Простые типы данных

In [1]:

```
my_integer = 10
type(my_integer)
```

Out[1]:

```
int
```

In [2]:

```
my_float = 5.5
type(my_float)
```

Out[2]:

```
float
```

In [3]:

```
type(5)
```

Out[3]:

```
int
```

In [4]:

```
my_string = 'Hello World!'
my_string_2 = "Hello World"
type(my_string_2)
```

Out[4]:

```
str
```

In [5]:

```
my_bool = True
# my_bool = False
type(my_bool)
```

Out[5]:

```
bool
```

In [6]:

```
x = 5
y = 1
print(type(x > y))
```

```
<class 'bool'>
```

In [7]:

```
# преобразование типов
salary = 1000
print('Ваша годовая зарплата составляет ', salary, ' условных единиц')
```

```
Ваша годовая зарплата составляет  1000  условных единиц
```

In [8]:

```
type(str(salary))
```

Out[8]:

```
str
```

In [9]:

```
print('Ваша годовая зарплата составляет ' + str(salary) + ' условных единиц')
```

```
Ваша годовая зарплата составляет 1000 условных единиц
```

In [10]:

```
int('1000') + 100
```

Out[10]:

```
1100
```

In [11]:

```
# неявное преобразование типов
print(1 + False)
# print(20 / 5.1)
```

```
1
```

Строки

In [12]:

```
1 + 2
```

Out[12]:

```
3
```

In [13]:

```
'1' + '2'
```

Out[13]:

```
'12'
```

In [14]:

```
my_string = 'Hello World'
```

In [15]:

```
my_string.upper()
```

Out[15]:

```
'HELLO WORLD'
```

In [16]:

```
my_string.lower()
```

Out[16]:

```
'hello world'
```

In [17]:

```
my_string.capitalize()
```

Out[17]:

```
'Hello world'
```

In [18]:

```
my_string.replace('Hello', 'Goodbye')
```

Out[18]:

```
'Goodbye World'
```

In [19]:

```
len(my_string)
```

Out[19]:

```
11
```

Индексация и срезы

In [20]:

```
my_string = 'Hello World'
```

In [21]:

```
my_string[2]
```

Out[21]:

```
'l'
```

In [22]:

```
my_string[-1]
```

Out[22]:

```
'd'
```

In [23]:

```
my_string[0:5]
```

Out[23]:

```
'Hello'
```

In [24]:

```
my_string = 'Hello World'
my_string[0:8:2]
```

Out[24]:

```
'HloW'
```

In [25]:

```
my_string[6:]
```

Out[25]:

```
'World'
```

In [26]:

```
my_string[:5]
```

Out[26]:

```
'Hello'
```

Как выделить номер часа из даты следующего формата?

In [27]:

```
date = '2019-08-27T23:59:00.932'
```

In [28]:

```
date[11:13]
```

Out[28]:

```
'23'
```

Проверка на вхождение элемента в объект

In [29]:

```
my_string = 'Hello World'
target_string = 'World'

if target_string in my_string:
    print('find!')
```

```
find!
```

f-строки

In [30]:

```
name = 'oleg'
lang = 'python'

t = f"Hello, {name.capitalize()}, i know {lang} a bit"

print(t)
print(type(t))
```

```
Hello, Oleg, i know python a bit
<class 'str'>
```

In [31]:

```
print(f'Hello {name}, i know {lang} a bit')
```

```
Hello oleg, i know python a bit
```

Списки

In [32]:

```
month_list = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep']

income_list = [13000, 14000, 14300, 15000, 13800, 13000, 14900, 15200, 15300]

income_by_months = [['Jan', 13000], ['Feb', 14000], ['Mar', 14300], ['Apr', 15000], ['May',
```

In [33]:

```
print(type(month_list))
print(type(income_list))
print(type(income_by_months))
```

```
<class 'list'>
<class 'list'>
<class 'list'>
```

In [34]:

```
# индексация элементов в списке
print(month_list[0])
print(month_list[-1])
print(income_by_months[-4][1])
```

```
Jan
Sep
13000
```

