



ФУНКЦИИ

В программировании функции принимают аргументы и возвращают значение. Функции в Python определяются с помощью инструкции `def`:

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

Функции позволяют упаковывать часть кода для его последующего повторного вызова. В примере выше определена функция с именем `sum`, которая принимает два параметра `x` и `y`, и возвращает результат их суммы. Обратившись к этой функции по имени и задав параметры, мы можем получить результат:

```
>>> sum(34, 12)  
46  
>>> sum('abc', 'def')  
'abcdef'
```

Инструкция `return` позволяет вернуть значение, которое нам необходимо. Это требуется для того, чтобы получить определенный результат и затем дальше использовать его в программе.

Функция может быть любой сложности, внутри конструкции `def -> return`, мы можем написать любой код. Смысл в функциях заключается в том, чтобы не писать один и тот же код повторно, а просто, в нужный момент, вызывать заранее написанную функцию. Так же функция может быть без параметров или может не возвращать какое-то конкретное значение или не заканчиваться инструкцией `return` вообще:

```
def fun():
    var = 'Python'
    if len(var) >= 6:
        print(var)
    return # В этом случае функция вернет значение None
```

Код под инструкцией `def` будет относиться к функции до тех пор, пока он вложен в эту инструкцию, то есть отступает от `def`.

ФУНКЦИИ БЫВАЮТ РАЗНЫХ ТИПОВ:

- **Глобальные функции** - такие функции доступны из любой части кода файла, в котором они написаны. Глобальные функции доступны из других модулей, но об этом мы расскажем в разделе "Подключение модулей".

```
# Объявляем функцию
def solve(s):
    c = []
    for i in range(len(s)-1):
        if i == 0 or i%2 == 0:
            c.append(s[i])
    return c
# вызываем функцию solve с заданными параметрами и выводим результат ее работы
print(solve([1, 2, 3, 4, 5, 6, 7, 8]))
```

- **Локальные функции** - функции, объявленные внутри других функций. Вызвать их можно только внутри функции, в которой они объявлены. Их удобно использовать, если необходима небольшая вспомогательная функция, которая больше нигде не используется.
- **Лямбда-функции** - особые, анонимные функции, имеющие ряд ограничений, по сравнению с обычными функциями. Они локально решают единственную задачу. Применение такой функции выглядит, как выражение, давайте посмотрим на примере:

```
# Обычная функция  
def search_len(arg_1):  
    return len(arg_1)  
  
# Лямбда-функция  
result = lambda x: len(x)
```

Обычно, лямбда-функции применяют при вызове функций, которые в качестве аргументов содержат функции. Проблема использования лямбда-функций состоит в том, что иногда усложняется читаемость кода.

```
>>> func = lambda x, y: x + y
>>> func(1, 2)
3
>>> func('a', 'b')
'ab'
>>> (lambda x, y: x + y)(1, 2)
3
>>> (lambda x, y: x + y)('a', 'b')
'ab'
```

Лямбда-функции не имеют имени, поэтому могут возникать проблемы с отловом ошибки.

Очень важно документировать описание к любой функции, чтобы каждый раз не разбирать написанное заново. Для этого используют строки, заключенные в тройные кавычки. Поскольку описание функции зачастую состоит более, чем из 1 строки, использование строк с тройными кавычками очень удобно. Обычно под документирование выделяют место между определением функции и началом основного кода:

```
def solve(s):  
    """ функция solve(s) принимает список  
        создает пустой список  
        находит элементы с четным индексом (включая 0)  
        заносит их в созданный список и возвращает его  
    """  
  
    c = []  
    for i in range(len(s)-1):  
        if i == 0 or i%2 == 0:  
            c.append(s[i])  
    return c
```

Переменные, которые объявляются внутри функций являются **локальными**. Изменение этих переменных и обращение к ним происходят только внутри функций, где они были объявлены. Если же переменные объявлены вне функций, они являются **глобальными**. С глобальными переменными надо обходиться осторожно. Их удобно использовать, потому что к ним можно обращаться из любой части кода и даже из других модулей, но, если в коде происходит неконтролируемое изменение глобальной переменной, то поиск ошибки может перерасти в головную боль. Рассмотрим пример:

```
var_1 = [1,2,3]

def func(a):
    var_1 = []
    for i in a:
        var_1.append(i**2)
    return var_1

print(func(var_1))
print(var_1)
```


Зеленым цветом обозначена **глобальная** переменная, **красным** - **локальная**. Глобальная переменная `var_1` в данном случае остается неизменной, т.к. она используется только в качестве параметра для функции и нигде не происходит манипуляций над ней. Внутри этой функции изменения происходят с локальной переменной `var_1`. Результат выполнения такой программы будет следующий:

```
[2, 4, 6]
[1, 2, 3]
```


ЗАДАНИЕ 1

Напишите программу, которая принимает арифметическое выражение в качестве аргумента и выводит результат этого выражения.

*Необходимо использовать функции. Программа должна поддерживать следующие арифметические операции: +, -, /, *, %(получение процента от числа), **(возведение в квадрат), **x (возведение в степень числа x). Запрещено подключать дополнительные модули. Для вывода результата необходимо использовать функцию print().*



- Подставьте "Входные данные" в свою программу и сравните результат с выходными данными.

- 1) Входные данные: $1+9$.

Выходные значения: 10.

- 2) Входные данные: 100%.

Выходные значения: 1.0.

- 3) Входные данные: 4^{**} .

Выходные значения: 16.

- 4) Входные данные: 9^{**3} .

Выходные значения: 729.

