Types of Arguments

- **Positional / Required Arguments:** The arguments which are required by the callee function to perform the given task are known as positional or required arguments.
  - They must be in the same order and same number as in the callee function while calling the callee function.

  For example
  Consider the following function

```
def interest( p , t , r ):
    si = (p * r * t ) / 100
    return si
```

  Now this function can be called in the following ways.

ans =interest (10000, 5, 6)	valid
ans =interest (pr , time , rate)	valid
ans =interest (10000 , 5 , 4.5)	valid
ans =interest (pr , rate , time)	invalid
ans =interest (pr , , 7.3)	invalid

- **Default Arguments:** The formal arguments are those which have their own predefined value which can be used when required value for those arguments is not passed or supplied while calling the function.
  - It means that the default arguments become optional in the function calling.
  - If the value for default argument is passed during function call then this value override the predefined value of that default argument.
  - **One very important thing to be noted is that all the default arguments must be on the rightmost side of the non-default arguments in the function definition.**

  For example
  Now consider the following function

```
def interest( p , t =2 , r = 0.10):
    si = (p * r * t ) / 100
    return si
```

  Now this function can be called in the following ways.

ans =interest (10000)	valid

ans =interest (10000 , 5)	valid
ans =interest (10000 , 5 , 7.3)	valid
ans =interest (pr , time , 7.3)	valid
ans =interest (pr , , 7.3)	invalid

- **Keyword / Named arguments:** These arguments are used during the function calling in those cases where we do not want to maintain the sequence of arguments / parameters.
  - ❖ It means, using named arguments we can reorder the arguments being passed to the callee function.
  - ❖ But we must have to write the name of the argument to which the value is to be passed.

For example

Consider the following function

```
def interest( p , t =2 , r = 0.10):
    si = (p * r * t ) / 100
    return si
```

Now this function can be called in the following ways.

```
ans =interest (p=10000 , t=5 , r=7.3)
ans =interest (t=5 , p=10000 , r=7.3)
ans =interest (p=10000 , r=7.3 , t=5 )
ans =interest (t=5 , p=10000 )
ans =interest (r=5 , p=10000 )
ans =interest (p=10000 )
```

and so on , All are Valid

Returning Multiple Values from Function

In Python we can return multiple values in the following ways:

- Returning values in the form of tuple variable.

Example

```
def squared(x,y,z):
    return x*x , y*y , z*z

t = squared (2,3,4)
print(t)
```

⟶ Here t is a tuple

- Directly unpacking received values of tuple by using no. of variables on left hand side of function calling.

Example

```
def squared(x,y,z):
    return x*x , y*y , z*z

v1 , v2 , v3 = squared (2,3,4)
print(v1, v2, v3)
```

Scope and Lifetime of Variable

The parts of program in which a variable can be used or accessed is called as the scope of the variable and the time period for which variable lives in memory is called its lifetime. The scope of variable is of following two types:

- Global Scope
  - Variables declared in top level segment or in (__main__) section of the program.
  - Variables declared outside body of all the function i.e. in the beginning

  These variables can be accessed inside the whole program i.e. in any block (if, for, while etc.) or in any function body.

- Local Scope
  - Variables declared inside the body of a particular function.

  These variables can be accessed only inside the function body in which it is declared.

For Example: a=10

```
def  f1(x,y):
        print(a)        Global variable          valid
        print(x)        local to f1( )           valid
        print(y)        local to f1( )           valid
        print(p)        local to f2( )           invalid     error
        print(n)        global variable declared in _main_   valid
def  f2(p,q):
        print(a)        Global variable          valid
        print(p)        local to f2( )           valid
        print(q)        local to f2( )           valid
        print(x)        local to f1( )           invalid     error
        print(n)        global variable declared in _main_   valid
# body of _main_ or Top level segment
n=20
print(a)        Global variable          valid
print(p)        local to f2( )           invalid     error
print(x)        local to f1( )           invalid     error
print(n)        global variable declared in _main_   valid
```