Functions in Python

- The purpose of Functions is that "To perform Certain Operation and Provides Code Re-Usability".

Definition of Function

- Sub Program of Main Program is called Function

(OR)

- A Part of Main Program is called Function.

Types of Fuctions

1. InBuilt Functions
2. User-defined Functions

1. InBuilt Functions

- InBuilt (BuiltIn) functions are those that are already defined in Python libraries and we can call them directly.
- Examples : - math, random, os, etc are InBuilt modules.

2. User-defined Functions

- User defined functions are those that we define ourselves in our program and then call them wherever we want.

Parts of Functions

- When we are dealing with Functions concept, we must ensure that there must exist 2 Parts. They are

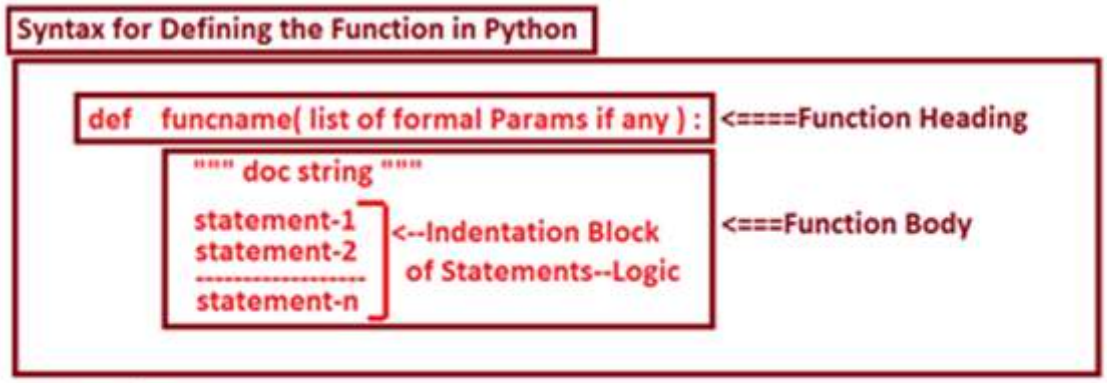
1. Function Definition
2. Function Call

- Programmatically, A Particular Function Definition Defined / Exist only Once.
- Programmatically, Function Calls exist many times.
- For Every Function Call, There Must Exist Function Definition otherwise we get NameError.
- Function Definition will execute through Function Calls otherwise Function Definition will not execute.

Phases in Functions

- When we are dealing with Functions concept, we must ensure that there must exist 3 Phases. They are

```
1. Every Function Takes INPUT
2. Every Function can do Processing
3. Every Function can give OUTPUT / RESULT
```



Explanation:

1. Here "def" is a keyword used for defining Functions in Python
2. Here "funcname" is a valid variable name used as Function Name and Function name is an object of <class, 'function'>
3. Here "list of formal parameters" represents list of variable names used in Function in Heading and they are used for storing the inputs coming from Function call(s).
4. here """doc string""" reprsents documentation String and it gives description about functionality of Function and It is optional to write.
5. Here statement-1,statement-2.....statement-n represents Indentation Block of statements and they are used for processing Inputs coming from Function call(s) and Indentation Block of statements called Business Logic.
6. In Function Body, we some variables, which are used for storing Temporary results and those variables are called "Local Variables".
7. The values of Formal parameters and local variables can be used in corresponding Function Definition but not possible to access in other part of the program.

Number of approaches to define Functions

- To define any Function in Python, we have 4 Approaches. They are

1) Approach-1

- In Approch-1, we do the following

```
Step-1 : INPUT Taking from Function Call
Step-2 : PROCESS doing in Function Body
Step-3 : RESULT / OUTPUT giving to Function Call
```

- **Examples**

In [19]:
```
# Program addition of two numbers by using Functions---Approach-1

# Creating a function defination.
def addop(a,b):     # Here a,b are called Formal Parameters.

    c=a+b           # Addition of two numbers.

    ''' Here return is a statement, used for returning the result of the
      function to function call.'''
    return c



#main program.

# INPUT Taking from Function Call.
a=int(input("Enter a value:"))
b=int(input("Enter b value:"))

result=addop(a,b)        # function call.

# RESULT / OUTPUT giving to Function Call.
print("Sum of ({} + {}) = {}".format(a,b,result))
```

```
Enter a value:12
Enter b value:23
Sum of (12 + 23) = 35
```

