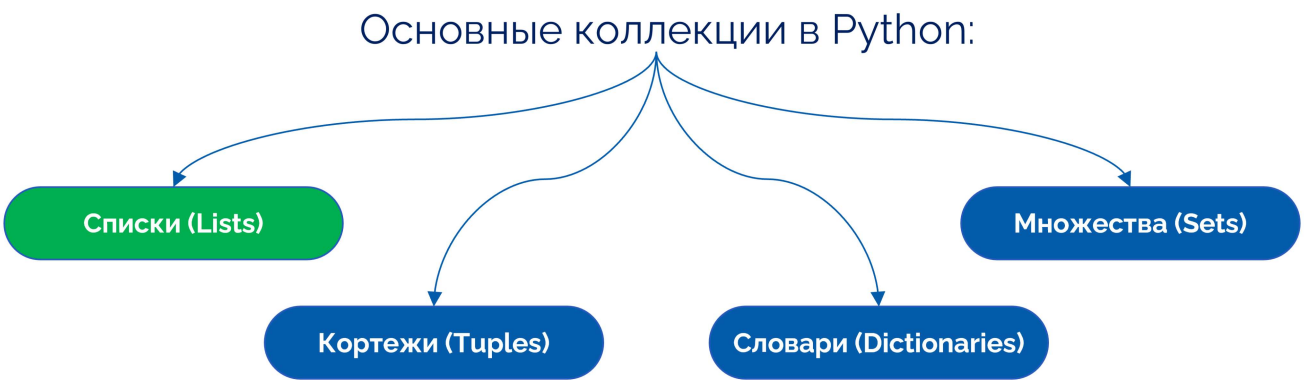
День №2

Погружение в структуры Python

¶

I. Контейнеры

Контейнеры (коллекции) в Python – особые структуры данных, которые позволяют хранить в себе информацию в различном виде.



Что вернет написанный ниже код?

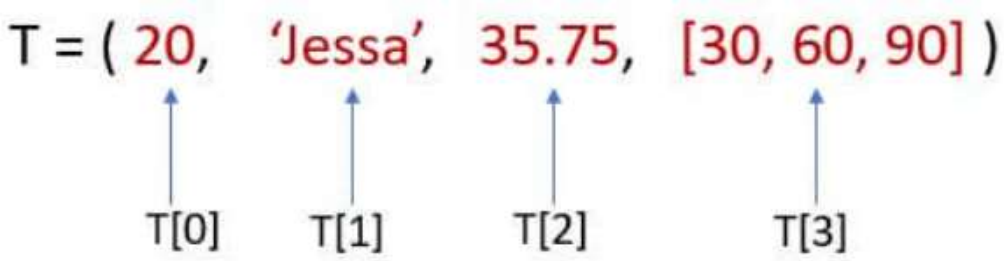
```
a = ['a', 'b', 'c’]
a.append(['d', 'e’]))
print(a)
```

Ввод [47]:

```
a = ['a', 'b', 'c']
a.append(['d', 'e'])
print(a)
```

```
['a', 'b', 'c', ['d', 'e']]
```

Кортежи – последовательность элементов, которая очень похожа на списки за тем исключением, что кортежи являются неизменяемыми. Для создания кортежей используются круглые скобки **(,)** или функция **tuple()**.



- ✓ **Ordered**: Maintain the order of the data insertion.
- ✓ **Unchangeable**: Tuples are immutable and we can't modify items.
- ✓ **Heterogeneous**: Tuples can contains data of types
- ✓ **Contains duplicate**: Allows duplicates data

Создается кортеж с помощью круглых скобок **( )** или функции **tuple( )**. Создадим пустой кортеж и проверим, какой тип данных ему присвоен

Ввод [48]:

```
# Создадим пустой кортеж и посмотрим, какой тип данных присвоил ему Python
my_tuple = tuple()
print(type(my_tuple))
```

```
<class 'tuple'>
```

Ввод [50]:

```
# Создадим кортеж и применим к нему срезы
my_tuple = (0, 1, 7, 2, 9)
print(my_tuple)
print(my_tuple[:2])
```

```
(0, 1, 7, 2, 9)
(0, 1)
```

