# Функции

In [1]:

```
# создаем функцию
def square(number):
    result = number ** 2
    return result

# print(square(10))
my_square = square(10)
print(my_square)
```

```
100
```

In [2]:

```
# help(print)
?print
```

In [3]:

```
# пишем docstring к своей функции
def square(number):
    """
    this is my function
    """
    result = number ** 2
    return result
```

In [4]:

```
?square
```

In [20]:

```
# функция без параметров
def square_2():
    user_input = int(input('Введите число'))
    result = user_input ** 2
    return result

square_2()
```

```
Введите число9
```

Out[20]:

```
81
```

In [11]:

```
# функция с двумя параметрамим
def power(number, number_2):
    result = number ** number_2
    return result

power(4, 10)
```

Out[11]:

```
1048576
```

In [22]:

```
# функция с параметром по умолчанию
def power(number, number_2=2):
    result = number ** number_2
    return result

power(10)
```

Out[22]:

```
100
```

In [19]:

```
# если не указан return, функция всегда возвращает None!
def square_3(number):
    result = number ** 2
    print(result)
#     return None
#     return
print(square_3(5))
```

```
25
```

## Области видимости

In [26]:

```
number = 5
power = 3

def power_2():
    number = 6
    power = 2
    return number ** power

print(power_2())
print(number ** power)
```

```
36
125
```

In [27]:

```
number = 5
power = 3

def power_2():
    number = 6
    return number ** power

print(power_2())
print(number ** power)
```

```
216
125
```

In [28]:

```
number = 5
power = 3

def power_2():
    return number ** power

power_2()
```

Out[28]:

```
125
```

In [29]:

```
def power_2():
    number = 6
    power = 2
    some_number = 1
    return number ** power

print(some_number)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-29-461955134ac4> in <module>
      5     return number ** power
      6
----> 7 print(some_number)

NameError: name 'some_number' is not defined
```

In [30]:

```
# global

name = 'James'
def say_hi():
    global name
    name = 'Oleg'
    print('Hello', name)

say_hi()

print(name)
```

```
Hello Oleg
Oleg
```

## lambda-функции

In [ ]:

```
def func(x, y):
    return x + y
```

In [ ]:

```
func = lambda x, y: x + y
```

In [31]:

```
func = lambda x, y: x + y
func(1, 6)
```

Out[31]:

```
7
```

In [32]:

```
nums = [1, 2, 3, 4, 5]
```

In [33]:

```
# even_numbers = lambda x: x % 2 == 0
odd_even_numbers = lambda x: 'even' if x % 2 == 0 else 'odd'
```

In [34]:

```
for num in nums:
#     print(even_numbers(num))
    print(odd_even_numbers(num))
```

```
odd
even
odd
even
odd
```

In [35]:

```
# функция map
list(map(odd_even_numbers, nums))
```

Out[35]:

```
['odd', 'even', 'odd', 'even', 'odd']
```

In [37]:

```
[odd_even_numbers(num) for num in nums]
```

Out[37]:

```
['odd', 'even', 'odd', 'even', 'odd']
```

In [38]:

```
def square(x):
    return x**2
```

In [39]:

```
for i in map(square, range(100, 105)):
    print(i)
```

```
10000
10201
10404
10609
10816
```

In [40]:

```
list(map(square, range(100,105)))
```

Out[40]:

```
[10000, 10201, 10404, 10609, 10816]
```

## Args and kwargs

In [ ]:

```
api_request(1, 2, 3)
```

In [ ]:

```
api_request(report=2, date=1, output=3)
```

In [41]:

```
def api_request(*params):
    date_start = params[0]
    date_end = params[1]
    print(params)
    print(date_start, date_end)
```

In [42]:

```
api_request('2019-01-01', '2019-01-31', ';sdfsdgsdg', 'sdgsdgsdg', 'sdgsdg')
```

```
('2019-01-01', '2019-01-31', ';sdfsdgsdg', 'sdgsdgsdg', 'sdgsdg')
2019-01-01 2019-01-31
```

In [43]:

```
def api_requets(**params):
    return params
```

In [45]:

```
api_requets(a=1, b=2, c=3, date='2020-01-01')
```

Out[45]:

```
{'a': 1, 'b': 2, 'c': 3, 'date': '2020-01-01'}
```

In [46]:

```
def api_request(**params):
    date_start = params['date_start']
    date_end = params['date_end']
    print(params)
    print(date_start, date_end)
```

In [48]:

```
api_request(date_start='2019-01-31', report='traffic', date_end='2019-01-01')
```

```
{'date_start': '2019-01-31', 'report': 'traffic', 'date_end': '2019-01-01'}
2019-01-31 2019-01-01
```

## Попрактикуемся

Хотим классифицировать жителей в файле по возрастам:

- до 18 лет - children
- 19-65 - young
- старше 65 - retiree

In [52]:

```
with open('adult.csv') as f:
    for line in f:
        print(line)
        break
```

```
age,workclass,fnlwgt,education,educational-num,marital-status,occupation,rel
ationship,race,gender,capital-gain,capital-loss,hours-per-week,native-countr
y,income
```