2) Approach-2

- In Approch-2, we do the following

```
Step-1 : INPUT Taking in Function Body
Step-2 : PROCESS doing in Function Body
Step-3 : RESULT / OUTPUT displayed in Function Body
```

- **Examples**

In [18]:
```
# Program addition of two numbers by using Functions---Approach-2

# Creating a function defination.
def addop():     # Here we are not taking any Formal Parameters.

    # INPUT Taking from Function Body.
    a=int(input("Enter a value:"))
    b=int(input("Enter b value:"))

    c=a+b           # Addition of two numbers.

    # RESULT / OUTPUT displayed in Function Body.
    print("Sum of ({} + {})={}".format(a,b,c))




#main program

addop()        # function call.
```

```
Enter a value:345
Enter b value:123
Sum of (345 + 123)=468
```

3) Approach-3

- In Approch-3, we do the following

```
Step-1 : INPUT Taking from Function Call
Step-2 : PROCESS doing in Function Body
Step-3 : RESULT / OUTPUT displayed in Function Body
```

- **Examples**

In [17]:
```
# Program addition of two numbers by using Functions---Approach-3

# Creating a function defination.
def addop(a,b):     # Here a,b are called Formal Parameters.

    c=a+b           # Addition of two numbers.

    # RESULT / OUTPUT displayed in Function Body.
    print("Sum of ({} + {})={}".format(a,b,c))



#main program.

# INPUT Taking from Function Call.
a=int(input("Enter a value:"))
b=int(input("Enter b value:"))

addop(a,b)        # function call.
```

```
Enter a value:321
Enter b value:123
Sum of (321 + 123)=444
```

4) Approach-4

- In Approch-4, we do the following

```
Step-1 : INPUT Taking in Function Body
Step-2 : PROCESS doing in Function Body
Step-3 : RESULT / OUTPUT giving to Function Call
```

- **Examples**

In [16]:
```
# Program addition of two numbers by using Functions---Approach-4

# Creating a function defination.
def addop():     # Here we are not taking any Formal Parameters.

    # INPUT Taking from Function Body.
    a=int(input("Enter a value:"))
    b=int(input("Enter b value:"))

    c=a+b           # Addition of two numbers.

    # In Python , a return statement can return one or more number of values.
    return a,b,c


#main program

# Function Call with Multi Line assignment.
a,b,c=addop()

# RESULT / OUTPUT displayed in Function Body.
print("Sum of ({} + {})={}".format(a,b,c))

print("-------------OR-------------------")
sid=addop() # Function Call with single line assigment

#print(sid, type(sid)) # sid is an object of <class,'tuple'>
print("-"*60)
print(sid,type(sid))
print("-"*60)

# RESULT / OUTPUT displayed in Function Body.
print("Sum of ({},{})={}".format(sid[-3],sid[-2],sid[-1]))
```

```
Enter a value:200
Enter b value:123
Sum of (200 + 123)=323
-------------OR-------------------
Enter a value:200
Enter b value:123
------------------------------------------------------------
(200, 123, 323) <class 'tuple'>
------------------------------------------------------------
Sum of (200,123)=323
```

- ## Difference Between return Vs print().
- **return**

  1) return is used to return a value from a function and exit the function. To return a value from a function, use the return keyword. You can use the returned value later by storing it in a variable.

- **print()**

  1) print() means displaying a value in the console. To print a value in Python, you call the print() function. After printing a value, you can no longer use it.

In [15]:
```
#Program for calculating area of rectangle by using Functions.

# Creating a function defination.
def area_of_rect():    # Here we are not taking any Formal Parameters.

    # INPUT Taking from Function Body.
    l=float(input("Enter the Length:"))
    b=float(input("Enter the Breadth:"))

    result=l*b        # calculating area of rectangle

    # RESULT / OUTPUT displayed in Function Body.
    print("Area of Rectangle ({} x {})={}".format(l,b,result))

#main program
area_of_rect()    # Function call
```

```
Enter the Length:10
Enter the Breadth:44
Area of Rectangle (10.0 x 44.0)=440.0
```