In [35]:

```
# срезы
print(income_by_months[1:3])  # 0, 1
print('--------------')
print(income_by_months[-8:-6])
print('--------------')
print(income_by_months[2:])
print('--------------')
print(income_by_months[:3])
```

```
[['Feb', 14000], ['Mar', 14300]]
--------------
[['Feb', 14000], ['Mar', 14300]]
--------------
[['Mar', 14300], ['Apr', 15000], ['May', 13800], ['Jun', 13000], ['Jul', 149
00], ['Aug', 15200], ['Sep', 15300]]
--------------
[['Jan', 13000], ['Feb', 14000], ['Mar', 14300]]
```

In [36]:

```
# можно обращаться к любому уровню вложенности
income_by_months = [['Jan', 13000], ['Feb', 14000], ['Mar', 14300], ['Apr', 15000], ['May',
income_by_months[0][0]
```

Out[36]:

```
'Jan'
```

In [37]:

```
income_by_months[0][1]
```

Out[37]:

```
13000
```

In [38]:

```
# изменение списков
income_by_months[0][1] = 13100
print(income_by_months)
```

```
[['Jan', 13100], ['Feb', 14000], ['Mar', 14300], ['Apr', 15000], ['May', 138
00], ['Jun', 13000], ['Jul', 14900], ['Aug', 15200], ['Sep', 15300]]
```

In [39]:

```
income_by_months[0:2] = [['Jan', 13200], ['Feb', 13900]]
print(income_by_months)
```

```
[['Jan', 13200], ['Feb', 13900], ['Mar', 14300], ['Apr', 15000], ['May', 138
00], ['Jun', 13000], ['Jul', 14900], ['Aug', 15200], ['Sep', 15300]]
```

In [40]:

```
a, b = [1, 2]
```

In [41]:

```
a
```

Out[41]:

```
1
```

In [42]:

```
b
```

Out[42]:

```
2
```

In [43]:

```
income_by_months_2 = [['Nov', 15400], ['Dec', 17000]]
income_by_month = income_by_months + income_by_months_2
print(income_by_month)
```

```
[['Jan', 13200], ['Feb', 13900], ['Mar', 14300], ['Apr', 15000], ['May', 138
00], ['Jun', 13000], ['Jul', 14900], ['Aug', 15200], ['Sep', 15300], ['Nov',
15400], ['Dec', 17000]]
```

Распаковка списков

In [44]:

```
data = ['первый', 'второй', 'третий']
first = data[0]
```

In [45]:

```
first, second, third = ['первый', 'второй', 'третий']
first
```

Out[45]:

```
'первый'
```

In [46]:

```
# когда число элементов неизвестно
first, *_ = ['первый', 'второй', 'третий']
first
```

Out[46]:

```
'первый'
```

In [47]:

```
_
```

Out[47]:

```
['второй', 'третий']
```

In [48]:

```
first, *other, last =  ['первый', 'второй', 'третий', 'четвертый']
```

In [49]:

```
other
```

Out[49]:

```
['второй', 'третий']
```

In [50]:

```
first
```

Out[50]:

```
'первый'
```

In [51]:

```
first, last
```

Out[51]:

```
('первый', 'четвертый')
```

Операции со списками

In [52]:

```
income_by_months
```

Out[52]:

```
[['Jan', 13200],
 ['Feb', 13900],
 ['Mar', 14300],
 ['Apr', 15000],
 ['May', 13800],
 ['Jun', 13000],
 ['Jul', 14900],
 ['Aug', 15200],
 ['Sep', 15300]]
```

In [53]:

```
# Удаляем элемент по индексу
del(income_by_months[-1])
income_by_months
```

Out[53]:

```
[['Jan', 13200],
 ['Feb', 13900],
 ['Mar', 14300],
 ['Apr', 15000],
 ['May', 13800],
 ['Jun', 13000],
 ['Jul', 14900],
 ['Aug', 15200]]
```

In [54]:

```
month_list
```

Out[54]:

```
['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep']
```

In [55]:

```
# удаляем элемент по значению
month_list.remove('Sep')
print(month_list)
```

```
['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug']
```

