DEPARTMENT OF COMPUTER SCIENCE

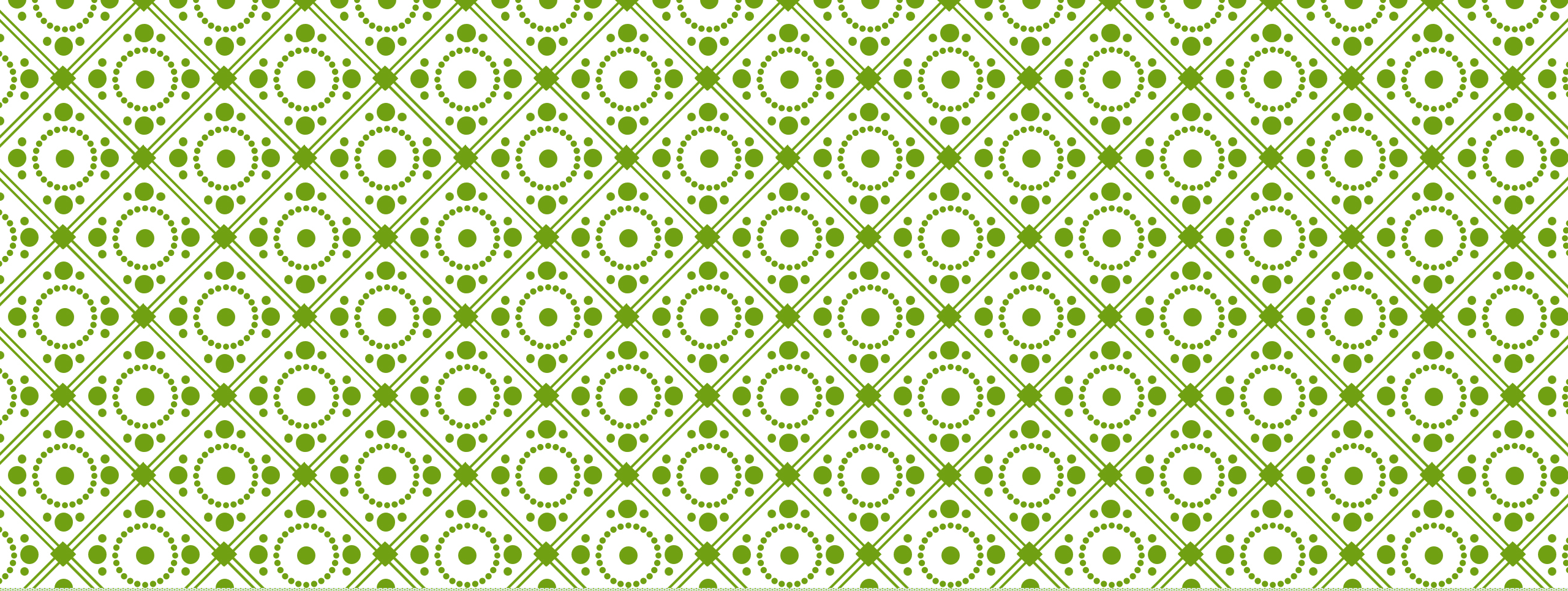
Semester : 3rd

Paper Code : SEC-1

Paper Name: Programming in Python

Topic: Function in Python

Unit: 5 (Creating Python Programs)



FUNCTION IN PYTHON

Monalisa Sardar
Asst. Prof

WHAT IS A FUNCTION?

- Function is a set group of related statements that performs a specific task.
- It help break our program into smaller and modular chunks.
- It avoids repetition and makes the code reusable.

DEFINE A FUNCTION

The keyword '**def**' is used to define a function in Python.

The name of the function must be unique (same rules which are used to define a variable).

Function docstring is used to describe the function.

Multiple arguments can be passed using "**,**".

'**:**" is used at the end of function header.

"**return**" statement is used to return a value from a function.

DEFINE A FUNCTION CONTD.

```
def function_name (Arg1, Arg2,.....Arg n):
    " docstring "
      statement 1
      statement 2
      statement 3
      return statement
```

FUNCTION CALL

1. function_name(arg1, arg2,…..arg n)

Eg: def names (arg1, arg2, arg3):

Call: names(‘Ram’, ‘Shyam’, ‘Sita’)

2. function_name ()

Eg: def names ():

Call: names()

EXAMPLE OF FUNCTION

```
def animal (arg1, arg2):
        print(arg1 + “and” + arg2)
```

Call a function : animal (‘Dogs’, ‘Cats’)

Output:

Dogs and Cats

FLOW OF EXECUTION

```
def subtract ( x, y):
    s = x – y
    return s
a= 10
b= 20
sum = a+b
diff = subtract(a,b)
print (“Sum is “ , sum)
print (“Difference is “ , diff)
```

Output:

Sum is 30

Difference is -10

FUNCTION ARGUMENTS

In Python arguments can be passed to the functions in following ways

1. Positional Arguments
2. Keyword Arguments
3. Default Arguments

POSITIONAL ARGUMENTS

Positional arguments are the arguments passed to a function in correct positional order.

The number of arguments in the function call should match exactly with the function definition.

```
def person (name, age):
    print(name, "and" , age)
```

Call a function : person ('Ram', 25) Output: Ram and 25

Call a function : person ('Ram') Output: Error (1 argument required)

Call a function : person () Output: Error

KEYWORD ARGUMENTS

When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.

