Cheat Sheet

Типы данных

Как мы их называем	Как они называются по-английски	Как они называются в Python
"Естественные" типы данных		
целое число	integer	`int`
вещественное число	floating-point number	`float`
логическая переменная	boolean / logical	`bool`
Упорядоченные типы данных (последовательности)		
строка	string	`str`
список	list	`lst`
кортеж	tuple	`tuple`
Неупорядоченные типы данных (коллекции)		
множество	set	`set`
словарь	dictionary	`dict`

Неизменяемые типы данных	Изменяемые типы данных
кортеж	список
строки	множество
числа целые и вещественные	словарь
логические переменные	

Тип данных	Тип структуры данных	Как обращаемся к элементу внутри?
кортежи	упорядоченный	по индексу
списки	упорядоченный	по индексу
строки	упорядоченный	по индексу
множество	неупорядоченный	не можем обратиться к элементу
Словари	неупорядоченный	по ключу

Основные функции

`input()`
считывает ввод (строку).

`print(x, sep=',', end='\n')`
вывести на экран. `x` - аргумент для печати. `sep` задает разделитель между аргументами

(по умолчанию пробел), `end` - какой сивол ставим в конце выведенной строки (по умолчанию переход на новую строку). Аргументы задаются через запятую.

`f'Hello, {name}!'`
синтаксис форматированной строки (`f-string`), вместо name будет подставлено значение переменной name. Например, если `name = 'Olya'` , строка из примера будет выглядеть как `'Hello, Olya!'`

`round(x, n)` округлить число x до n знаков после точки

In [89]:
```
n1 = 2.53242
n2 = 3.521515

print(n1, n2, sep=';', end='!\n')
print(f'n1 * n2 = {round(n1 + n2, 2)}')
```

```
2.53242;3.521515!
n1 * n2 = 6.05
```

`map(название функции, последовательность)`
применяет функцию ко всем элементам последовательности

In [19]:
```
# с помощью map приводим строки к нижнему регистру
print(list(map(str.lower, ['кОт', 'пЕс', 'УткОнОс'])))
```

```
['кот', 'пес', 'утконос']
```

In [18]:
```
# с помощью map переводим считанный список строк в список целых чисел
numbers = input().split()
print(list(map(int, numbers)))
```

```
[1, 2, 3]
```

Функции для превращения в определенные типы данных

`float()`
вещественное число

`int()`
целое число

`str()`
строка

`list()`
список

`set()`
множество

`dict()`
словарь

Арифметические операции

In [3]:
```
print(11 + 2)   # Складывать
print(11 - 2)   # Вычитать
print(11 * 2)   # Умножать
```

```
print(11 ** 2)  # Возводить в степень
print(11 / 2)   # Делить. Результатом, как и на калькуляторе, будет дробное число
print(11 // 2)  # Деление нацело даст целое число в качестве результата
print(11 % 2)   # Остаток от деления нацело
```

```
13
9
22
121
5.5
5
1
```

Логические операции

In [9]:
```
print(2 + 2 == 4) # равно
print(2 + 2 != 4) # не равно
print(2 + 2 < 4)  # меньше
print(2 + 2 > 4)  # больше
print(2 + 2 >= 4) # больше или равно
print(2 + 2 <= 4) # меньше или равно
print('o' in 'Hello') #  проверка на наличие в последовательности
print('o' not in 'Hello') # проверка на отсутствие в последовательности
```

```
True
False
False
False
True
True
True
False
```

In [11]:
```
age = 19
print((age >= 18) and (age < 65)) # возраст 18 или старше И младше 65
print((age < 18) or (age >= 65)) # возраст меньше 18 ИЛИ 65 и старше
not(age == 19) # логическое отрицание
```

```
True
False
```

Out[11]:
```
False
```

`and`
возвращает `True`, если оба условия истинны, и `False` во всех других случаях

`or`
возвращает `True`, когда одно или оба условия истинны, `False`, когда оба ложны.

Условный оператор: пример

In [12]:
```
sex = 'male'
if sex == 'male':
    print('Мужской пол')
elif sex == 'female':
    print('Женский пол')
else:
    print('Пол не указан')
```

```
Мужской пол
```

Цикл while: пример

In [2]:
```
# напечатайте числа от 1 до x включительно
x = int(input())
```

```
n = 1
while n <= x:
    print(n)
    n = n + 1
```

```
1
2
3
4
5
```

In [21]:
```
# сохраняйте в список животных, пока пользователь не введет "конец"
animals = []
while True:
    animal = input()
    if animal == 'конец':
        break
    animals.append(animal)

print(animals)
```

```
['кот', 'капибара', 'утконос']
```

Цикл for: примеры

С помощью цикла for можно перебирать элементы строк, списков, кортежей, множеств и ключи словарей.