In [56]:

```
# добавляем элемент в конец списка
income_by_months.append(['Dec', 17000])
income_by_months
```

Out[56]:

```
[['Jan', 13200],
 ['Feb', 13900],
 ['Mar', 14300],
 ['Apr', 15000],
 ['May', 13800],
 ['Jun', 13000],
 ['Jul', 14900],
 ['Aug', 15200],
 ['Dec', 17000]]
```

In [57]:

```
income_list
```

Out[57]:

```
[13000, 14000, 14300, 15000, 13800, 13000, 14900, 15200, 15300]
```

In [58]:

```
# добавляем элемент по нужному индексу
income_list.insert(2, 1111111)
print(income_list)
```

```
[13000, 14000, 1111111, 14300, 15000, 13800, 13000, 14900, 15200, 15300]
```

In [59]:

```
# считаем количество вхождений элемента в список
income_list.count(13000)
```

Out[59]:

```
2
```

In [60]:

```
income_list
```

Out[60]:

```
[13000, 14000, 1111111, 14300, 15000, 13800, 13000, 14900, 15200, 15300]
```

In [61]:

```
# узнаем индекс элемента в списка (только первое вхождение!)
income_list.index(13000, 1)
```

Out[61]:

```
6
```

In [62]:

```
# разворачиваем список
month_list.reverse()
month_list
```

Out[62]:

```
['Aug', 'Jul', 'Jun', 'May', 'Apr', 'Mar', 'Feb', 'Jan']
```

In [63]:

```
# узнаем длину списка
len(income_list)
```

Out[63]:

```
10
```

In [64]:

```
# сумма элементов
sum(income_list)
```

Out[64]:

```
1239611
```

In [65]:

```
income_list
```

Out[65]:

```
[13000, 14000, 1111111, 14300, 15000, 13800, 13000, 14900, 15200, 15300]
```

In [66]:

```
# максимальный элемент элементов
max(income_list)
```

Out[66]:

```
1111111
```

In [67]:

```
# минимальный элемент элементов
min(income_list)
```

Out[67]:

```
13000
```

In [68]:

```
# сортировка по возрастанию
sorted(income_list)
```

Out[68]:

```
[13000, 13000, 13800, 14000, 14300, 14900, 15000, 15200, 15300, 1111111]
```

In [69]:

```
income_list
```

Out[69]:

```
[13000, 14000, 1111111, 14300, 15000, 13800, 13000, 14900, 15200, 15300]
```

In [70]:

```
# изменить порядок сортировки
sorted(income_list, reverse= True)
```

Out[70]:

```
[1111111, 15300, 15200, 15000, 14900, 14300, 14000, 13800, 13000, 13000]
```

In [71]:

```
# а это сортировка строк по алфавиту
sorted(month_list)
```

Out[71]:

```
['Apr', 'Aug', 'Feb', 'Jan', 'Jul', 'Jun', 'Mar', 'May']
```

Изменение списков

В примере ниже переменная a и b на самом деле указывают на один и тот же объект. В результате, при добавлении в b очередного элемента этот элемент добавляется и в исходном листе a

In [72]:

```
a = [1, 2, 3]
b = a
```

In [73]:

```
b.append(4)
'a = {}'.format(a)
```

Out[73]:

```
'a = [1, 2, 3, 4]'
```

In [74]:

```
id(a), id(b)
```

Out[74]:

```
(2966418903808, 2966418903808)
```

Используем модуль copy

In [75]:

```
import copy
```

In [76]:

```
a = [1, 2, 3]
b = copy.copy(a)
```

In [77]:

```
id(a), id(b)
```

Out[77]:

```
(2966419302656, 2966418906432)
```

In [78]:

```
b.append(4)
print('a = {}'.format(a))

print('b = {}'.format(b))
```

```
a = [1, 2, 3]
b = [1, 2, 3, 4]
```

Списки и строки

In [79]:

```
queries_string = "смотреть сериалы онлайн,новости спорта,афиша кино,курс доллара,сериалы эт
```

In [80]:

```
# преобразование строки в список (например, из CSV-файла)
queries_list = queries_string.split(',')
queries_list
```

