Функции

Вы уже знаете, что…

Это функции

```
bool()
float()
int()
list()
range()
str()
tuple()
abs()
chr()
dir()
enumerate()
help()
input()
len()
max()
min()
ord()
reversed()
print()
round()
sorted()
sum()
```

Функции можно вызывать. В функции можно передавать аргументы.

```
print('Do you wanna build a snowman?’)
```

Функции могут возвращать значение

```
x = input('Or ride our bike around the halls?')
```

Функции

Функции:

- помогают использовать написанный код в программах многократно
- делают его более понятным за счёт деления на блоки.

Синтаксис создания функции:

```
def <имя функции> (<аргументы функции>):
    <тело функции>
```

- Имя функции – придумывается по тем же правилам, что и имя переменной (буквы, цифры, знаки подчёркивания)
- Аргументы функции – её параметры, которые в теле функции становятся переменными
- Тело функции – код, который работает после вызова функции, а затем возвращает значение с помощью `return`

Функции

Напишем функцию, которая проверяет, что список целых чисел, передаваемый ей как аргумент, содержит только чётные числа:

```
def only_even(numbers):
    result = True
    for x in numbers:
        if x % 2 != 0:
            result = False
            break
    return result


print(only_even([2, 4, 6]))     # True
print(only_even([1, 2, 3]))     # False
```

```
def only_even(numbers):
    for x in numbers:
        if x % 2 != 0:
            return False
    return True


print(only_even([2, 4, 6]))     # True
print(only_even([1, 2, 3]))     # False
```

return

Оператор return используется в теле функции, чтобы вернуть значение функции.

Можно использовать несколько операторов return. Первый сработавший оператор return остановит выполнение функции и вернёт указанное значение.

Функция всегда возвращает значение. Если в ней нет оператора return, либо он есть, но без возвращаемого значения, то функция возвращает None.

```
print(print("Одну копейку буду должна"))    # None
```

None - специальный тип данных в Python, который используется для обозначения «ничего».

None

Пример: функция может вернуть либо что-то, либо ничего.

```
def find_kotik_or_zaika(s):
    for word in s.split():
        if word in ('котик', 'зайка'):
            return word
    return None


print(find_kotik_or_zaika('зайказайка котик зайка'))   # котик
print(find_kotik_or_zaika('зайказайка котикзайка'))    # None

result = find_kotik_or_zaika('мишка зайка слоник')
if result:
    print('Найден', result)
else:
    print('Котик или зайка не найден')
# Найден зайка
```

Области видимости

- **Глобальная область видимости** – то, что снаружи.
- **Локальная область видимости** – то, что внутри функции.
- **Глобальная переменная** – переменная, которая находится в глобальной области видимости.
- **Локальная переменная** – переменная, которая находится в локальной области видимости.

Аргумент функции является локальной переменной этой функции.

```
def find_kotik_or_zaika(s):
    for word in s.split():
        if word in ('котик', 'зайка'):
            return word
    return None

print(s)    # NameError: name 's' is not defined
```

Локальные переменные недоступны в глобальной области видимости.

Области видимости

Глобальные переменные доступны в локальной области видимости.

```
kotik_zaika = ('котик', 'зайка')            # глобальная переменная

def find_kotik_or_zaika(s):
    for word in s.split():
        if word in kotik_zaika:             # обращаемся к гл. переменной из локальной о.в.
            return word
    return None

print(find_kotik_or_zaika('зайказайка котик зайка'))   # котик
print(find_kotik_or_zaika('зайказайка котикзайка'))    # None
```

Области видимости

Но есть одно «НО». Если попытаться записать в глобальную переменную новое значение, то вместо этого будет создана новая локальная переменная.

```
def make_x_access():
    x = 'Access'                # 2.1 создаётся локальная переменная
    print(x)                    # 2.2 выводится 'Access'

x = 'Python'                # 1. глобальная переменная

make_x_access()                 # 2. вызываем функцию
print(x)                    # 3. выводится 'Python'

# Access
# Python
```

Области видимости

Второе «НО». Если попытаться изменить глобальную переменную, то она меняется, а локальная переменная не создаётся.

Но это работает только если тип изменяемый – например – list.

```
def list_modify_1(list_arg):
    list_arg = [1, 2, 3, 4]     # создаётся новый локальный список

def list_modify_2(list_arg):
    list_arg += [4]             # меняется исходный внешний список

sample_1 = [1, 2, 3]
sample_2 = [1, 2, 3]
list_modify_1(sample_1)
list_modify_2(sample_2)
print(sample_1, sample_2)     # [1, 2, 3] [1, 2, 3, 4]
```

Пример 1

Функция принимает в качестве аргумента строку s и возвращает строку “Виват, <s>!”.

```
def vivat(s):
    return f'Виват, {s}!'

print(vivat('Лицей'))   # Виват, Лицей!
```

Пример 2

Функция принимает в качестве аргумента целое число, прибавляет к нему 1 и возвращает результат.

Увеличим число на 3, используя эту функцию.

```
def increase(x):
    return x + 1

x = 0
x = increase(x)
x = increase(x)
x = increase(x)
print(x)        # 3
```

Пример 3

Функция принимает в качестве аргумента список seq и значение value. Затем, если значения нет в списке, добавляет его в список.

```
def add_value(seq, value):
    if value not in seq:
        seq.append(value)

seq = [1, 2, 3]
add_value(seq, 1)
add_value(seq, 11)
add_value(seq, 3)
add_value(seq, 33)
print(seq)          # [1, 2, 3, 11, 33]
```

Пример 4

Функция принимает в качестве аргумента список чисел seq и возвращает новый список, состоящий из чётных чисел списка seq.

```
def filter_even(seq):
    result = []
    for elem in seq:
        if elem % 2 == 0:
            result.append(elem)
    return result

seq = [1, 2, 3, 4, 5, 6]

print(filter_even(seq))     # [2, 4, 6]
print(seq)          # [1, 2, 3, 4, 5, 6]
```

Пример 5

Функция принимает строку и выводит её, если она не была выведена ранее.

```
history = []

def print_unique(s):
    if s not in history:
        history.append(s)
        print(s)


print_unique('Не отпускай')          # Не отпускай
print_unique('Меня')                 # Меня
print_unique('Не отпускай')
print_unique('Не отпускай')
print_unique('Меня')
print_unique('Вдруг кто увидит')     # Вдруг кто увидит
```