In [10]:
```
# перебор элементов списка
students = ['Маша', 'Миша', 'Глаша']
for st in students:
    print(st)
```

```
Маша
Миша
Глаша
```

In [11]:
```
# перебор индексов с помощью диапазона чисел range()
students = ['Маша', 'Миша', 'Глаша']
for i in range(len(students)):
    print(students[i])
```

```
Маша
Миша
Глаша
```

In [14]:
```
# вложенный цикл - печатаем таблицу умножения для 1, 2, 3
for i in range(1,4):
    for j in range(1,4):
        print(f'{i} x {j} = {i * j}')
    print('-'*9)
```

```
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
---------
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
---------
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
---------
```

In [33]:
```
# вложенный цикл - перебераем список списков
students = [['Masha', 5], ['Petya', 4]]
```

```
for st in students:
    for item in st:
        print(item)
    print('-'*9)
```

```
Masha
5
---------
Petya
4
---------
```

Операции с последовательностями (строки, кортежи, списки)

Операции в ячейке ниже работают не только со строками, но и со списками и кортежами.

In [83]:
```
print('Hello' + ' Oleg') # склейка
print('Hello' * 2)       # повторение
print('Hello'[1])        # обращение к элементу по индексу, вернет второй символ
print('Hello'[-1])       # обращение к элементу по индексу, вернет последний символ
print(len('Hello'))      # возвращает количество элементов в строке
```

```
Hello Oleg
HelloHello
e
o
5
```

Если обратиться по индексу элемента, которого нет в последовательности -- будет ошибка.

Срезы

Срезы позволяют обратиться сразу к нескольким символам внутри строки или элементам последовательности:

1. `'Hello'[2:4]` позволяет обратиться к символам слова Hello с индексами 2 и 3: `ll`;
2. `'Hello'[:3]` позволяет обратиться ко всем символам от символа с индексом 0 до символа с индексом 2: `Hel`;
3. `'Hello'[2:]` позволяет обратиться ко всем символам от символа с индексом 2 до конца строки: `llo`;
4. `'Hello'[:]` позволяет обратиться ко всем символам строки: `Hello`;
5. `'Hello'[-4:-1]` позволяет обратиться к символам с 4 до 1 (при нумерации с конца строки): `ell`
6. `'Hello'[::2]` достает каждую букву слова с шагом в два символа `Hlo`

In [28]:
```
print(f'Срез строки "Hello"[2:4]: {"Hello"[2:4]}')
print(f'Срез строки "Hello"[:3]: {"Hello"[:3]}')
print(f'Срез строки "Hello"[2:]: {"Hello"[2:]}')
print(f'Срез строки "Hello"[:]: {"Hello"[:]}')
print(f'Срез строки "Hello"[-4:-1]: {"Hello"[-4:-1]}')
print(f'Срез строки "Hello"[::2]: {"Hello"[::2]}')
```

```
Срез строки "Hello"[2:4]: ll
Срез строки "Hello"[:3]: Hel
Срез строки "Hello"[2:]: llo
Срез строки "Hello"[:]: Hello
Срез строки "Hello"[-4:-1]: ell
Срез строки "Hello"[::2]: Hlo
```

Списки

In [9]:
```
students = ['Маша', 'Миша', 'Глаша']
students[0] = 'Даша' # замена элемента списка
print(students)
print(*students) # 'распаковка' списка -- печать элементов по отдельности
```

```
['Даша', 'Миша', 'Глаша']
Даша Миша Глаша
```

- `.append()` добавляет в существующий список новый элемент;
- `.remove()` удаляет из существующего списка указанный элемент;
- `.count()` подсчитывает, сколько раз указанный элемент входит в список;
- `.index()` находит индекс первого элемента, совпадающего с указанным.

Методы `.append()` и `.remove()` изменяют исходный список.

Методы `.count()` и `.index()` можно применять и к кортежам.