In [14]:
```
#Program calculating simple interest and total amount to pay by using functions.

# Creating a function defination.
def simpleint(p,t,r):    # Here p,t,r are called Formal Parameters.

    # Calculate simple interest.
    si=(p*t*r)/100

    # Calculate total amount to pay.
    totalamt=p+si

    # Displaying all the result on console.
    print("Principle Amount:{}".format(p))
    print("Time:{}".format(t))
    print("Rate of Interest:{}".format(r))
    print("Simple Interest on Amount:{}".format(si))
    print("Total Amount to Pay:{}".format(totalamt))


#main program

# INPUT Taking from Function Call.
# Ask the user to enter the principle amount.
p=float(input("Enter Principle Amount:"))

# Ask the user to enter time in months.
t=float(input("Enter Time in months:"))

# Ask the user to enter rate of interest.
r=float(input("Enter rate of interest:"))

simpleint(p,t,r)  # Function call
```

```
Enter Principle Amount:100000
Enter Time in months:2
Enter rate of interest:2
Principle Amount:100000.0
Time:2.0
Rate of Interest:2.0
Simple Interest on Amount:4000.0
Total Amount to Pay:104000.0
```

In [13]:
```
''' WAPP which will compute the addition of Even numbers and Odd numbers
    separately from the given list by using functions.'''


# Creating a function defination.
def sum_even_odd(lst):

    # Initialization for sum of even numbers.
    even_sum=0

    # Initialization for sum of odd numbers.
    odd_sum=0

    # Using for loop
    for i in lst:

        # If value is divided by 2 then add in even numbers.
        if (i%2==0):
            even_sum=even_sum+i

        # If value is not divided by 2 then add in odd numbers.
        else:
            odd_sum=odd_sum+i
    # printing the addition of even and odd numbers on console.
    print(even_sum,odd_sum)

#main program

# function call with user given list.
sum_even_odd([10,2,4,5,3,7,6,13,18])
```

```
40 28
```

In [12]:
```
#Program for accepting list of values and display by using Functions.

# Creating a function.
def acceptvalues():

    # creating empty list.
    lst=list()

    # Ask the user How Many Values u want place in list.
    n=int(input("Enter How Many Values u want place in list:"))

    # If user choosed less than or equal to "0" then return empty list.
    if (n<=0):
        return lst     # we are returning empty list

    # If user choosed greater than "0".
    else:
        # Using for loop to ask the user to enter the individual value.
        for i in range(1,n+1):

            # Ask the user to enter the choosed number of values.
            val=float(input("Enter {} value:".format(i)))

            # Adding those user entered values in to a list.
            lst.append(val)
        return lst           # we are returning non-empty list

#main program
lstdata=acceptvalues()        # Function Call

# If user not enterd any thing then print "List is empty".
if (len(lstdata)==0):
    print("List is Empty")

# If user enterd choosed number of values, then print the content if list.
else:
    print("Content of list={}".format(lstdata))
```

```
Enter How Many Values u want place in list:5
Enter 1 value:12
Enter 2 value:45
Enter 3 value:78.90
Enter 4 value:45
Enter 5 value:34
Content of list=[12.0, 45.0, 78.9, 45.0, 34.0]
```

In [11]:
```
#Program for accepting list of values and sum and average by using Functions.

# Creating a function.
def acceptvalues():

    # creating empty list.
    lst=list()

    # Ask the user How Many Values u want place in list.
    n=int(input("Enter How Many Values u want place in list:"))

    # If user choosed less than or equal to "0" then return empty list.
    if (n<=0):
        return lst     # we are returning empty list

    # If user choosed greater than "0".
    else:
        # Using for loop to ask the user to enter the individual value.
        for i in range(1,n+1):

            # Ask the user to enter the choosed number of values.
            val=float(input("Enter {} value:".format(i)))

            # Adding those user entered values in to a list.
            lst.append(val)
        return lst           # we are returning non-empty list

# Creating another function to find sum and average for a given list.
def findsumavg(lst):

    # Initialization for sum.
    sum=0
    for a in lst:

        # Calculating the sum of the list.
        sum=sum+a

        # Calculating the average of the list.
        avrg=sum/len(lst)

    # returning the sum and average of the list.
    return sum,avrg


#main program
lstdata=acceptvalues()     # Function Call

# If user not enterd any thing then print "List is empty".
if(len(lstdata)==0):
    print("List is Empty:")

# If user enters number of values, then print the content if list.
else:
    print("Content of List={}".format(lstdata))
    sum1,avg1=findsumavg(lstdata)        # Function Call for sum, average.
```

```
    # print the sum and average of the list.
    print("Sum({})={}".format(lstdata,sum1))
    print("Avg({})={}".format(lstdata,avg1))
```

```
Enter How Many Values u want place in list:5
Enter 1 value:12
Enter 2 value:45
Enter 3 value:23.90
Enter 4 value:45
Enter 5 value:78
Content of List=[12.0, 45.0, 23.9, 45.0, 78.0]
Sum([12.0, 45.0, 23.9, 45.0, 78.0])=203.9
Avg([12.0, 45.0, 23.9, 45.0, 78.0])=40.78
```

