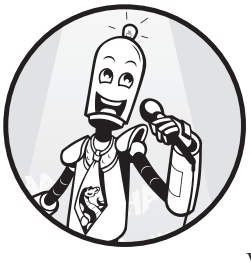
# 2

# Трюки Python



*Трюками* мы будем называть способы необычайно быстрого или легкого решения задач. В книге вы встретите массу различных трюков и методик повышения лаконичности кода, которые к тому же позволят ускорить его реализацию. Хотя приемы Python встретятся вам во всех технических главах данной книги, эта посвящена самому очевидному: трюкам, существенно ускоряющим написание кода, которые можно взять на вооружение быстро и без особых усилий.

Эта глава также играет роль фундамента для последующих, более продвинутых глав. Для понимания материала, который будет изложен далее, вам необходимо освоить навыки, заложенные в однострочниках из текущей главы. В частности, мы охватим широкий спектр простейшей функциональности Python, благодаря которой можно писать эффективный код, в том числе списковые включения, доступ к файлам, функции `map()` и `reduce()`, лямбда-функции, срезы, присваивание срезам, функции-генераторы и функцию `zip()`.

Если вы уже опытный программист, то можете пролистать эту главу и решить самостоятельно, какие вопросы хотите изучить подробнее, а в каких уже и так хорошо разбираетесь.

## Поиск самых высокооплачиваемых работников с помощью спискового включения

В данном разделе вы познакомитесь с прекрасной, очень эффективной и полезной возможностью Python для создания списков: списковым включением (list comprehension). Оно пригодится нам во множестве однострочников далее в книге.

### *Общее описание*

Представьте, что вы работаете в отделе кадров большой компании и вам нужно найти всех сотрудников, зарабатывающих по крайней мере 100 000 долларов в год. Выходные результаты должны представлять собой список кортежей, каждый из которых состоит из двух значений: имени сотрудника и его годовой зарплаты. Ниже представлен соответствующий код:

```
employees = {'Alice' : 100000,
             'Bob' : 99817,
             'Carol' : 122908,
             'Frank' : 88123,
             'Eve' : 93121}

top_earners = []
for key, val in employees.items():
    if val >= 100000:
        top_earners.append((key,val))

print(top_earners)
# [('Alice', 100000), ('Carol', 122908)]
```

И хотя код работает правильно, существует более простой и намного более лаконичный, а значит, и удобочитаемый способ получить тот же результат. При прочих равных условиях решение, занимающее *меньше строк*, будет понятнее для читающего код.

В Python существует замечательный способ создания новых списков: *списковое включение*. Оно описывается простой формулой:

```
[выражение + контекст]
```

Внешние квадратные скобки указывают, что результат представляет собой новый список. *Контекст* указывает, какие элементы списка необходимо

взять. *Выражение* описывает способ модификации элементов списка перед добавлением результата в список. Пример выглядит так:

```
[x * 2 for x in range(3)]
```

Выделенная жирным шрифтом часть, **`for x in range(3)`**, представляет собой контекст, а остальная часть, `x * 2`, — выражение. Выражение удваивает значения `0`, `1`, `2`, сгенерированные контекстом. Таким образом, результат спискового включения представляет собой следующий список:

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

Как выражение, так и контекст могут быть произвольной степени сложности. Выражение может представлять собой функцию от любой описанной в контексте переменной и выполнять любые вычисления — и даже вызывать внешние функции. Задача выражения — модифицировать каждый из элементов списка перед добавлением его в новый список.

Контекст может состоять из одной или нескольких переменных, описанных с помощью одного или нескольких вложенных циклов `for`. Можно также ограничить контекст, задействовав операторы `if`. В данном случае новое значение добавляется в список только при соблюдении заданного пользователем условия.

Списковое включение лучше всего пояснить на примере. Внимательно изучите следующие примеры, и вы поймете, что оно собой представляет:

```
print([❶x ❷for x in range(5)])
# [0, 1, 2, 3, 4]
```

**Выражение ❶**: тождественная функция (не меняет контекст переменной `x`).

**Контекст ❷**: переменная контекста `x` принимает все значения, возвращаемые функцией `range`: `0`, `1`, `2`, `3`, `4`.

```
print([❶(x, y) ❷for x in range(3) for y in range(3)])
# [(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)]
```

**Выражение ❶**: создает новый кортеж из переменных контекста `x` и `y`.

**Контекст ❷**: переменная контекста `x` проходит в цикле по всем значениям, возвращаемым функцией `range (0, 1, 2)`; то же делает и переменная контекста `y`. Эти два цикла `for` — вложенные, вследствие чего переменная

контекста `y` повторяет итерации своего цикла для каждого из значений переменной контекста `x`. Таким образом, получается $3 \times 3 = 9$ сочетаний переменных контекста.

```
print([❶x ** 2 ❷for x in range(10) if x % 2 > 0])
# [1, 9, 25, 49, 81]
```