In [53]:

```
# подготовим файл для работы, разобьем строки по запятой и удалим пробельные символы
i = 0
with open('adult.csv', 'r') as f:
    for line in f:
        print(line.strip().split(','))

        i = i + 1
        if i > 5:
            break
```

```
['age', 'workclass', 'fnlwgt', 'education', 'educational-num', 'marital-stat
us', 'occupation', 'relationship', 'race', 'gender', 'capital-gain', 'capita
l-loss', 'hours-per-week', 'native-country', 'income']
['25', 'Private', '226802', '11th', '7', 'Never-married', 'Machine-op-inspc
t', 'Own-child', 'Black', 'Male', '0', '0', '40', 'United-States', '<=50K']
['38', 'Private', '89814', 'HS-grad', '9', 'Married-civ-spouse', 'Farming-fi
shing', 'Husband', 'White', 'Male', '0', '0', '50', 'United-States', '<=50
K']
['28', 'Local-gov', '336951', 'Assoc-acdm', '12', 'Married-civ-spouse', 'Pro
tective-serv', 'Husband', 'White', 'Male', '0', '0', '40', 'United-States',
'>50K']
['44', 'Private', '160323', 'Some-college', '10', 'Married-civ-spouse', 'Mac
hine-op-inspct', 'Husband', 'Black', 'Male', '7688', '0', '40', 'United-Stat
es', '>50K']
['18', '?', '103497', 'Some-college', '10', 'Never-married', '?', 'Own-chil
d', 'White', 'Female', '0', '0', '30', 'United-States', '<=50K']
```

In [54]:

```
# выделим первый "столбец" в отдельную переменную age
i = 0
with open('adult.csv', 'r') as f:
    for line in f:
        age, *other_columns = line.strip().split(',')

        if i > 0:
            print(age)
            print(other_columns[1])
        i += 1
        if i > 5:
            break
```

```
25
226802
38
89814
28
336951
44
160323
18
103497
```

Добавим классификацию возрастов

In [55]:

```
i = 0
with open('adult.csv', 'r') as f:
    for line in f:
        age, *other_columns = line.strip().split(',')

        if i > 0:
            if int(age) <= 18:
                age_group = 'children'

            elif int(age) <= 60:
                age_group = 'young'

            else:
                age_group = 'retiree'

            print(age, age_group)

        i += 1
```

```
25 young
38 young
28 young
44 young
18 children
34 young
29 young
63 retiree
24 young
55 young
65 retiree
36 young
26 young
58 young
48 young
43 young
20 young
43 young
37 young
```

Что тут нехорошо:

- сложно потестировать все случаи (в нашем цикле нужных случаев может и не оказаться)
- наш цикл стал довольно громоздким, а мы только начали
- эта классификация может потребоваться еще в 100500 местах кода
- данные могут быть кривыми и скрипт будет падать с ошибкой

Условия вычисления возрастной группы:

1. В строке 15 столбцов
2. Столбец с возрастом первый по счету
3. Возраст должен быть целым числом в адекватных пределах
4. Могут добавиться еще требования, о которых мы пока не знаем

In [56]:

```
def age_is_correct(age, lower_age=0, upper_age=120):
    """
    Проверка корректности возраста age по следующим правилам:
    1. Целое число
    2. В адекватных пределах

    Возвращает True или False. Пример
    age_is_correct(15)
    True

    age_is_correct(121)
    False

    age_is_correct(-5)
    False
    """

    if str.isnumeric(age):
        if lower_age <= int(age) <= upper_age:
            return True

    return False

age_is_correct('5')
```

Out[56]:

```
True
```

In [58]:

```
age_is_correct('125aaa')
```

Out[58]:

```
False
```

In [59]:

```
def number_of_columns(line, separator=','):
    """Возвращает количество столбцов в строке line с разделителем separator"""

    return len(line.split(separator))
```

In [60]:

```
def line_is_correct(line):
    """
    Проверка строки на корректность. Проверяются условия:
    1. В строке 15 столбцов (пригодится с упражнения)
    2. Столбец с возрастом первый по счету
    3. Возраст должен быть целым числом в адекватных пределах
    """
    age = line.strip().split(',')[0]

    if number_of_columns(line) == 15:
        if age_is_correct(age):
            return True

    return False
```

In [61]:

```
# Добавляем функции в наш цикл
i = 0
with open('adult.csv', 'r') as f:
    for line in f:

        if line_is_correct(line):
            age, *other_columns = line.strip().split(',')
            if int(age) <= 18:
                age_group = 'children'

            elif int(age) <= 60:
                age_group = 'young'

            else:
                age_group = 'retiree'

            print(age, age_group)

        i += 1
        if i > 10:
            break
```

```
25 young
38 young
28 young
44 young
18 children
34 young
29 young
63 retiree
24 young
55 young
```

In [62]:

```
# вынесем классификацию возраста в отдельную функцию
def age_classification(age):
    """
    Возвращает возрастную категорию для возраста age (можно передать как строку).
    Классификация категорий:
        - до 18 лет - children
        - 19-60 - young
        - старше 65 - retiree

    Пример
    age_classification('18')
    'children'

    age_classification(65)
    'retiree'
    """

    if int(age) <= 18:
        return 'children'
    elif int(age) <= 65:
        return 'young'
    elif int(age) > 65:
        return 'retiree'
    return 'boom'
```

In [63]:

```
age_classification(140)
```

Out[63]:

```
'retiree'
```

In [64]:

```
age_classification(18)
```

Out[64]:

```
'children'
```

In [67]:

```
?age_classification
```

In [65]:

```
# реализуем главную функцию
def main():
    with open('adult.csv', 'r') as f:
        for line in f:
            if line_is_correct(line):
                age, *colums = line.strip().split(',')
                print(age, age_classification(age))
```

In [66]:

```
main()
```

```
25 young
38 young
28 young
44 young
18 children
34 young
29 young
63 young
24 young
55 young
65 young
36 young
26 young
58 young
48 young
43 young
20 young
43 young
37 young
```