Out[80]:

```
['смотреть сериалы онлайн',
 'новости спорта',
 'афиша кино',
 'курс доллара',
 'сериалы этим летом',
 'курс по питону',
 'сериалы про спорт']
```

In [81]:

```
len(queries_list[0].split(' '))
```

Out[81]:

```
3
```

In [82]:

```
len(queries_list[0].split(' '))
```

Out[82]:

```
3
```

In [83]:

```
# Преобразование списка в строку

print(','.join(['Столбец 1', 'Столбец 2', 'Столбец 3']))
```

```
Столбец 1,Столбец 2,Столбец 3
```

In [84]:

```
# проверка вхождения элемента в список:

# 'Москва' in ['Ленинград', 'Одесса', 'Севастополь', 'Москва']
'Москва' not in ['Ленинград', 'Одесса', 'Севастополь', 'Москва']
```

Out[84]:

```
False
```

Tuple (кортежи)

In [85]:

```
salary_tuple = (1000, 1200, 1300, 900, 800)
# type(salary_tuple)
type(list(salary_tuple))
```

Out[85]:

```
list
```

In [86]:

```
list(salary_tuple)
```

Out[86]:

```
[1000, 1200, 1300, 900, 800]
```

In [87]:

```
# print(salary_tuple[0])
salary_tuple[0] = 500
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-87-f9322f03a403> in <module>
      1 # print(salary_tuple[0])
----> 2 salary_tuple[0] = 500

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

In [88]:

```
# кортеж из одного элемента задается так:
t = ('one', )
```

In [89]:

```
type(t)
```

Out[89]:

```
tuple
```

In [90]:

```
# без запятой получится строка
type( ('one') )
```

Out[90]:

```
str
```

In [91]:

```
# функция zip
salaries = [1000, 1200, 1300, 900, 800, 1000]
names = ['Robert', 'Jane', 'Liza', 'Richard', 'John']

salaries_by_names = zip(names, salaries)
print(salaries_by_names)
```

```
<zip object at 0x000002B2AC68DB40>
```

In [92]:

```
list(salaries_by_names)
```

Out[92]:

```
[('Robert', 1000),
 ('Jane', 1200),
 ('Liza', 1300),
 ('Richard', 900),
 ('John', 800)]
```

Циклы

Цикл while

In [93]:

```
x = 5
while x != 0:
    x -= 1
    print(x)
    # break

print('Я прекратил работу')
```

```
4
3
2
1
0
Я прекратил работу
```

In [94]:

```
x = 7
while x != 0:
    if x % 2 == 0:
        print(x, '- четное число')
    else:
        print(x, '- нечетное число')
    x = x - 1
```

```
7 - нечетное число
6 - четное число
5 - нечетное число
4 - четное число
3 - нечетное число
2 - четное число
1 - нечетное число
```

Цикл for

In [95]:

```
# итерация по строкам
company_name = 'Orange'
for letter in company_name:
    print(letter)
#     letter = letter.capitalize()
#     print(letter)
# print(letter)
```

```
O
r
a
n
g
e
```

In [96]:

```
# итерация по спискам
companies_capitalization = [
    ['Orange', 1.3],
    ['Maxisoft', 1.5],
    ['Headbook', 0.8],
    ['Nicola', 2.2]
]
```

In [97]:

```
for company in companies_capitalization:
#     print(company)
    print(company[0], 'capitalization is', company[1])
#     print('end of iteration')
```

```
Orange capitalization is 1.3
Maxisoft capitalization is 1.5
Headbook capitalization is 0.8
Nicola capitalization is 2.2
```

break, pass, continue

In [98]:

```
phrase = '640Кб должно хватить для любых задач. Билл Гейтс (по легенде)'
```

In [99]:

```
for letter in phrase:
    if letter == ' ':
        break
    print(letter, end='')
```

```
640Кб
```

In [100]:

```
for letter in phrase:
    if letter == ' ':
        continue
    print(letter)
print('finish loop')
```

```
б
4
0
К
б
д
о
л
ж
н
о
х
в
а
т
и
т
ь
д
л
я
л
ю
б
ы
х
з
а
д
а
ч
.
Б
и
л
л
Г
е
й
т
с
(
п
о
л
е
г
е
н
д
е
)
finish loop
```

In [101]:

```
for letter in phrase:
    if letter == ' ':
        pass
    print(letter)
print('finish loop')
```

```
б
4
0
К
б

д
о
л
ж
н
о

х
в
а
т
и
т
ь

д
л
я

л
ю
б
ы
х

з
а
д
а
ч
.

