы для закрепления	111

ЧАСТЬ II
БИБЛИОТЕКИ DATA SCIENCE

Глава 7. Библиотека NumPy	114
Установка и импорт NumPy	115
Создание массивов	116
Индексация и слайсинг	119
Поэлементные операции	121
Фильтрация значений	122
Представления и копии	124
Методы массива	125
Бродкастинг	129
Математические функции NumPy	130
Резюме	132
Вопросы для закрепления	132
Глава 8. Библиотека SciPy	134
Обзор SciPy	135
Подмодуль <code>scipy.misc</code>	135
Подмодуль <code>scipy.special</code>	136
Подмодуль <code>scipy.stats</code>	137
Дискретные распределения	137
Непрерывные распределения	140
Резюме	144
Вопросы для закрепления	144
Глава 9. Библиотека Pandas	145
Структура датафреймов	146
Создание датафреймов	146
Создание датафреймов из словаря	146
Создание датафреймов из списка списков	148
Создание датафрейма из файла	149
Взаимодействие с данными датафреймов	150
Головы и хвосты	150

Описательная статистика	152
Доступ к данным	155
Синтаксис со скобками	155
Оптимизированный доступ по метке	158
Оптимизированный доступ по индексу	160
Маски и фильтрация	161
Булевы операторы библиотеки Pandas	163
Управление датафреймами	164
Управление данными	167
Метод <code>replace</code>	169
Интерактивный дисплей	171
Резюме	172
Вопросы для закрепления	172
Глава 10. Библиотеки визуализации данных	173
Библиотека <code>matplotlib</code>	174
Оформление графиков	175
Маркировка данных	179
Построение графиков для множества наборов данных	180
Объектно-ориентированный стиль	182
Библиотека <code>Seaborn</code>	184
Темы <code>Seaborn</code>	185
Библиотека <code>Plotly</code>	188
Библиотека <code>Vokeh</code>	189
Другие библиотеки визуализации	191
Резюме	191
Вопросы для закрепления	192
Глава 11. Библиотеки машинного обучения	193
Популярные библиотеки машинного обучения	194
Принцип работы машинного обучения	194
Преобразования	195
Разделение тестовых и тренировочных данных	196

Обучение и тестирование	197
Подробнее о Scikit-learn	198
Резюме	198
Вопросы для закрепления	198
Глава 12. Инструментарий естественного языка (NLTK)	199
Образцы текстов NLTK	200
Частотное распределение	202
Текстовые объекты	206
Классификация текста	207
Резюме	210
Упражнения	211

ЧАСТЬ III СРЕДНИЙ УРОВЕНЬ PYTHON

Глава 13. Функциональное программирование	214
Знакомство с функциональным программированием	215
Область видимости и состояние	215
Зависимость от глобального состояния	216
Изменение состояния	217
Изменение изменяемых данных	218
Функции функционального программирования	219
Списковые включения	222
Базовый синтаксис списковых включений	222
Замена map и filter	222
Множественные переменные	224
Словарные включения	224
Генераторы	224
Выражения-генераторы	225
Функции-генераторы	226
Резюме	228
Вопросы для закрепления	228

Глава 14. Объектно-ориентированное программирование	229
Связывание состояния и функции	230
Классы и экземпляры	230
Закрытые методы и переменные	233
Переменные класса	233
Специальные методы	234
Методы представления	235
Расширенные методы сравнения	236
Методы математических операторов	238
Наследование	239
Резюме	243
Вопросы для закрепления	243
Глава 15. Прочие вопросы	245
Сортировка	246
Списки	246
Чтение и запись файлов	250
Контекстные менеджеры	250
Объекты <code>datetime</code>	251
Регулярные выражения	254
Наборы символов	255
Классы символов	255
Группы	256
Именованные группы	256
Найти все	257
Найти итератор	257
Замена	258
Замена с использованием именованных групп	258
Компиляция регулярных выражений	258
Резюме	259
Вопросы для закрепления	260

Приложение А. Ответы к вопросам в конце глав	261
Глава 1	261
Глава 2	261
Глава 3	261
Глава 4	262
Глава 5	262
Глава 6	262
Глава 7	263
Глава 8	263
Глава 9	263
Глава 10	264
Глава 11	264
Глава 12	264
Глава 13	265
Глава 14	265
Глава 15	266

3

Последовательности

Ошибок при использовании неадекватных данных
намного меньше, чем при отсутствии данных вообще.

Чарльз Бэббидж

В этой главе

- Общие операции с последовательностями.
- Списки и кортежи.
- Строки и строковые методы.
- Диапазоны.

В главе 2 вы узнали о типах коллекций. Здесь же мы познакомимся с группой встроенных типов, называемых *последовательностями*. Последовательность — это упорядоченная конечная коллекция. Вы можете представить ее как полку в библиотеке, где у каждой книги есть свое место, доступ к которой можно легко получить,

если вы знаете это место. Книги упорядочены, у каждой (кроме стоящих в конце) есть стоящие перед ней и после нее. Вы можете положить книгу на полку, взять ее оттуда, но эта полка может быть и пустой. Встроенные типы, составляющие последовательность, — это списки, кортежи, строки, бинарные строки и диапазоны. В этой главе рассмотрены их общие характеристики и особенности.

