Unit 3

Functions and Modules

Syllabus :

- 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.

Need of Functions :

- Functions are very important part of programming language.
- They help program to be modular. Means they help in writing small parts of a program which are meaningful.
- These small parts (modules) can be used again and again at different places in a program.
- **Example :**
- Max( ) is a function to find max element from given list.
- It can be used again and again.
- Print( ) is a most commonly used function.

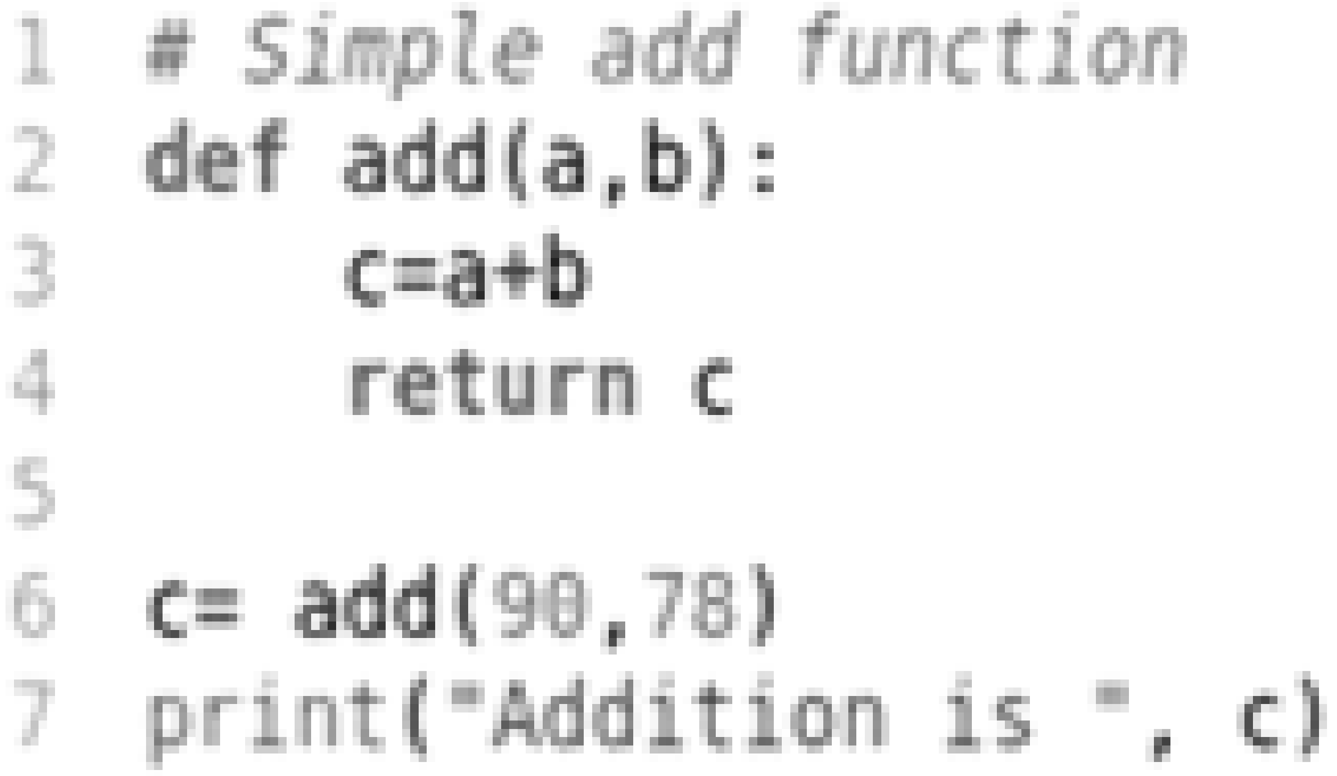
Defining functions :

- When new function is to be used we need to first define it.
- Local variables or objects cannot be accessed outside a function.
- **Note :** Function name cannot contain spaces or special characters except underscore (–).