In [20]:
```
shopping_list = ['молоко', 'хлеб', 'сметана', 'молоко']
print(f'Исходный список: {shopping_list}')  # Выведем исходный список

shopping_list.append('кефир')               # Добавим в исходный список кефир
print(f'Список с кефиром: {shopping_list}')  # Выведем список с кефиром

shopping_list.remove('сметана')                 # Удалим из списка сметану
print(f'Список без сметаны: {shopping_list}')   # Выведем список без сметаны

print(f'Молоко встречается {shopping_list.count("молоко")} раза.')
print(f'Первый раз молоко встречается под индексом {shopping_list.index("молоко")}.'
print(f'Первый раз кефир встречается под индексом {shopping_list.index("кефир")}.')

print()
shopping_tuple = ('молоко', 'хлеб', 'сметана', 'молоко', 'кефир')
print(f'Молоко встречается {shopping_tuple.count("молоко")} раза.')
print(f'Первый раз молоко встречается под индексом {shopping_tuple.index("молоко")}.
print(f'Первый раз кефир встречается под индексом {shopping_tuple.index("кефир")}.')
```

```
Исходный список: ['молоко', 'хлеб', 'сметана', 'молоко']
Список с кефиром: ['молоко', 'хлеб', 'сметана', 'молоко', 'кефир']
Список без сметаны: ['молоко', 'хлеб', 'молоко', 'кефир']
Молоко встречается 2 раза.
Первый раз молоко встречается под индексом 0.
Первый раз кефир встречается под индексом 3.

Молоко встречается 2 раза.
Первый раз молоко встречается под индексом 0.
Первый раз кефир встречается под индексом 4.
```

Методы строк

- `.split()` разбивает строку по пробельным символам (в качестве аргумента можно задать свой разделитель)
- `.join()` создаёт из списка строк новую строку. Строка, к которой был применён метод, станет разделителем между элементами.

In [24]:
```
names = 'Olya, Anya, Petya'.split(', ') # разбиваем строку с именами по ', '
print(names)
print(';'.join(names)) # объединяем список строк в строку через ';'
```

```
['Olya', 'Anya', 'Petya']
Olya;Anya;Petya
```

- `.isdigit()` позволяет проверить, что строка состоит только из цифр;
- `.islower()` позволяет проверить, что строка состоит только из букв нижнего регистра (строчных);
- `.isalpha()` позволяет проверить, что строка состоит только из букв;
- `.isalnum()` позволяет проверить, что строка состоит только из букв и цифр;
- `.replace()` позволяет создать новую строчку, заменив что-то в изначальной;
- `.count()` позволяет понять, сколько раз последовательность символов встречалась в строке;

In [15]:
```
s = ' 4242HelloHello42!   '
print(f'Проверим их работу на примере строки "{s}""')
print(f'Состоит ли строка только из цифр: {s.isdigit()}')
print(f'Состоит ли строка только из строчных букв: {s.islower()}')
print(f'Состоит ли строка только из букв: {s.isalpha()}')
print(f'Состоит ли строка только из букв или цифр: {s.isalnum()}')
print(f'Удалим боковые пробелы: "{s.strip()}"')
print(f'Заменим число 42 на слово "Ответ": "{s.replace("42","Ответ")}"')
print(f'Найдём первый и последний индекс для цифр "42" : {s.find("42")}, {s.rfind("4
print(f'Сколько раз последовательность цифр "42" встречаются в строке: {s.count("42"
```

```
Проверим их работу на примере строки " 4242HelloHello42!   ""
Состоит ли строка только из цифр: False
Состоит ли строка только из строчных букв: False
Состоит ли строка только из букв: False
Состоит ли строка только из букв или цифр: False
Удалим боковые пробелы: "4242HelloHello42!"
Заменим число 42 на слово "Ответ": " ОтветОтветHelloHelloОтвет!   "
Найдём первый и последний индекс для цифр "42" : 1, 15
Сколько раз последовательность цифр "42" встречаются в строке: 3
```

- `.find()` и `.rfind()` позволяют найти индекс самой первой и самой последней найденной последовательности символов в строке.
- `.lower()` создаёт новую строчку, где все буквы будут маленькими;
- `.upper()` создаёт новую строчку, где все буквы будут большими;
- `.strip()` создаёт новую строчку, где по бокам исходной строки удалены лишние символы. Без указания аргумента удаляет пробелы, но можно указать строку из символов, которые нужно считать лишними: `.strip('.')` будет считать лишним символом только точки;
- `.lstrip()` и `.rstrip()` работают так же, как и `.strip()`, но удаляют символы только слева или только справа.

