Unit – III

Function:-

Functions are the most important aspect of an application. A function can be defined as the organized block of reusable code which can be called whenever required.

Python allows us to divide a large program into the basic building blocks known as function. The function contains the set of programming statements enclosed by {}. A function can be called multiple times to provide reusability and modularity to the python program.

In other words, we can say that the collection of functions creates a program. The function is also known as procedure or subroutine in other programming languages.

Python provide us various inbuilt functions like range() or print(). Although, the user can create its functions which can be called user-defined functions.

Need of function:-

- By using functions, we can avoid rewriting same logic/code again and again in a program.
- We can call python functions any number of times in a program and from any place in a program.
- We can track a large python program easily when it is divided into multiple functions.
- Reusability is the main achievement of python functions.
- However, Function calling is always overhead in a python program.

Defining a Function

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

- Function blocks begin with the keyword **def** followed by the function name and parentheses (( )).
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or *doc string*.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Syntax:-

```
def  function-name(parameter):
        Instruction to be processed.
        Return statement
```

Examples

Ex 1
```
def hello_world():
        print("hello world")

# Now you can call printme function
hello_world()
```

Ex 2
```
#defining the function
def func (name):
        print("Hi ",name);

#calling the function
func("Ayush")
```

Ex 3
```
def sum (a,b):
        return a+b;

#taking values from the user
a = int(input("Enter a: "))
b = int(input("Enter b: "))

#printing the sum of a and b
print("Sum = ",sum(a,b))
```

Ex 4
```
def printme( str ):
        print str
        return;

# Now you can call printme function
printme("first call to user defined function!")
printme(" second call to the same function")
```

- Parameter part in function is optional it may or may not be used.
- All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function
- There is an exception in the case of mutable objects since the changes made to the mutable objects like string do not revert to the original string rather, a new string object is made, and therefore the two different objects are printed.

Ex 1
```
#defining the function  immutable items
def change_list(list1):
   list1.append(20);
   list1.append(30);
   print("list inside function = ",list1)

#defining the list
list1 = [10,30,40,50]

#calling the function
change_list(list1);
print("list outside function = ",list1);
```

Ex 2
```
#defining the function  mutable objects like (string)
def change_string (str):
   str = str + " Hows you";
   print("printing the string inside function
```

```
    :",str);

    string1 = "Hi I am there"

    #calling the function
    change_string(string1)

    print("printing the string outside function :",string1)
```

Types of arguments

We can pass different types of arguments at the time of function calling.

- Required arguments
- Keyword arguments
- Default arguments
- Variable-length arguments

Required Argument:-

Argument is provided at the time of function calling. As far as the required arguments are concerned, these are the arguments which are required to be passed at the time of function calling with the exact match of their positions in the function call and function definition. If either of the arguments is not provided in the function call, or the position of the arguments is changed, then the python interpreter will show the error.

Example 1

```
#the argument name is the required argument to the function func
def func(name):
   message = "Hi "+name;
   return message;

name = input("Enter the name?")
print(func(name))
```

Output:

```
Enter the name?gauraw
Hi gauraw
```

Example 2

```
#the function simple_interest accepts three arguments and returns the simple interest accordingly
def simple_interest(p,t,r):
        return (p*t*r)/100

p = float(input("Enter the principle amount? "))
r = float(input("Enter the rate of interest? "))
t = float(input("Enter the time in years? "))
```

```
print("Simple Interest: ",simple_interest(p,r,t))

Output:
Enter the principle amount? 10000
Enter the rate of interest? 5
Enter the time in years? 2
Simple Interest:  1000.0
```

Example 3

```
#the function calculate returns the sum of two arguments a and b
def calculate(a,b):
   return a+b

calculate(10) # this causes an error as we are missing a required arguments b.

Output:
TypeError: calculate() missing 1 required positional argument: 'b'
```

Keyword arguments:-

Python allows us to call the function with the keyword arguments. This kind of function call will enable us to pass the arguments in the random order.
The name of the arguments is treated as the keywords and matched in the function calling and definition. If the same match is found, the values of the arguments are copied in the function definition.

Example 1

```
#function func is called with the name and message as the keyword arguments
def func(name,message):
        print("printing the message with",name,"and ",message)
func(name = "John",message="hello")

Output:
printing the message with John and  hello
```

Example 2

providing the values in different order at the calling

```
#The function simple_interest(p, t, r) is called with the keyword arguments the order of arguments doesn't matter in this case
def simple_interest(p,t,r):
   return (p*t*r)/100
```

```
print("Simple Interest: ",simple_interest(t=10,r=10,p=1900))
```

Output:
Simple Interest: 1900.0

If we provide the different name of arguments at the time of function call, an error will be thrown.

Example 3

```
#The function simple_interest(p, t, r) is called with the keyword arguments.
def simple_interest(p,t,r):
   return (p*t*r)/100

print("Simple Interest: ",simple_interest(time=10,rate=10,principle=1900))  # doesn't find the exact match of the name of the arguments (keywords)
```

Output:

TypeError: simple_interest() got an unexpected keyword argument 'time'

The python allows us to provide the mix of the required arguments and keyword arguments at the time of function call. However, the required argument must not be given after the keyword argument, i.e., once the keyword argument is encountered in the function call, the following arguments must also be the keyword arguments.

