

# Software Engineering from Innopolis

Vladimir Maximov

17 июля 2023 г.

## 1 Итерируемые объекты в Python

### 1.1 Цель

Изучить коллекции в Python и работу с ними.

### 1.2 Задачи

- Ознакомиться со списками в Python.
- Рассмотреть популярные операции, используемые над списками slice, filter, map, reduce.
- Ознакомиться с кортежем в Python.
- Разобрать различные подходы создания коллекций.
- Ознакомиться с множествами в Python.
- Понять, чем руководствоваться при выборе коллекции для хранения объектов.

### 1.3 Список

Список - упорядоченная и изменяемая коллекция объектов:

```
1 my_list = [1, 2, 3, None, [0], "word"]
```

Слайс (срез) списка - создание нового списка, используя элементы уже существующего:

```
1 my_list[2:4:1]
```

Берем со 2 элемента включительно по 4 элемент неключительно с шагом 1.

Краткая форма создания списка называется List comprehension (генератор списка):

```
1 [x for x in range(10)]
```

## 1.4 Кортеж

Кортеж - упорядоченная и неизменяемая коллекция:

```
1 (1, 2, 3, 4, 5)
```

## 1.5 Множество

Множество - неупорядоченная коллекция значений, в которой не допускаются повторения и не может содержать не хешируемых объектов:

```
1 {1, 2, 3, "some string"}
```

## 1.6 Словари

Словарь - изменяемая коллекция объектов, которые обладают ключевыми словами. В Python словарь можно описать указанием ключей и значений элементов или генерирующим выражением:

```
1 {"key1" : "value1",  
2  "key2" : "value2",  
3  "key3" : "value3"}
```

## 1.7 Использование памяти

Для хранения коллекций выделяется чуть больше памяти, чем фактически необходимо. Делается это для того, чтобы интерпретатор Python при добавлении элементов выделял память реже.

В Python в коллекции можно хранить разные объекты, каждый из которых может быть непредвиданного размера. Поэтому в Python в списке хранятся указатели на нужные объекты.

## 1.8 Itertools

itertools - модуль, который предоставляет различные функции для работы с итерируемыми объектами. Его можно использовать для упрощения записи операций над итерируемыми объектами. Примеры методов:

itertools.combinations - метод для поиска подмножеств итерируемого объекта, возвращает генератор:

```
1 import itertools
2 itertools.combinations("ABCD", 2)
3 list(itertools.combinations("ABCD", 2))
4 # Вывод: [(A, B), (A, C), (A, D), (B, C), (B, D), (C, D)]
```

itertools.compress - метод, который выбирает из исходного итерируемого объекта элементы, согласно селектору (маске):

```
1 import itertools
2 itertools.compress([1,2,3,4], [1,0,1,0])
3 list(itertools.compress([1,2,3,4], [1,0,1,0]))
4 # Вывод: [1,3]
```

## 2 Функции над итерируемыми объектами

### 2.1 Цель

Ознакомиться с тем, какие операции могут быть сделаны на итерируемых объектах и что можно представить в качестве итерируемого объекта.

### 2.2 Задачи

- Ознакомиться как еще можно работать со списками
- На примерах понять, какие объекты можно представлять в качестве итерируемого объекта.

## 2.3 Itertools (продолжение)

Если стандартного набора библиотеки `itertools` не хватает, то можно использовать сторонний пакет - [more\\_itertools](#), он расширяет возможности работы с итерируемыми объектами. Примеры функций:

Функция `chunked`:

```
1 from more_itertools import chunked
2 iterable = [0, 1, 2, 3, 4, 5, 6, 7, 8]
3 list(chunked(iterable, 3))
4 # Вывод: [[0, 1, 2], [3, 4, 5], [6, 7, 8]]
```

Функция `flatten`:

```
1 from more_itertools import flatten
2 iterable = [(0, 1), (2, 3)]
3 list(flatten(iterable))
4 # Вывод: [0, 1, 2, 3]
```

Функция `split_at`:

```
1 from more_itertools import split_at
2 list(split_at('abdcdba', lambda x: x == 'b'))
3 # Вывод: [['a'], ['c', 'd', 'c'], ['a']]
```