Б
и
л
л

Г
е
й
т
с

(
п
о
```

```
л
е
г
е
н
д
е
)
finish loop
```

Функции range и enumerate

In [102]:

```
range(10)
type(range(10))
```

Out[102]:

```
range
```

In [103]:

```
list(range(2, 10, 5))
```

Out[103]:

```
[2, 7]
```

In [104]:

```
range(10**12)
```

Out[104]:

```
range(0, 1000000000000)
```

In [105]:

```
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

In [106]:

```
# с указанием левой и правой границы
for i in range(3, 20):
    print(i)
```

```
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

In [107]:

```
# третий аргумент - шаг
for i in range(3, 20, 5):
    print(i)
```

```
3
8
13
18
```

In [108]:

```
# enumerate позволяет получить индекс каждого элемента
# enumerate([1, 2, 3, 4, 5])
list(enumerate([1, 2, 3, 4, 5]))
```

Out[108]:

```
[(0, 1), (1, 2), (2, 3), (3, 4), (4, 5)]
```

In [109]:

```
companies_capitalization = [
    ['Orange', 1.3],
    ['Maxisoft', 1.5],
    ['Headbook', 0.8],
    ['Nicola', 2.2]
]

for i, company in enumerate(companies_capitalization):
    print(i+1, company[0], 'capitalization is', company[1])
```

```
1 Orange capitalization is 1.3
2 Maxisoft capitalization is 1.5
3 Headbook capitalization is 0.8
4 Nicola capitalization is 2.2
```

In [110]:

```
for i, company in enumerate(companies_capitalization):
    print(company)

    if i > 2:
        break
```

```
['Orange', 1.3]
['Maxisoft', 1.5]
['Headbook', 0.8]
['Nicola', 2.2]
```

Попрактикуемся

Имеется структура данных cook_book, в которой хранится информация об ингредиентах блюд и их количестве в расчете на одну порцию и переменная, в которой хранится количество людей, на которых необходимо приготовить данные блюда:

In [111]:

```
cook_book = [
  ['салат',
      [
        ['картофель', 100, 'гр.'],
        ['морковь', 50, 'гр.'],
        ['огурцы', 50, 'гр.'],
        ['горошек', 30, 'гр.'],
        ['майонез', 70, 'мл.'],
      ]
  ],
  ['пицца',
      [
        ['сыр', 50, 'гр.'],
        ['томаты', 50, 'гр.'],
        ['тесто', 100, 'гр.'],
        ['бекон', 30, 'гр.'],
        ['колбаса', 30, 'гр.'],
        ['грибы', 20, 'гр.'],
      ],
  ],
  ['фруктовый десерт',
      [
        ['хурма', 60, 'гр.'],
        ['киви', 60, 'гр.'],
        ['творог', 60, 'гр.'],
        ['сахар', 10, 'гр.'],
        ['мед', 50, 'мл.'],
      ]
  ]
]
```

In [112]:

```
person = 5
```

Необходимо вывести пользователю список покупок необходимого количества ингредиентов для приготовления блюд на определенное число персон в следующем виде:

Салат:
картофель, 500гр.
морковь, 250гр.
огурцы, 250гр.
горошек, 150гр.
майонез, 350мл.

Пицца:
сыр, 250гр.
томаты, 250гр.
тесто, 500гр.
бекон, 150гр.
колбаса, 150гр.
грибы, 100гр.

Фруктовый десерт:
хурма, 300гр.
киви, 300гр.
творог, 300гр.
сахар, 50гр.
мед, 250мл.

In [113]:

```
for item in cook_book:
    print(item[0].capitalize())

    for line in item[1]:
        meal, weight, gr = line
        weight = weight * person

        print(meal, weight, gr)

    print('')
```

```
Салат
картофель 500 гр.
морковь 250 гр.
огурцы 250 гр.
горошек 150 гр.
майонез 350 мл.