Общие операции

У семейства последовательностей есть довольно много общих функций. В частности, есть способы использования последовательностей, применимых для большинства членов группы. Есть операции, относящиеся к последовательностям конечной длины, для доступа к элементам в последовательности и создания новой последовательности на основе ее содержимого.

Проверка принадлежности

Проверить, принадлежит ли элемент последовательности, можно с помощью операции `in`. Она возвращает значение `True`, если последовательность содержит элемент, который оценивается как равный тому, что в запросе, и `False` — в противном случае. Ниже приведены примеры использования `in` с разными типами последовательностей:

```
'first' in ['first', 'second', 'third']  
True
```

```
23 in (23,)  
True
```

```
'b' in 'cat'  
False
```

```
b'a' in b'ieojjza'  
True
```

Можно использовать `not` в сочетании с `in`, чтобы проверить отсутствие чего-либо в последовательности:

```
'b' not in 'cat'  
True
```

Два самых распространенных случая использования сочетания `in` и `not` — это интерактивная сессия для изучения данных и как часть оператора `if` (см. главу 5).

Индексирование

Поскольку последовательность — это упорядоченная серия элементов, вы можете получить доступ к элементу в ней, используя его местоположение, или *индекс*. Индексы начинаются с нуля и увеличиваются до числа, на единицу меньшего количества элементов. Например, в последовательности из восьми элементов у первого будет индекс ноль, а у последнего — семь.

Чтобы с помощью индекса получить доступ к элементу, заключите номер индекса в квадратные скобки. В следующем примере объявляется строка и осуществляется доступ к ее первой и последней подстрокам с помощью индексных номеров:

```
name = "Ignatius"  
name[0]  
'I'  
  
name[7]  
's'
```

С помощью отрицательных индексов вы можете индексировать обратный отсчет с конца последовательности:

```
name[-1]  
's'  
  
name[-2]  
'u'
```

Слайсинг

Вы можете использовать индексы для создания новых последовательностей, которые будут представлять собой подпоследовательности оригинала. В квадратных скобках укажите начальный и конечный индексные номера последовательности, разделив их двоеточием, — так будет возвращена новая последовательность:

```
name = "Ignatius"  
name[2:5]  
'nat'
```

Возвращаемая подпоследовательность содержит элементы от первого индекса и до последнего, не включая его. Если вы пропустите начальный индекс, подпоследовательность начнется с родительской последовательности. А если последний, то подпоследовательность перейдет в конец последовательности:

```
name[:5]  
'Ignat'
```

```
name[4:]  
'tius'
```

Отрицательные индексные номера можно использовать для создания срезов (слайсов)¹, начинающих отсчет с конца последовательности. Ниже показано, как захватить последние три буквы строки:

```
name[-3:]  
'ius'
```

Если вы хотите, чтобы срез пропускал элементы, предоставьте третий аргумент, указывающий на то, как вести подсчет. Так, если у вас есть список последовательности целых чисел, вы можете создать срез, просто используя начальный и конечный индексные номера:

```
scores = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]  
scores[3:15]  
[3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

Но вы можете указать и размер шага. Например, считать по три:

```
scores[3:15:3]  
[3, 6, 9, 12]
```

Для обратного отсчета используйте отрицательный шаг:

```
scores[18:0:-4]  
[18, 14, 10, 6, 2]
```

Сбор информации

Вы можете выполнять общие операции над последовательностями, чтобы собрать информацию о них. Последовательность конечна, и у нее есть длина, которую можно узнать с помощью функции `len`:

```
name = "Ignatius"  
len(name)  
8
```

Используйте `min` и `max`, чтобы найти минимальные и максимальные элементы:

¹ Срез (слайс, slice) — извлечение из данной строки одного символа или некоторого фрагмента подстроки или подпоследовательности. — *Примеч. пер.*

```
scores = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]
min(scores)
0
```

```
max(name)
'u'
```

Эти методы предполагают, что содержимое последовательности можно сравнить способом, подразумевающим упорядочивание. В последовательностях, допускающих смешанные типы элементов, могут возникнуть ошибки, если содержимое нельзя будет сравнить:

```
max(['Free', 2, 'b'])
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-15-d8babe38f9d9> in <module>()
----> 1 max(['Free', 2, 'b'])
TypeError: '>' not supported between instances of 'int' and 'str'
```

Выяснить, сколько раз появляется элемент в последовательности, можно с помощью `count`:

```
name.count('a')
1
```

Получить индекс элемента в последовательности можно с помощью `index`:

```
name.index('s')
7
```

Можно использовать метод `index` для создания среза до элемента. Например, до буквы в строке:

```
name[:name.index('u')]
'Ignati'
```

Математические операции

Вы можете выполнять операции сложения и умножения с последовательностями одного типа. При этом эти операции проводятся именно с последовательностями, а не с их содержимым. Так, в результате сложения списка [1] и [2] получится [1,2], а не [3]. Ниже приведен пример использования оператора плюс (+) для создания новой строки из трех отдельных строк:

```
"prefix" + "-" + "postfix"  
'prefix-postfix'
```

Оператор умножения (*) работает, выполняя многократное сложение всей последовательности, а не ее содержимого:

```
[0,2] * 4  
[0, 2, 0, 2, 0, 2, 0, 2]
```

Это полезный способ настройки последовательности со значениями по умолчанию. Допустим, вы хотите отследить баллы для заданного количества участников в списке. Вы можете инициализировать список с помощью умножения так, что в нем будут исходные баллы для каждого участника:

```
num_participants = 10  
scores = [0] * num_participants  
scores  
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Списки и кортежи

Списки и кортежи — последовательности, которые могут содержать объекты любого типа. Их содержимое может быть смешанным, поэтому у вас в одном списке могут быть строки, целые числа, экземпляры, числа с плавающей точкой и другие объекты. Разделители в списках и кортежах — запятые. Элементы в списке заключены в квадратные скобки, а в кортеже — в круглые. Основное различие между списками и кортежами в том, что первые изменяемы, а вторые — нет. Это означает, что вы можете изменить содержимое списка, но как только создан кортеж, изменить его уже невозможно. Чтобы изменить содержимое кортежа, нужно создать новый на основе текущего. Благодаря своей способности изменяться списки более функциональны, но они используют больше памяти.

Создание списков и кортежей

Вы создаете список с помощью функции — конструктора списков `list()` или синтаксиса квадратных скобок. Например, чтобы создать список исходных значений, просто укажите значения в квадратных скобках:

```
some_list = [1,2,3]  
some_list  
[1, 2, 3]
```

Кортежи создаются с помощью функции — конструктора кортежей `tuple()` или круглых скобок. Чтобы создать кортеж с одним элементом, поставьте за ним запятую, иначе Python воспримет круглые скобки не как обозначение кортежа, а как логическую группировку. Вы можете создать кортеж и без круглых скобок — просто поставьте запятую после элемента. Ознакомьтесь с примерами создания кортежа в листинге 3.1.

Листинг 3.1. Создание кортежа

```
 tup = (1,2)
 tup
 (1,2)

 tup = (1,)
 tup
 (1,)

 tup = 1,2,
 tup
 (1,2)
```

ОБРАТИТЕ ВНИМАНИЕ

Распространенная, но незначительная ошибка случается, когда вы оставляете висящую запятую после аргумента функции. Это превращает аргумент в кортеж, содержащий исходный аргумент. Поэтому вторым аргументом функции `my_function(1, 2,)` будет `(2,)`, а не `2`.

Вы можете применять конструкторы списка и кортежа для последовательности в качестве аргумента. В следующем примере используется строка и создается список элементов в ней:

```
 name = "Ignatius"
 letters = list(name)
 letters
 ['I', 'g', 'n', 'a', 't', 'i', 'u', 's']
```

Добавление и удаление элементов списка

Вы можете добавлять элементы в список и удалять их из него. Чтобы понять, как это работает, представьте, что список — это стопка книг. Самый эффективный способ добавить элемент в список — использовать метод `append`. Он добавляет элемент в конец списка, как если бы вы положили книги наверх стопки. Чтобы добавить элемент в другую позицию списка, используйте метод `insert`, указав в качестве аргумента номер индекса, в котором вы хотите разместить новый элемент. Это менее эффективно, чем применение `append`, так как может потребоваться переместить другие элементы в списке, чтобы

освободить место для нового. Но это является проблемой обычно только при работе с очень большими списками. В листинге 3.2 показаны примеры добавления и вставки.

Листинг 3.2. Добавление и вставка элементов списка

```
flavours = ['Chocolate', 'Vanilla']
flavours
['Chocolate', 'Vanilla']

flavours.append('SuperFudgeNutPretzelTwist')
flavours
['Chocolate', 'Vanilla', 'SuperFudgeNutPretzelTwist']

flavours.insert(0, "sourMash")
flavours
['sourMash', 'Chocolate', 'Vanilla', 'SuperFudgeNutPretzelTwist']
```

Для удаления элемента из списка используйте `pop`. При отсутствии аргументов этот метод удаляет последний элемент. Но, используя необязательный аргумент индекса, вы можете указать определенный. В обоих случаях элемент удаляется из списка и возвращается.

В следующем примере из списка извлекается последний элемент, а затем элемент с индексом 0. Вы можете видеть, что оба элемента возвращаются при извлечении, а затем исчезают из списка:

```
flavours.pop()
'SuperFudgeNutPretzelTwist'

flavours.pop(0)
'sourMash'

flavours
['Chocolate', 'Vanilla']
```

Чтобы добавить содержимое из одного списка в другой, используйте `extend`:

```
desserts = ['Cookies', 'Water Melon']
desserts
['Cookies', 'Water Melon']

desserts.extend(flavours)
desserts
['Cookies', 'Water Melon', 'Chocolate', 'Vanilla']
```

Этот метод изменяет первый список так, что теперь к его содержимому добавляется содержимое второго.