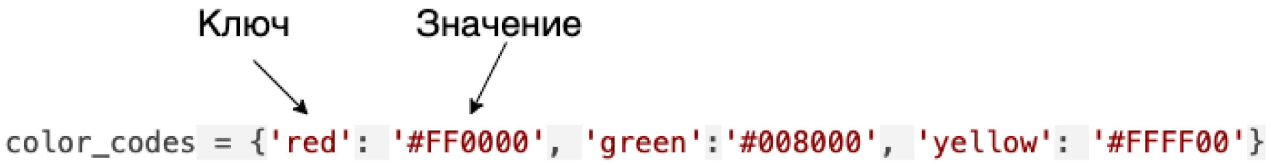
Ввод [51]:

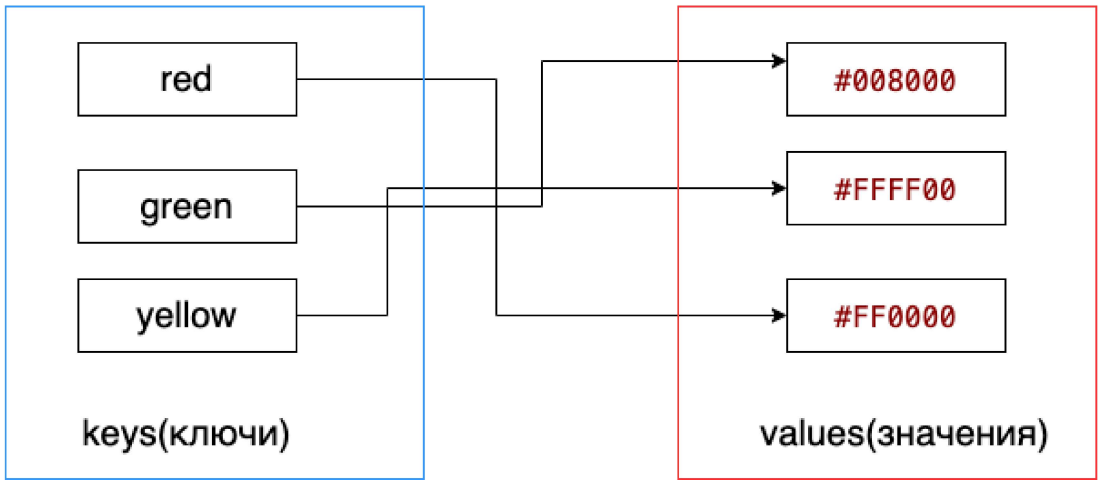
```
# Попробуем изменить кортеж
my_tuple[0] = 10
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-51-89bfecd96303> in <module>
      1 # Попробуем изменить кортеж
----> 2 my_tuple[0] = 10

TypeError: 'tuple' object does not support item assignment
```

Словари – ассоциативный массив (или хэш), то есть коллекция, содержащая пары «Ключ : Значение». Для создания словарей используются фигурные скобки **{ }** или функция **dict()** ,а в качестве аргументов передаются «Ключи : Значения».





Создаются словари с помощью фигурных скобок **{ }** или функции **dict( )**, где в обязательном порядке передаются ключ-значение. Создадим пустой словарь и проверим, какой тип данных ему присвоен

Ввод [52]:

```
# Создадим пустой словарь и посмотрим, какой тип данных вернет Python
my_dict = dict()
print(type(my_dict))
```

```
<class 'dict'>
```

Ввод [57]:

```
# Создадим словарь с именами студентов как ключ и возрастом как значение
my_dict = {'Alex':20, 'Liza':19, 'Tom':21}
print(my_dict)

print(my_dict['Liza'])
print(my_dict.get('Liza'))
my_dict['Jack'] = 20
print(my_dict)
print(my_dict['Jack'])
```

```
{'Alex': 20, 'Liza': 19, 'Tom': 21}
19
19
{'Alex': 20, 'Liza': 19, 'Tom': 21, 'Jack': 20}
20
```

Python поддерживает следующие основные методы словарей:

