PARAMETERS AND ARGUMENTS

Inside the function, the a**rguments** are assigned to variables **called parameters**. Here is a definition for a function that takes an argument:

Function Arguments

Types of Formal arguments:

- Required arguments
- Default arguments
- Keyword arguments
- Variable-length arguments

1. Required Arguments

Required arguments are the arguments passed to a function in correct positional order. Here, the number of arguments in the function call should match exactly with the function definition.

Example:

```
>>>def add(a,b):     # add() needs two arguments, if not it shows error
        return a+b
>>>a=10
>>>b=20
>>>print("Sum of ", a ,"and ", b, "is" , add(a,b))
```

Output:

Sum of 10 and 20 is 30

2. Default Arguments:

A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.

Example:

```
>>>def add(a,b=0):
        print ("Sum of ", a ,"and ", b, "is" ,a+b)
>>>a=10
>>>b=20
>>>add(a,b)
>>>add(a)
```

Output:

Sum of 10 and 20 is 30

Sum of 10 and 0 is 10

3. ***Keyword Arguments:***

Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.

Example:

```
>>>def add(a,b):
        print ("Sum of ", a ,"and ", b, "is" ,a+b)
>>>a=10
>>>b=20
>>>add(b=a,a=b)
```

Output:

Sum of 20 and 10 is 30

4. ***Variable-Length Arguments:***

The special syntax *args in function definitions in python is used to pass a variable number of arguments to a function. It is used to pass a non-keyworded, variable-length argument list.

- The syntax is to use the symbol * to take in a variable number of arguments; by convention, it is often used with the word args.
- What *args allows you to do is take in more arguments than the number of formal arguments that you previously defined. With *args, any number of extra arguments can be tacked on to your current formal parameters

Example:

```
>>>def myFun(*argv):
        for arg in argv:
            print (arg)
>>>myFun('Hello', 'Welcome', 'to', 'Learn Python')
```

Output:

Hello
Welcome
to
Learn Python

The Anonymous Functions or Lambda Functions

In Python, anonymous function is a function that is defined without a name. While normal functions are defined using the def keyword, in Python anonymous functions are defined using the lambda keyword. Hence, anonymous functions are also called lambda functions.

Syntax:

```
lambda arguments: expression
```

Example:

```
>>>double = lambda x: x * 2
        print(double(5))
```

Output:

10

In the above program, lambda x: x * 2 is the lambda function. Here x is the argument and x * 2 is the expression that gets evaluated and returned.

The same Anonymous function can be written in **normal function** as

```
>>>def double(x):
        return x * 2
>>>double(5)
```