In [10]:
```
#Program for accepting list of values and find max and min by using Functions.

# Creating a function.
def acceptvalues():

    # creating empty list.
    lst=list()

    # Ask the user How Many Values u want place in list.
    n=int(input("Enter How Many Values u want place in list:"))

    # If user choosed less than or equal to "0" then return empty list.
    if (n<=0):
        return lst     # we are returning empty list

    # If user choosed greater than "0".
    else:
        # Using for loop to ask the user to enter the individual value.
        for i in range(1,n+1):

            # Ask the user to enter the choosed number of values.
            val=float(input("Enter {} value:".format(i)))

            # Adding those user entered values in to a list.
            lst.append(val)
        return lst           # we are returning non-empty list

# Creating another function to find max and min value for a given list.
def findmaxmin(lstdata):

    # Initializing 1st value as max value.
    maxv=lstdata[0]
    # Initializing 1st value as min value.
    minv=lstdata[0]

    for i in range(1,len(lstdata)):

        # Comparing each values and finding max value.
        if(lstdata[i]>maxv):
            maxv=lstdata[i]

        # Comparing each values and finding min value.
        if(lstdata[i]<minv):
            minv=lstdata[i]

    # returning the max and min of the list.
    return maxv,minv



#main program
lstdata=acceptvalues()     # Function Call

# If user not enterd any thing then print "List is empty".
if(len(lstdata)==0):
    print("List is Empty:")
```

```
# If user enters number of values, then print the content if list.
else:
    print("Content of List={}".format(lstdata))
    maxv,minv=findmaxmin(lstdata) # Function Call for max, min.

    # print the max and min value from the list.
    print("max({})={}".format(lstdata,maxv))
    print("minv({})={}".format(lstdata,minv))
```

```
Enter How Many Values u want place in list:4
Enter 1 value:100
Enter 2 value:200.0
Enter 3 value:300.00
Enter 4 value:400
Content of List=[100.0, 200.0, 300.0, 400.0]
max([100.0, 200.0, 300.0, 400.0])=400.0
minv([100.0, 200.0, 300.0, 400.0])=100.0
```

Types of Arguments in python

- Arguments are those, which are used in Function Calls either in the form of Variables and Values.
- Based on the Relation, arguments are 5 Types. They are

    1. Positional Arguments
    2. Default Arguments
    3. Keyword Arguments
    4. Variable Length Arguments
    5. Keyword Variable Length Arguments

- **Function with no arguments**

In [9]:
```
# Creating a function.
def even_num():

    # Taking start and stop inputs from users.
    start=int(input("Enter the start value:"))
    stop=int(input("Enter the stop value:"))

    # Using "for loop" for itaration of range of values.
    for i in range(start, stop+1):

        # # If value is divided by 2, then print those even numbers.
        if (i%2==0):
            print(i)

#main program
even_num()
```

```
Enter the start value:10
Enter the stop value:20
10
12
14
16
18
20
```

1. Positional Arguments

- The Concept of Positional Parameters (or) arguments says that "The Number of Arguments (Actual arguments) must be equal to the number of formal parameters ".
- This Parameter mechanism also recommends Order and Meaning of Parameters for Higher accuracy.
- To pass the Specific Data from Function Call to Function Definition then we must take Positional Argument Mechanism.
- The default Argument Passing Mechanism is Positional Arguments (or) Parameters.
- In Positional Arguments, the Number of Arguments (Actual arguments) is not equal to the number of formal parameters than we get TypeError.
- Positional Arguments maintains insertion order, it will take 1st argument as 1st, 2nd as 2nd and so on.