- **dict.clear()** - очищает словарь.
- **dict.copy()** - возвращает копию словаря.
- **classmethod dict.fromkeys(seq[, value])** - создает словарь с ключами из seq и значением value (по умолчанию None).
- **dict.get(key[, default])** - возвращает значение ключа, но если его нет, не бросает исключение, а возвращает default (по умолчанию None).
- **dict.items()** - возвращает пары (ключ, значение).
- **dict.keys()** - возвращает ключи в словаре.
- **dict.pop(key[, default])** - удаляет ключ и возвращает значение. Если ключа нет, возвращает default (по умолчанию бросает исключение).
- **dict.popitem()** - удаляет и возвращает пару (ключ, значение). Если словарь пуст, бросает исключение KeyError. Помните, что словари неупорядочены.
- **dict.setdefault(key[, default])** - возвращает значение ключа, но если его нет, не бросает исключение, а создает ключ со значением default (по умолчанию None).
- **dict.update([other])** - обновляет словарь, добавляя пары (ключ, значение) из other. Существующие ключи перезаписываются. Возвращает None (не новый словарь!).
- **dict.values()** - возвращает значения в словаре.

Множества (Sets)

Множества – неупорядоченная коллекция уникальных элементов. Для создания множеств используются круглые скобки **{ }** или функция **set()**.

Ключевые особенности множеств:

- Множества – неупорядоченные коллекции произвольных данных
- Множества содержат уникальные элементы
- Элементами множества могут быть только неизменяемые объекты
- Могут быть гетерогенными

Создаются множества с помощью функции **set( )**, которой в качестве аргумента передается последовательность. Создадим пустое множество и проверим, какой тип данных ему присвоен

Ввод [58]:

```
# Создадим пустое множество и проверим тип данных
my_set = set()
print(type(my_set))
```

```
<class 'set'>
```

Ввод [61]:

```
# Создадим множество содержащее числа от 0 до 5
my_set = set((0, 7, 8, 9, 18, 29, 0))
print(my_set)
set()
```

```
{0, 7, 8, 9, 18, 29}
```

Язык Python поддерживает ряд следующих операций над множествами:

- in – проверка элемента на вхождение в множество;
- – (минус) – разность множеств;
- | – объединение множеств;
- & – пересечение множеств.

Ввод [64]:

```
# проверка элемента на вхождение
my_set_1 = set((0, 7, 8, 9, 18, 29, 0))
my_set_2 = set((1, 9, 8, 7, 9, 8, 10, 11))

print(my_set_1, my_set_2)
print(7 in my_set_1)
```

```
{0, 7, 8, 9, 18, 29} {1, 7, 8, 9, 10, 11}
True
```

Ввод [65]:

```
# разность множеств
my_set_1 - my_set_2
```

Out[65]:

```
{0, 18, 29}
```

Ввод [67]:

```
# объединение множеств
my_set_1 | my_set_2
```

Out[67]:

```
{0, 1, 7, 8, 9, 10, 11, 18, 29}
```

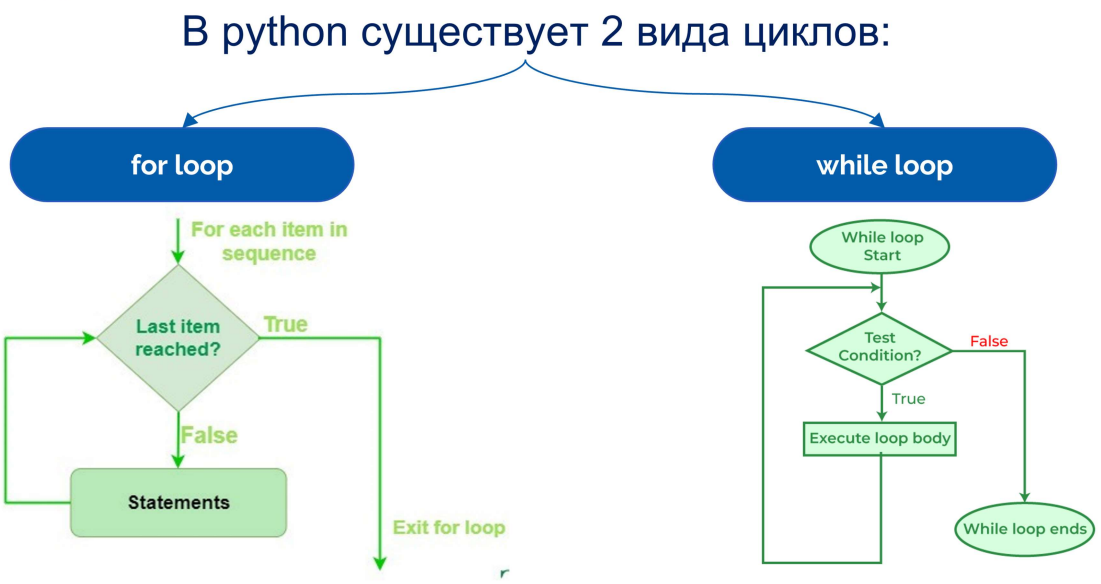
Ввод [68]:

```
# пересечение множеств
my_set_1 & my_set_2
```

