UNIT: III	Functions and Modules	Lectures: 08 Hrs

- **Need for functions:**
- **Function:** definition, call, variable scope and lifetime, the return statement.
- Defining functions, Lambda or anonymous function,
- documentation string,
- good programming practices.
- **Introduction to modules**,
- **Introduction to packages in Python**,
- **Introduction to standard library modules**.

Function

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

Defining a Function

•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 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.

Function

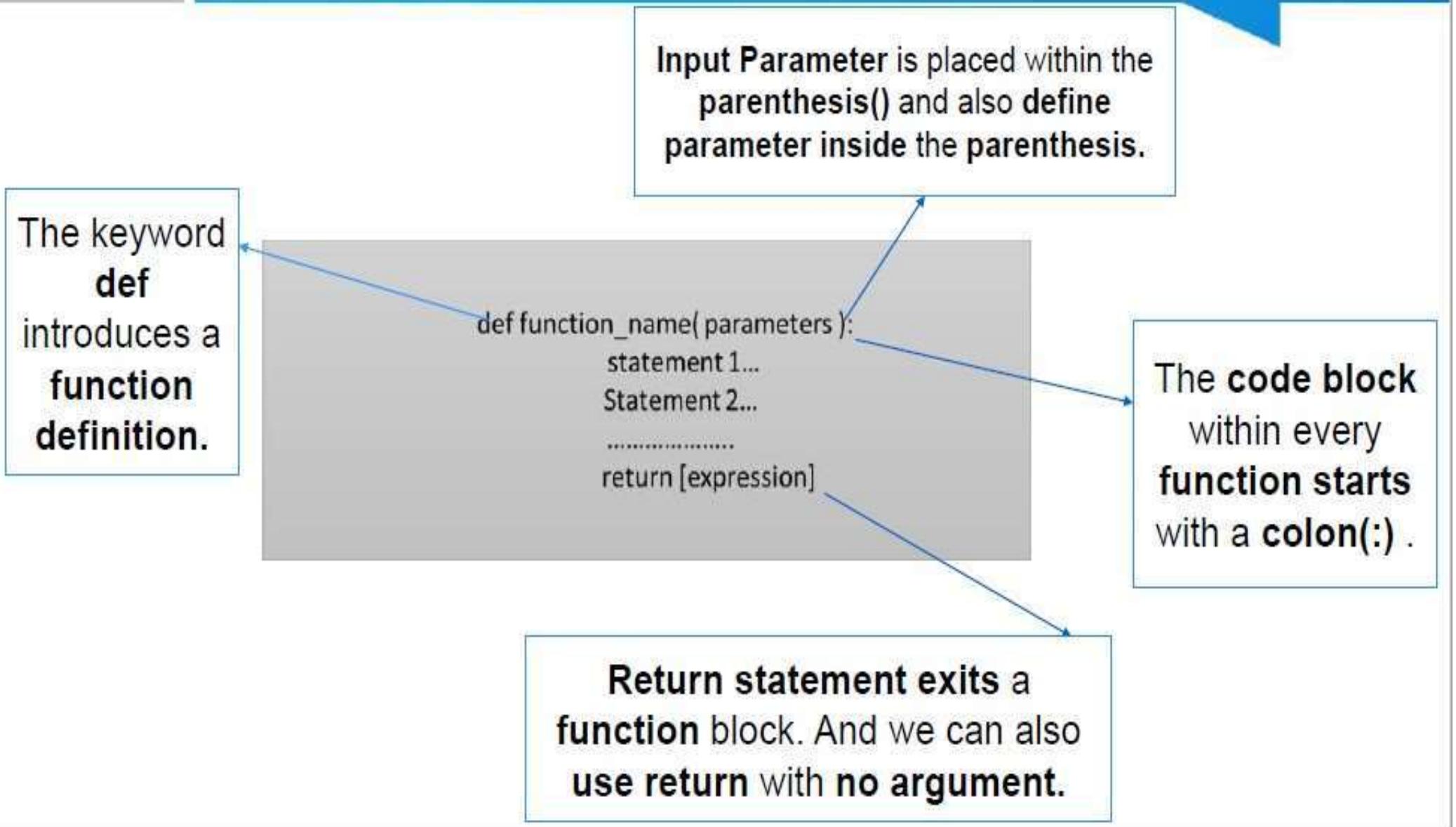
Syntax:

def function_name (parameter) :

def marks the start of function
Function_name to uniquely identify a function.

