

Unit - III

- Function :- A function can be defined as the organized block of reusable code which can be called whenever required.
- Divide a large portion program into the basic building blocks known as function.
- The function contains the set of programming statements enclosed by {}
- The functions is also known as procedure in other programming languages.

Collection of function creates a program

- Need of function :- can be called multiple times.
- By using functions, we can avoid rewriting same code again in a program.
- We can call python functions any number of times and from any place in program.
- We can track a large python program easily when it is divided into multiple functions

• Defining a function :-

- Function blocks begin with the keyword def followed by the function name and parentheses ().
- Any input parameters or argument should be placed within these parentheses.
- The first statement of a function can be an optional statement.
- The code block within every functions starts with a colon (:) and is indented.
- A return statement with no argument is the same as return None.

Syntax :-

def function-name (parameter):

Instruction to be processed

Return statement

Example:-

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

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

```
③ def sum(a,b):  
    return a+b;  
a=int(input("Enter a:"))  
b=int(input("Enter b:"))  
print("Sum=", sum(a,b))
```

```
④ def printme(str):  
    print str  
    return;  
printme("first call to user defined function!")  
printme("second call to the same function")
```

```
⑤ def func(name, message):  
    print("printing the message with", name, "and", message)  
func(name="John", message="hello")
```

O/P →

printing the message with John and hello.

* To create a function we use def keyword followed by the function_name()

∴ def function_name():

Types of Arguments

① Required Arguments :- used at time of function calling.

these are the arguments which are required to be passed at time of function calling with exact match of positions

```
def func(name):
```

O/P

```
    message = "Hi" + name;
```

Enter the name? Any an

```
    return message;
```

Hi Anyan

```
name = input("Enter name?")
```

```
print(func(name))
```

② Keyword Arguments :-

call function with keywords.

argument = keywords

with random order function call.

Keywords are matched in function call and replaced in function definition.

```
def func(name, message):
```

```
    print("printing message with", name, "and", message)
```

```
func(name = "John", message = "hello")
```

--- keywords

O/P

printing message with John and hello

③ Default Arguments :- if value of any argument is not provided at time of function call, then that argument can be initialized with the value given in the definition.

```
def printme(name, age = 22):
```

.... default Argument

```
    print("My name is", name, "and age is", age)
```

```
printme(name = "john")
```

age is considered in function

O/P

My name is john and age is 22.

④ Variable length Arguments:-

We may not know the no. of arguments to pass.
in such cases python provides comma separated values
(tuples) at function call.

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

print me ("john", "David", "Smith", "nick") *variable*

O/P:-

type of passed arguments is <class 'tuple'>
printing the passed arguments:-
john
David
Smith

* Scope of variables:-

depend upon the location where the variable is being declared.

Global variables
Local variables

* Scope of Variables :- depends upon the location where the variable is being declared. variable declared in one part of program may not be accessible to other parts

LOCAL VARIABLE

① declared inside a function of computer program

② Accessible only within the function in the functions program

③ Created \rightarrow function starts executing
destroyed \rightarrow execution complete

④ More reliable and secure
value cannot be changed

GLOBAL VARIABLE

① declared outside a function of computer program

② Accessible by all functions in the functions program

③ Remain forever entire time in the program is executing

④ value can be changed

• Anonymous Function (lambda) :-

- contains small piece of code.
- called anonymous because they are not declared in a standard manner by using def keyword.
- Can use lambda keyword to create small anonymous fun^{cn}.
- can take any no. of arguments.
- Returns just one value in the form of expression.
- can't contain multiple expressions.
- can't be a direct call to print.
- have their own local namespace

Syntax :- lambda arguments : expression

ex.

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

O/P \Rightarrow 30

* The Anonymous function (lambda):-

- The anonymous function contains a small piece of code.
- 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.
- Lambda requires an expression.
- Lambda functions have their own local namespace.
- cannot access variables.

Syntax:

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

```
① def myfunc(n):
    return lambda a: a * n
```

```
x = myfunc(2)
```

```
11
```

```
print(x(11))
```

* Python Module :-

- allows you to logically organize your python code.
- Grouping code into module makes code easier to understand and use.
- module is a python object.
- Module is a file consisting of python code.
- can define functions, classes, variables.
- used to import functionality of one module into other.
- Instead of importing the whole module into namespace python provides flexibility to import only the specific attributes of a module.
- using from-import statement

• Python packages:-

- Facilitate the developer with application development environment by providing a hierarchical dictionary
- packages contains sub-packages, module and sub-modules
- used to categorize the application level code efficiently

• class account:

```
def __init__(self, account no, balance):
```

```
    self.account no = account no.
```

```
    self.balance = balance
```

```
def withdraw (self, amount)
```

```
    self.balance = balance - amount
```

```
    print("Amount of", self.balance)
```

```
acc1 = account (1234, 5000)
```

```
acc1.withdraw (500)
```

O/P

⇒ 4500

• class vehicle:

```
def __init__(self, model, colour)
```

```
    self.model = model
```

```
    self.colour = colour
```

```
def show (self)
```

```
    print (self.colour)
```

```
    print (self.model)
```

```
veh = vehicle ("A234", "orange")
```

```
veh.show()
```

O/P ⇒ orange

⇒ A234

Learning functions

```
def greet():  
    print("Hello")  
    print("How do you do?")
```

```
greet()
```

• Fun

can also n no. of times

O/P

Hello

How do you do?

John & Graham in
their class

Function with Arguments

```
def greet(name):  
    print("Hello", name)  
    print("How do you do?")
```

```
greet("Jack")
```

• For name = Jack

O/P

Hello Jack

How do you do?

```
def add-numbers(n1, n2):  
    result = n1 + n2  
    print("The sum is", result)
```

number 1 = 5.4

number 2 = 6.7

```
add-numbers(number 1, number 2)
```

O/P → The sum is 12.1

```
def add-numbers(n1, n2):  
    result = n1 + n2  
    return result
```

number 1 = 5.4

number 2 = 6.7

```
result = add-numbers(number 1, number 2)
```

```
print("The sum is", result)
```



```
def greet():  
    print("Hello")  
    return  
    print("How do you do?")  
greet("Jack")
```

Q/P →

Hello Jack

→ return statement terminates program