Functions in Python with Examples

A function in Python is an organized block of reusable code, which avoid repeating the tasks again and again. If you want to do a task repeatedly in a code, then just make a function, set the task in it and call the function multiple times whenever you need it. Functions in Python begins with the keyword **def** and is followed by the name of the function and parentheses.

Create a function in Python

Functions in Python, use the **def** keyword as in the below syntax:

```
def function_name( parameters ):
    function_suite
    return [expression]
```

Above,

- The **def** keyword is used to define a function name,
- The parameters are optional to add,
- To return a value, the return statement is used.

Note: We will run below example programs in Python to understand the concept of Functions in Python.

Call a function in Python

Code is structured when we define a function. Execution of this function is done by calling. Functions in Python work the same as in any other programming language. To call, the function name is followed by parentheses.

Let us see an example and create a function **demo** and call it in

Python:

```
#!/usr/bin/python

# Defining a Function i.e. Function Definition
def demo( ):
  print("We provide free tutorials and interview questions.");
  print("We also provide interview questions and answers with q

# call the demo function
demo()
```

The output is as follows:

```
We provide free tutorials and interview questions.
We also provide interview questions and answers with quizzes to
```

In the above example, we created a function named **demo()** and added two print statements to it. The text in this statements is visible only when the **demo()** function is called:

Function Parameters

The names entered in the function at the time of defining it, are Function Parameters. Adding Parameters to a function in Python is optional. Let us see the syntax for Function Parameters in Python:

```
def function_name(parameter1, parameter2, … ):
```

Above, **parameter1** and **parameter2** are the parameters which is added to the function while defining it. Let us now see the above example and add a parameter to it:

```
#!/usr/bin/python

# Defining a Function i.e. Function Definition with parameter s
def demo(str):
  print("Device:",str);

# call the demo function
demo("Laptop")

#call again
demo("Desktop")
```

The output is as follows:

```
Device: Laptop
Device: Desktop
```

The above function takes string **str** as a parameter. The string "**Device**:" is passed along the device name i.e. **Laptop** and **Desktop**.

Function Arguments in Python

In Python,

- Required arguments
- Keyword arguments
- Arbitrary Keyword Arguments
- Default arguments
- Variable-length/ Arbitrary arguments

Let us learn about the above types of Function Arguments, one by one,

Required arguments in Python

The required arguments as the name suggests are the required number of arguments to be passed on a function call. When you call a function, the arguments in the call are to match with the function definition for the program to run correctly. An error occurs even if the position of the argument is changed.

Let us see an example wherein we have two arguments and we are calling with the same number and position of arguments:

```
#!/usr/bin/python

# Defining a Function with parameters l and b
# length for length and b for breadth of a rectangle
def area(l,b):
  print("Area of Rectangle = ",l*b);

# call the area() function
area(20, 30)
```

The output is as follows:

```
Area of Rectangle = 600
```

Keyword arguments in Python

Call a function with keyword arguments in Python. Through this, you can skip arguments or even pass them in any order while calling a function. As we saw above, this is definitely different that Required arguments wherein the order in which the arguments are passed is the same while calling.

Let us see an example wherein we are calling the function with two arguments passed in random order:

```
#!/usr/bin/python

# Defining a Function with parameters name and rank
def details(name,rank):
  print("Student Name = ",name);
  print("Student Rank = ",rank);

# Call the details() function with arguments passed in any orde
details(rank = 5, name = "Jack")
```

The output is as follows:

```
Student Name = Jack
Student Rank = 5
```

Let us see another wherein we are calling the function with three arguments passed in random order:

```
#!/usr/bin/python

# Defining a Function with parameters product, pid, price
def details(name, pid, price):
  print("Produce Name = ",name);
  print("Product ID = ",pid);
  print("Product Price = ",price);

# Call the details() function with arguments passed in random o
details(price = 550.50, name = "Jack", pid = 567)
```

The output is as follows:

```
Produce Name = Jack
Product ID = 567
Product Price = 550.5
```

