



---

# **Subject: Python Programming**

**Module Number: 2.1**

**Module Name: Advanced Concepts**

**Version Code: PP4, PP5**  
**Released Date: 4-OCT-2019**

## Syllabus:

- **Functions Functions, the Building Blocks of Code:** Use of functions, Scopes and name resolution, Input parameters, Return values, Recursive functions, Anonymous functions, Function attributes, Built-in functions, Importing objects.
- **Saving Time and Memory:** map, zip, and filter, Comprehensions, Generators, Some performance considerations, Name localization, and Generation behavior in built-ins.
- **Advanced Concepts – OOP, Decorators, and Iterators:** Decorators, Class and object namespaces, Attribute shadowing, Initializing an instance, Accessing a base class, Multiple inheritance, Static and class methods, Private methods and name mangling, The property decorator, Operator overloading, Polymorphism

## AIM:

To understand advance concepts in python like function and Object oriented concepts



## Objectives:

The Objectives of this module is to:

- To understand and implement Function in Python
- Comprehension, Generator, map, filter and zip in python
- To understand and implement OOP concepts, Decorators, and Iterators

## Outcomes:

At the end of this module, you are expected to:

- Write functions in the python program
- Understand the concept of Comprehension, Generators, map, filter and zip in python
- Implementing OOPs and Decorators, and Iterators concepts while writing Python Program

## Contents

- Functions in Python



## Introduction to Python 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.
- As you already know, Python gives you many built-in functions like `print()`, etc. but you can also create your own functions. These functions are called user-defined functions.
- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.

## Benefits of Functions

Functions are used for the following reasons as listed below:-

1. They reduce the code duplication in a program by performing specific ask when it is being called.
2. They help in splitting a complex task or procedure in to smaller blocks
3. They improve traceability
4. They improve readability
5. Hide the implementation details



## Creating Function in Python

Function In Python created by using the **def keyword** as shown below:

```
def my_function():
```

```
    print("Hello from a function")
```

## Calling a Function in Python

- To call a function, use the function name followed by parenthesis as shown below:

```
def my_function():
```

```
    print("Hello from a function")
```

**Defination of a function**

```
my_function()
```

**Calling a function**

## Scope and Name Resolution

**Example:** This program illustrate Local and Global scope in Python

```
def my_function():
```

```
    test = 1          # local scope
```

```
    print("my_function is:",test)
```

```
test=0                # global scope
```

```
my_function()
```

```
print("global is:" test)
```

## Scope and Name Resolution

Once the code is executed the out put is

Inside my\_function the value of test is: 1

The value of test(global variable) outside the function :0

Hence its clear that test=1 shadows the test=0 assignment in my\_function. In global scope the value of test is still 0

## Input Parameter

Passing parameter Parameters are specified after the function name, inside the parentheses.

You can add as many parameters as you want, just separate them with a comma.

```
def my_function(fname):  
    print("The first name is " + fname)
```

Parameter(fname)

```
my_function("john")
```

Value of the parameter passed

## Input Parameter

There three key points of input parameter:

1. Argument –passing- Assigning an object to a local variable name
2. Assigning an object to an argument name inside a function doesn't affect the caller
3. Changing a mutable object argument in a function affects the caller

## Argument Passing Parameter

**Example :-**

```
x=3
```

```
def func(y):
```

```
    print(y)
```

```
func(x)  # it prints 3
```

## Assignment to argument name doesn't affect caller

Example :

```
x=3
```

```
def func(x):
```

```
    x=7
```

```
    print(x)    # it prints 7
```

```
func(x)
```

```
print(x)    # it prints 3
```



## Changing a mutable affects the caller

Example :

```
x=[1,2,3]
```

```
def func(x):
```

```
    x[2]=42
```

```
    x="hello world"
```

```
func(x)
```

```
print(x)    # it prints [1,42,3]
```

## How to specify the input parameters

Different ways to specify the input parameters are:

- Positional arguments
- Keyword Arguments
- Variable Positional Arguments
- Variable keyword argument
- Keyword only argument

## Positional Arguments

**Example :**

```
def func(a,b,c):
```

```
    print(a,b,c)
```

```
func(1,2,3)    # prints 1,2,3
```

## Keyword Arguments

**Example :**

```
def func(a,b,c)
```

```
    print(a,b,c)
```

```
func(a=1,c=2,b=3)    # prints 1,3,2
```

In the above example when the func is called the value a is assigned to 1, b is assigned to 3 and the value of c is assigned to 2 irrespective of the position on a,b,c