In [21]:
```
s = ' .Test. '
print(f'Исходная строка: "{s}"')
print(f'Строка, в которой нет боковых пробелов: "{s.strip()}"')
print(f'Строка, в которой нет боковых пробелов справа: "{s.rstrip()}"')
print(f'Строка, в которой нет боковых пробелов слева: "{s.lstrip()}"')
print(f'Строка, в которой нет боковых пробелов и точек: "{s.strip(" .")}"')
print(f'Строка, в которой все буквы большие: "{s.upper()}"')
print(f'Строка, в которой все буквы маленькие: "{s.lower()}"')
print(f'Все буквы маленькие и нет лишних пробелов: "{s.lower().strip()}"')
print(f'Исходная строка не изменилась: "{s}"')
```

```
Исходная строка: " .Test. "
Строка, в которой нет боковых пробелов: ".Test."
Строка, в которой нет боковых пробелов справа: " .Test."
Строка, в которой нет боковых пробелов слева: ".Test. "
Строка, в которой нет боковых пробелов и точек: "Test"
Строка, в которой все буквы большие: " .TEST. "
Строка, в которой все буквы маленькие: " .test. "
```

```
Все буквы маленькие и нет лишних пробелов: ".test."
Исходная строка не изменилась: " .Test. "
```

- `.startswith()` проверяет, начинается ли строка с нужной нам последовательностью символов (или одной из нужных нам последовательностей символов);
- `.endswith()`, аналогично, проверяет, кончается ли строка нужной нам последовательностью символов (или одной из нужных нам последовательностей символов);

In [25]:
```python
print(f'Оканчивается ли "hse.ru" на ".ru"? {"hse.ru".endswith("ru")}')
print(f'Оканчивается ли "hse.ru" на ".ru" или ".рф"? {"hse.ru".endswith(("ru", ".рф"
print(f'Начинается  ли строка "https://cbr.ru" с "https"? {"https://cbr.ru".startswi
```

```
Оканчивается ли "hse.ru" на ".ru"? True
Оканчивается ли "hse.ru" на ".ru" или ".рф"? True
Начинается  ли строка "https://cbr.ru" с "https"? True
```

Множества

In [29]:
```python
pets = {'кошка', 'ужик', 'питон'}
pets.add('собака') # добавим элемент в множество
pets.remove('кошка') # удаляем элемент из множества
print(pets)
```

```
{'ужик', 'собака', 'питон'}
```

Операции над множествами

In [29]:
```python
my_pets_list = {'кошка', 'рыбка', 'шиншилла', 'ужик'}
friend_pets_list = {'собака', 'питон', 'ужик', 'кошка', 'хамелеон'}

# объединение -- элементы, которые входят хотя бы в одно множество
print(my_pets_list | friend_pets_list)

# пересечение -- элементы, которые входят в оба множества
print(my_pets_list & friend_pets_list)

# разность: входят в my_pets_list, но не входят в  friend_pets_list
print(my_pets_list - friend_pets_list)

# разность: входят в friend_pets_list, но не входят в my_pets_list
print(friend_pets_list - my_pets_list)

# симметрическая разность: входят в одно из множества, но не в оба сразу
print(my_pets_list ^ friend_pets_list)
```

```
{'кошка', 'питон', 'собака', 'хамелеон', 'рыбка', 'шиншилла', 'ужик'}
{'кошка', 'ужик'}
{'шиншилла', 'рыбка'}
{'питон', 'собака', 'хамелеон'}
{'питон', 'шиншилла', 'рыбка', 'собака', 'хамелеон'}
```

Словари

In [30]:
```python
eng_rus = {'apple': 'яблоко', 'hello': 'привет', 'world': 'мир'}

print(eng_rus['apple']) # обращаемся к значению "яблоко" по его ключу - apple
eng_rus['hello'] = 'здравствуйте' # перезаписываем значение ключа
print(eng_rus) # печатаем измененный словарь
del eng_rus['world'] # удаляем пару ключ:значение
print(eng_rus) # печатаем измененный словарь
eng_rus['python'] = 'питон' # создаем новую пару ключ-значение
print(eng_rus) # печатаем измененный словарь
```

```
яблоко
{'apple': 'яблоко', 'hello': 'здравствуйте', 'world': 'мир'}
{'apple': 'яблоко', 'hello': 'здравствуйте'}
{'apple': 'яблоко', 'hello': 'здравствуйте', 'python': 'питон'}
```