This allows you to place them out of order.

Arguments can be passed as: parameter_name = value

```
def person (name, age):
    print(name , "and" , age)
```

Call: person (age=25, name = 'Ram') Output: Ram and 25

DEFAULT ARGUMENT

A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.

```
def person (name, age=25):
    print(name , "and" , age)
```

Call: person ('Ram', 35) Output: Ram and 35

Call: person ('Ram') Output: Ram and 25

MIXING POSITIONAL & KEYWORD ARGUMENT

We can also mix positional arguments and keyword arguments in a function call.

The only requirement is that positional arguments must appear before any keyword arguments.

```
def person (name, age, address ):
    print(name , "and" , age , "and", address)
```

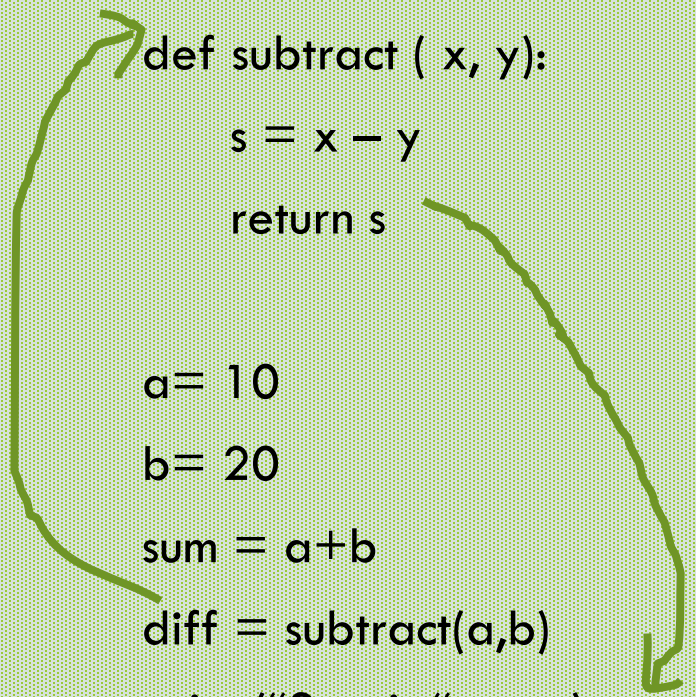
Call: person('Ram', address = 'Kolkata', age =25) Output: Ram and 25 and Kolkata

Call: person (name='Ram', 25, address='Kolkata') Output: Error

FLOW OF EXECUTION

```
def subtract ( x, y):
    s = x – y
    return s


a= 10
b= 20
sum = a+b
diff = subtract(a,b)
print (“Sum is “ , sum)
print (“Difference is “ , diff)
```



Output:

Sum is 30

Difference is -10

THE RETURN STATEMENT

The "**return**" statement is used to exit a function and go back to the place from where it was called.

Syntax of return

return [value/ expression]

The statements after the return statement are not executed.

It is not mandatory for a function to return something.

If the return statement is without any expression then a special value None is returned.

EXAMPLE

```
def abs_value(num):
    if num > 0:
        return num
    else:
        return –num
print(abs_value(2))
print(abs_value(-55))
```

OUTPUT:

2

55

SCOPE OF VARIABLES

The scope of a variable refers to the part of the program where it can be accessed or recognized.

There are two types of variables based on scope

1. Local variable
2. Global variable

LOCAL VARIABLES

The variable we create inside the function are called local variables.

Local variables can only be accessed inside the body of the function in which it is defined.

As soon as the function ends the local variable is rendered as garbage value.

Trying to access a local variable from outside the function results in error.

EXAMPLE

```
def test():
    num = 20
    print("Value inside function =", num)
test()
print("Value outside function =", num)
```

OUTPUT:

Value inside function = 20

Error (Variable not defined)

EXAMPLE

```
def test():
    num = 20
    print("Value inside function =", num)
test()
num = 30
print("Value outside function=", num)
```

OUTPUT:

Value inside function = 20

Value outside function = 30

EXAMPLE

```
def test1():
        num= 20
        print("Value in test1()=", num)

def test2():
        num= 40
        print("Value in test2()=", num)
test1()
test2()
```

OUTPUT:

Value in test1() = 20
Value in test2() = 40

GLOBAL VARIABLES

The Global variable are variables that are defined outside of any functions.

The scope of global variable starts from the point they are defined and continues on until the program ends.

When we define a variable outside of a function, it is global by default.

They can be accessed from anywhere in the program.

They can be modified inside a function only using "**global**" keyword.

EXAMPLE

```
num =10
a=10
b=20
def test():
        print("Value of num inside function=", num)
        c = a + b
        print("Value of c =", c)
 test()
```

OUTPUT:

Value of num inside function = 10

Value of c = 30

EXAMPLE

```
num =10
a=10
b=20
def test():
        num = num + 100
        print("Value of num inside function=", num)
        c = a + b
        print("Value of c =", c)
 test()
```

OUTPUT:

Error

EXAMPLE

```
num =10
a=10
b=20
def test():
        global num
        num = num + 100
        print("Value of num inside function=", num)
        c = a + b
        print("Value of c =", c)
 test()
Print("Value of num outside function=", num)
```

OUTPUT:

Value of num inside function= 110
Value of c = 30
Value of num outside function=110