Python Functions is a block of related statements designed to perform a computational, logical, or evaluative task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.

Functions can be both built-in or user-defined. It helps the program to be concise, non-repetitive, and organized.

Syntax:

```
def function_name(parameters):
    """docstring"""
    statement(s)
    return expression
```

Creating a Function

We can create a Python function using the **def** keyword.

Example: Python Creating Function

- Python3

```
# A simple Python
function

def fun():
  print("Welcome to
GFG")
```

Calling a Function

After creating a function we can call it by using the name of the function followed by parenthesis containing parameters of that particular function.

Example: Python Calling Function

- Python3

```
# A simple Python function

def fun():
  print("Welcome to GFG")

# Driver code to call a
function
fun()
```

Output

```
Welcome to GFG
```

Arguments of a Function

Arguments are the values passed inside the parenthesis of the function. A function can have any number of arguments separated by a comma.

Example: Python Function with arguments

In this example, we will create a simple function to check whether the number passed as an argument to the function is even or odd.

- Python3

```
# A simple Python function to
check
# whether x is even or odd


def evenOdd(x):
    if (x % 2 == 0):
        print("even")
    else:
        print("odd")


# Driver code to call the
function
evenOdd(2)
evenOdd(3)
```

Output

```
even
odd
```

Types of Arguments

Python supports various types of arguments that can be passed at the time of the function call. Let’s discuss each type in detail.

Default arguments

A default argument is a parameter that assumes a default value if a value is not provided in the function call for that argument. The following example illustrates Default arguments.

- Python3

```
# Python program to demonstrate
# default arguments


def myFun(x, y=50):
    print("x: ", x)
    print("y: ", y)


# Driver code (We call myFun()
with only
# argument)
myFun(10)
```

Output

```
('x: ', 10)
('y: ', 50)
```

Like C++ default arguments, any number of arguments in a function can have a default value. But once we have a default argument, all the arguments to its right must also have default values.

Keyword arguments

The idea is to allow the caller to specify the argument name with values so that the caller does not need to remember the order of parameters.

- Python3

```
# Python program to demonstrate Keyword
Arguments
def student(firstname, lastname):
    print(firstname, lastname)


# Keyword arguments
student(firstname='Geeks',
lastname='Practice')
student(lastname='Practice',
firstname='Geeks')
```

Output

```
('Geeks', 'Practice')
('Geeks', 'Practice')
```

Variable-length arguments

-----------X--------------X
The following topic(variable length non-keywords argument and docstring) is a bit advanced for learners who are new to python and can be skipped or taught based on the response of the class.
---------X--------------------X

In Python, we can pass a variable number of arguments to a function using special symbols. There are two special symbols:

- *args (Non-Keyword Arguments)
- **kwargs (Keyword Arguments)

Example 1: Variable length non-keywords argument

- Python

```
# Python program to illustrate
# *args for variable number of arguments


def myFun(*argv):
    for arg in argv:
        print(arg)


myFun('Hello', 'Welcome', 'to',
'GeeksforGeeks')
```

Output

```
Hello
Welcome
to
GeeksforGeeks
```

Example 2: Variable length keyword arguments

- Python3

```
# Python program to illustrate
# *kargs for variable number of keyword
arguments


def myFun(**kwargs):
    for key, value in kwargs.items():
        print("%s == %s" % (key, value))


# Driver code
myFun(first='Geeks', mid='for',
last='Geeks')
```

Output

```
first == Geeks
mid == for
last == Geeks
```

Docstring

The first string after the function is called the Document string or Docstring in short. This is used to describe the functionality of the function. The use of docstring in functions is optional but it is considered a good practice.

The below syntax can be used to print out the docstring of a function:

Syntax: `print(function_name.__doc__)`

Example: Adding Docstring to the function

- Python3

```
# A simple Python function to check
# whether x is even or odd


def evenOdd(x):
    """Function to check if the number is even
or odd"""

    if (x % 2 == 0):
        print("even")
    else:
        print("odd")


# Driver code to call the function
print(evenOdd.__doc__)
```

Output

```
Function to check if the number is even or odd
```

The return statement

The function return statement is used to exit from a function and go back to the function caller and return the specified value or data item to the caller.

Syntax: `return [expression_list]`

The return statement can consist of a variable, an expression, or a constant which is returned to the end of the function execution. If none of the above is present with the return statement a None object is returned.