Example 4

```
def func(name1,message,name2):
   print("printing the message with",name1,",",message,",and",name2)
func("John",message="hello",name2="David") #the first argument is not the keyword argument
```

Output:
printing the message with John , hello ,and David

The following example will cause an error due to an in-proper mix of keyword and required arguments being passed in the function call.

Example 5

```
def func(name1,message,name2):
   print("printing the message with",name1,",",message,",and",name2)
func("John",message="hello","David")
```

Output:
SyntaxError: positional argument follows keyword argument

Default Arguments:-

Python allows us to initialize the arguments at the function definition. If the value of any of the argument is not provided at the time of function call, then that argument can be initialized with the value given in the definition even if the argument is not specified at the function call.

Example 1

```
def printme(name,age=22):
   print("My name is",name,"and age is",age)
printme(name = "john") #the variable age is not passed into the function however the default value of age is considered in the function

Output:
My name is john and age is 22
```

Example 2

```
def printme(name,age=22):
   print("My name is",name,"and age is",age)
printme(name = "john") #the variable age is not passed into the function however the default value of age is considered in the function
printme(age = 10,name="David") #the value of age is overwritten here, 10 will be printed as age

Output:
My name is john and age is 22
My name is David and age is 10
```

Variable length Arguments:-

Sometimes we may not know the number of arguments to be passed in advance. In such cases, Python provides us the flexibility to provide the comma separated values which are internally treated as tuples at the function call.

However, at the function definition, we have to define the variable with * (star) as *<variable - name >.

Example

```
def printme(*names):
   print("type of passed argument is ",type(names))
   print("printing the passed arguments...")
   for name in names:
      print(name)
```

```
printme("john","David","smith","nick")

Output:
type of passed argument is  <class 'tuple'>
printing the passed arguments...
john
David
smith
```

Scope of variables:-

The scopes of the variables depend upon the location where the variable is being declared. The variable declared in one part of the program may not be accessible to the other parts.

Global variables
Local variables

The variable defined outside any function is known to have a global scope whereas the variable defined inside a function is known to have a local scope.

Example 1

```
def print_message():
   message = "hello !! I am going to print a message." # the variable message is local to the function itself
   print(message)
print_message()
print(message) # this will cause an error since a local variable cannot be accessible here.
Output:

hello !! I am going to print a message.
  File "/root/PycharmProjects/PythonTest/Test1.py", line 5, in
   print(message)
NameError: name 'message' is not defined
```

Example 2

```
def calculate(*args):
   sum=0
   for arg in args:
      sum = sum +arg
   print("The sum is",sum)
sum=0
calculate(10,20,30) #60 will be printed as the sum
print("Value of sum outside the function:",sum) # 0 will be printed
```

Output:

The sum is 60
Value of sum outside the function: 0

The Anonymous function (lambda):-

- The anonymous function contains a small piece of code
- These functions are called anonymous because they are not declared in the standard manner by using the def keyword. You can use the lambda keyword to create small anonymous functions.
- Lambda forms can take any number of arguments.
- Return just one value in the form of an expression.
- They cannot contain commands or multiple expressions.
- An anonymous function cannot be a direct call to print because lambda requires an expression
- Lambda functions have their own local namespace and cannot access variables other than those in their parameter list and those in the global namespace.
- Although it appears that lambda's are a one-line version of a function, they are not equivalent to inline statements in C or C++, whose purpose is by passing function stack allocation during invocation for performance reasons.

Syntax:-

lambda arguments : expression

E.g.

```
x = lambda a:a+10
print("sum = ",x(20))
```

```
x = lambda a,b:a+b
print("sum = ",x(20,10))
```

Why to use lambda

When we want to execute some task with in function with some input which will be processing parameter passed to function.
E.g.

```
def myfunc(n):
  return lambda a : a * n

x = myfunc(2)          // will execute first fun with parameter 2 then it passed on to lambda with x i.e. 11
print(x(11))
```

Python modules

- A module allows you to logically organize your Python code.
- Grouping related code into a module makes the code easier to understand and use.
- A module is a Python object with arbitrarily named attributes that you can bind and reference.
- Module is a file consisting of Python code.
- Module can define functions, classes and variables. A module can also include runnable code.

Import statement

- The import statement is used to import all the functionality of one module into another

- We can use the functionality of any python source file by importing that file as the module into another python source file.
- We can import multiple modules with a single import statement.

Syntax:-

import module1,module2,..... module n
ex. import numpy

From-import statement

- Instead of importing the whole module into the namespace, python provides the flexibility to import only the specific attributes of a module.
- This can be done by using from? import statement. The syntax to use the from-import statement is given below.

Syntax:-
from < module-name> **import** <name 1>, <name 2>..,<name n>

calculation.py:

```
#place the code in the calculation.py
def summation(a,b):
   return a+b
def multiplication(a,b):
   return a*b;
def divide(a,b):
```

```
    return a/b;
```

Main.py:

```
from calculation import summation
#it will import only the summation() from calculation.py
a = int(input("Enter the first number"))
b = int(input("Enter the second number"))
print("Sum = ",summation(a,b))
```

Renaming import file

```
import <module-name> as <specific-name>
```

Python packages

The packages in python facilitate the developer with the application development environment by providing a hierarchical directory structure where a package contains sub-packages, modules, and sub-modules. The packages are used to categorize the application level code efficiently.