Пицца
сыр 250 гр.
томаты 250 гр.
тесто 500 гр.
бекон 150 гр.
колбаса 150 гр.
грибы 100 гр.

Фруктовый десерт
хурма 300 гр.
киви 300 гр.
творог 300 гр.
сахар 50 гр.
мед 250 мл.
```

List comprehension

In [114]:

```
# Дана последовательность чисел. Мы хотим оставить только те, что делятся на 5
sequence = range(0, 40, 3)
list(sequence)
```

Out[114]:

```
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39]
```

In [115]:

```
# решение в лоб
for num in sequence:
    if num % 5 == 0:
        print(num)
```

```
0
15
30
```

In [116]:

```
# если хотим получить отфильтрованный лист, то будет даже так
filtered_sequence = []

for num in sequence:
    if num % 5 == 0:
        filtered_sequence.append(num)

print(filtered_sequence)
```

```
[0, 15, 30]
```

In [117]:

```
[num for num in sequence if num % 5 == 0]
```

Out[117]:

```
[0, 15, 30]
```

In [118]:

```
[num for num in sequence if num % 5 == 0]
```

Out[118]:

```
[0, 15, 30]
```

In [119]:

```
[w for w in sequence if w % 5 == 0]
```

Out[119]:

```
[0, 15, 30]
```

In [120]:

```
# с list comprehension получается покороче

[x**2 for x in sequence if x % 5 == 0]
```

Out[120]:

```
[0, 225, 900]
```

Пример вычисления метрики из набора списков. Столбцы в каждой строке:

- дата
- номер счетчика

- количество визитов

Найдем среднее количество визитов по нашим данным

In [121]:

```
api_response = [
    ['2017-12-26', '777', 184],
    ['2017-12-27', '111', 146],
    ['2017-12-28', '777', 98],
    ['2017-12-29', '777', 206],
    ['2017-12-30', '111', 254],
    ['2017-12-31', '777', 89],
    ['2018-01-01', '111', 54],
    ['2018-01-02', '777', 68],
    ['2018-01-03', '777', 74],
    ['2018-01-04', '111', 89],
    ['2018-01-05', '777', 104],
    ['2018-01-06', '777', 99],
    ['2018-01-07', '777', 145],
    ['2018-01-08', '111', 184],
]
```

In [122]:

```
sum([element[2] for element in api_response if element[1] == '777'])
```

Out[122]:

```
1067
```

In [123]:

```
for element in api_response:
    print(element[2])
```

```
184
146
98
206
254
89
54
68
74
89
104
99
145
184
```

In [124]:

```
sum([x[2] for x in api_response])/len(api_response)
```

Out[124]:

```
128.14285714285714
```

Множества (set)

Набор неповторяющихся элементов в случайном порядке

In [125]:

```
data_scientist_skills = set(['Python', 'R', 'SQL', 'Tableau', 'SAS', 'Git'])
data_engineer_skills = set(['Python', 'Java', 'Scala', 'Git', 'SQL', 'Hadoop'])
```

In [126]:

```
set([1, 2, 1])
```

Out[126]:

```
{1, 2}
```

In [127]:

```
# логическое ИЛИ - что нужно знать data-scientst, который по совместительству data-engineer
print(data_scientist_skills.union(data_engineer_skills))
print(data_scientist_skills | data_engineer_skills)
```

```
{'SAS', 'Git', 'R', 'Java', 'Python', 'Hadoop', 'Tableau', 'SQL', 'Scala'}
{'SAS', 'Git', 'R', 'Java', 'Python', 'Hadoop', 'Tableau', 'SQL', 'Scala'}
```

In [128]:

```
# логическое И - что нужно знать и data-scientist и data-engineer
print(data_scientist_skills.intersection(data_engineer_skills))
print(data_scientist_skills & data_engineer_skills)
```

