Functions

- Whenever you want execute a logic more than one time either in the same program or in the different then the corresponding logic must be define in separate named block is called “Function”
- “Function” is self-contained block which is having a statement or group of statements to perform particular task
- Functions are classified into 2 types
  - Predefined Functions
    - The functions which are provided by the Python interpreter then those function are called predefined functions also known built-in function
      - Eg: print( ) | type( ) | id( ) | exit( ) …
  - Userdefined Function
    - The Functions which are defined by us for our application requirements called user defined functions also known as customized function
- Advantages :
  - Code reusability
  - Code optimization
  - Memory Optimization
  - Efficiency of the program will be increased
  - Debugging is becomes easy
    - It is the process of executing the program in our style in-order to identify the mistakes in the code

Shashi Kumar - SSST

```
def <functionName>([list of args]):
....Local variables
....Statement(s)
....[return  <variable | exp | value>]
```

Based in the application requirements we can define the function in the following ways

1. **Function without arguments and without return value**
2. **Function Without arguments and with return value**
3. **Function with arguments without return value**
4. **Function with arguments with return value**
5. **Function with arguments with multiple returns**

Example:

```
def greetings():
    print("Hello ")
    print("MyDear")
    print("Have a nice Day...!!!")
    print("-"*30)

#calling
greetings()
greetings()
greetings()
```

Shashi Kumar - SSST

Example 2:

```
def myAdd(x,y):  #x,y are formal parameters
    s=x+y
    print("Result is : ",s)

#calling
myAdd(10,20)   #10,20 are actual arguments
myAdd(5+5+5,10*3)  #45
a=90
b=90
myAdd(a,b)
```

Note :

- Formal parameter are the parameter which are declared in the function definition
- Formal parameter are copies of actual arguments
- Formal parameters will works as local variable
- Actual arguments the arguments which we are passing by the time of calling function
- Actual arguments may be the values | variables | expressions
- Memory locations are different for both formal and actual arguments

Shashi Kumar - SSSIT

Example 3:

```
def sq(x):      # sq(x) -> int
   s=x*x
   return s

def cu(x):   #cu(x) -> int
   c=x*x*x
   return c

#calling
r=sq(2)
print("Result is : ",r) #NameError

a=cu(2)
print("Result is : ",a)
```

Note:

- Whenever you want use the value calculated by function from outside of that function then we have to define function with return

Shashi Kumar - SSSIT

Example :

```
def findAreaOfCircle(rad):
    area=3.14*rad*rad
    return area

#calling
#findAreaOfCircle(rad) -> float

r=int(input("Enter Radius of Circle "))
ac=findAreaOfCircle(r)
print("Area of Circle is : ",ac)

#Example with multiple Return values
def myCalc(x,y):    # myCalc(x,y) -> int,int,int
    a=x+y  #12
    s=x-y   #  8
    m=x*y # 20
    return a,s,m

#calling
i,j,k=myCalc(10,2)
print("Add of Two is : ",i)
print("Sub of two is : ",j)
print("Mut of two is : ",k)
```

Shashi Kumar - SSSIT

Note:

- **If a function returns more than one value , we can store them into a variable, but that variables will be acts a tuple Collection**

```
def myCalc(x,y):    # myCalc(x,y) -> values
   a=x+y  #12
   s=x-y   #  8
   m=x*y # 20
   return a,s,m

#calling
t=myCalc(10,2)
print("Type is : ",type(t))
print("Result is : ",t)
print("Mul is : ",t[2])
```

Shashi Kumar - SSSIT