Software Engineering from Innopolis

Vladimir Maximov

18 июля 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]])
```

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 инстументарием можно воспользоваться для работы с табличными данными.

3.3 Табличное представление данных, именованные данные, работа с ними

Когда мы работае с данными, часто так бывает что они представлены в виде таблиц. То есть представлят 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 резаультата 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 данныими 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 типизорованных связанных таблиц подзодит 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

4 numpy / pandas

4.1 Цель

Ознакомиться с функционалом основного инструмента анализа данных.

4.2 numpy

Самый фундаментальный пакет для работы с данными. Часто он выступает основой для других пакетов работы с данными. В частности это происходит потому что разработчики библиотеки на очень глубинном уровне переработали взаимедействие python с коллекциями (особенно многомерными). В результате чего работа с numpy.ndarray происходит значительны быстрее по скорости чем со стандартным list.

Основной пакет, который будет использоваться - numpy.linalg. Все алгоритмы, связанные с линейной алгеброй есть в данном пакете.

4.3 pandas

Обычно при работе с данными мы наблюдаем что каждый кусочек данных помимо числовых значений может относиться к разным категориям, быть логически отделимым от другого кусочка таких же данных по набору признаков. Тут для анализа данных появляется такой инструмент как pandas. Он реализует работу с кусками данных. Тип объекта с данными который загружен с помощью pandas будет называтья DataFrame, а одна отдельная запись из датафрейма будет называться series.

Внутри себя pandas использует алгебру реализованную в NumPy. pandas имеет под собой зависимость в numpy.

Series (серия)- это одномерный, именованный (помеченный) массив, который может хранить данные любого типа, включая целые числа, числа с плавающей точкой, строки и объекты Python. Базовая структура серии состоит из двух компонентов: данных и индекса. В серии данные представляют собой одномерный массив NumPy, содержащий фактические значения данных.

```
data_frame_covid.T.iloc[:,0]
# Вывод:
# country         Afghanistan
# continent              Asia
# date             2020-04-12
# day                      12
# month                     4
# year                   2020
# cases                    34
# deaths                    3
# country_code            AFG
# population       37172386.0
# Name: 0, dtype: object

type(data_frame_covid.T.iloc[:,0])
# Вывод: pandas.core.series.Series
```

Индекс - это структура, похожая на массив, которая помечает каждый элемент массива данных. По умолчанию индекс представляет собой последовательность целых чисел, начинающуюся с нуля, но также можно предоставить пользовательские метки для индекса. Это облегчает обращение и манипулирование элементами данных на основе их меток. Чтобы создать серию, можно использовать конструктор pd.Series() и передать список, словарь или массив NumPy в качестве данных. Также можно передать необязательный параметр индекса, чтобы указать пользовательские метки для элементов данных.

DataFrame (ДатаФрейм, Таблица данных) - это двумерная, изменяемая в размере и гетерогенная структура табличных данных с маркированными осями (строками и столбцами). Его можно рассматривать как коллекцию объектов Series, где каждый столбец представляет собой объект Series. DataFrames предназначены для обработки широкого спектра типов данных, что делает их универсальным и мощным инструментом для обработки и анализа данных. Базовая структура DataFrame состоит из трех компонентов: данных, индекса и столбцов. В DataFrame данные хранятся в виде коллекции из одного или нескольких объектов Series, каждый столбец которых представляет собой отдельную Series. Данные в каждой Series могут быть различных типов, таких как целые числа, числа с плавающей точкой или строки.

Столбцы являются структурой, подобной массиву, которая помещает метку на каждый столбец в DataFrame.

```
1   data_frame_covid.columns
2   # Вывод: Index(['country', 'continent', 'date', 'day',
3   #       'month', 'year', 'cases',
3   #       'deaths', 'country_code', 'population'],
4   #       dtype='object')
```

Как Series так и DataFrame предоставляют методы для анализа данных, такие как:

- Выборка данных: можно указать особые строки и колонки с которыми вы будете работать, используя индексы, слайсы, условия
- Очистка данных: можно заполнять пропущенные данные, удалять записи в которых не хватает данных.
- Преобразование данных: можно применять функции по всей длинне фрейма или по его отдельным частям
- Аггрегация данных: можно вычислять такие оценочные величины как сумма, среднее, количество.А также группировать данные по своим критериям при помощи groupby.
- Сортировка: можно упорядочить данные основываясь на значениях в колонках, а также по своим критериям
- Функциональность временных рядов: вы можете работать с данными временных рядов, пересчитывать данные с разной частотой и выполнять вычисления с использованием скользящего окна в DataFrame или Series.

В качестве основных типов данных для работы, pandas использует типы данных из NumPy по пораву наследования, ознакомиться с которыми можно по ссылке.

Во многом успех работы с данными зависит в скорости доступа к данным и возможности быстро отфильтровать ненужное. Для ускорения работы нам могут помочь индексы. При создании датафрейма у него имеется стандартный индекс для быстрого доступа через него до строк данных. Выбирать такие строки можно с помощью функций *.loc*() и *.iloc*() (i==integer). Для датафрейма можно установить новый индекс (при этом либо заменив старый либо создав новый датафрейм из существующего с новым индексом).

Комбинирование фреймов:

```
1   data_frame_covid[1:3]
2       .append(data_frame_covid[-3:-1], ignore_index=True)
```

	country	continent	date	day	month	year	cases	deaths	country_code	population
0	Albania	Europe	2020-04-12	12	4	2020	17	0	ALB	2866376.0
1	Algeria	Africa	2020-04-12	12	4	2020	64	19	DZA	42228429.0
2	Yemen	Asia	2020-04-12	12	4	2020	0	0	YEM	28498687.0
3	Zambia	Africa	2020-04-12	12	4	2020	0	0	ZMB	17351822.0

Рис. 1: результат комбинирования фреймов

Для того чтобы выбрать несколько условий для фильтрации датафрейма можно воспользоваться методом query(), он позволяет в одну строку записать все условия фильтрации которые вам могут быть нужны при выборе данных. при этом в результате вы получите новый фрейм с данными.

```
1   data_frame_covid.query("cases > 5")
```

	country	continent	date	day	month	year	cases	deaths	country_code	population
0	Afghanistan	Asia	2020-04-12	12	4	2020	34	3	AFG	37172386.0
1	Albania	Europe	2020-04-12	12	4	2020	17	0	ALB	2866376.0
2	Algeria	Africa	2020-04-12	12	4	2020	64	19	DZA	42228429.0
3	Andorra	Europe	2020-04-12	12	4	2020	21	2	AND	77006.0
7	Argentina	America	2020-04-12	12	4	2020	162	7	ARG	44494502.0
...	...	...	...	...	...	...	...	...	...	...
194	United_Arab_Emirates	Asia	2020-04-12	12	4	2020	376	4	ARE	9630959.0
195	United_Kingdom	Europe	2020-04-12	12	4	2020	8719	839	GBR	66488991.0
198	United_States_of_America	America	2020-04-12	12	4	2020	28391	1831	USA	327167434.0
199	Uruguay	America	2020-04-12	12	4	2020	7	0	URY	3449299.0
200	Uzbekistan	Asia	2020-04-12	12	4	2020	172	1	UZB	32955400.0

Рис. 2: фильтрация фрейма