Syntax :

- def<space><function_name>(<parameters>):
- <tab> ……
- <tab> ……
- <tab> return <variables to be returned>

Program : Write a program to add two numbers using a function :

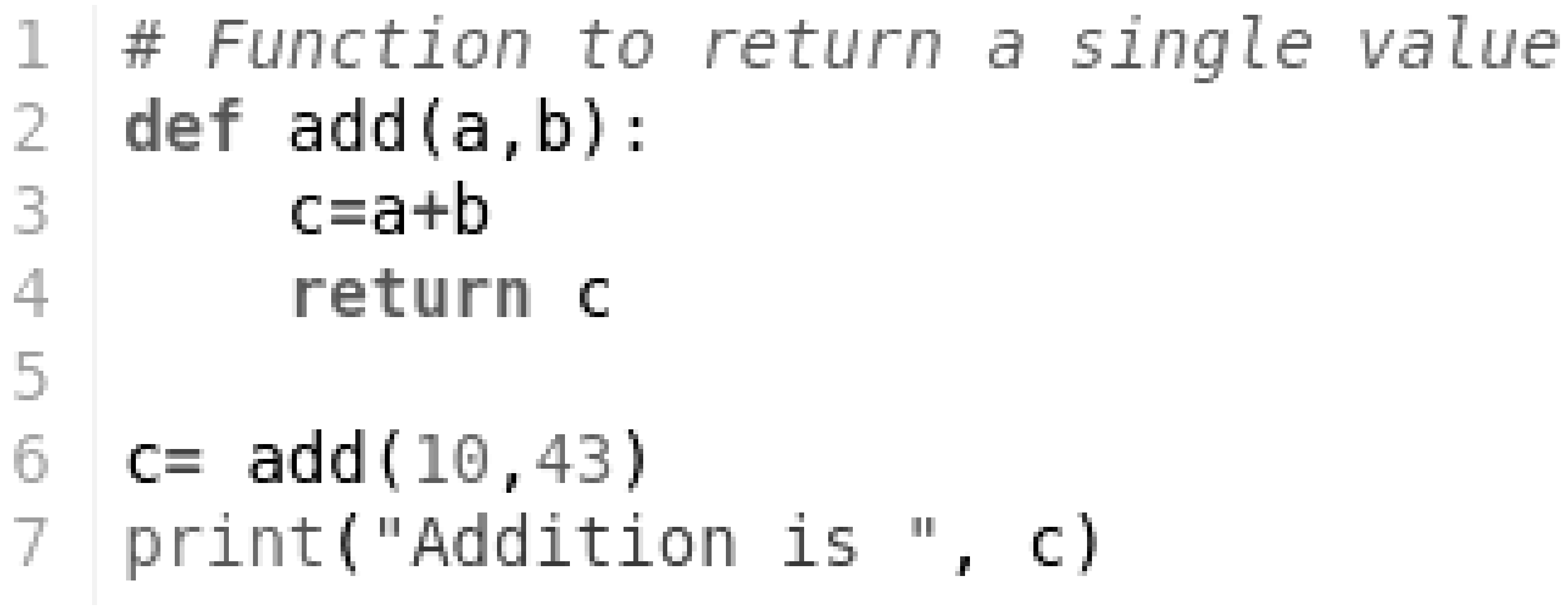


```python
# Simple add function
def add(a,b):
        c=a+b
        return c

c= add(90,78)
print("Addition is ", c)
```

Call to a function :

- A function can be called from any place in python script.
- Place or line of call of function should be after place or line of declaration of function.
- Example: In following code, add function is called on line 6.
- Add() function definition start on line 2 and ends on line 4.
- Then add() is called on line 6.