Out[68]:

```
{7, 8, 9}
```

Циклы (loops) используются для повторения определенного блока кода.



Ввод [69]:

```
# Выведем на печать 10 раз фразу "Hello, world" с помощью циклов
for element in range(10):
    print('Hello, world')
```

```
Hello, world
Hello, world
Hello, world
Hello, world
Hello, world
Hello, world
Hello, world
Hello, world
Hello, world
Hello, world
```

Ввод [73]:

```
# Создадим лист из 5 элементов и с помощью цикла for напечатаем все элементы
print(my_list)
for num in my_list:
    print(num, num**2)
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
0 0
2 4
4 16
6 36
8 64
10 100
12 144
14 196
16 256
18 324
20 400
```

Ввод [ ]:

```
# Попробуем рассчитать сумму всех элементов от 0 до 1000 с помощью циклов
pass
```

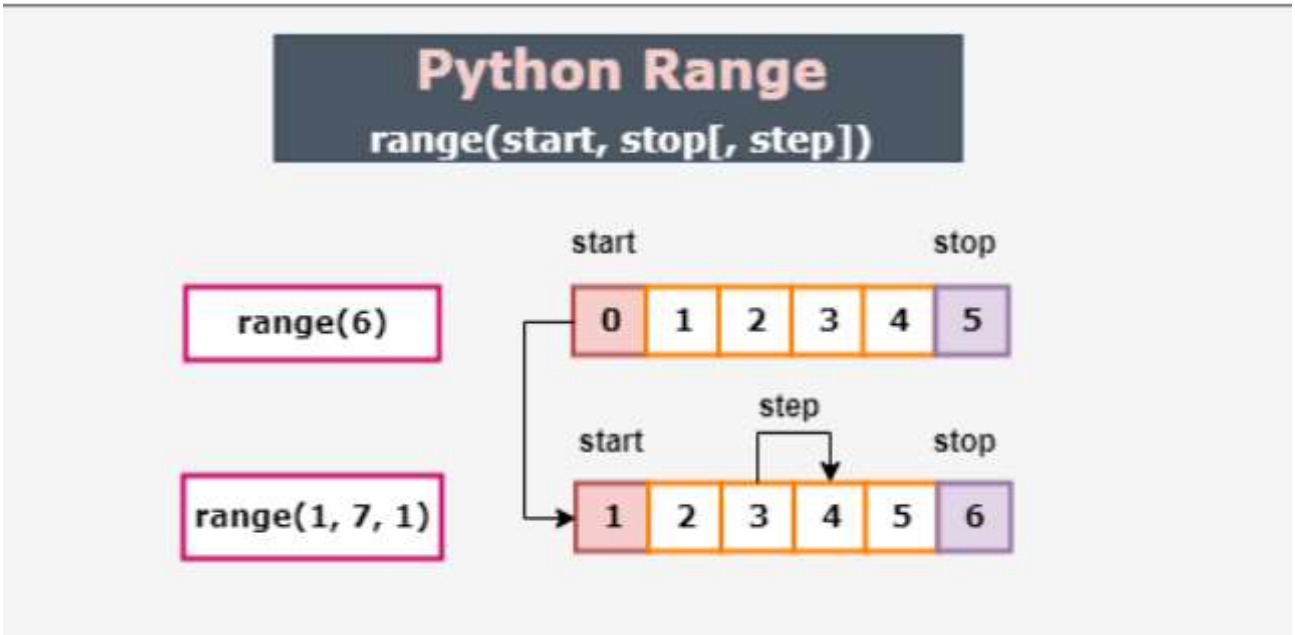
Ввод [75]:

```
# Попробуем сделать то же самое с помощью цикла while
number = 1
summa = 0
while number!= 345+1:
    summa += number
    number +=1

print(summa)
```

```
59685
```