Argument to pass a value in function
colon(:) to mark end of function header

Function Syntax



Function Type

THERE ARE TWO TYPES OF FUNCTION IN PYTHON

BUILT-IN FUNCTION:

Example: **print(), input(), eval().**

USERDEFINE FUNCTION:

Example: **def** my_addition(x,y):

sum = x + y

return sum

THERE ARE 68 BUILT-IN FUNCTION VERSION 3.4.

PASSING ARGUMENTS TO FUNCTIONS

- In programming, there are two ways in which arguments can be passed to functions :-
  - **Pass by Value**:
    - Function creates a copy of the variable(Object in Python) passed to it as an argument.
    - The actual object is not affected.
    - Object is of **immutable type,**because immutable objects cannot be modified.
  - **Pass by Reference**:
    - The actual object is passed to the called function.
    - All the changes made to the object inside the function affect its original value.
    - Object is **mutable type**, as mutable objects can be changed, the passed objects are updated.

Example

```
# Defining function print_str(str1)
''' This function prints the string being passed as an
argument '''
def print_str(str1):
    print(str1)
    return

# Calling user-defined function print_str(str1)
print_str("Calling the user defined function")
```

Function is printing the argument 'str1'

Function Call

Return Statement

It is statement to return from a function to its previous function who called this function.

After return control goes out of the current function.

All local variables which were allocated memory in current function will be destroyed.

Return statement is optional in python.

Any function can return multiple arguments.

Return Statement

Example

```
return              #This returns none value
return None         #This returns none value
return a, b         #This returns two values
return a            #This returns single value
```

These all are valid examples of a return statement.

- Position
- Keyword
- Default
- Variable length
- lambda

DEFAULT ARGUMENT VALUES

- **Default Argument-** argument that **assumes a default value** if a **value is not provided** in the **function call** for that argument.
- The **default value** is **evaluated only once.**
- **EXAMPLE:**

```
def function1( a,b=90 ):
#"This prints a passed info into this function"
    print ("value of a: ", a)
    print ("value of b", b)
    return
# call function1 function
function1(a="Monika", b=50)
function1(a="Lalit")
```

In this code, argument '**b**' has given a **default value. ie(b=90)**

When the **value of 'b'** is **not passed in function** ,then it takes **default argument.**

Output:

```
value of a:  Monika
value of b 50
value of a:  Lalit
value of b 90
```

DEFAULT ARGUMENTS

- The **default value** is **evaluated only once**. This makes a difference when the **default is a mutable object** such as **a list, dictionary, or instances of most classes.**
- **Example:**

```
def function1(b, L=[]):
    L.append(b)
    return L
print(function1('Devashish'))
print(function1('gunjan'))
print(function1('Monika'))
print(function1('Lalit'))
print(function1('Purva'))
```

Output:

```
['Devashish']
['Devashish', 'gunjan']
['Devashish', 'gunjan', 'Monika']
['Devashish', 'gunjan', 'Monika', 'Lalit']
['Devashish', 'gunjan', 'Monika', 'Lalit', 'Purva']
```

- if we **don't want the default value** to be **shared** between **subsequent calls**, then
- **Example :**

```
def function1(b, L=None):
    if L is None:
        L = []
    L.append(b)
    return L
print(function1('lm'))
print(function1('mm'))
```

Output:

```
['lm']
['mm']
```

KEYWORD ARGUMENTS

- **Keyword arguments** are related to the **function calls**.
- Using **keyword arguments** in a **function call**, the **caller identifies** the **arguments by the parameter name**.
- Allows to **skip arguments** or **place them out of order**, the **Python interpreter use the keywords** provided to **match the values with parameters.**
- **Example:**

```
def function1( date,year ):
    print ("date: ", date)
    print ("year: ", year)
    return
function1(year=1975, date=20)
```

- Caller **identifies** the **keyword argument** by the **parameter name.**
- Calling of the **argument Order** doesn't matter .

Output:

```
date:  20
year:  1975
```

KEYWORD ARGUMENTS

❑ Example 1:

```
def function2(date,year=1989,name="Monika",title="Singh"):
    print(date,year,name,title)
#calling of function function2
function2(22)
function2(date=19)
function2(date=15,name="Purva")
function2(24,1897,'Gunjan','Singh')
```

Output:

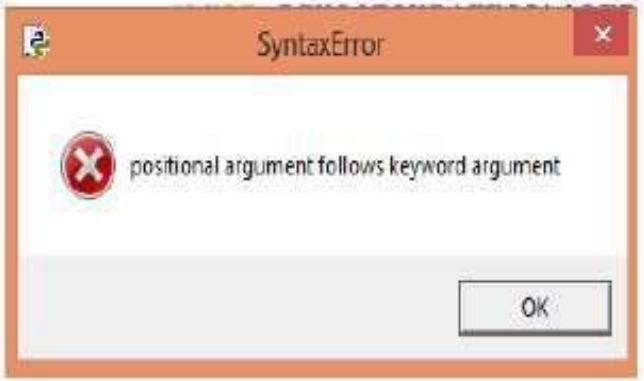
```
22 1989 Monika Singh
19 1989 Monika Singh
15 1989 Purva Singh
24 1897 Gunjan Singh
```

❑ Example 2:

```
def function2(date,year=1989,name="Monika",title="Singh"):
    print(date,year,name,title)

function2(date=19,23) #non keyword argument after keyword argument
function2()  #required argument missing
function2(age=45) #unknown keyword argument
function2('25',date=31) #duplicate value for the same argument
```

Output:



Anonymous Functions / Lambda Functions

Functions containing only single operation can be converted into an anonymous function.

'Lambda' is the keyword used to create such anonymous functions.

Syntax

Lambda < space > <parameter> : < operation >

Example

```
my_addition = lambda x, y : x + y
print("addition is ", my_addition(20, 30))
```

Output = 50

LAMBDA EXPRESSIONS

- **Small functions** can be **created** with the **lambda keyword** also known as **Anonymous function**.
- Used wherever **functions object** are **required**.
- These **functions** are called **anonymous** because they are **not declared** in the **standard manner** by using the **def keyword.** .
- **Syntax :**

```
lambda [arg1 [,arg2,.....argn]]:expression
```

- **EXAMPLE:**

```
def function1(n):
    return lambda x: x*n
a=function1(3)
print(a(4))
```

- The function returns the multiply of two arguments.
- Lambda function is restricted to a single expressions.

Output: 12

Variable Scope and Lifetime

1. In functions, there are two kinds of variables, local and global.

Local Variables / Objects :

2. Variables or objects which are used only within given function are local variables or local objects.

3. Local objects include parameters and created in a given function.

Example

In following code, mult() function has two local variables

Local variable ‘a’ and local variable ‘b’.

Global variables / objects

1. **Objects which can be accessed throughout the script/program are global variables or objects.**
2. **Global variables or objects are created in python script outside any function.**
3. **Global objects are available after “global” keyword defined in the script.**

Global variables / objects

1. **Reading Global variable value**

Example

In following example global variable is accessed for printing / reading purpose.

No modification to global variable is done here.

Global variables / objects

```
#No modification in global variable id made
def add_gv(a,b):
    c=a+b+gv
    print("in function value of gv is =",gv)
    print("The addition is :",c)

gv=100
print("The initial value of gv =", gv)
add_gv(10,20)
print("After function value of gv =",gv)
```

Global variables / objects

Modification of Global Variable Value

‘global’ keyword is used to modify a global variable inside a function.

Example

In following example “global” keyword is used inside the function.

Now global variable can be modified within the function.

Modifications made in the function (after using “global”) will stay after the function as well.

Global variables / objects

```
#Modification in global variable is made
def add_gv(a,b):
    global gv
    gv=150
    print(gv)
    c=a+b+gv
    return c
    gv=100
print(gv)
x=add_gv(10,20)
print(x)
print(gv)
```

Documentation String

In python, programmer can write a documentation for every function.

This documentation can be accessed by other functions.

Advantage of Document string

It is useful when we want to know about any function in python.

Programmer can simply print the document string of that function and can know what that function does.

Documentation String

```
def func( ):
    """Welcome to Coulomb"""
    return
print(func.__doc__)
```

Standard Libraries in Python

1. **Math (import math)**

- **This is a package for providing various functionalities regarding mathematical operations.**

2. **Random (import random)**

- **This is the module which supports various functions for generation of random numbers and setting seed of random number generator.**

3. **Numpy (import numpy)**

- **This is a package in python which supports various numeric operations. It supports multidimensional arrays or matrices and their calculations.**

4. **Scipy (import scipy)**

- **This is the package for various scientific computations.**

Introduction to Modules

- **Modules make python programs re-usable.**
- **Every python code (.py) file can be treated as a module.**
- **A module can be accessed in other module using import statement.**
- **A single module can have multiple functions or classes.**
- **Each function or class can be accessed separately in import statement.**

Introduction to Modules

Example to create your own module

- **Create a file named sample.py in your directory.**
- **Write function add() in it. (as we have seen in previous sections)**
- **Now create another file trial.py in same directory**
- **In trial.py write**
- **import sample.add**

  print(“addition is “, sample.add(10,20))
- **Now run trial.py.**
- **Now the output will be 30.**