Program :



```
# Function to return a single value
def add(a,b):
    c=a+b
    return c

c= add(10,43)
print("Addition is ", c)
```

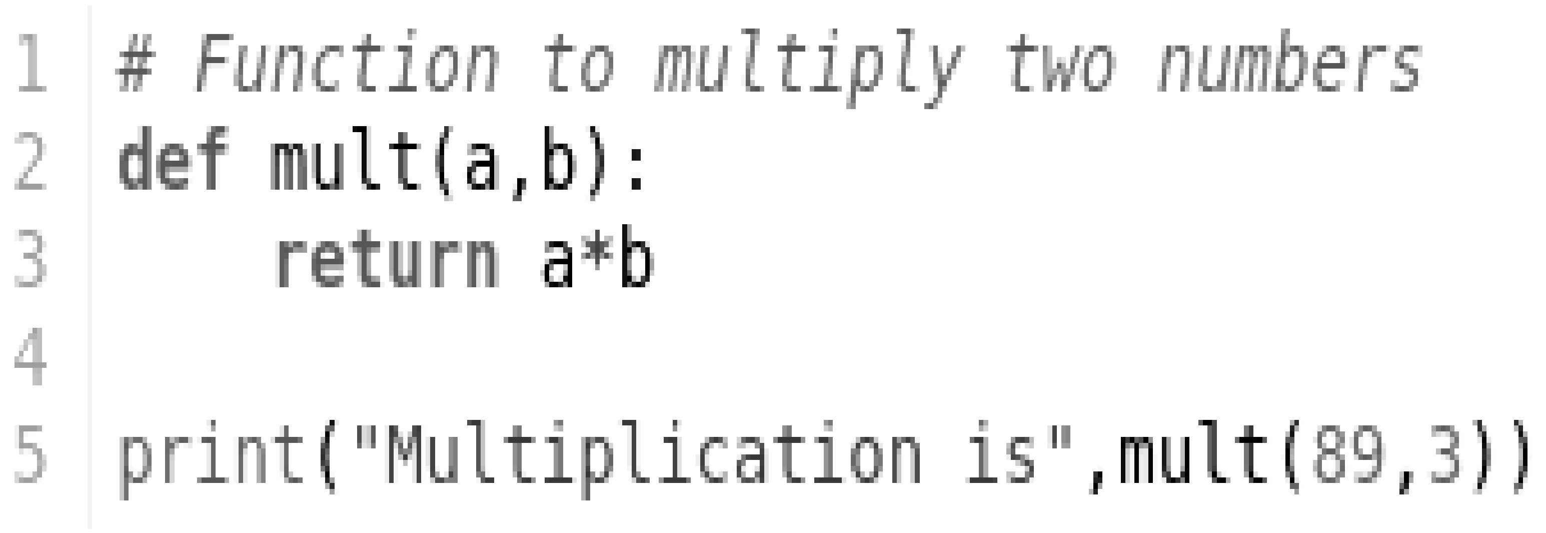
Variable Scope and Lifetime :

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

Local Variables / Objects :

- Variables or objects which are used **only** within given function are local variables or local objects.
- Local objects include parameters and any variable / object which is created in a given function.
- **Example**
- In following code, mult() function has two local variables
- Local variable a and local variable b.

Program :



```python
# Function to multiply two numbers
def mult(a,b):
    return a*b

print("Multiplication is",mult(89,3))
```

Global variables / objects :