--------X-------------X

It is not important for the return function to return *int* data type only. It can return anything and everything, including but not limited to list, tuples, dictionary, string and all other data types. It will be further explained in this course that a function can even return a python object. This is part of Object oriented programming.

Example: Python Function Return Statement

- Python3

```
def square_value(num):
    """This function returns the
square
    value of the entered
number"""
    return num**2


print(square_value(2))
print(square_value(-4))
```

Output:

```
4
16
```

Is Python Function Pass by Reference or pass by value?

One important thing to note is, in Python every variable name is a reference. When we pass a variable to a function, a new reference to the object is created. Parameter passing in Python is the same as reference passing in Java.

Example:

- Python3

```
# Here x is a new reference to same
list lst
def myFun(x):
    x[0] = 20


# Driver Code (Note that lst is
modified
# after the function call.
lst = [10, 11, 12, 13, 14, 15]
myFun(lst)
print(lst)
```

Output

```
[20, 11, 12, 13, 14, 15]
```

When we pass a reference and change the received reference to something else, the connection between the passed and received parameter is broken. For example, consider the below program.

- Python3

```
def myFun(x):

    # After below line link of x with
previous
    # object gets broken. A new object is
assigned
    # to x.
    x = [20, 30, 40]


# Driver Code (Note that lst is not
modified
# after the function call.
lst = [10, 11, 12, 13, 14, 15]
myFun(lst)
print(lst)
```

Output

```
[10, 11, 12, 13, 14, 15]
```

Another example to demonstrate that the reference link is broken if we assign a new value (inside the function).

- Python3

```
def myFun(x):

    # After below line link of x with
previous
    # object gets broken. A new object is
assigned
    # to x.
    x = 20
```

```
# Driver Code (Note that lst is not
modified
# after the function call.
x = 10
myFun(x)
print(x)
```

Output

```
10
```

Exercise: Try to guess the output of the following code.

- Python3

```
def swap(x,
y):
    temp = x
    x = y
    y = temp


# Driver code
x = 2
y = 3
swap(x, y)
print(x)
print(y)
```

Output

```
2
3
```

----X--------X

Anonymous functions also known as lambda functions are very important when it comes to dealing with dictionary data types. It allows easy sorting and can be passed as a reference to the sort function and many other in-built python functions.

Anonymous functions:

In Python, an anonymous function means that a function is without a name. As we already know the def keyword is used to define the normal functions and the lambda keyword is used to create anonymous functions. Please see this for details.

- Python3

```
# Python code to illustrate the cube of a
number
# using lambda function


def cube(x): return x*x*x

cube_v2 = lambda x : x*x*x

print(cube(7))
print(cube_v2(7))
```

Output

```
343
```

Python Function within Functions

A function that is defined inside another function is known as the inner function or nested function. Nested functions are able to access variables of the enclosing scope. Inner functions are used so that they can be protected from everything happening outside the function.

- Python3

```
# Python program to
# demonstrate accessing of
# variables of nested
functions

def f1():
    s = 'I love
GeeksforGeeks'

    def f2():
        print(s)

    f2()

# Driver's code
f1()
```

Output

```
I love GeeksforGeeks
```

Difference between print and return

It can confuse the learner as to what is the difference between returning a function and printing a function. Since, printing inside a function and printing the returned function end up giving the same result.

The difference arises in the functionality of the return function.

If we print a statement inside the function and do not return it, the statement will be available at the output and could not be used as a variable in any other part of the code.

However, if we return a statement, we can store that statement in a variable and can use that variable in a different part of the code.

```python
def fun():
    s = 'Welcome to GeeksforGeeks'
    print(s)
```

```python
fun()
```

```
Welcome to GeeksforGeeks
```

As visible, we print inside the function without a return statement. If I try to store the output of the function in a variable, it will be stored as a NoneType variable.

```python
def fun():
    s = 'Welcome to GeeksforGeeks'
    print(s)
```

```python
var = fun()
print(type(var))
```

```
Welcome to GeeksforGeeks
<class 'NoneType'>
```

The function will print the statement whenever the function is called but it won't be able to store the statement in a variable.
However, if we use the return statement, we can store it for further usage.

```python
def fun():
    s = 'Welcome to GeeksforGeeks'
    return s
```

```python
var = fun()
print(var)
print(type(var))
```

```
Welcome to GeeksforGeeks
<class 'str'>
```