WORKING WITH PYTHON FUNCTIONS:

Function means Code Reusability.It performs a specific task
OR
A function is a block of organized, reusable code. Functions provide better modularity (high degree of code reusing)

Types of Functions
Basically functions are two types:

Built-in functions:
Functions that are built into Python like:
all(), any(), bin(), bool(), bytes(), callable(), chr(), print(), file(),len(), input()...etc.!

User-defined functions (Building our Own Functions)
Functions defined by the users themselves as per client or business requirements..!!

Simple rules to define a function in Python:
1 Function blocks begin with the keyword 'def', the function name & parentheses ( ).
2 Any input parameters or arguments should be placed within these parentheses.
3 The first statement of a function can be an optional statement, It is docstring.
4 The code block within every function starts with a colon (:) and is indented.
5 A return statement with no arguments is the same as return None.

SYNTAX:
```
def Function_Name( argument1, argument2,....,argumentN):
Statement1
Statement2
return [expression]
```

Example:
```
#Definition of the function
#Name of the function
def MyMsgs_Shows():#Indentation of the function (right)
    #Body of the function
    #Logical part of the function
    #Called Part of the function
    #Argument declaration section
    print("Hey Welcome to Functions")
    print('Good One')
    #End of the function
    #Always Optional
    #It is the statement
    return()
#Calling Part
#Passing Parameters
#Passed values are parameters
#Code recalling/Reusable
MyMsgs_Shows()
MyMsgs_Shows()
```

Example:

```
def MyMsgs_Shows():
    print("Hey Welcome to Functions")
    print('Good One')
    return()
MyMsgs_Shows()
```

The Return statement in function
It is used to return a value from a function, return statement without an expression argument returns none.

Syntax:

```
def function_name(argument1, argument2, ...) :
    statement_1
    statement_2
    ....
    return expression
function_name(arg1, arg2)
```

Example:

```
def Sum( arg1, arg2 ):
   total = arg1 + arg2
   print("The Total is:",total)
   return;
total = Sum(30,20);
```

Example:

```
def Tot_Sub(a,b):
    Tot=a+b
    Sub=a-b
    return(Tot,Sub)
x,y=Tot_Sub(10,20)
print("The Sum is :",x)
print("The Subtraction is :",y)
```

Example:

```
def Input():
    x=input("Enter Any Number: ")
    y=input("Enter Any Number: ")
    z=int(x)+int(y);
    print(z)
    return()
Input()
```

Various Forms of Function Arguments
You can call a function by using the following types of formal arguments:
1 Required Arguments or Positional Arguments
2 Keyword Arguments
3 Default Arguments
4 Variable-Length/Arbitary Arguments

Positional Arguments
Arguments passed should match the function parameter from left to right

Syntax:

```
def FunName(arg1,arg2):
     SuiteOfTheFunction
```

Example:

```
def fun(a,b,c):
  print(a,b,c)
fun(1,2,3)
```

Example:

```
def printMy( str ):
    print(str)
    return;
printMy()
```

NOTE:

```
TypeError: printMy() missing 1 required positional argument: 'str'
```

Keyword Arguments(Parameters)

We can call a function by specifying the keyword argument in the form argument name= value

Syntax:

```
def FunctionName(arg1,arg2,arg3):
      SuiteOfStatements
```

Example:

```
def fun(a,b,c):
  print(a,b,c)
fun(c=3,b=2,a=1)
```

Example:

```
def printMy(str):
    print(str)
    return;
printMy(str = "RoboticProcessAutomation-RPA")
```

Example:

```
def printinfo( name, exp ):
    print("Name: ", name)
    print("Experience: ", exp)
    return;
printinfo( exp='15+Yrs', name="KSRaju" )
```

Example:

```
def nsquare(x, y = 2):
    return (x*x+y*y)
print("The square of the sum of 2 and 3 is : ", nsquare(2))
print("The square of the sum of 2 and 3 is : ", nsquare(2,4))
```

Default Argument Values

We can assign default value for arguments to receive if the call passes too few values

OR

A default value can be written in the format "argument1 = value", therefore we will have the option to declare or not declare a value for those arguments.

Syntax:

```
def Default_Args(arg=value,arg=value):
      Suite of Statements
```

Example:

```
def fun(a,b=2,c=3):
  print(a,b,c)
fun(1)
```

Example:

```
def printinfo( name, exp = '15+' ):
    print("Name: ", name)
    print("Experience: ", exp)
    return;
printinfo( exp=14, name="KSRaju" )
printinfo( name="NareshIT" )
```

Example:

```
def Scores(BigData,Hadoop=85,Spark=80):
    print(BigData,Hadoop,Spark)
Scores(71,77)
Scores(65,Spark=74)
Scores(BigData=70,Hadoop=90,Spark=75)
```

Arbitrary Argument lists or Variable-length arguments
Add an arbitary argument in the function definition start the variable name with *

Syntax:

```
def functionname([formal_args,] *var_args_tuple ):
  "doc_string"
   function_suite
   return [expression]
```

Example:

```
def fun(*arr):
  for val in arr:
    print(val)
fun(1,2,3,4)
```

Example:

```
def fun1(*var):
   for x in var:
      print(x)
fun( 70, 60, 50 )
```