## Variable Positional Arguments

**Example :**

```
def minimum(*n):          #n=(1,3,-7,9)
```

```
    if n:
```

```
        mn=n[0]
```

```
        for value in n[1:]:
```

```
            if value<mn:
```

```
                mn = value
```

```
    print(mn)
```

```
minimum(1,3,-7,9)         # prints -7
```

```
minimum()                 # prints nothing
```

## Variable Positional Arguments

- In the above example Passing the variable numbers of positional arguments to a function (minimum(\*n)) by using the operator “\*” followed by variable name where “\*” indicates that the parameter will be collecting a variable number of positional argument based on the function call.
- Hence  $n=(1,3,-7,9)$  .
- Within the function n is a tuple and this function calculates the minimum number from the input values.

## Variable Keyword Arguments

Example :

```
def func(**kwargs):
```

```
    print(kwargs)
```

```
    print(type(kwargs))
```

```
func(a=1,b=42)
```

```
func(**{'a':1,'b':42})
```

```
func(**dict(a=1,b=42))
```

**Output ->**

```
{'a': 1, 'b': 42}
```

```
<class 'dict'>
```

```
{'a': 1, 'b': 42}
```

```
<class 'dict'>
```

```
{'a': 1, 'b': 42}
```

```
<class 'dict'>
```

## Keyword-Only Arguments

**Example :**

```
def kwo(*a,c):
```

```
    print(a,c)
```

```
kwo(1,2,3,c=7)    #prints (1,2,3) 7
```

```
kwo(c=7)          #prints (),7
```

```
Kwo(1,2)          #Displays error:- "TypeError: func() missing 1 required keyword-only argument:  
    'c' "
```

Python 3 supports keyword only argument



## Default Parameter

The following example shows how to use a default parameter value. If we call the function without parameter, it uses the default value:

```
def my_function(country = "Norway"):  
    print("I am from " + country)
```

```
my_function("Sweden")
```

```
my_function("India")
```

```
my_function()
```

```
my_function("Brazil")
```

**Output ->**

Sweden

India

Norway

Brazil

## Return value in function

Return Values -To let a function return a value, use the return statement:

```
def my_function(x):
```

```
    return 5 * x
```

```
print(my_function(3))
```

```
Print(my_function(4))
```

**Output ->**

15

20

## Return multiple values in function

To return multiple values is very easy ,where the returned values are separated by comma as shown below. The results are returned in tuple.

```
def func(a,b):  
    x=a+b  
    y=a*b  
    return (x,y)  
  
print(func(20,7))
```

**Output ->**

(27, 140)

## Recursive Function

- When a function call itself to produce a result it is said to be recursive
- A base condition is must in every recursive programs otherwise it will continue to execute forever like an infinite loop
- Recursive function is called by some external code. If the base condition is met then the program do something meaningful and exits.
- Otherwise, function does some required processing and then call itself to continue recursion.

## Example of Recursive Function

```
def fact(n):  
    if n == 0:  
        return 1  
    else:  
        return n * fact(n-1)  
  
print(fact(0))           # Prints 1  
print(fact(5))           #Prints 120
```

## Anonymous Function

In Python, anonymous function is a function that is defined without a name. While normal functions are defined using the `def` keyword, in Python anonymous functions are defined using the `lambda` keyword.

Hence, anonymous functions are also called lambda functions

A lambda function can take any number of arguments, but can only have one expression.

### Syntax :

`lambda arguments : expression`

## Anonymous Function

### Example 1: Lambda Function with one argument

A lambda function that adds 10 to the number passed in as an argument, and print the result:

```
x = lambda a : a + 10
```

```
print(x(5))
```

# 15 (Output)

## Anonymous Function

### Example 2: Lambda Function with multiple argument

```
x = lambda a,b: a+10+b
```

```
print(x(5,5))
```

#20(Output)



## Built-in Function

Python supports lot of Built in Function some of them are listed below:

Function	Description
<u>abs()</u>	Returns the absolute value of a number
<u>all()</u>	Returns True if all items in an iterable object are true
<u>any()</u>	Returns True if any item in an iterable object is true
<u>ascii()</u>	Returns a readable version of an object. Replaces none-ascii characters with escape character
<u>bin()</u>	Returns the binary version of a number
<u>bool()</u>	Returns the boolean value of the specified object
<u>bytearray()</u>	Returns an array of bytes
<u>bytes()</u>	Returns a bytes object
<u>callable()</u>	Returns True if the specified object is callable, otherwise False
<u>chr()</u>	Returns a character from the specified Unicode code.