```
{'SQL', 'Git', 'Python'}
{'SQL', 'Git', 'Python'}
```

In [129]:

```
# разность множеств - что знает data-scientist, но не знает data-engineer (и наоборот)
# print(data_scientist_skills.difference(data_engineer_skills))
# print(data_scientist_skills - data_engineer_skills)
print(data_engineer_skills.difference(data_scientist_skills))
print(data_engineer_skills - data_scientist_skills)
```

```
{'Java', 'Hadoop', 'Scala'}
{'Java', 'Hadoop', 'Scala'}
```

In [130]:

```
# симметричная разность множеств - что такого знают data-scientist и data-engineer, чего не
# print(data_scientist_skills.symmetric_difference(data_engineer_skills))
# print(data_scientist_skills ^ data_engineer_skills)
print(data_engineer_skills.symmetric_difference(data_scientist_skills))
print(data_engineer_skills ^ data_scientist_skills)
```

```
{'SAS', 'Scala', 'Java', 'Hadoop', 'Tableau', 'R'}
{'SAS', 'Scala', 'Java', 'Hadoop', 'Tableau', 'R'}
```

In [131]:

```
# Из списка можно убрать все повторения просто обратив его в set!
```

Словари

In [132]:

```
salaries = {
    'John': 1200,
    'Mary': 500,
    'Steven': 1000,
    'Liza': 1500
}
```

In [133]:

```
# обращение к элементу словаря
salaries['John']
```

Out[133]:

```
1200
```

In [134]:

```
# удаляем элемент из словаря
del(salaries['Liza'])
salaries
```

Out[134]:

```
{'John': 1200, 'Mary': 500, 'Steven': 1000}
```

In [135]:

```
# добавляем элемент в словарь
salaries['Mary'] = 2000
salaries
```

Out[135]:

```
{'John': 1200, 'Mary': 2000, 'Steven': 1000}
```

In [136]:

```
# безопасно получаем значение по ключу
# salaries['Oleg']
# print(salaries.get('Oleg', 'Not found'))
salaries.get('Mary', 'Not Found')
```

Out[136]:

```
2000
```

In [137]:

```
salaries['Andrey'] += 2000
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-137-35882ddf7fae> in <module>
----> 1 salaries['Andrey'] += 2000

KeyError: 'Andrey'
```

In [138]:

```
# проверка на наличие ключа в словаре
recruit = 'Amanda'

if recruit in salaries:
    print('Значение для ключа уже существует')
else:
    print('Добавляю новый ключ')
    salaries[recruit] = 2200

print(salaries)
```

```
Добавляю новый ключ
{'John': 1200, 'Mary': 2000, 'Steven': 1000, 'Amanda': 2200}
```

In [139]:

```
# Можно использовать метод setdefault
# setdefault не изменяет значение, если ключ уже был в словаре

# salaries.setdefault('Mary', 3000)
# salaries
salaries.setdefault('Paul', 3000)
salaries
```

Out[139]:

```
{'John': 1200, 'Mary': 2000, 'Steven': 1000, 'Amanda': 2200, 'Paul': 3000}
```

In [140]:

```
# перейдем к более сложному словарю
staff_dict = {
    'Robert': {'salary': 800, 'bonus': 200},
    'Jane': {'salary': 200, 'bonus': 300},
    'Liza': {'salary': 1300, 'bonus': 200},
    'Richard': {'salary': 500, 'bonus': 1200}
}
```

In [141]:

```
staff_dict['Robert']['bonus']
```

Out[141]:

```
200
```

In [142]:

```
staff_dict['Oleg'] = {'salary': 1000000, 'bonus': 300}
staff_dict
```

Out[142]:

```
{'Robert': {'salary': 800, 'bonus': 200},
 'Jane': {'salary': 200, 'bonus': 300},
 'Liza': {'salary': 1300, 'bonus': 200},
 'Richard': {'salary': 500, 'bonus': 1200},
 'Oleg': {'salary': 1000000, 'bonus': 300}}
```

In [143]:

```
# получаем только ключи/значения из словаря (очень пригодиться в циклах)
# print(staff_dict.keys())
# print(staff_dict.values())
# print(staff_dict.items())

print(list(staff_dict.keys()))
print(list(staff_dict.values()))
print(list(staff_dict.items()))
```

```
['Robert', 'Jane', 'Liza', 'Richard', 'Oleg']
[{'salary': 800, 'bonus': 200}, {'salary': 200, 'bonus': 300}, {'salary': 13
00, 'bonus': 200}, {'salary': 500, 'bonus': 1200}, {'salary': 1000000, 'bonu
s': 300}]
[('Robert', {'salary': 800, 'bonus': 200}), ('Jane', {'salary': 200, 'bonu
s': 300}), ('Liza', {'salary': 1300, 'bonus': 200}), ('Richard', {'salary':
500, 'bonus': 1200}), ('Oleg', {'salary': 1000000, 'bonus': 300})]
```

In [144]:

```
# итерация по словарям
# так бы было без цикла
print("Robert's salary:", staff_dict['Robert']['salary'])
print("Jane's salary:", staff_dict['Jane']['salary'])
print("Richard's salary:", staff_dict['Richard']['salary'])
```

```
Robert's salary: 800
Jane's salary: 200
Richard's salary: 500
```

In [145]:

```
for person in staff_dict:
    print(person)
```

```
Robert
Jane
Liza
Richard
Oleg
```

In [146]:

```
for key in staff_dict.keys():
    print(key)
```

```
Robert
Jane
Liza
Richard
Oleg
```

In [147]:

```
for value in staff_dict.values():
    print(value)
```

```
{'salary': 800, 'bonus': 200}
{'salary': 200, 'bonus': 300}
{'salary': 1300, 'bonus': 200}
{'salary': 500, 'bonus': 1200}
{'salary': 1000000, 'bonus': 300}
```

In [148]:

```
for key, value in staff_dict.items():
    print(key, value['salary'])
```

```
Robert 800
Jane 200
Liza 1300
Richard 500
Oleg 1000000
```

In [149]:

```
for i, person in enumerate(staff_dict):
    print(i+1, person)
```

```
1 Robert
2 Jane
3 Liza
4 Richard
5 Oleg
```

In [150]:

```
# используем цикл
for person, info in staff_dict.items():
#     print(person, info)
    print(person, "'s salary: ", info['salary'], sep='')
```

```
Robert's salary: 800
Jane's salary: 200
Liza's salary: 1300
Richard's salary: 500
Oleg's salary: 1000000
```

In [151]:

```
# добавим уровень з/п
for person, info in staff_dict.items():
#     print(person)
    if info['salary'] > 1000:
        info['status'] = 'above average'
    else:
        info['status'] = 'below average'
#     print(f"{person}'s salary: {info['salary']} ({status})")

staff_dict
```

Out[151]:

```
{'Robert': {'salary': 800, 'bonus': 200, 'status': 'below average'},
 'Jane': {'salary': 200, 'bonus': 300, 'status': 'below average'},
 'Liza': {'salary': 1300, 'bonus': 200, 'status': 'above average'},
 'Richard': {'salary': 500, 'bonus': 1200, 'status': 'below average'},
 'Oleg': {'salary': 1000000, 'bonus': 300, 'status': 'above average'}}
```

In [152]:

```
# функция zip
categories = ['Еда', 'Авто', 'Политика', '346346']
audience = [100, 200, 300]

dict(zip(categories, audience))

# categories_dict = dict(zip(categories, audience))
# print(categories_dict)
```

Out[152]:

```
{'Еда': 100, 'Авто': 200, 'Политика': 300}
```

Dict comprehension

Похоже на list comprehension

In [153]:

```
[x**2 for x in range(10)]
```

Out[153]:

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

In [154]:

```
{n: n**2 for n in range(10)}
```

Out[154]:

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

In [155]:

```
results = [('date', '2018-01-01'), ('counter', '777'), ('visits', 154)]
```

In [156]:

```
{metric: value for (metric, value) in results}
```

Out[156]:

```
{'date': '2018-01-01', 'counter': '777', 'visits': 154}
```

In [ ]:

```

```