In [92]:
```python
eng_rus = {'apple': 'яблоко', 'hello': 'привет', 'world': 'мир'}

print('apple' in eng_rus) # проверяем наличие ключа в словаре
print('apple' not in eng_rus) # проверяем отсутствие ключа в словаре
print(eng_rus.values()) # все значения словаря
print(eng_rus.values()) # все ключи словаря
print(eng_rus.items()) # пары ключ-значение в форме списка кортежей
```

```
True
False
dict_values(['яблоко', 'привет', 'мир'])
dict_values(['яблоко', 'привет', 'мир'])
dict_items([('apple', 'яблоко'), ('hello', 'привет'), ('world', 'мир')])
```

In [5]:
```python
# перебираем все ключи словаря
eng_rus = {'apple': 'яблоко', 'hello': 'привет', 'world': 'мир'}

for key in eng_rus:
    print(key)
```

```
apple
hello
world
```

In [6]:
```python
# перебираем все значения словаря
eng_rus = {'apple': 'яблоко', 'hello': 'привет', 'world': 'мир'}

for key in eng_rus:
    print(eng_rus[key])

# перебираем все значения словаря: второй способ
for value in eng_rus.values():
    print(value)
```

```
яблоко
привет
мир
яблоко
привет
мир
```

In [8]:
```python
# печатаем ключ и соответствующее ему значение
eng_rus = {'apple': 'яблоко', 'hello': 'привет', 'world': 'мир'}

for key in eng_rus:
    print(key, eng_rus[key])
```

```
apple яблоко
hello привет
world мир
```

In [ ]:
```python
eng_rus = {'apple': 'яблоко', 'hello': 'привет', 'world': 'мир'}
eng_rus.items()
```

Словарь списков

In [36]:
```python
temperature = {
    '1 января': [-5, -12],
    '2 января': [-4, -8]
}
# Выведем весь словарь температур -- для двух дней:
```

```
print(f'Температура по дням (дневная, ночная): {temperature}')

# Выведем список температур -- для 1 января:
print(f'Температура 1 января (дневная, ночная): {temperature["1 января"]}')

# Выведем дневную температуру 1 января:
print(f'Температура 1 января (дневная): {temperature["1 января"][0]}')

# Выведем среднюю температуру за 1 января:
print(f'Температура 1 января (средняя): {sum(temperature["1 января"]) / len(temperat
```

```
Температура по дням (дневная, ночная): {'1 января': [-5, -12], '2 января': [-4, -8]}
Температура 1 января (дневная, ночная): [-5, -12]
Температура 1 января (дневная): -5
Температура 1 января (средняя): -8.5
```

Подсчет статистики с помощью словаря

In [91]:
```
# посчитаем. сколько раз каждая буква встречается в тексте
twister = 'two toads terribly tired trotted along the road said toad number one to t

letters = ''.join(twister.split()) # избавляемся от пробелов

stats = {} # создаем пустой словарь для подсчета
for l in letters:
    if l not in stats:
        stats[l] = 1
    else:
        stats[l] += 1

print(stats)
```

```
{'t': 13, 'w': 2, 'o': 10, 'a': 6, 'd': 7, 's': 2, 'e': 8, 'r': 7, 'i': 3, 'b': 3,
'l': 2, 'y': 1, 'n': 4, 'g': 1, 'h': 2, 'u': 2, 'm': 2}
```

Сумма, минимум, максимум, сортировка

`sum()`

суммирует элементы списка

`max()`

находит максимум списка, кортежа, множества

`min()`

находит минимум списка, кортежа, множества

In [16]:
```
numbers = [5,7,30]
print(f'Минимум списка {numbers}: {min(numbers)}')
print(f'Максимум списка {numbers}: {max(numbers)}')
print(f'Сумма списка {numbers}: {sum(numbers)}')

numbers = (5,7,30)
print(f'Минимум кортежа {numbers}: {min(numbers)}')
print(f'Максимум кортежа {numbers}: {max(numbers)}')
print(f'Сумма кортежа {numbers}: {sum(numbers)}')

s = '112358'
print(f'Минимальная цифра в строке {s}: {min(s)}, максимальная: {max(s)}')
```

```
Минимум списка [5, 7, 30]: 5
Максимум списка [5, 7, 30]: 30
Сумма списка [5, 7, 30]: 42
Минимум кортежа (5, 7, 30): 5
```

```
Максимум кортежа (5, 7, 30): 30
Сумма кортежа (5, 7, 30): 42
Минимальная цифра в строке 112358: 1, максимальная: 8
```