## Built-in Function

<u>pow()</u>	Returns the value of x to the power of y
<u>print()</u>	Prints to the standard output device
<u>property()</u>	Gets, sets, deletes a property
<u>range()</u>	Returns a sequence of numbers, starting from 0 and increments by 1 (by default)
<u>repr()</u>	Returns a readable version of an object
<u>reversed()</u>	Returns a reversed iterator
<u>round()</u>	Rounds a numbers
<u>set()</u>	Returns a new set object
<u>setattr()</u>	Sets an attribute (property/method) of an object

## Importing Objects

**Module** contents are made available to the caller with the import statement. Python modules can get access to code from another module by importing the file/function using import. When import is used, it searches for the module initially in the local scope by calling `__import__()` function. The value returned by the function are then reflected in the output of the initial code.

The import statement takes many different forms, some common import are shown in the next slide

## Importing Objects

### Syntax:

```
import module_name
```

**Example :** import math

```
x=math.pi
```

```
print(x)
```

#Output -> 3.141592653589793

## Importing Objects

**from module import module name**

In the above code module math is imported, and its variables can be accessed by considering it to be a class and pi as its object.

In the below code the value of pi is returned by `__import__()` pi as whole can be imported into our initial code, rather than importing the whole module.

## Importing Objects

**Syntax :**

```
from module_name import function_name
```

**Example :** from math import pi

## Importing Objects

**from module\_name import \***

In the above code module math is not imported, rather just pi has been imported as a variable.

All the functions and constants can be imported using \*.

**Syntax :**

**from module\_name import \***

**Example :**

```
from math import *  
print(pi)  
print(factorial(6))
```

## Saving Time and Memory

In this section will discuss about the important function which are listed below:

- **Map()**
- **Filter()**
- **Zip()**
- **Comprehension**
- **Generator**



## map()

The map() function executes a specified function for each item in a iterable .map() is a function which takes two arguments where the first parameter is the function and the second parameter is the sequence.

Syntax : `r = map(func, seq)`

where ,

function(func) -> A Function to be run for each item in the iterable

sequence(iterable) -> The iterable to be filtered

## map()

**Example :** Program to double a number using map function without using lambda function

```
def double(num):  
    return num+num  
  
num=(1,2,3,4)  
print(list(map(double,num)))      # output ->[2,4,6,8]
```

## map()

**Example :** Program to double a number using map function using lambda function

```
num=(1,2,3,4)
```

```
print(list(map(lambda num:num+num,num))) # output -
```

```
>[2,4,6,8]
```

## filter()

The filter() function returns an iterator where the items are filtered through a function to test if the item is accepted or not.

Syntax: **filter(function, sequence)**

where ,

function -> A Function to be run for each item in the iterable

sequence(iterable) -> The iterable to be filtered

Filter offers an elegant way to filter out all the elements from a "sequence", for which the function returns True. This function will be applied to every element of the list or tuple .

## Example on filter()

**Example :**Program to filter even numbers from the series 1...10 without using lambda

```
def iseven(num):
```

```
    return num%2==0
```

```
num=(1,2,3,4,5,6,7,8,9,10)
```

```
print(list(filter(iseven,num))) # Output -> [2, 4, 6, 8, 10]
```

## Example on filter()

**Example :**Program to filter even numbers from the series 1...10 with using lambda

```
num=(1,2,3,4,5,6,7,8,9,10)
```

```
x=list(filter(lambda num:num%2==0,num))
```

```
print(x)
```

**# Output -> [2, 4, 6, 8, 10]**

## zip()

The zip() function returns a zip object, which is an iterator of tuples where the first item in each passed iterator is paired together, and then the second item in each passed iterator are paired together etc.

If the passed iterators have different lengths, the iterator with the least items decides the length of the new iterator.

### Syntax :

```
zip(iterator1, iterator2, iterator3 ...)
```

where,

iterator1, iterator2, iterator3 ...Iterator objects that will be joined together

## Example of zip()

**Example:** Program to zip a and b

```
a = ("John", "Charles", "Mike")
```

```
b = ("Jenny", "Christy", "Monica", "Vicky")
```

```
x = zip(a, b)          # (('John', 'Jenny'), ('Charles', 'Christy'), ('Mike', 'Monica'))
```



## Comprehension

Comprehensions provide a concise way to create sequence. Python offers different types of comprehensions :list,dict and set.

By using comprehension sequence can be created easily which saves memory and time.

