



python

# Лекция 10

## Функции. Часть 1



## Функции

**Функция** - фрагмент программного кода (подпрограмма), к которому можно обратиться из другого места программы.

**Функции** - это самые основные программные структуры в языке Python, обеспечивающие многократное использование программного кода и уменьшающие его избыточность.



## Как создать функцию в Python

Функцию нужно сначала объявить (инициализировать) и только потом, использовать.

Объявление и последующая инициализация функции в Python выполняется с помощью оператора **def**. Синтаксис объявления функции таков:

```
def имя_функции (список_параметров):
```

```
    Тело функции (последовательность операторов)
```



## Особенности работы функций

Функция может возвращать результат своей работы. В таком случае в теле функции должен присутствовать оператор **return**. То значение, которое следует после **return**, и будет результатом работы функции.

Если функция не должна возвращать результат вычислений, тогда оператор **return** не обязательный.

**Внимание!** Функцию нельзя использовать до тех пор, пока вы ее не объявите.



## Пример объявления функции

**Имя функции**



```
def calculate_summa(a, b):  
    c = a + b  
    return c
```



**Параметры. В этом примере их два.**

**Оператор для объявления.**

В этом примере функция возвращает сумму параметров. Поэтому в конце идет оператор **return**, после которого идет переменная, в которой хранится сумма параметров.



## Как использовать функции

Для использования функции нужно написать ее имя, после чего в круглых скобках указываются параметры, которые вы передадите в функцию. Это называется **«вызов функции»**.

```
def calculate_summa(a, b):  
    c = a+b  
    return c
```

```
d = calculate_summa(10, 10)
```

```
print(d)
```

**Параметры, которые  
передаются в функцию**



**Вызов функции. Туда передается два параметра.**





## Часто используемая терминология при описании функции

Параметры, которые использовались при описании функции, называются **формальными параметрами**.

Параметры, которые были использованы при вызове функции, называются **фактическими параметрами**.

Переменные, объявленные внутри функции, а также формальные параметры являются **локальными** для этой функции. Это означает, что за пределами функции к этим переменным обратиться нельзя.



## Как работают функции

При вызове функции сначала проверяется соответствие количества фактических и формальных параметров. После этого формальные параметры присваиваются фактическими и управление передается в тело функции. Когда работа функции заканчивается, то управление возвращается в тело вызывающей программы.





## Как работают функции

```
def calculate_summa(a, b):  
    c = a+b  
    return c  
  
d = calculate_summa(10, 10)  
print(d)
```

1. def calculate\_summa(a, b):  
c = a+b  
return c

2. d = calculate\_summa(10, 10)

3. return c

4. d = calculate\_summa(10, 10)

**1. Описание функции.**

**2. Вызов функции.**

**3. При вызове управление происходит присвоение «a = 10», «b = 10» и управление передается в функцию.**

**4. После оператора return управление возвращается.**



## Некоторые вопросы о функциях

**Сколько параметров может иметь функция ?**

Произвольное количество.

**Сколько значений может возвращать функция?**

Произвольное количество. Просто укажите их через запятую после оператора `return`. В этом случае они будут автоматически упакованы в кортеж.

**Сколько функций можно описать в программе?**

Произвольное количество.



## Замечания о написании функций

1) Старайтесь описывать функции так, чтобы их длина не превышала 30-ти строк. В случае, когда тело функции превышает предел в 30-ть строк, лучшим решением будет разбиение этой функции на две и более функций с меньшей длиной.

2) Старайтесь чтобы ваша функция выполняла только одно логически завершенное действие. Не нужно объединять несколько разных завершенных действий. Пусть одна функция считывает данные с клавиатуры, вторая ищет введенные данные в списке и т.д.



## Локальные переменные функции

Все переменные, объявленные в теле функции, а также формальные параметры являются **локальными**.

Это означает, что они существуют только во время работы функции и «видны» только в теле функции. Таким образом, если за пределами функции существует переменная с таким же именем, как у переменной в теле функции, то это две разные переменные.



## Локальные переменные функции

```
def calculate_summa(a, b):  
    summa = a + b  
    return summa
```

```
summa = 10
```

```
print(calculate_summa(4,5))
```

```
print(summa)
```

В приведенном примере в теле функции и вне её существуют две переменные `summa`. Это две разные переменные и изменение значения одной не повлияет на значение другой.



## О параметрах функции

Если параметры функции неизменяемого типа, то изменение параметра в теле функции не влияет на фактический параметр.

Если формальным параметром выступает изменяемый тип, то изменение формального параметра повлияет и на фактический.



## Пример функции с параметром неизменяемого типа данных

```
def calculate_summa(a, b):  
    a = a + 3  
    summa = a + b  
    return summa  
  
a = 5  
b = 3  
  
print(calculate_summa(a, b))  
print(a)
```

И хотя в теле функции изменяется значение формального параметра **a**, на значение фактического параметра это не влияет. Потому что, в данном случае, параметром является неизменяемый тип данных.