Важной функцией, которая связана с циклами в Python, является функция range. Функция позволяет генерировать арифметическую прогрессию с заданными параметрами, которую можно использовать для циклов.



Ввод [3]:

```
# С помощью цикла и функции range выведем на печать значения от 1 до 20
pass
```

Функцию range можно использовать для того, чтобы обращаться к элементам списка по индексам. Попробуем вывести каждый второй элемент из списка с помощью индексов из функции range

Ввод [ ]:

```
# Цикл for и применение функции range
pass
```

Давайте попрактикуемся!

1. Дана случайная текстовая строка. Верните новую строку, в которой каждая буква или символ исходной строки повторняется дважды.

Пример: "Hello, Lesh" - "HHeellllloo, LLeesshh"

Ввод [81]:

```
my_text = 'Hello, Lesh'
empty_text = ''

for i in my_text:
    if i not in [',', ' ']:
        empty_text += i*2
    else:
        empty_text += i

print(empty_text)
```

```
HHeellllloo, LLeesshh
```

2. Посчитайте, сколько високосных годов было в нашей эре (с 0 года)

Ввод [82]:

```
2023//4
```

Out[82]:

```
505
```

3. Вы - разработчик программы лайков в соцсети. Вам нужно написать код, который принимает список пользователей, которые поставили лайки посту, и возвращает текстовое описание в следующем виде:

[] --> "no one likes this"
["Peter"] --> "Peter likes this"
["Jacob", "Alex"] --> "Jacob and Alex like this"
["Max", "John", "Mark"] --> "Max, John and Mark like this"
["Alex", "Jacob", "Mark", "Max"] --> "Alex, Jacob and 2 others like this"

Ввод [85]:

```
likes_list = ["Alex", "Jacob"]

if not likes_list:
    print('no one likes this')
elif len(likes_list)==1:
    print(likes_list[0] + ' likes this')
elif len(likes_list)==2:
    print(likes_list[0] + ' and '+ likes_list[1] + ' likes this')
elif len(likes_list)==3:
    print(likes_list[0] + ' and '+ likes_list[1] + ' likes this')
else:
    print('else condition')
```

```
Alex and Jacob likes this
```

4. Вам дан случайный список из положительных и отрицательных чисел, верните сумму всех положительных

Ввод [35]:

```
pass
```

5. Напишите код, который принимать на вход текст, а возвращает тот же самый текст, но слова длиннее 5 символов возвращаются в обратном порядке.

Например, spinWords("Hey fellow warriors") => returns "Hey wollef sroirraw"
spinWords("This is a test") => returns "This is a test"
spinWords("This is another test")=> returns "This is rehtona test"

Подсказка: разделить текст на слова по пробелам можно с помощью метода .split()

Ввод [36]:

```
1 pass
```

II. Модули (библиотеки) в Python

Как и любой другой язык программирования, в Python есть возможность загрузить внешние модули (их еще называют библиотеками). Эти модули могут быть как частью стандартных библиотек, которые уже включены в python, так и глобальными внешними модулями. Например, для получения значения числа pi существует модуль math, который входит в стандартную библиотеку.