Функция `transpose`:

```
1 from more_itertools import transpose
2 list(transpose([(1, 2, 3), (11, 22, 33)]))
3 # Вывод: [['a'], ['c', 'd', 'c'], ['a']]
```

Функция `windowed`:

```
1 from more_itertools import windowed
2 all_windows = windowed([1, 2, 3, 4, 5], 3)
3 list(all_windows)
4 # Вывод: [(1, 2, 3), (2, 3, 4), (3, 4, 5)]
```

## 2.4 Filter, Map, Reduce

Иногда для целесообразного использования памяти лучше написать часть программы в функциональном стиле. Особенно это применимо когда нужно пройти по большому количеству элементов. Для этого в `python` существуют встроенные функции. Их использование будет рассмотрено на простых примерах.

### 2.4.1 Filter

Filter - функция, которая позволяет выбрать из любого итерируемого объекта элементы удовлетворяющие условию. В качестве аргумента принимает функцию или лямбда-выражение и список, из которого отфильтрует значения:

```
1 items = [-5,-4,-3,-2,-1,0,1,2,3,4,5]
2 def my_filter_expr(item):
3     return item > 0
4 positive_items = tuple(filter(my_filter_expr, items))
5 print(positive_items)
6 # Вывод: (1, 2, 3, 4, 5)
```

Важно понимать что получившийся объект не хранит в себе все элементы получившегося списка. Он хранит информацию о том как можно получить каждый из последующих элементов списка. Важно понимать как это работает чтобы в дальнейшем не попадать на ошибки. Самый близкий родственник объекту к получившемуся фильтру - итератор. Итератор это специальный объект в python который выдает по одному элементу и по нему можно пройти только один раз.

### 2.4.2 Map

Map - функция, которая позволяет применить функцию ко всему списку значений. В качестве аргумента принимает функцию или лямбда-выражение и список, к элементам которого будет применена функция:

```
1 list(map(lambda a: a[0]**a[1], [(0, 2), (1, 2), (2, 2)]))
2 # Вывод: [0, 1, 4]
```

### 2.4.3 Reduce

Reduce - функция, применяющая другую функцию к последовательным парам значений в списке, аналог функции Fold в Wolfram Mathematica:

```
1 from functools import reduce
2 many_items = [1,2,3,4]
3 product = reduce(lambda a, b: a * b, many_items)
4 # Вывод: 24
```

В начале в качестве переменной  $a$  функция берет значение 1, в качестве переменной  $b$  - 2, на втором шаге переменная  $a$  - результат функции на предыдущей итерации,  $b$  - 3 и т.д. У функции есть третий необязательный аргумент - начальное значение.

## 2.5 Метод скользящего окна

Метод скользящего окна представляет собой процесс, при котором окно фиксированного размера последовательно перемещается по набору данных. Значения внутри окна анализируются и обрабатываются для получения новых результатов, например, вычисления среднего или медианного значения.

## 2.6 Операции из линейной алгебры

Линейная алгебра это раздел математики, основными конструкциями в которой являются списочный типы данных. Матрицы, векторы, тензоры - это все понятия из линейной алгебры. Самые базовые математические операции в линейной алгебре - это операции на этих структурах определенных в линейном пространстве - транспонирование, перемножение матриц и векторов, нахождение определителей, решение системы линейных уравнений. Пример перемножения матриц в помощью `more_itertools`:

```
1  from more_itertools import matmul
2  matrix1 = [
3      [1,2,3],
4      [4,5,6],
5      ]
6  matrix2 = [
7      [7, 8],
8      [9, 10],
9      [11, 12],
10     ]
11  list(matmul(matrix1, matrix2))
12  # Вывод: [[58, 64], [139, 154]]
```

## 2.7 Линейная алгебра с numpy

numpy - библиотека, которая заточена под выполнение операций на матрицах. Основные функции можно посмотреть по [ссылке](#). Пример перемножения матриц с помощью numpy:

```
1 import numpy as np
2 matrix1 = np.array([
3     [1,2,3],
4     [4,5,6],
5 ])
6 matrix2 = np.array([
7     [7, 8],
8     [9, 10],
9     [11, 12],
10 ])
11 matrix1 @ matrix2
12 # Вывод: array([[58, 64], [139, 154]])
```

## 3 Базы данных и SQL, работа с бд из Python

### 3.1 Цель

Рассмотреть базы данных как один из возможных источников данных для работы.