**Выражение ❶**: функция возведения в квадрат переменной контекста `x`.

**Контекст ❷**: переменная контекста `x` проходит в цикле по всем значениям, возвращаемым функцией `range` — `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, — но только нечетным, то есть когда `x % 2 > 0`.

```
print([❶x.lower() ❷for x in ['I', 'AM', 'NOT', 'SHOUTING']])
# ['i', 'am', 'not', 'shouting']
```

**Выражение ❶**: строковая функция приведения к нижнему регистру переменной контекста `x`.

**Контекст ❷**: переменная контекста `x` проходит в цикле по всем строковым значениям в списке: `'I'`, `'AM'`, `'NOT'`, `'SHOUTING'`.

Теперь вы сможете понять, что происходит во фрагменте кода, который будет показан ниже.

## Код

Рассмотрим уже обсуждавшуюся ранее задачу с зарплатами сотрудников: по ассоциативному массиву со строковыми ключами и целочисленными значениями создать новый список из кортежей (ключ, значение), таких, что соответствующее ключу значение больше или равно 100 000. Соответствующий код приведен в листинге 2.1.

**Листинг 2.1.** Однострочное решение для спискового включения

```
## Данные
employees = {'Alice' : 100000,
             'Bob' : 99817,
             'Carol' : 122908,
             'Frank' : 88123,
             'Eve' : 93121}

## Однострочник
top_earners = [(k, v) for k, v in employees.items() if v >= 100000]
```

```
## Результат
print(top_earners)
```

Каковы же будут результаты выполнения этого фрагмента кода?

### Принцип работы

Рассмотрим этот однострочник подробнее.

```
top_earners = [❶(k, v) ❷for k, v in employees.items() if v >= 100000]
```

**Выражение ❶**: создает простой кортеж (ключ, значение) для переменных контекста `k` и `v`.

**Контекст ❷**: метод ассоциативного массива `dict.items()` обеспечивает проход переменной контекста `k` в цикле по всем ключам ассоциативного массива, а переменной контекста `v` — в цикле по соответствующим переменной контекста `k` значениям, но только если значение переменной контекста `v` равно или больше 100 000, в соответствии с условием `if`.

Результат выполнения этого однострочника выглядит следующим образом:

```
print(top_earners)
# [('Alice', 100000), ('Carol', 122908)]
```

В этой простой однострочной программе мы познакомились с важным понятием *спискового включения*. Такие включения используются во многих местах данной книги, поэтому хорошо разберитесь с примерами в текущем разделе, прежде чем читать дальше.

## Поиск информативных слов с помощью спискового включения

В этом однострочнике мы еще более углубимся в изучение обладающей большими возможностями функциональности списковых включений.

### Общее описание

Поисковые системы ранжируют текстовую информацию по степени соответствия запросу пользователя. Для этого поисковые системы анализируют

содержимое текста, в котором необходимо произвести поиск. Любой текст состоит из слов. В одних содержится немало информации о содержимом текста, а в других — нет. Примеры первых слов — *white*, *whale*, *Captain*, *Ahab*[1] (узнали, откуда это?). Примеры слов второго типа — *is*, *to*, *as*, *the*, *a* и *how*, поскольку они содержатся в большинстве текстов. При реализации поисковых систем часто отфильтровывают слова, не несущие особого значения. Простейший эвристический подход — отфильтровывать все слова из трех или менее букв.

## Код

Наша цель — решить следующую задачу: создать на основе многострочного строкового значения список списков, каждый из которых состоит из всех слов одной из строк, причем слова эти длиной три символа и более. В листинге 2.2 приведены данные и решение.

**Листинг 2.2.** Однострочное решение для поиска информативных слов

```
## Данные
text = '''
Call me Ishmael. Some years ago - never mind how long precisely - having
little or no money in my purse, and nothing particular to interest me
on shore, I thought I would sail about a little and see the watery part
of the world. It is a way I have of driving off the spleen, and regulating
the circulation. - Moby Dick'''

## Однострочник
w = [[x for x in line.split() if len(x)>3] for line in text.split('\n')]

## Результат
print(w)
```

Какими же будут результаты выполнения этого фрагмента кода?

## Принцип работы

Данный однострочник создает список списков с помощью двух вложенных выражений для спискового включения:

- во внутреннем выражении для спискового включения `[x for x in line.split() if len(x)>3]` используется строковая функция `split()`

[1] Белый, кит, капитан, Ахав.

для разбиения заданной строки на последовательность слов. Мы проходим по всем словам `x` и добавляем в список те из них, длина которых не менее трех символов;

- во внешнем выражении для спискового включения создается строковое значение `line`, используемое в предыдущем операторе. Опять же, для разбиения текста по символам новой строки `'\n'` применяется функция `split()`.