Example

In [5]:
```
# WAPP to display the student information by using Positional Arguments.

# Creating a function.
def  dispstudinfo(sno,sname,marks):

    # print the student information on console.
    print("\t{}\t{}\t{}".format(sno,sname,marks))


#main program
print("-"*50)
# printing sno, sname, marks for disply purpose.
print("\tStno\tName\t\tMarks")
print("-"*50)
dispstudinfo(10,"Prasanna",77.77)       # Function Call-1.
dispstudinfo(20,"Umamaheswari",66.66)   # Function Call-2.
```

```
--------------------------------------------------
        Stno    Name            Marks
--------------------------------------------------
        10      Prasanna        77.77
        20      Umamaheswari    66.66
```

- **In Positional Arguments, the Number of Arguments (Actual arguments) is not equal to the number of formal parameters than we get TypeError.**

In [85]:
```
# WAPP to display the student information by using Positional Arguments.

# Creating a function.
def  dispstudinfo(sno,sname,marks):

    # print the student information on console.
    print("\t{}\t{}\t{}".format(sno,sname,marks))
```

In [83]:
```
''' In this function call we are assigning two arguments but it
   accepts three arguments.'''
dispstudinfo(10,"Prasanna")      # Function Call-1 with Positional Arguments.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_6588\446586449.py in <module>
      1 ''' In this function call we are assigning two arguments but it
      2    accepts three arguments.'''
----> 3 dispstudinfo(10,"Prasanna")      # Function Call-1 with Positional Argu
ments.

TypeError: dispstudinfo() missing 1 required positional argument: 'marks'
```

In [86]:
```
''' In this function call we are assigning four arguments but it
    accepts only three arguments.'''
dispstudinfo(10,"Prasanna",56.89,"Python")      # Function Call-1.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_6588\3393230008.py in <module>
      1 ''' In this function call we are assigning four arguments but it
      2     accepts only three arguments.'''
----> 3 dispstudinfo(10,"Prasanna",56.89,"Python")      # Function Call-1.

TypeError: dispstudinfo() takes 3 positional arguments but 4 were given
```

2. Default Arguments

- When there is a Common Value for family of Function Calls then Such type of Common Value(s) must be taken as default parameter with common value (But not recommended to pass by using Positional Parameters).
- We can provide a default value to an argument by using the assignment operator(=).
- When we use default parameters in the function definition, they must be used as last Parameter(s) otherwise we get Error (SyntaxError: non-default argument (Positional) follows default argument).

In [21]:
```
# WAPP to display the student information by using Default Arguments.
# In this program "course='Python'" is a Default Argument.


# Creating a function.
def  dispstudinfo(sno,sname,marks,course="Data Science"):

    # print the student information on console.
    print("\t{}\t{}\t{}\t{}".format(sno,sname,marks,course))


#main program
print("-"*50)
# printing sno, sname, marks, course for disply purpose.
print("\tStno\tName\t\tMarks\t\tCourse")
print("-"*50)
dispstudinfo(10,"Prasanna",77.77)       # Function Call-1
dispstudinfo(20,"Umamaheswari",66.66)   # Function Call-2
```

```
--------------------------------------------------
        Stno    Name            Marks           Course
--------------------------------------------------
        10      Prasanna        77.77   Data Science
        20      Umamaheswari    66.66   Data Science
```

In [87]:
```
# WAPP to display the student information by using Default Arguments.
# In this program "course='Python'" is a Default Argument.
'''We can provide a default value to an argument by using the
   assignment operator(=).'''
# In function call-1, we are assigning course value as 'Python'.


# Creating a function.
def  dispstudinfo(sno,sname,marks,course="Data Science"):

    # print the student information on console.
    print("\t{}\t{}\t{}\t{}".format(sno,sname,marks,course))


#main program
print("-"*50)
# printing sno, sname, marks, course for disply purpose.
print("\tStno\tName\t\tMarks\t\tCourse")
print("-"*50)
dispstudinfo(10,"Prasanna",77.77,"Python")       # Function Call-1
dispstudinfo(20,"Umamaheswari",66.66)   # Function Call-2
```

```
--------------------------------------------------
        Stno    Name            Marks           Course
--------------------------------------------------
        10      Prasanna        77.77   Python
        20      Umamaheswari    66.66   Data Science
```

- **When we use default parameters in the function definition, they must be used as last Parameter(s) otherwise we get Error (SyntaxError: non-default argument (Positional) follows default argument).**