### 3.2 Задачи

- Разобраться как работать с табличными данными
- Понять, как можно обращаться с табличными данными из python
- Выяснить, каким инструментарием можно воспользоваться для работы с табличными данными.

### 3.3 Табличное представление данных, именованные данные, работа с ними

Когда мы работаем с данными, часто так бывает что они представлены в виде таблиц. То есть представляют собой последовательные наборы строк с данными, в которой каждый отдельный элемент может быть ассоциирован с наименованием и/или типом данных.

### 3.4 Реляционные базы данных

Табличное представление свойственно для реляционных БД. База данных (БД) - это специальным образом организованная совокупность данных о некоторой предметной области, хранящаяся во внешней памяти компьютера. В таких таблицах все поля не только уникально проименованы, но и имеют строгий тип данных, а также могут иметь ограничения по уникальности и связи с другими таблицами. Пройденный курс по SQL с заданиями в СУБД MySQL по [ссылке](#).

### 3.5 Работа из python: sqlite3, psycopg2, pandas

Для работы с данными из реляционных БД в языке программирования python существуют библиотеки для низкоуровневого взаимодействия с ними. Для связи с БД sqlite, которая не запускается на отдельном сервере а хранится файлом в локальной файловой системе, существует модуль sqlite3, который не нужно устанавливать отдельно, он встроен в стандартную библиотеку python. В его стандартный функционал входит соединение с базой, исполнение sql выражений и работа с полученным от результата выражением как с объектом знакомым языку программирования (итерируемый объект).

```
1     import sqlite3
2     from contextlib import closing
3
4     sqlite_filename = "datasets/movies.sqlite"
5     con = sqlite3.connect(sqlite_filename)
6     def dict_factory(cursor, row):
7         d = {}
8         for idx, col in enumerate(cursor.description):
9             d[col[0]] = row[idx]
10        return d
11
12    con.row_factory = dict_factory
13
14    with closing(con.cursor()) as cur:
15        for row in cur.execute("SELECT * FROM movies JOIN\
16        \directors ON movies.director_id = directors.id"):
17            print(row)
```

В данном случае мы с использованием sqlite3 получаем все фильмы из таблицы *movies*, режиссеры которых присутствуют в таблице *directors*.



Самым классическим инструментом для работы с датасетами является pandas. Помимо того чтобы работать с данными из электронных таблиц, он может забрать данные из sql-query, если в качестве параметра ему передать соединение с базой. Данные, полученные с помощью sql select можно сразу же загрузить в датафрейм. у pandas есть такой инструмент.

```
1 import pandas as pd
2 import pandas.io.sql as psql
3
4 connection = psycopg2.connect("dbname=test user=student1\
5 \password=student1 host=212.109.218.158 port=5432")
6 dataframe = psql.read_sql('SELECT * FROM files JOIN\
7 \person ON files.owner = person.id;', connection)
```

### 3.6 Краткое объяснение нереляционных баз данных на примере MongoDB

Не все БД занимаются хранением связанных прямоугольных таблиц. Это происходит потому что тип хранилища состоящий из строго типизированных связанных таблиц подходит не для всех типов задач. Один из примеров такой базы - MongoDB.

Mongo - популярная хостируемая на сервере документная (не реляционная, NoSQL) БД. Структура данных в этой и подобных базах отличается от всего пройденного до этого. Напоминает словари в python или же json файл. Причинами выбора именно нереляционной базы для вашего проекта могут быть следующие:

- В проекте (или его части) не нужны взаимоотношения между объектами, вы собираетесь хранить много данных в коллекции, каждый объект в которой самостоятелен сам по себе. Часто это фронтенд-ориентированные проекты или части проектов. (отсутствие связей между объектами и быстрый поиск по коллекции позволяют быстро выдавать данные для отображения на сайте или мобильном устройстве)
- Заранее не известна модель хранимых данных
- Отличная масштабируемость