### List Comprehension:

List comprehension consist of “[ “]” containing an expression followed by a for clause,if clause.

The result will be a new list resulting from evaluating the expression in the context of the for and if clause which follow it.

## Comprehension

### Syntax of list comprehension:

1. [expr for val in collection]
2. [expr for val in collection if <condition>] -> the result will be added if the condition satisfied.

## Comprehension

**Example : Program on finding the squares of the number and create list.**

```
squares=[]  
for i in range(1,6):  
    squares.append(i**2)  
print (squares)
```

**# Output -> [1, 4, 9, 16, 25]**

## Comprehension

The above code can be rewritten as shown below Using list comprehension :

```
squares=[i**2 for i in range(1,6)]  
print (squares)           # Output -> [1, 4, 9, 16, 25]
```

## Set Comprehension

Set comprehensions are pretty similar to list comprehensions. The only difference between them is that set comprehensions use curly brackets { } containing an expression followed by a for clause, if clause. The result will be a new set resulting from evaluating the expression in the context of the for and if clause which follow it.

### Example: Program to create set using comprehension

```
Squares={ }
```

```
squares={i for i in range(1,6)}
```

```
print (squares)           # Output -> {1, 2, 3, 4, 5}
```

## Dictionary Comprehension

Dictionary comprehension exactly the same as list and set only there is a difference in syntax.

**Syntax :**

`x=dict((c,k) for k,c in enumerate(string))` or `x={c:k for k,c in enumerate(string)}`

## Example of Dictionary Comprehension

**Example:** Program to assign to each key with its position in the string

```
word="hello"
```

```
x=dict((c,k) for k,c in enumerate(word))
```

```
print(x)           # Out put -> {'h': 0, 'e': 1, 'l': 3, 'o': 4}
```

## Generators

Generators are very powerful tool in Python. They are based on the concepts of iteration.

Generators are of two types :

- **Generator Function** :These are very similar to regular function ,but instead of returning the result through return statements ,they use to yield,which allows them to suspend and resume their state between each call.
- **Generator expression**: This are similar to list comprehension. Instead of returning the list they return an object that produces results one by one.



## Example of Generators

**Example :**Program that yields 1,2,3 using function

```
simpleGeneratorFunc()
```

```
def simpleGeneratorFun():
```

```
    yield 1
```

```
    yield 2
```

```
    yield 3
```

```
# Driver code to check above generator function
```

```
for value in simpleGeneratorFun():
```

```
    print(value)          # Output ->1
```

**2**

**3**

## Self Assessment Question

1. What is the output of the following code?

```
def foo(fname, val):  
    print(fname(val))  
foo(max, [1, 2, 3])  
foo(min, [1, 2, 3])
```

- a. 3 1
- b. 1 3
- c. Error
- d. none of the mentioned

**Answer: A**

## Self Assessment Question

2. What is the output of the following code?

```
def foo():  
    return total + 1  
total = 0  
print(foo())
```

- b. 0
- c. 1
- d. Error
- e. none of the above

**Answer: B**

## Self Assessment Question

3. What is the output of the following code?

```
def foo(i, x=[]):
```

```
    x.append(i)
```

```
    return x
```

```
for i in range(3):
```

```
    print(foo(i))
```

b. [0] [1] [2]

c. [0] [0, 1] [0, 1, 2]

d. [1] [2] [3]

e. [1] [1, 2] [1, 2, 3]

**Answer: A**

## Self Assessment Question

4. Fill in the line of code for calculating the factorial of a number.

```
def fact(num):
```

```
    if num == 0:
```

```
        return 1
```

```
    else:
```

```
        return _____
```

- b. `num*fact(num-1)`
- c. `(num-1)*(num-2)`
- d. `num*(num-1)`
- e. `fact(num)*fact(num-1)`

**Answer: A**

## Self Assessment Question

5. What is the output of the following piece of code?

```
def test(i,j):  
    if(i==0):  
        return j  
    else:  
        return test(i-1,i+j)  
print(test(4,7))
```

- a. 13
- b. 7
- c. Infinite loop
- d. 17

**Answer: A**

## Self Assessment Question

6. What happens if the base condition isn't defined in recursive programs?
- b. Program gets into an infinite loop
  - c. Program runs once
  - d. Program runs n number of times where n is the argument given to the function
  - e. An exception is thrown