In [24]:
```
# WAPP to display the student information by using Default Arguments.
# In this program "course='Python'" is a Default Argument.
# In this program default argument is assigned in between.

# Creating a function.
def  dispstudinfo(sno,sname,course="Data Science",marks,):

    # print the student information on console.
    print("\t{}\t{}\t{}\t{}".format(sno,sname,marks,course))


#main program
print("-"*50)
# printing sno, sname, marks, course for disply purpose.
print("\tStno\tName\t\tMarks\t\tCourse")
print("-"*50)
dispstudinfo(10,"Prasanna",77.77)       # Function Call-1
dispstudinfo(20,"Umamaheswari",66.66)   # Function Call-2
```

```
  File "C:\Users\Administrator\AppData\Local\Temp\ipykernel_6588\4043455161.py"
, line 6
    def  dispstudinfo(sno,sname,course="Data Science",marks,):
                                                            ^
SyntaxError: non-default argument follows default argument
```

3. Keyword Arguments

- In some of the circumstances, we know the function name and formal parameter names and we don't know the order of formal Parameter names and to pass the data / values accurately we must use the concept of Keyword Parameters (or) arguments.
- The implementation of Keyword Parameters (or) arguments says that all the formal parameter names used as arguments in Function call(s) as keys.
- If we want to assign the positional arguments in keyword argument, we must ensure that it should be used before keyword
  arguments otherwise we get SyntaxError.
- Keyword Arguments maintains insertion order, it will take 1st argument as 1st, 2nd as 2nd and so on.

In [26]:
```
#Program for demonstrating Keyword arguments

def  disp(a,b,c):
    print("\t{}\t{}\t{}".format(a,b,c))
```

In [27]:
```
# calling the function with keyword arguments.
disp(a=10,b=30,c="Ram")   # Function call-1
disp(c="Ram",a=10,b=30,)   # Function call-2
disp(a=10,c="Ram",b=30)   # Function call-3
```

```
        10      30      Ram
        10      30      Ram
        10      30      Ram
```

- **If we want to assign the positional arguments in keyword argument, we must ensure that it should be used before keyword arguments otherwise we get SyntaxError.**

In [28]:
```
#Program for demonstrating Keyword arguments

def  disp(a,b,c):
    print("\t{}\t{}\t{}".format(a,b,c))
```

In [30]:
```
# calling the function with Positional and keyword arguments.
disp(10,b=30,c="Ram")   # Function call-1
disp(100,300,c="Ram")   # Function call-2
```

```
        10      30      Ram
        100     300     Ram
```

In [31]:
```
# calling the function with Positional and keyword arguments.
disp(a=10,"Hari",c="Ram")   # Function call-1
```

```
  File "C:\Users\Administrator\AppData\Local\Temp\ipykernel_6588\3592847165.py"
, line 2
    disp(a=10,"Hari",c="Ram")   # Function call-1
                             ^
SyntaxError: positional argument follows keyword argument
```

In [32]:
```
# calling the function with Positional and keyword arguments.
disp(a=10,b=20,1000)   # Function call-1
```

```
  File "C:\Users\Administrator\AppData\Local\Temp\ipykernel_6588\742091981.py",
line 2
    disp(a=10,b=20,1000)   # Function call-1
                         ^
SyntaxError: positional argument follows keyword argument
```

4. Variable Length Arguments (Arbitary Arguments)

- When we have family of multiple function calls with Variable number of values / arguments then with normal python programming, we must define multiple function definitions. This process leads to more development time. To overcome this process, we must use the concept of Variable length Parameters.

- To Implement, Variable length Parameters concept, we must define single Function Definition and takes a formal Parameter preceded with a symbol called asterisk (* param) and the formal parameter with asterisk symbol is called Variable length Parameters and whose purpose is to hold / store any number of values coming from similar function calls and whose type is <class, 'tuple'>.
- Rule: - The *param must always written at last part of Function Heading and it must be only one (but not multiple).
- Rule: - When we use Variable length and default parameters in function Heading, we use default parameter as last and before we use variable length parameter and in function calls, we should not use default parameter as Key word argument bcoz Variable number of values are treated as Positional Argument Value(s).