In [37]:
```
marks = [8.4, 4, 8, 9.4]
print(sorted(marks))  # отсортировали список оценок по возрастанию
print(sorted(marks, reverse=True))  # а теперь по убыванию
```

```
[4, 8, 8.4, 9.4]
[9.4, 8.4, 8, 4]
```

In [39]:
```
fruits = ['apple', 'Яблоко', 'Апельсин', 'Orange', 'манго']
print(sorted(fruits)) # сортировка строк без ключа (регистр не игнорируется)
print(sorted(fruits, key=str.lower)) # сортировка строк с ключом (регистр игнорирует
```

```
['Orange', 'apple', 'Апельсин', 'Яблоко', 'манго']
['apple', 'Orange', 'Апельсин', 'манго', 'Яблоко']
```

In [45]:
```
# вывод словаря с сортировкой по ключам
marks_students = {'Antonova Anna':10, 'Ivanov Alexey':8, 'Bobylev Sergey':5, 'Yanova

for key in sorted(marks_students):
  print(f'Студент {key} получил оценку {marks_students[key]}')
```

```
Студент Antonova Anna получил оценку 10
Студент Bobylev Sergey получил оценку 5
Студент Ivanov Alexey получил оценку 8
Студент Yanova Lena получил оценку 5
```

In [48]:
```
# вывод словаря с сортировкой по значениям
marks_students = {'Antonova Anna':10, 'Ivanov Alexey':8, 'Bobylev Sergey':5, 'Yanova

for value in sorted(set(marks_students.values())):
    for key in marks_students:
        if marks_students[key] == value:
            print(f'Оценку {value} получит студент {key}')
```

```
Оценку 5 получит студент Bobylev Sergey
Оценку 5 получит студент Yanova Lena
Оценку 8 получит студент Ivanov Alexey
Оценку 10 получит студент Antonova Anna
```

In [51]:
```
# минимум / максимум среди ключей словаря
marks_students = {'Antonova Anna':10, 'Ivanov Alexey':8, 'Bobylev Sergey':5, 'Yanova

max_key = max(marks_students)
print(f'Студент, чья фамилия идет позже остальных: {max_key}')
```

```
Студент, чья фамилия идет позже остальных: Yanova Lena
```

In [56]:
```
# минимум / максимум среди значений словаря

min_value = min(marks_students.values())

for key in marks_students:
    if marks_students[key] == min_value:
        print(f'Минимальную оценку {min_value} получит студент {key}')
```

```
Минимальную оценку 5 получит студент Bobylev Sergey
Минимальную оценку 5 получит студент Yanova Lena
```

Функции

In [57]:
```
# Определим функцию, которая считает возраст по году рождения
def get_age(year):
    print(2020 - year)
```

```
birth_year = 2005
get_age(birth_year)
```

```
15
```

In [58]:
```
# Определим функцию, считающую,
# сколько было лет в какой-то год
def check_age(birth_year, year):
    print(year - birth_year)

check_age(1996, 2020)
```

```
24
```

In [60]:
```
# определим функцию с двумя возвращаемыми значениями по условию
# функция проверяет, родился ли человек в тот год, для которого проверяем возраст
def check_age(birth_year, year):
    if year < birth_year:
        return 'Ошибка'
    else:
        return year - birth_year

print(check_age(1996, 1980))
print(check_age(1996, 2000))
```

```
Ошибка
4
```

In [62]:
```
# вызов функции внутри функции

# переводим год рождения в возраст
def get_age(age):
    return 2020 - age

# находим минимальный и максимальный возраст в списке
def get_min_max_age(list_of_ages):
    return get_age(min(list_of_ages)), get_age(max(list_of_ages))

get_min_max_age([1987, 1990, 2000, 2004, 1999])
```

Out[62]:
```
(33, 16)
```

In [64]:
```
# возвращаем несколько значений

# возвращаем длину слова и последнюю букву для игры в слова
def check_name(word):
    return len(word), word[-1]

petya_score, last_letter = check_name('Москва')
print(petya_score)
print(last_letter)
```

```
6
a
```

Анонимные функции

In [2]:
```
def squared(a):
    return a ** 2

print(list(map(squared, [2, 3, 4])))
# анонимная функция аналогична squared
print(list(map(lambda a: a ** 2, [2, 3, 4])))
```

```
[4, 9, 16]
[4, 9, 16]
```

Сортировка словаря по значению с анонимной функцией.