- Objects which can be accessed throughout the script/program are global variables or objects.
- Global variables or objects are created in python script outside any function.
- 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)
```

Arguments to a Function :

- A function may accept arguments or it may not accept any arguments or parameters.
- Arguments or Parameters to a function are treated as local variable for that function.
- While defining the function, number of parameters has to be specified as sequence of variables.

Types of Arguments :

There are different types of arguments :

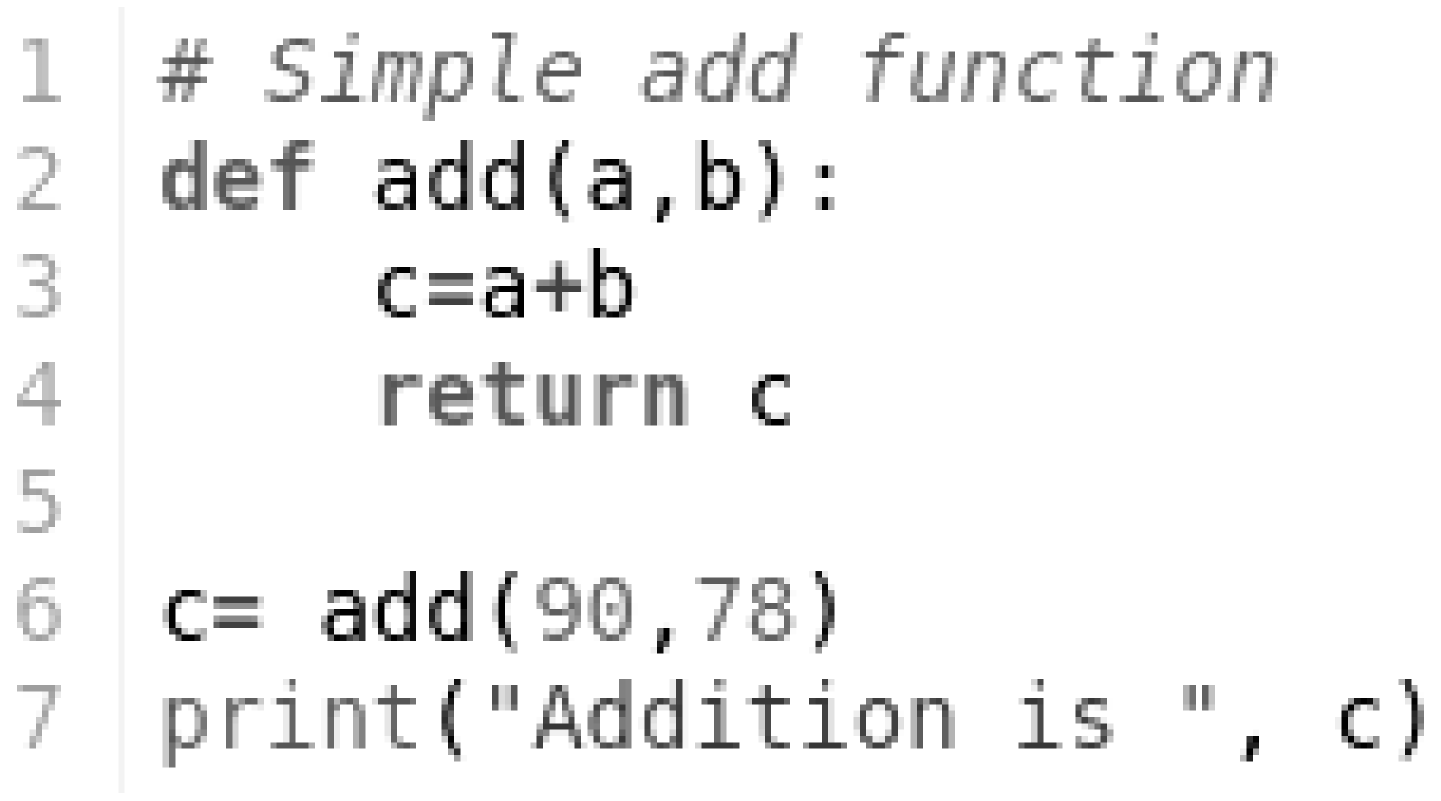
- Positional Arguments
- Default Arguments
- Unlimited-Positional Arguments
- Keyword Arguments

Types of Arguments :

Positional Arguments :

- These arguments are passed to function based on their position
- Any normal arguments are positional arguments
- Example. In following example add functional takes two positional arguments a and b.
- When function is called add(90,78) then arguments are assigned by their position.
- First position is of a so value 90 will be assigned to variable a.
- Second position is of b so value 78 will be assigned to variable b.

Example of Positional Arguments :



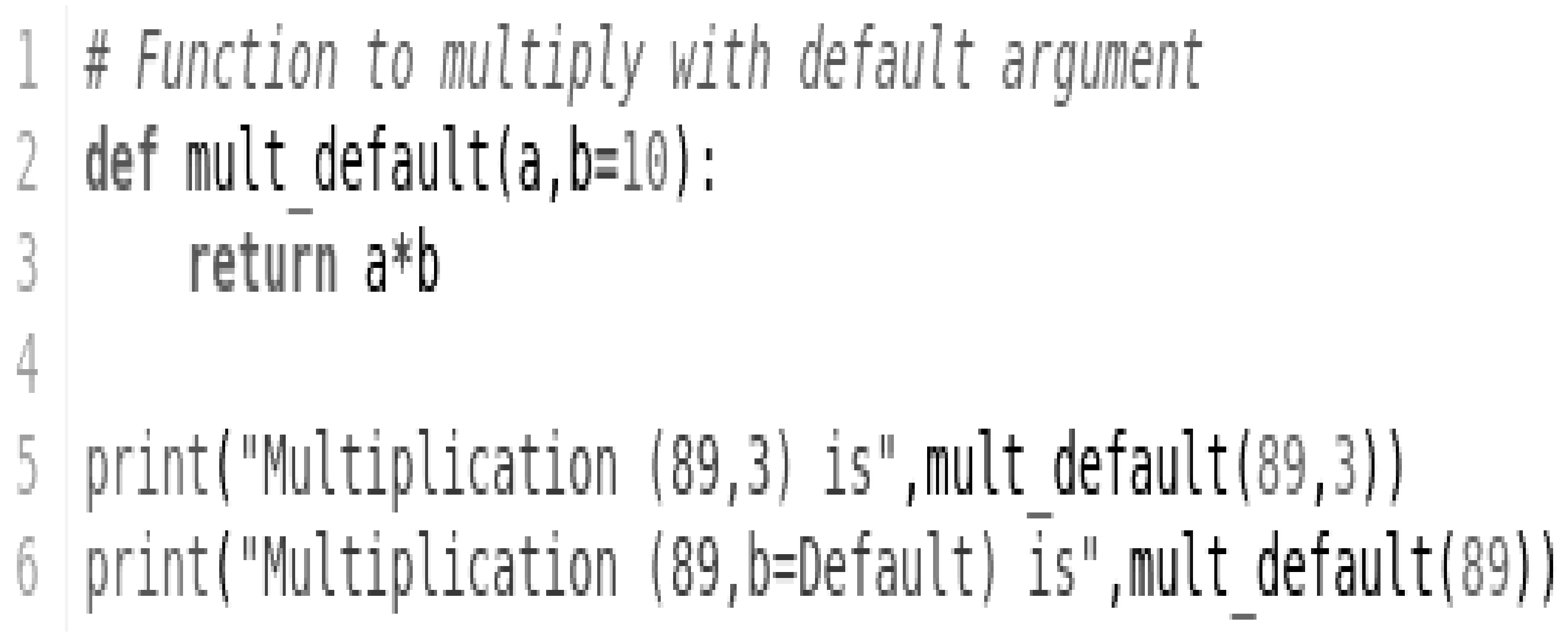
```python
# Simple add function
def add(a,b):
    c=a+b
    return c