In [33]:
```
#Program for demonstrating Variable length arguments.

def  dispvalues( *names ):  # Here *names is called Variable Length Parameter.
    print(names,type(names))
```

In [35]:
```
# Calling the function with Positional arguments.
dispvalues(10) # Function Call-1
dispvalues(10,20) # Function Call-2
dispvalues(10,20,30) # Function Call-3
```

```
(10,) <class 'tuple'>
(10, 20) <class 'tuple'>
(10, 20, 30) <class 'tuple'>
```

In [ ]:
```
# Program for demonstrating Variable length arguments.
# Here we are using positional arguments with Variable length arguments.

def  dispvalues(sno,name, *names ):
    print(names,city)
```

In [55]:
```
dispvalues(10,"Hari",67.89) # Function Call-1
dispvalues(10,20,"Ram") # Function Call-2
dispvalues(10,20,30,56.78,"Kiran","DS") # Function Call-3
```

```
(10, 'Hari', 67.89) Hyd
(10, 20, 'Ram') Hyd
(10, 20, 30, 56.78, 'Kiran', 'DS') Hyd
```

- **If we are trying to assign positional arguments after vareable length arguments, then we get TypeError.**

In [56]:
```
# Program for demonstrating Variable length arguments.
# Here we are using positional arguments after Variable length arguments.

def  dispvalues( *names,sno,name, ):
    print(names,city)
```

In [58]:
```
dispvalues(10,"Hari",67.89,23,100) # Function Call-1
dispvalues(10,20,"Ram") # Function Call-2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_6588\643787400.py in <module>
----> 1 dispvalues(10,"Hari",67.89,23,100) # Function Call-1
      2 dispvalues(10,20,"Ram") # Function Call-2

TypeError: dispvalues() missing 2 required keyword-only arguments: 'sno' and 'n
ame'
```

In [53]:
```
# Program for demonstrating Variable length arguments.
# Here we are using default arguments with Variable length arguments.

def  dispvalues( *names,city="Hyd" ):
    print(names,city)
```

In [54]:
```
dispvalues(10) # Function Call-1
dispvalues(10,20) # Function Call-2
dispvalues(10,20,30) # Function Call-3
```

```
(10,) Hyd
(10, 20) Hyd
(10, 20, 30) Hyd
```

- **If we are trying to assign positional arguments after default arguments and before variable arguments, then we get SyntaxError.**

In [67]:
```
# Program for demonstrating Variable length arguments.
# Here we are using default arguments before Variable length arguments.

def  dispvalues(city="Hyd",sno,*names):
       print(city,sno,names)


dispvalues(10) # Function Call-1
dispvalues(10,20) # Function Call-2
dispvalues(10,20,30) # Function Call-3
```

```
  File "C:\Users\Administrator\AppData\Local\Temp\ipykernel_6588\2046617696.py"
, line 4
    def  dispvalues(city="Hyd",sno,*names):
                                   ^
SyntaxError: non-default argument follows default argument
```

Variable length arguments precedence

- Positional arguments.
- Veriable length arguments.
- Default arguments.

5. Keyword Variable Length Arguments

- When we have family of multiple function calls with Key Word Variable number of values / arguments then with normal python programming, we must define multiple function definitions. This process leads to more development time. To overcome this process, we must use the concept of Keyword Variable length Parameters.
- To Implement, Keyword Variable Length Parameters concept, we must define single Function Definition and takes a formal Parameter preceded with a symbol called double asterisk (** param) and the formal parameter with double asterisk symbol is called Keyword Variable length Parameters and whose purpose is to hold / store any number of (Key, Value) coming from similar function calls and whose type is <class, 'dict'>.
- Rule: - The **param must always written at last part of Function Heading and it must be only one (but not multiple).

Keyword variable length arguments precedence :

- Positional Arguments.
- Veriable length arguments.
- Default arguments.
- Keyword variable length arguments.

In [73]:
```
# Program for demonstrating Keyword Var length arguments.

# Creating function with Keyword Var length arguments.
def  dispinfo(**k):
    #Here  **k is called Keyword Var length param and its type <class,'dict'>

    # print the k and type of k.
    print(k,type(k))

#main program
dispinfo(eno=10, ename="RS", sal=3.4, dsg="Author") # Function Call-1
dispinfo(sno=20, sname="TR", marks=33.33) # Function Call-2
```

```
{'eno': 10, 'ename': 'RS', 'sal': 3.4, 'dsg': 'Author'} <class 'dict'>
{'sno': 20, 'sname': 'TR', 'marks': 33.33} <class 'dict'>
```

