Software Engineering from Innopolis

Vladimir Maximov

27 июня 2023 г.

1 Итеритуемые 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 невключительно с шагом 1.

Краткая форма создания списка называется List comprehention (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 - неупоярдоченная 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:

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

Функция flatten:

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

Функция split_at:

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

Функция transpose:

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

Функция windowed:

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

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

Важно понимать что получившийс 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 родственный объект к получившемуся фильтру - итератор. Итератор это специальный объект в python который выдает по одном элементу и по нему можно пройтись 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:

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

```
from functools import reduce
many_items = [1,2,3,4]
product = reduce(lambda a, b: a * b, many_items)
# 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 линейныхуравнений. Пример перемнжения 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 фукнции 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]])
```