In [6]:
```
d = {'cat':124, 'barsik':76, 'meow':50}
# сортировка по ключу -- первому элементу кортежа
print(sorted(d.items()))
# сортировка по значению -- второму элементу кортежа
print(sorted(d.items(), key=lambda x:x[1]))
```

```
[('barsik', 76), ('cat', 124), ('meow', 50)]
[('meow', 50), ('barsik', 76), ('cat', 124)]
```

Передача строк, кортежей, чисел

Рассмотрим, как передаются в функции строки, кортежи, целые и дробные числа.

In [1]:
```
# определяем функцию, которая будет обновлять строку
# входные параметры: строка
# возвращаемые значения: строка, к которой приписано' + добавка'
def update_string(s):
    s += ' + добавка'
    return s
```

In [3]:
```
str1 = 'исходный текст'
str2 = update_string(str1)
print('str2:', str2)
print('str1:', str1) # исходная строка не изменилась
```

```
str2: исходный текст + добавка
str1: исходный текст
```

In [5]:
```
# определяем функцию, которая будет обновлять кортеж
# входные параметры: кортеж
# возвращаемые значения: другой кортеж
def update_tuple(t):
    t = ('другой', 'кортеж')
    return t
```

In [6]:
```
t1 = ('исходный', 'кортеж')
t2 = update_tuple(t1)
print('t2:', t2)
print('t1:', t1) # исходный кортеж не изменился
```

```
t2: ('другой', 'кортеж')
t1: ('исходный', 'кортеж')
```

In [7]:
```
# определяем функцию, которая будет обновлять число
# входные параметры: число (целое или дробное)
# возвращаемые значения: то же число, умноженное на два
def update_number(n):
    n *= 2 # другой вариант записи: n = n * 2
    return n
```

In [8]:
```
n1 = 5
n2 = update_number(n1) # вызовем функцию для целого числа
print('n2:', n2)
print('n1:', n1)
```

```
n2: 10
n1: 5
```

In [9]:
```
n3 = 7.2
n4 = update_number(n3) # вызовем функцию для дробного числа
print('n4:', n4)
print('n3:', n3)
```

```
n4: 14.4
n3: 7.2
```

Передача списков, словарей, множеств

Рассмотрим, как передаются в функции списки, словари и множества.

In [10]:
```python
# определяем функцию, которая будет обновлять список
# входные параметры: список
# возвращаемые значения: список, к которому добавлен элемент '+ добавка'
def update_list(L):
    L.append('+ добавка')
    return L
```

In [12]:
```python
list1 = ['исходный', 'список']
list2 = update_list(list1)
print('list2:', list2)
print('list1:', list1) # исходный список изменился
```

```
list2: ['исходный', 'список', '+ добавка']
list1: ['исходный', 'список', '+ добавка']
```

In [13]:
```python
# определяем функцию, которая будет обновлять множество
# входные параметры: множество
# возвращаемые значения: множество, к которому добавлен элемент '+ добавка'
def update_set(s):
    s.add('+ добавка')
    return s
```

In [14]:
```python
s1 = {'исходное', 'множество'}
s2 = update_set(s1)
print('s2:', s2)
print('s1:', s1) # исходное множество изменилось
```

```
s2: {'множество', '+ добавка', 'исходное'}
s1: {'множество', '+ добавка', 'исходное'}
```

In [ ]:
```python
# определяем функцию, которая будет обновлять словарь
# входные параметры: словарь
# возвращаемые значения: словарь, к которому добавлен доп.ключ и доп.значение
def update_dict(d):
    d['доп.ключ'] = 'доп.значение'
    return d
```

In [ ]:
```python
d1 = {'ключ 1': 'значение 1', 'ключ 2': 'значение 2' }
d2 = update_dict(d1)
print('d2:', d2)
print('d1:', d1) # исходный словарь изменился
```

```
d2: {'ключ 1': 'значение 1', 'ключ 2': 'значение 2', 'доп.ключ': 'доп.значение'}
d1: {'ключ 1': 'значение 1', 'ключ 2': 'значение 2', 'доп.ключ': 'доп.значение'}
```