In [76]:
```
#Program for demonstrating Keyword Var length arguments

# Creating function with Keyword Var length arguments.
def  dispinfo(**k):

    print("-"*50)                       # Disply purpose

    # using for loop to extract the key,value from given data.
    for key,val in k.items():
        print("\t{}\t{}".format(key,val))

    print("-"*50)                       # Disply purpose

#main program
dispinfo(eno=10,ename="RS",sal=3.4,dsg="Author") # Function Call-1
dispinfo(sno=20,sname="TR",marks=33.33) # Function Call-2
dispinfo(tno=100,tname="KN") # Function Call-3
```

```
--------------------------------------------------
        eno     10
        ename   RS
        sal     3.4
        dsg     Author
--------------------------------------------------
--------------------------------------------------
        sno     20
        sname   TR
        marks   33.33
--------------------------------------------------
--------------------------------------------------
        tno     100
        tname   KN
--------------------------------------------------
```

In [77]:
```
# Program for demonstrating Keyword Var length arguments.
# Using positional arguments before Keyword Var length arguments.

# Creating function with Keyword Var length arguments.
def  findtotalmarks(sno,sname,cls, **kvr  ):

    print("-"*50)                                # Disply purpose.

    print("Roll Number:{}".format(sno))        # print the roll no.

    print("Name of Student:{}".format(sname)) # print the student name.

    print("Class Name:{}".format(cls))         # print the class name.

    print("-"*50)                                # Disply purpose.

    # print "Subject Name and Subject Marks" separately for disply purpose.
    print("\tSubject Name\tSubject Marks")
    print("-"*50)                                # Disply purpose.

    # Initialization total marks as "Zero".
    totmarks=0

    # Use "for loop" to disply key, value pair from the dict.
    for Subject,Marks in kvr.items():

        # printing keys as subject names and values as marks.
        print("\t{}\t\t{}".format(Subject,Marks))

        # Addition of Total marks.
        totmarks=totmarks+Marks

    print("-"*50)                                 # Disply purpose.

    # Print the total marks on console.
    print("\tTotal Marks:{}".format(totmarks))

    print("-"*50)                                 # Disply purpose.



#main program
# function call-1
findtotalmarks(10,"Rossum","X", Eng=50,Telugu=60,Hindi=55,Maths=70,Sci=67,Soc=80)

# function call-2
findtotalmarks(20,"Rakesh","XII",Maths=70,Phy=58,Che=57)

# function call-3
findtotalmarks(30,"Ramesh","B.Tech", C=50,CPP=60,OS=55,DBMS=70,DS=67)

# function call-4
findtotalmarks(40,"Travis","Research")
```

```
--------------------------------------------------
Roll Number:10
```

```
Name of Student:Rossum
Class Name:X
--------------------------------------------------
        Subject Name    Subject Marks
--------------------------------------------------
        Eng             50
        Telugu          60
        Hindi           55
        Maths           70
        Sci             67
        Soc             80
--------------------------------------------------
        Total Marks:382
--------------------------------------------------
--------------------------------------------------
Roll Number:20
Name of Student:Rakesh
Class Name:XII
--------------------------------------------------
        Subject Name    Subject Marks
--------------------------------------------------
        Maths           70
        Phy             58
        Che             57
--------------------------------------------------
        Total Marks:185
--------------------------------------------------
--------------------------------------------------
Roll Number:30
Name of Student:Ramesh
Class Name:B.Tech
--------------------------------------------------
        Subject Name    Subject Marks
--------------------------------------------------
        C               50
        CPP             60
        OS              55
        DBMS            70
        DS              67
--------------------------------------------------
        Total Marks:302
--------------------------------------------------
--------------------------------------------------
Roll Number:40
Name of Student:Travis
Class Name:Research
--------------------------------------------------
        Subject Name    Subject Marks
--------------------------------------------------
--------------------------------------------------
        Total Marks:0
--------------------------------------------------
```

Recussive Function or Recurison

- We know that in Python, a function can call other functions. It is even possible for the function to call itself. These type of construct are termed as "Recussive or Recurison Function".

In [80]:
```
''' WAPP which will find the factorial of the given number by using
    Recussive Function.'''

# Defining function to find the factorial.
def fact(num):

    # If num is less than or equal to "0", then return "1".
    if num<=0:
        return 1

    # If num is greater than "0", then return "num*fact(num-1)".
    else:
        return num*fact(num-1)

#main program
# call the function with value, it will find the factorial of that value.
fact(5)      # function call with (5).
```

Out[80]:
```
120
```