# Функции

# Функции

Функции представляют блок кода, который выполняет определенную задачу и который можно повторно использовать в других частях программы.

```
def имя_функции (параметры):
    инструкции
```

# Пример

Определим словарь

```
def say_hello():
    print("Hello")
```

# Вызов функции

Пока вы не вызовете функцию, работать она не будет. Вызов функции выглядит следующим образом:

```
имя_функции (параметры)
```

# Пример

```
def say_hello():   # определение функции say_hello
   print("Hello")
say_hello()       # вызов функции say_hello
say_hello()
say_hello()
```

# Пример 2

```
def say_hello():
    print("Hello")
def say_goodbye():
    print("Good Bye")


say_hello()
say_goodbye()
```

# Функция main

В программе может быть определено множество функций. И чтобы всех их упорядочить, одним из способов их организации является добавление специальной функции (обычно называется main), в которой потом уже вызываются другие функции

# Параметры функции

Функция может принимать параметры. Через параметры в функцию можно передавать данные. Пример - функция print(), которая с помощью параметра принимает значение, выводимое на консоль.

# Удаление

Теперь определим и используем свою функцию с параметрами:

```
def say_hello(name):
    print(f"Hello, {name}")

say_hello("Tom")
say_hello("Bob")
say_hello("Alice")
```

# Метод get

При вызове функции значения передаются параметрам по позиции. Например, определим и вызовем функцию с несколькими параметрами:

```
def print_person(name, age):
    print(f"Name: {name}")
    print(f"Age: {age}")
print_person("Tom", 37)
```

# Значения по умолчанию

Некоторые параметры функции мы можем сделать необязательными, указав для них значения по умолчанию при определении функции. Например:

```
def say_hello(name="Tom"):
    print(f"Hello, {name}")
say_hello()  # здесь name = "Tom"
say_hello("Bob")    # здесь name = "Bob"
```

# Неопределенное количество параметров

С помощью символа звездочки можно определить параметр, через который можно передавать неопределенное количество значений. Это может быть полезно, когда мы хотим, чтобы функция получала несколько значений, но мы точно не знаем, сколько именно.

# Копирование словаря

```
def sum(*numbers):
    result = 0
    for n in numbers:
        result += n
    print(f"sum = {result}")
sum(1, 2, 3, 4, 5)      # sum = 15
sum(3, 4, 5, 6)         # sum = 18
```

# Оператор return

Функция может возвращать результат. Для этого в функции используется оператор return, после которого указывается возвращаемое значение:

```
def имя_функции (параметры):
    инструкции
    return возвращаемое_значение
```

# Пример

```
def double(number):
    return 2 * number

result1 = double(4)     # result1 = 8
result2 = double(5)     # result2 = 10
print(f"result1 = {result1}")   # result1 = 8
print(f"result2 = {result2}")   # result2 = 10
```

# Задание

- Напишите функцию, которая возвращает True, если переданное в параметрах число четное и False, если число нечетное
- Написать и протестировать функцию, выводящую на экран сумму, среднее, максимум, минимум и количество всех чисел, переданных через параметры.

# Лямбда-выражения

Лямбда-выражения в языке Python представляют небольшие анонимные функции, которые определяются с помощью оператора lambda. Формальное определение лямбда-выражения:

```
lambda параметры : инструкция
```

# Лямбда-выражения

Определим простейшее лямбда-выражение:

```
message = lambda: print("hello")

message()   # hello
```

# Пример 2

Определим простейшее лямбда-выражение:

```
square = lambda n: n * n

print(square(4))    # 16
print(square(5))    # 25
```

# Задание

- Напишите лямбда выражение, приветствующее пользователя по указанному в параметрах имени.