## Пример функции с параметром изменяемого типа данных

```
def calculate_summa(a):  
    a[0] = a[0] + 3  
    summa = 0  
    for element in a:  
        summa = summa + element  
    return summa  
a = [3, 5, 10]  
  
print(calculate_summa(a))  
print(a)
```

В данном случае параметром является изменяемый тип данных. И изменение в теле функции формального параметра влияет на фактический.





## Как из локальной переменной сделать глобальную

Если возникает необходимость объявлении в теле функции глобальной переменной (т. е. видимой за пределами функции) используется оператор **global**.

Синтаксис его применения таков:

```
global имя_переменной
```

В таком случае переменная, объявленная таким образом, станет доступна во всем модуле.



# Как из локальной переменной сделать глобальную

```
def calculate_summa(a):  
    global b  
    b=a  
    summa = a + b  
    return summa  
  
print(calculate_summa(5))  
print(b)
```

В этом примере объявлена глобальная переменная с именем **b**. И ее можно использовать за пределами функции.

**Внимание!! Это хоть и существующая возможность, ее применение при написании функций считается не самой удачной практикой. Поэтому используйте эту возможность только в случае, когда иных решений нет.**



## Рекурсия

В программировании **рекурсия** — вызов функции (процедуры) из нее же самой непосредственно (простая рекурсия) или через другие функции (сложная или косвенная рекурсия).

Количество вложенных вызовов функции или процедуры называется **глубиной рекурсии**.

Структурно рекурсивная функция на верхнем уровне всегда представляет собой команду ветвления (выбор одной из двух или более альтернатив в зависимости от условия (условий), которое в данном случае уместно назвать «условием прекращения рекурсии»), имеющей две или более альтернативные ветви, из которых хотя бы одна является **рекурсивной** и хотя бы одна **терминальной**.

**Рекурсивная** ветвь выполняется, когда условие прекращения рекурсии ложно, и содержит хотя бы один рекурсивный вызов — прямой или опосредованный вызов функцией самой себя.

**Терминальная** ветвь выполняется, когда условие прекращения рекурсии истинно. Она возвращает некоторое значение, не выполняя рекурсивного вызова.



## Пример рекурсии

```
def summa (number_list, i):  
    if i == len(number_list) - 1:  
        return number_list[i] ← Терминальная ветвь  
    else:  
        return number_list[i] + summa(number_list, i+1)
```

↑  
**Рекурсивная ветвь**

```
list_one = [2, 6, 9]
```

```
print(summa(list_one, 0))
```

В этом примере описана рекурсивная функция, которая подсчитает сумму элементов списка без использования циклов.



## Некоторые вопросы о рекурсии

**Рекурсия кажется мне сложноватой, часто ли ее используют в разработке ?**

Нет. Это больше для решения «алгоритмических» задач повышенной сложности. В обычной разработке рекурсивные функции используются редко.

**Можно ли заменить рекурсивные функции на иной инструмент ?**

Да. Любая задача, которая может быть решена с помощью рекурсивной функции, может быть решена с использованием циклов и наоборот.

**Если забыть условие выхода из цикла он зациклится. Что будет, если забыть терминальную ветвь в рекурсивной функции?**

Каждый вызов функции занимает некоторое количество памяти. Поскольку количество вызовов функции не ограничено, то будет исчерпана память и программа аварийно завершится.



## Список использованной литературы

- 1) Лутц М. Изучаем Python, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011 - 461 — 472 с.
- 2) Саммерфилд М. Программирование на Python 3. Подробное руководство. - Пер. с англ. - СПб.:Символ-Плюс, 2009. - 203 — 216 с.



## Домашнее задание

- 1) Напишите функцию, которая вернет максимальное число из списка чисел.
- 2) Реализуйте функцию, параметрами которой являются - два числа и строка. Возвращает она конкатенацию строки с суммой чисел.
- 3) Реализуйте функцию рисующую на экране прямоугольник из звездочек «\*». Ее параметрами будут целые числа, которые описывают длину и ширину такого прямоугольника.
- 4) Напишите функцию, которая реализует линейный поиск элемента в списке целых чисел. Если такой элемент в списке есть, то верните его индекс, если нет, то верните число «-1».
- 5) Напишите функцию, которая вернет количество слов в строке текста.



## Дополнительное домашнее задание.

1) Существуют такие последовательности чисел:

0,2,4,6,8,10,12

1,4,7,10,13

1,2,4,8,16,32

1,3,9,27

1,4,9,16,25

1,8,27,64,125

Реализуйте программу, которая выведет следующий член этой последовательности (либо подобной им) на экран. Последовательность пользователь вводит с клавиатуры в виде строки. Например, пользователь вводит строку 0,5,10,15,20,25 и ответом программы должно быть число 30.

2) Число-палиндром с обеих сторон (справа налево и слева направо) читается одинаково. Самое большое число-палиндром, полученное умножением двух двузначных чисел:  $9009 = 91 \times 99$ . Найдите самый большой палиндром, полученный умножением двух трехзначных чисел. Выведите значение этого палиндрома и то, произведением каких чисел он является.