Модули

In [68]:
```python
import math # импорт модуля math с дополнительными математическими фукнциями
print(math.sqrt(25)) # вызов функции из модуля
```

```
5.0
```

In [67]:
```python
import string # модуль string содержит много переменных с подмножествами символов
print(string.digits) # все цифры
print(string.punctuation) # все знаки препинания
```

```
0123456789
```

```
!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
```

Работа с файлом

Чтобы файл открылся без указания полного пути (только по названию) -- он должен лежать в той же папке, что и файл с кодом Python. На платформе online.hse.ru, чтобы считать файл, обязательно нужно указывать только название файла без полного пути (см. примеры).

Также не забывайте указывать кодировку `encoding='UTF-8'`, чтобы не было проблем с кириллицей.

Мы открываем файл с помощью конструкции `with open() as`, которая потом и закрывает файл.

Мы используем три режима открытия файлов:

1. `mode='r'` — *read,* ***чтение***. При отсутствии указанного файла выдается ошибка. Это режим по умолчанию.
2. `mode='w'` — *write,* ***запись***. При отсутствии указанного файла создается новый. При наличии файла содержимое перезаписывается.
3. `mode='a'` — *append,* ***дозапись***. При отсутствии указанного файла создается новый. При наличии файла новое содержимое дозаписывается после имеющегося.

In [70]:
```
# считываем текст из файла в строку, способ 1

text = ''                # создаем пустую строку, в которой будем хранить текст из ф
with open('text_file.txt', encoding='UTF-8') as infile:
    for line in infile:  # сохраняем поочередно каждую строку файла infile в перемен
        text += line     # добавляем новую строку к переменной text

print(text)
```

```
Привет! Это первая строка в файле.
А это вторая строка.

Третья строка была пустой, а это уже четвертая строка.
```

In [71]:
```
# считываем текст из файла в строку, способ 2

with open('text_file.txt', encoding='UTF-8') as infile:
    text = infile.read()

print(text)
```

```
Привет! Это первая строка в файле.
А это вторая строка.

Третья строка была пустой, а это уже четвертая строка.
```

In [72]:
```
# считываем строки из файла в список, способ 1

lines = [] # создаем пустой список
with open('text_file.txt', encoding='UTF-8') as infile:
    for line in infile: # проходимся по файлу построчно
        lines.append(line.strip()) # добавляем строку + удаляем знак перехода на нов
print(lines)
```

```
['Привет! Это первая строка в файле.', 'А это вторая строка.', '', 'Третья строка бы
```

```
ла пустой, а это уже четвертая строка.']
```

In [74]:
```python
# считываем строки из файла в список, способ 2

lines = [] # создаем пустой список
with open('text_file.txt', encoding='UTF-8') as infile:
    lines = infile.readlines()

# обратите внимание, в этом способе сохранился знак перехода на новую строку
# и надо будет его почистить позже, если это важно для задачи
print(lines)
```

```
['Привет! Это первая строка в файле.\n', 'А это вторая строка.\n', '\n', 'Третья стр
ока была пустой, а это уже четвертая строка.\n']
```

In [78]:
```python
# считать файл .csv и разбить по разделителям

lines = []

with open('csv_file.csv', encoding='UTF-8') as infile: # открываем файл под псевдони
    for line in infile:  # сохраняем поочередно каждую строку файла infile в перемен
        lines.append(line.strip().split(', ')) # добавляем в список строку + избавля

print(lines)
```

```
[['Фамилия', 'Имя', 'Отчество', 'год рождения'], ['Иванов', 'Иван', 'Иванович', '198
0'], ['Петрова', 'Ирина', 'Михайловна', '2000']]
```

In [80]:
```python
# запись в файл -- создается новый файл (или затирается старый с таким же именем) и

important_text = 'Это очень важный текст. В нем даже несколько строк. \nОни отделены
with open('info.txt', 'w', encoding='UTF-8') as outfile:    # открываем файл, указыв
    print(important_text, file=outfile)
```

In [82]:
```python
# дозапись в файл -- открывается существующий файл и в его конец дозаписывается текс

important_text_3 = 'Это третий важный текст. Надеемся, что он запишется после второг
with open('info.txt', 'a', encoding='UTF-8') as outfile: # открываем тот же файл, ук
    print(important_text_3, file=outfile)
```