Default arguments in Python

While calling a function, if the value isn't passed, a default value is assumed for that argument. Let us see an example and display the default value. We have a function below with total 2 arguments:

```
#!/usr/bin/python

# Defining a Function with parameters name and rank
def details(name,rank = 10):
  print("Student Name = ",name);
  print("Student Rank = ",rank);

# Call the details() function
details(name = "John", rank = 5)
details(name = "Mark", rank = 8)

# Default value assumed for rank parameter
# since we haven't passed value for rank
details(name = "Jacob")
```

The output is as follows:

```
Student Name = John
Student Rank = 5
Student Name = Mark
Student Rank = 8
Student Name = Jacob
Student Rank = 10
```

Let us now see another example wherein the default value is assumed for the argument, whose value isn't specified. We have a function below with total 3 arguments:

```
#!/usr/bin/python

## Defining a Function with parameters name, rank amd score
def details(name,rank = 10, score = 60.50):
  print("Student Name = ",name);
  print("Student Rank = ",rank);
  print("Student Score = ",score);
  print("----------------------");

## Call the details() function
```

```
# All the values passed while calling
details(name = "John", rank = 5, score = 20.50)

# Default value assumed for score parameter
# since we haven't passed value
details(name = "Mark", rank = 8)

# Default value assumed for rank and score parameter
# since we haven't passed values for both
details(name = "Jacob")
```

The output is as follows:

```
Student Name = John
Student Rank = 5
Student Score = 20.5
----------------------
Student Name = Mark
Student Rank = 8
Student Score = 60.5
----------------------
Student Name = Jacob
Student Rank = 10
Student Score = 60.5
----------------------
```

Variable-length/ Arbitrary arguments in Python (*args)

Arbitrary arguments work when you have no idea about the number of arguments to be passed. These are variable length arguments, allowing you to pass any number of arguments, defined as * **(asterisk)**. That means, include a * before the parameter name while defining the function. Let us now see an example:

```
#!/usr/bin/python

## Defining a Function with variable number of arguments
def demo(*sports):
  print("Sports 1 = ",sports[0]);
  print("Sports 2 = ",sports[1]);
  print("Sports 3 = ",sports[2]);

# Call the details() function
demo("Football", "Hockey", "Cricket")
```

The output is as follows:

```
Sports 1 = Football
```

```
Sports 2 = Hockey
Sports 3 = Cricket
```

We passed ***sports** as variable-length/arbitrary argument above.

Let us see another example, wherein we will use for loop to display all the variable-length argument values:

```
#!/usr/bin/python

# Defining a Function with variable number of arguments
def demo(*sports):
  print("Displaying passed arguments...");

  for name in sports:
    print(name)

# Call the details() function
demo("Football", "Hockey", "Cricket", "Squash", "Volleyball")
```

The output is as follows:

```
Displaying passed arguments...
Football
Hockey
Cricket
Squash
Volleyball
```

Arbitrary Keyword Arguments in Python (**kwargs)

Add two asterisks ** before the parameter name in a function definition, if you don't know in advance how many **keyword arguments** will be passed to your function. A dictionary format gets created for such arguments i.e. a Dictionary of arguments

Let us see an example:

```
#!/usr/bin/python

## Defining a Function with multiple keyword arguments **kwargs
def demo(**stu):
  print("Student name: " + stu["student"])
  print("Student section: " + stu["section"])

#Call the demo function
```

```
demo(student = "Jack",section = "AD")
```