**Answer :A**

## Self Assessment Question

7. What is the name of anonymous function

- b. Recursive
- c. Generator
- d. Lambda
- e. map

**Answer :C**



## Self Assessment Question

8. Which function is used to select the element from the list efficiently

- b. Map
- c. Filter
- d. Lambda
- e. Zip

**Answer: B**

## Self Assessment Question

9. What is the output of the following program :

```
y = 8
```

```
z = lambda x : x * y
```

```
print (z(6))
```

- b. 48
- c. 14
- d. 20
- e. None of the above

**Answer: A**

## Self Assessment Question

10. What is the output of the following?

```
k = [print(i) for i in my_string if i not in "aeiou"]
```

- b. prints all the vowels in my\_string
- c. prints all the consonants in my\_string
- d. prints all characters of my\_string that aren't vowels
- e. prints only on executing print(k)

**Answer: C**

## Self Assessment Question

11. What is the output of the following?

```
generator=(2*i for i in range(1,4))
```

```
for val in generator:
```

```
    print(val)
```

- b. 2  
4  
6
- b. 1,2,3
- c. 2,4,6
- d. None of the above

**Answer: A**

## Assignment

1. Write a Python code to perform `+,*,/,//,%` using function
2. Write a Python code to perform HCF of two numbers using function
3. Write a Python code to perform LCM of two numbers using function
4. Write a Python code to illustrate the generator
5. Write a Python code to illustrate comprehension
6. Write a python code to find the sum of 10 numbers using recursion
7. Write a python code to find the fibonacci series using function

## Summary

- These are very similar to regular function ,but instead of returning the result through return statements ,they use to yield, which allows them to suspend and resume their state between each call.
- Comprehensions provide a concise way to create sequence. Python offers different types of comprehensions :list,dict and set.
- Set comprehensions are pretty similar to list comprehensions. The only difference between them is that set comprehensions use curly brackets { } containing an expression followed by a for clause, if clause.
- Recursive function is called by some external code. If the base condition is met then the program do something meaningful and exits.

## Document Links

Topics	URL	Notes
Functions in python	<a href="https://www.w3schools.com/python/python_functions.asp">https://www.w3schools.com/python/python_functions.asp</a>	This link explains about the Function in python
Lambda function	<a href="https://www.w3schools.com/python/python_lambda.asp">https://www.w3schools.com/python/python_lambda.asp</a>	This explains about the lambda function in python
Zip Function in Python	<a href="https://www.w3schools.com/python/ref_func_zip.asp">https://www.w3schools.com/python/ref_func_zip.asp</a>	This explains about the zip function in python
Map and Filter in Python	<a href="https://www.python-course.eu/python3_lambda.php">https://www.python-course.eu/python3_lambda.php</a>	This explains about the map and filter function in python
Map in python	<a href="https://www.w3schools.com/python/ref_func_map.asp">https://www.w3schools.com/python/ref_func_map.asp</a>	This explains about map function with example function in python
Filter in python	<a href="https://www.w3schools.com/python/ref_func_filter.asp">https://www.w3schools.com/python/ref_func_filter.asp</a>	This explains about the lambda function in python

## Document Links

Topics	URL	Notes
Generators in Python	<a href="https://www.learnpython.org/en/Generators">https://www.learnpython.org/en/Generators</a>	This link explains about the Generator with example in python
Generator Expression	<a href="https://djangostars.com/blog/list-comprehensions-and-generator-expressions/">https://djangostars.com/blog/list-comprehensions-and-generator-expressions/</a>	This link explains about the Generator expression with example in python
Built in function in python	<a href="https://www.w3schools.com/python/python_ref_functions.asp">https://www.w3schools.com/python/python_ref_functions.asp</a>	This link explains about the Built in function used in python



## Video Links

Topics	URL	Notes
Function in python	<a href="https://www.youtube.com/watch?v=BVfCWuca9nw">https://www.youtube.com/watch?v=BVfCWuca9nw</a>	This link explains function in python
Function Argument in python	<a href="https://www.youtube.com/watch?v=ijXMGpoMkhQ">https://www.youtube.com/watch?v=ijXMGpoMkhQ</a>	This explains about the Input parameter of function in python

## Books

Book Name	Chapter	Page Number	Comments
Learn Python Programming by Fabrizio Romano ,2 <sup>nd</sup> Edition ,Packt Publication	. Chapter 4	Page No: 109-141	Explains Function ,Input Parameter,Typesof Parameter
Learn Python Programming by Fabrizio Romano ,2 <sup>nd</sup> Edition ,Packt Publication	. Chapter 5	Page No: 143-175	Explains Zip,Filter ,Reduce,Generator,Comprehension,