c= add(90,78)
print("Addition is ", c)
```

Types of Arguments :

Default Arguments

- One of the argument to a function may have its default value.
- For example laptop has default built-in speakers. So if no speaker is connected it will play default speaker.
- Similarly in function argument, a default value can be assigned to an argument.
- Now if value for this argument is not passed by the user then function will consider that arguments default value.
- Calling functions with very large number of arguments can be made easy by default values
- For example, in following code mult_default function takes b argument as default.
- So, even if value of b is not passed, then default value of b will be 10.
- It is clear from the result that call mult_default(89) results in 890.
- Means a=89 and b = 10. So result= 89 * 10 = 890

Example of Default Arguments :



```
# Function to multiply with default argument
def mult_default(a,b=10):
    return a*b

print("Multiplication (89,3) is",mult_default(89,3))
print("Multiplication (89,b=Default) is",mult_default(89))
```

Types of Arguments :

Unlimited Positional Arguments

- Some functions can have some compulsory arguments and after that there can be any number of arguments.
- Example is print().
- In print() function we can pass any number of strings separated by comma.
- And all strings will get printed.
- So, programmer can also create such a function taking unlimited arguments.

Types of Arguments :

Keyword Arguments

- These are another special category of arguments supported in python.
- Here arguments are passed in format “key=value”
- All key-word arguments can be taken in a special variable with **.

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.

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

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 Example:

```
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.