The output is as follows:

```
Student name: Jack
Student section: AD
```

Recursion in Python

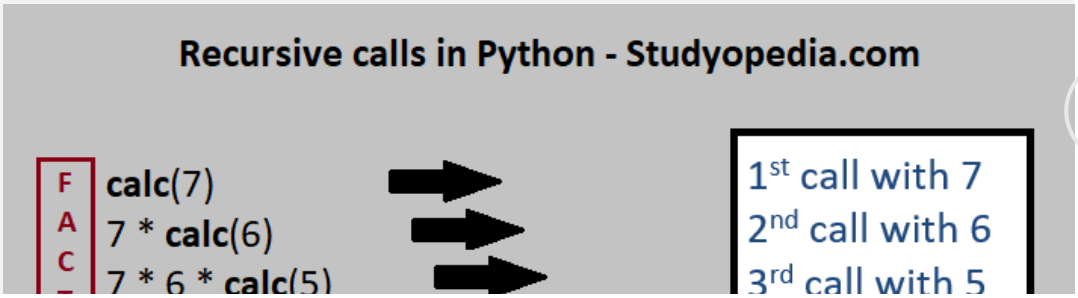
When a function calls itself, it is called Recursion. In another sense, with Recursion, a defined function can call itself. Recursion is a programming approach, which makes code efficient and reduces LOC. Let us see an example wherein we will be calculating factorial with Recursion:

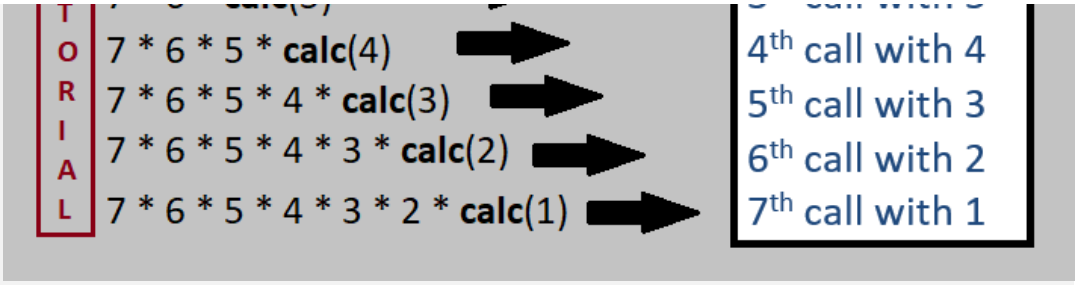
```
#function to calculate factorial
def calc(num):

    # Condition for 1! (1 factorial)
    if num == 0:
        return 1
    else:
        return num * calc(num-1)

#function call
print("0! = ",calc(0));
print("1! = ",calc(1));
print("7! = ",calc(7));
print("10! = ",calc(10));
```

The output is as follows:

```
0! = 1
1! = 1
7! = 5040
10! = 3628800
```

The above program calls in a recursive way. To get the value of 7!, the function **calc()** works like:





The return statement in Python

The return statement in Python is used to terminate/exit the function and is placed at the end. It can have an expression or can be used without any expression. A return statement with an expression gets evaluated first and then returned to the function called. However, a return statement with no expression returns **None**. Let us now see an example:

```
#!/usr/bin/python

# Defining a Function
def demo(num):
  return 10 + num

print(demo(2));
print(demo(10))
```

The output is as follows:

```
12
20
```

Let us see another example:

```
#!/usr/bin/python

# Defining a function
def demo( a, b, c ):

   res = a + b + c
   return res;

# Displaying result of sum
res = demo( 5, 7, 15);
print ("Result = ", res )

res = demo(50, 500, 1000);
print ("Result = ", res )
```

The output is as follows:

```
Result = 27
Result = 1550
```

Let us now see the above Python program without using a return statement, since using a return is optional:

```
#!/usr/bin/python

# Defining a function
def demo( a, b, c ):
   print ("Result = ", a+b+c)

# Displaying result of sum
demo( 5, 7, 15)
demo( 50, 500, 1000)
```

The output is as follows:

```
Result = 27
Result = 1550
```

In this lesson, we learned Functions in Python with several examples. We also saw how to work with function arguments, including default, required, keyword, arbitrary arguments, etc.

Read More

- Scope of Variables
- Python Operators
- Python Numbers
- Type Conversion
- Python Strings
- Loops in Python
- Decision Making Statements
- Python Tuples
- Python Dictionary
- Python Lists
- Python DateTime