Для импорта модуля используют команду import
Попробуем импортировать модуль math и вывести на экран число pi

Ввод [88]:

```
import math
print(math.pi)

import os
os.getcwd()
```

```
3.141592653589793
```

Out[88]:

```
'C:\\Users\\Admin\\Desktop\\ЛЭШ\\Анализ данных 2023\\Занятие №2'
```

В Python огромное количество модулей (больше 175 тысяч). Среди них есть самые популярные, которые позволяют применять python в абсолютно разных сфера. У каждого модуля есть документация, в которой описано, зачем модуль нужен и как с ним работать.

В python большое количество полезных базовых модулей, которые применяются практически в каждом проекте. Познакомимся с некоторыми из них.

- os - модуль для работы с файловой системой
- random - модуль для работы со случайными числами
- datetime - модуль для работы с датами
- pandas - модуль для работы с табличными пространствами
- matplotlib - модуль для визуализации
- numpy - модуль для работы с массивами

III. Pandas - основа большинства проектов по анализу данных и машинному обучению

Библиотека pandas является самым популярным инструментов в анализе данных. Она дает возможность преобразовывать, очищать, агрегировать, транформировать, визуализировать и анализировать любые структурированные данные. Она также тесно связана с другими библиотеками,

которые позволяют строить модели машинного обучения, нейросети и т.д.

1. Полезные ссылки:

1) Документация библиотеки pandas (https://pandas.pydata.org/docs/user_guide/index.html) (https://pandas.pydata.org/docs/user_guide/index.html)
2) 10 minutes to pandas (https://pandas.pydata.org/docs/user_guide/10min.html) (https://pandas.pydata.org/docs/user_guide/index.html)
3) Pandas Cookbook (https://pandas.pydata.org/docs/user_guide/cookbook.html) (https://pandas.pydata.org/docs/user_guide/cookbook.html)
4) Top pandas guide youtube (https://www.youtube.com/watch?v=vmEHCJofslg) (https://www.youtube.com/watch?v=vmEHCJofslg)

2. Первые шаги в pandas

2.1. Установка библиотеки

Перед тем, как начать работать с pandas, ее нужно установить. Библиотека входит в стандарный пакет anaconda.

>Если библиотека pandas у вас не установлена, в командной строке (win + R для ОС Windows) выполните команду **pip install pandas**

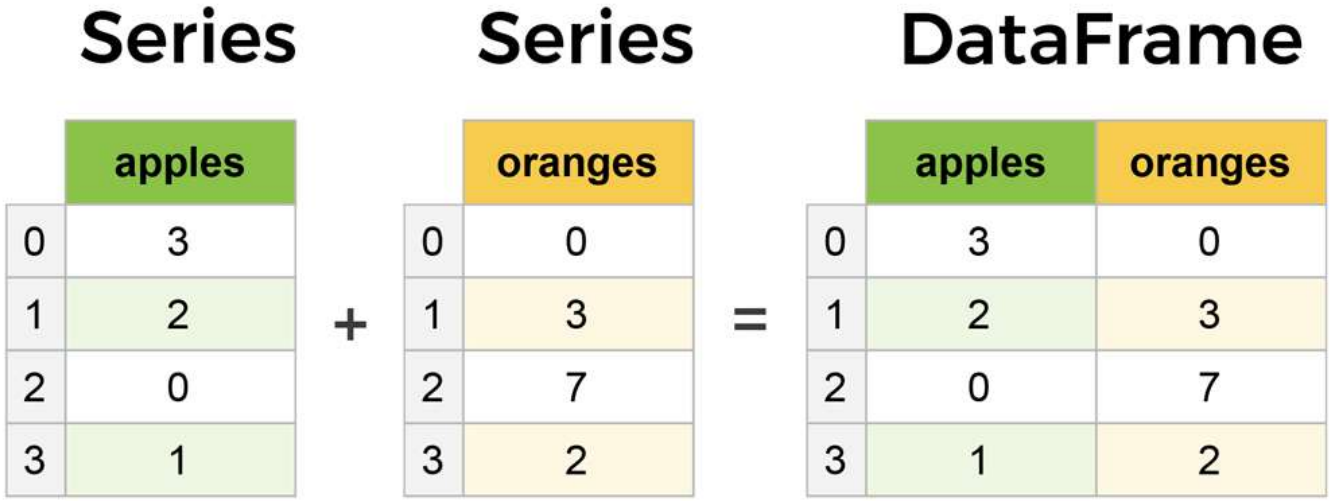
Давай попробуем импортировать библиотеку и понять, какой она у нас версии. Сделать это можно с помощью команды **import** и встроенного "магического метода" **version**

Ввод [89]:

```
# Импорт библиотеки и проверка её версии
import pandas as pd
print(pd.__version__)
```

```
1.2.4
```

В pandas существует 2 основных структуры данных - Series (серия) и DataFrame (датафрейм).