Конечно, необходимо научиться думать на языке списковых включений, поэтому в первое время они могут показаться сложными. Но когда вы закончите читать данную книгу, списковые включения станут для вас обыденными и вы будете быстро писать код на языке Python в подобном стиле.

## Чтение файла

В этом разделе мы прочитаем данные из файла и сохраним результат в виде списка строковых значений (по одному на строку). Мы также удалим из прочитанных строк все ведущие и хвостовые пробельные символы.

### *Общее описание*

В Python чтение файла не представляет трудности, но требует обычно нескольких строк кода (и кое-какого поиска в Google). Ниже представлен один из стандартных способов чтения данных из файла в языке Python:

```
filename = "readFileDefault.py" # этот код

f = open(filename)
lines = []
for line in f:
    lines.append(line.strip())

print(lines)
"""
['filename = "readFileDefault.py" # этот код',
'',
'f = open(filename)',
'lines = []',
'for line in f:',
'lines.append(line.strip())',
'',
'print(lines)']
"""
```

Предполагается, что этот фрагмент кода сохранен в файле `readFileDefault.py` в текущем каталоге. Код открывает данный файл, создает пустой список, `lines`, и заполняет его строковыми значениями с помощью операции `append()` в теле цикла `for`, в котором проходит по всем строкам файла. Мы также воспользовались строковым методом `strip()` для удаления всех ведущих и хвостовых пробельных символов (в противном случае в строках бы оказались и символы новой строки `'\n'`).

Для доступа к файлам на компьютере необходимо знать, как их открывать и закрывать. Получить доступ к файлу данных можно только после его открытия. Если файл был закрыт, значит, все данные уже в него записаны. Python может создавать буфер и ожидать некоторое время, пока не запишет весь буфер в файл (рис. 2.1). Причина этого проста: доступ к файлам осуществляется довольно медленно. Из соображений эффективности Python не записывает биты по отдельности, а ждет, пока буфер наполнится достаточным количеством байтов, после чего сбрасывает весь буфер в файл целиком.

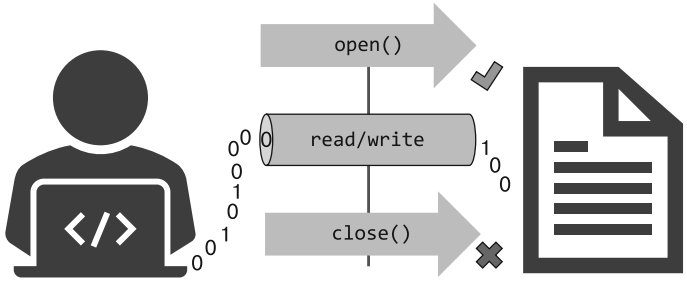



**Рис. 2.1.** Открытие и закрытие файла на языке Python

Именно поэтому рекомендуется с помощью команды `f.close()` закрывать файл после записи в него данных, чтобы гарантировать, что все данные записаны должным образом, а не остались во временной памяти. Однако существует несколько исключений, когда Python закрывает файл автоматически, в частности, когда счетчик ссылок уменьшается до нуля, как вы увидите в следующем коде.

## *Код*

Наша задача: открыть файл, прочитать все строки, удалить ведущие и хвостовые пробельные символы и сохранить результаты в списке. Соответствующий однострочник приведен в листинге 2.3.

**Листинг 2.3.** Однострочное решение для построчного чтения файла

```
print([line.strip() for line in open("readFile.py")])
```

Попробуйте догадаться, какие результаты будут выведены при выполнении этого фрагмента кода, прежде чем читать следующий подраздел.

### Принцип работы

Для вывода полученного списка в командную оболочку мы применили оператор `print()`. Этот список был создан с помощью спискового включения (см. раздел «Поиск самых высокооплачиваемых работников с помощью спискового включения» на с. 42). В части *выражение* спискового включения используется метод `strip()` строковых объектов.

*Контекст* спискового включения проходит в цикле по всем строкам файла.

В результате выполнения этого однострочника будет выведен просто он сам (поскольку он предназначен для чтения своего файла исходного кода на Python — `readFile.py`), обернутый в строковый объект и вставленный в список:

```
print([line.strip() for line in open("readFile.py")])
# ['print([line.strip() for line in open("readFile.py")])']
```

Этот подраздел демонстрирует, что повышение лаконичности кода делает его более удобочитаемым, не нанося какого-либо ущерба эффективности.

## Лямбда-функции и функция map

В этом разделе я познакомлю вас с двумя важными возможностями Python: лямбда-функциями и функцией `map()` — ценными инструментами в наборе Python-разработчика. Мы воспользуемся ими для поиска конкретного значения в списке строковых значений.

### Общее описание

Из главы 1 вы узнали, как описать новую функцию с помощью выражения `def x`, за которым следует тело функции. Однако это не единственный способ описания функции в языке Python. Можно также воспользоваться