Functions

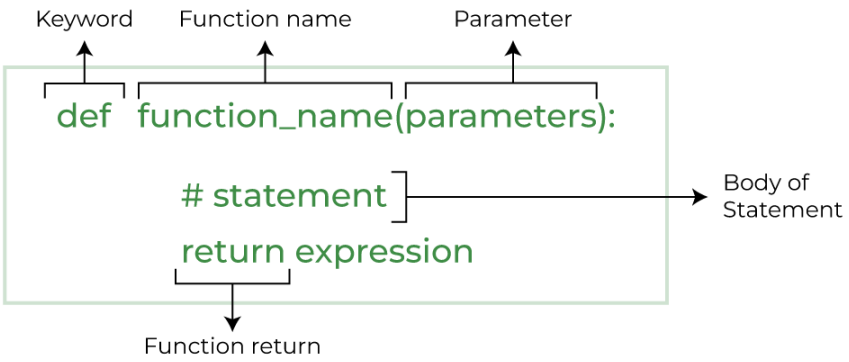
- **Python Functions** is a block of statements that return the specific task.
- The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in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

❖ Advantages of Python Functions

Pause We can stop a program from repeatedly using the same code block by including functions.

- ➢ Once defined, Python functions can be called multiple times and from any location in a program.
- ➢ Our Python program can be broken up into numerous, easy-to-follow functions if it is significant.
- ➢ The ability to return as many outputs as we want using a variety of arguments is one of Python's most significant achievements.
- ➢ However, Python programs have always incurred overhead when calling functions.

❖ Function Declaration



❖ Types of Functions in Python

There are mainly two types of functions in Python.

➢ Built-in library function

Built in functions are already defined in python.
A user has to remember the name and parameters of a particular function.
Some widely used functions are:

Function	Description
len()	Returns the length of a python object
abs()	Returns the absolute value of a number
max()	Returns the largest item in a python iterable
min()	Returns the largest item in a python iterable
sum()	Sum() in Python returns the sum of all the items in an iterator
type()	The type() in Python returns the type of a python object
help()	Executes the python built-in interactive help console
input()	Allows the user to give input
format()	Formats a specified value
bool()	Returns the boolean value of an object

➢ User-defined function:

These functions are defined by a programmer to perform any specific task or to reduce the complexity of big problems and use that function according to their need.

❖ Creating a Function in Python

We can create a user-defined function in Python, using the **def** keyword. We can add any type of functionalities and properties to it as we require.

Example:

```
# A simple Python function


def fun():

print("Welcome to Topper World")
```

Output:

```
Welcome to Topper World
```

❖ Calling a Python Function

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

Example:

```
# A simple Python function

def fun():

        print("Welcome to Topper World")


# Driver code to call a function

fun()
```

Output:

```
Welcome to Topper World
```

❖ Function Arguments

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

Types of Function Arguments

Python supports various types of arguments that can be passed at the time of the function call.

In Python, we have the following 4 types of function arguments.

- **Default argument**
- **Keyword arguments (named arguments)**
- **Positional arguments**
- **Arbitrary arguments** (variable-length arguments *args and **kwargs)

❖ 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.

Example:

```
# 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:15
```

❖ 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.

Example:

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


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

Output:

```
Topper Practice
Topper Practice
```

❖ Positional Arguments

We used the Position argument during the function call so that the first argument (or value) is assigned to name and the second argument (or value) is assigned to age.

Example:

```
def nameAge(name, age):
        print("Hi, I am", name)
        print("My age is ", age)

# You will get correct output because  argument is given in order
print("Case-1:")
nameAge("Kritika", 19)
# You will get incorrect output because argument is not in order
print("\nCase-2:")
nameAge(19, "Kritika")
```

Output:

```
Case-1:
Hi, I am Kritika
My age is  19
Case-2:
Hi, I am 19
My age is  Kritika
```

❖ Arbitrary Keyword Arguments

In Python Arbitrary Keyword Arguments, *args, and **kwargs can pass a variable number of arguments to a function using special symbols. There are two special symbols:

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

➔ 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__)
```

➔ Return Statement in Python Function

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 at the end of the function execution.
- If none of the above is present with the return statement a None object is returned.

➔ Pass by Reference and 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:

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


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

Output:

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