Series (серия) выступает в роли одномерного массива. Создать серию можно с помощью функции **pandas.Series()**, которая в качестве аргумента принимает на вход список (не только). Серии очень похожи на списки, но отличаются по поведению. Операции применяются к списку целиком, а в случае с сериями - к ее элементам.

Задача: создать лист из целых чисел, а также серию из созданного листа, после чего вывести их на экран.

Ввод [94]:

```
# Создание листа с целыми числа, серии и вывод на экран
print(my_list)
s = pd.Series(my_list)
print(s)
print(type(s))
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
0      0
1      2
2      4
3      6
4      8
5     10
6     12
7     14
8     16
9     18
10    20
dtype: int64
<class 'pandas.core.series.Series'>
```

В отличие от Series, Dataframe - это многомерные массивы (фактически таблицы). У них есть строки (rows), столбцы (columns) и ячейки (cells). Технически датафрейм состоит из набора серий. Создать датафрейм можно с помощью функции **pandas.DataFrame()**, которая на вход принимает различные последовательности.

Задача: создать датафрейм, состоящий из 2 столбцов целых чисел, вывести на экран тип данных и сам датафрейм

Ввод [96]:

```
# Создание датафрейма из 2 колонок
df = pd.DataFrame(data={'col_1':[0, 1, 2, 3], 'col_2':[4,5,6,7]})
print(type(df))
df
```

```
<class 'pandas.core.frame.DataFrame'>
```

Out[96]:

	col_1	col_2
0	0	4
1	1	5
2	2	6
3	3	7

2.2. Манипулирование данными

Пока мы научились создавать серии и датафреймы. Давайте пройдем по базовым командам по выбору полей и работе с ними.

Задача: создать датафрейм, состоящий из 5 столбцов и 100 строк случайных чисел от 0 до 200. Подсказка: используйте функцию np.random.randint() для создания массива данных

Ввод [98]:

```
# Создание случайного датафрейма
import numpy as np
import pandas as pd

df = pd.DataFrame(np.random.randint(0, 200, (100, 5)), columns=['a', 'b', 'c', 'd',
df
```

Out[98]:

	a	b	c	d	e
0	32	36	121	20	15
1	180	91	150	51	150
2	154	128	41	95	108
3	151	33	48	199	42
4	30	120	68	114	143
...	...	...	...	...	...
95	122	171	150	14	38
96	41	110	168	96	114
97	175	144	163	98	90
98	31	150	188	93	153
99	64	23	8	132	195

100 rows × 5 columns

Ввод [101]:

```
# Выберем столбец a и разделим каждый из его элементов на
# 2 и присвоим новые значение столбцу "new_a"
df['new_a'] = df['a']/2
df.head()
```

Out[101]:

	a	b	c	d	e	new_a
0	32	36	121	20	15	16.0
1	180	91	150	51	150	90.0
2	154	128	41	95	108	77.0
3	151	33	48	199	42	75.5
4	30	120	68	114	143	15.0

Ввод [106]:

```
# Выберем столбец c и разделим его на столбец e,
# присвоим новые значение столбцу "new_c"
df['new_c'] = df['c']/df['e']
df.head()
```

Out[106]:

	a	b	c	d	e	new_a	new_c
0	32	36	121	20	15	16.0	8.066667
1	180	91	150	51	150	90.0	1.000000
2	154	128	41	95	108	77.0	0.379630
3	151	33	48	199	42	75.5	1.142857
4	30	120	68	114	143	15.0	0.475524

