FUNCTIONS

- If a group of statements is repeatedly required then it is not recommended to write these statements every time separately have to define these statements as a single unit and we can call that unit any number of times based on our requirement without rewriting. This unit is nothing but function.

- The main advantage of functions is code Reusability.

- Note: In other languages functions are known as methods, procedures, subroutines etc

- Python supports 2 types of functions

1) **Built in Functions**

2) **User Defined Functions**

1) Built in Functions

The functions which are coming along with Python software automatically, are called built in functions or pre-defined functions.

Eg: id()

type()

input()

eval()

etc..

```
>>> for e in __builtins__.__dict__:
...         print(e)
```

2) User Defined Functions

The functions which are developed by programmer explicitly according to business requirements, are called user defined functions.

Syntax to Create User defined Functions:

```
def function_name(parameters):
""" doc string"""
----
-----
return value
```

Note: While creating functions we can use 2 keywords

1) **def (mandatory)**

2) **return (optional)**

Eg 1: Write a function to print Hello

```
def wish():

print("Hello Good Morning")

wish()
```

Parameters

Parameters are inputs to the function. If a function contains parameters, then at the time of calling, compulsory we should provide values otherwise, otherwise we will get error.

Eg: Write a function to take name of the student as input and print wish message by name.

```
def wish(name):

  print("Hello",name," Good Morning")


wish("Sai")

wish("Ravi")
```

Eg: Write a function to take number as input and print its square value

```
def squareIt(number):

   print("The Square of",number,"is", number*number)

squareIt(4)

squareIt(5)
```

Return Statement

Function can take input values as parameters and executes business logic, and returns output to the caller with return statement.

Q) Write a Function to accept 2 Numbers as Input and return Sum

```
def add(x,y):

  return x+y

result=add(10,20)

print("The sum is",result)

("The sum is",add(100,200))
```

If we are not writing return statement then default return value is None.

```
def f1():

  print("Hello")

f1()

print(f1())
```

Returning Multiple Values from a Function

In other languages like C, C++ and Java, function can return atmost one value. But in Python, a function can return any number of values.

Eg 1:

```
def sum_sub(a,b):
  sum=a+b
  sub=a-b
  return sum,sub


x,y=sum_sub(100,50)

print("The Sum is :",x)

print("The Subtraction is :",y)
```

Types of Arguments

def f1(a,b):

------

f1(10,20)

a, b are formal arguments whereas 10,20 are actual arguments.

There are 4 types are actual arguments are allowed in Python.

1) **Positional Arguments**

2) **Keyword Arguments**

3) **Default Arguments**

4) **Variable Length Arguments**

1) **Positional Arguments:**

These are the arguments passed to function in correct positional order.

```
def sub(a, b):

print(a-b)

sub(100, 200)

sub(200, 100)
```

The number of arguments and position of arguments must be matched. If we change the order then result may be changed.

If we change the number of arguments then we will get error.

2) **Keyword Arguments:**

We can pass argument values by keyword i.e by parameter name.

```
def wish(name,msg):

  print("Hello",name,msg)

wish(name="Sai",msg="Good Morning")

wish(msg="Good Morning",name="Durga")
```

Output

Hello Sai Good Morning

Hello Durga Good Morning

Here the order of arguments is not important, but number of arguments must be matched.

Note: We can use both positional and keyword arguments simultaneously. But first we have to take positional arguments and then keyword arguments, otherwise we will get syntax error.

1) def wish(name,msg):

2) print("Hello",name,msg)

3) wish("Durga","GoodMorning") -> Valid

4) wish("Durga",msg="GoodMorning") → Valid

5) wish(name="Durga","GoodMorning") → Invalid

SyntaxError: positional argument follows keyword argument

3) Default Arguments

Sometimes we can provide default values for our positional arguments.

```
def wish(name="Guest"):
  print("Hello",name,"Good Morning")
```

3) **wish("Sai")**

4) **wish()**

Output

Hello Sai Good Morning

Hello Guest Good Morning

If we are not passing any name then only default value will be considered.

***Note:

After default arguments we should not take non default arguments.

1) **def wish(name="Guest",msg="Good Morning")**: → Valid

2) **def wish(name,msg="Good Morning")**: → Valid

3) **def wish(name="Guest",msg)**: → Invalid

SyntaxError: non-default argument follows default argument

4) Variable Length Arguments

Sometimes we can pass variable number of arguments to our function, such type of arguments is called variable length arguments.

We can declare a variable length argument with * symbol as follows

def f1(*n):

We can call this function by passing any number of arguments including zero number.

Internally all these values represented in the form of tuple.

1)
```
def sum(*n):
     total=0
     for n1 in n:
       total=total+n1
     print("The Sum=",total)


sum()

sum(10)

sum(10,20)

sum(10,20,30,40)
```

Note: We can mix variable length arguments with positional arguments.

1)
```
def f1(n1,*s):
      print(n1)
      for s1 in s:
        print(s1)
```

```
f1(10)

f1(10,20,30,40)

f1(10,"A",30,"B")
```

Note: After variable length argument, if we are taking any other arguments then we should provide values as keyword arguments.

```
def f1(*s,n1):

  for s1 in s:

    print(s1)

  print(n1)

f1("A","B",n1=10)
```