Ввод [111]:

```
# Выберем строки по значению в конкретном столбец
df[df['b']>150]
```

Out[111]:

	a	b	c	d	e	new_a	new_c
8	76	155	199	56	159	38.0	1.251572
10	64	173	158	165	159	32.0	0.993711
15	164	153	33	120	79	82.0	0.417722
17	152	195	63	149	10	76.0	6.300000
19	34	191	44	99	194	17.0	0.226804
22	173	187	183	89	189	86.5	0.968254
24	35	198	33	109	173	17.5	0.190751
28	159	182	14	18	3	79.5	4.666667
29	168	179	198	164	105	84.0	1.885714
35	125	184	100	72	120	62.5	0.833333
42	196	191	107	117	13	98.0	8.230769
58	118	172	82	199	64	59.0	1.281250
65	77	175	185	111	134	38.5	1.380597
66	178	187	69	151	77	89.0	0.896104
69	105	177	67	61	38	52.5	1.763158
74	86	157	90	162	75	43.0	1.200000
78	164	194	67	175	163	82.0	0.411043
93	26	177	39	54	77	13.0	0.506494
94	148	198	111	169	61	74.0	1.819672
95	122	171	150	14	38	61.0	3.947368

Ввод [116]:

```
# Выберем строки, где столбцы a и b больше 80
df[(df['b']>150)&(df['c']>150)]
```

Out[116]:

	a	b	c	d	e	new_a	new_c
8	76	155	199	56	159	38.0	1.251572
10	64	173	158	165	159	32.0	0.993711
22	173	187	183	89	189	86.5	0.968254
29	168	179	198	164	105	84.0	1.885714
65	77	175	185	111	134	38.5	1.380597

Ввод [121]:

```
# превратим значения столбца e, которые больше 95, в список
print(df['a'].tolist())
```

```
[32, 180, 154, 151, 30, 107, 5, 12, 76, 87, 64, 195, 82, 153, 45, 164, 17
2, 152, 10, 34, 134, 182, 173, 20, 35, 31, 146, 130, 159, 168, 21, 178, 17
5, 90, 38, 125, 179, 131, 81, 148, 169, 50, 196, 81, 25, 61, 79, 66, 101,
19, 59, 143, 10, 122, 9, 99, 59, 141, 118, 6, 36, 94, 24, 133, 157, 77, 17
8, 185, 95, 105, 60, 69, 189, 174, 86, 175, 110, 134, 164, 138, 62, 146, 1
89, 5, 75, 29, 35, 141, 182, 147, 145, 78, 77, 26, 148, 122, 41, 175, 31,
64]
```

Ввод [119]:

```
# отсортируем данные по какому-нибудь столбцу
df.sort_values('a', ascending=False)
```

Out[119]:

	a	b	c	d	e	new_a	new_c
42	196	191	107	117	13	98.0	8.230769
11	195	92	171	142	16	97.5	10.687500
72	189	118	156	166	122	94.5	1.278689
82	189	120	26	197	14	94.5	1.857143
67	185	147	15	77	159	92.5	0.094340
...	...	...	...	...	...	...	...
52	10	112	95	173	45	5.0	2.111111
54	9	87	9	192	116	4.5	0.077586
59	6	43	13	168	30	3.0	0.433333
6	5	34	35	20	147	2.5	0.238095
83	5	4	56	170	149	2.5	0.375839

100 rows × 7 columns

2.3. Базовые математические операторы

Pandas - мощный инструмент для вычислений. Пройдемся по базовым командам, позволяющим рассчитать различные статистические показатели.

Задача: создать датафрейм, состоящий из 5 столбцов и 5000 строк случайных чисел от 0 до 100. Подсказка: используйте функцию np.random.randint() для создания массива данных

Ввод [45]:

```
# Создание датафрейма
pass
```

Ввод [63]:

```
# Запишем в отдельные столбцы датафрейма минимальное, среднее и максимальное значени
pass
```

Ввод [ ]:

```
# Часть из математических методов датафреймов позволяют применять их не только к сто
pass
```

Ввод [ ]:

```

```

Ввод [ ]:

```
1
```