Chapter – 4

WORKING WITH PYTHON LIBRARY

WHAT IS A LIBRARY?

A library is a collection of modules (and packages) that together cater to a specific type of applications or requirements. A library can have multiple modules in it. Some commonly used Python libraries are as listed below.

(i) Python standard library: This library is distributed with Python that contains module for various types of functionalities. Some commonly used modules of Python standard library are:

- **math Module:** Which provides mathematical functions to support different types of Calculations.
- **cmath Module:** Which provides mathematical functions for complex numbers.
- **random Module:** Which provides functions for generating pseudo-random numbers?
- **statistics Module:** Which provides mathematical statistics functions.
- **urllib Module:** Which provides URL handling functions so that you can access websites from within your program.

(ii) NumPy library. This library provides some advance math functionalities along tools to create and manipulate numeric arrays.

(iii) SciPy library. This is another useful library that offers algorithmic and mathematical-tools for scientific calculations.

(iv) tkinter library. This library provides traditional Python user interface tool kit and helps you to create user friendly GUI interface for different types of applications.

(v) matplotlib library. This library offers many functions and tools to produce quality output in variety of formats such as plots, charts, graph set c. A library can have multiple modules in it.

What is a Module?

The act of partitioning a program into individual components (known as modules*)* is modularity. A module is a separate unit. The justification for partitioning a program is that

- it reduces its complexity to some degree and
- it creates several well-defined, documented boundaries within the program.

Another useful feature of having modules, is that its contents can be reused in other programs without having to rewrite or recreate them.

Structure of a Python Module:

A Python module can contain much more than just *functions.* A Python module is a normal Python file *(.py* file) containing one or *more* of the following objects related to a particular task:

Docstrings: Triple quoted comments; useful for documentation purposes.

Variables and constants-: labels for data.

Classes: Templates/blueprint t o create objects of a certain kind.

Objects: Instances of classes.

Statements: Instructions

Functions: Named group of instructions.

Python comes loaded with some predefined modules that you can use and you can even create your own modules. The Python modules that come preloaded with Python are called standard library modules.

Let us create a user defined module namely tempConversion.

#tempConversaion.py

“”” conversion functions between Fahrenheit and centigrade”””

#Functions

```
def to_centigrade(x):
        returns 5*(x-32)/9.0
def to_fahrenheit(x):
        returns 9*x/5.0+32
#constants
  FREEZING_C=0.0
  FREEZING_F=32.0
  The elements shown in the module are:
  Name of the module: tempconversion
  Module file name: tempConversion.py
```

Contains: Two functions (i) to_centigrade() (ii) to_fahrenheit()

Two constants (i) FREEZING_C (ii) FREEZING_F

Three docstrings (triple quotes strings)

The module can be imported through an import statement. After importing the module if we write:

help(tempconversion)

Python will display all docstrings along with module name, functions’ name, and constants as shown below.

```
>>> import tempConversion
>>> help (tempConversion)
Help on module tempConversion :
Name
    tempConversion – Conversion functions between Fahrenheit and centigrade
FILE
    C:\python37\pythonwork\tempconversion.py
FUNCTIONS
    to_centigrade(x)
        Returns : x converted to centigrade
    To_fahrenheit(x)
        Returns : x converted to Fahrenheit
```

DATA

FREEZING_C = 0.0

FREEZING_F = 32.0

There is one more function dir( ) when applied to a module, gives you names of all that is defined inside the module, (See below)

```
>>> import tempConversion
>>>dir (tempConversion)
['FREEZING_C', 'FREEZING_F', '_____ built-ins __,' '__doc__,' ___file ___,
'__name__', ' ___package ___, to _centigrade', to __ fahrenheit' ]
```

Importing Modules In A Python Program

As mentioned before, in Python if you want to use the definitions inside a module, then you need to first import the module in your program. Python provides import statement to import modules in a program. The import statement can be used in two forms:

(i) To import entire module: the **import <module>**command

(ii) To import selected objects from a module: the **from <module> import<object>** command

Importing Entire Module

The import statement can be used to import entire module and even for importing selected items. To import entire module, the import statement can be used as per following syntax.

import module1 [, module2 [, ... module]]

For example, to import a module, say time, you will write:

import time → Module namely time being imported

To import two modules namely decimals and fractions, you will write:

import decimals, fractions → Two modules namely decimals and fractions beings imported with one import statement.

After importing a module, you can use any function/definition of the imported module as per following syntax:

<module-name>. <function-name>( )

This way of referring to a module's object is called **dot notation**.

For example, consider the module tempConversion given in figure. To use its function to_centrigrade( ), we will be writing :

```
import temConversion
tempConversion.to_centrigrade(98.6)
```

→ calling function to_centrigrade( )of imported module tempConversion.

You can give an alias name to imported module as :

import <module> as <aliasname>

```
import tempConversion as tc
```

Importing Select Objects from a Module

If you want to import some selected items, not all from a module, then you can use from <module> import statement as per following syntax:

From<module> import<objectname>[,<objectname>[….]]*

To import Single Object

If you want to import a single object from the module like this so that do not have to prefix the module's name, you can write the name of object after keyword import. For instance, to import just the constant pi from module math, you can write:

from math import pi

Now, the constant pi can be used, and you need not prefix it with module name. That is, to print the value of pi, after importing like above, you will be writing.

print(pi)

Not this

print(math.pi) → Do not use module name with imported object if imported through

From <module> import command.

Do not use module name with imported object if imported through from <module> import command because now the imported object is part of your program's environment.

To Import Multiple Objects

If you want to import multiple objects from the module like this so that you do not have to prefix the module's name, you can write the comma separated list of objects after keyword import. For instance, to import just two functions sqrt( ) and pow( ) from math module, you will write :

from math import sqrt,pow

To Import All Objects of a Module

If you want to import all the items from the module like this so that you do not have to prefix the module's name, you can write.

from<modulename>import *

That is, to import all the items from module math, you can write:

from math import*

Now you can use all the defined functions, variable etc from math module, without having to prefix module's name to the imported item name.

Using Python's Built-in Functions

The Python interpreter has several functions built into it that are always available you need not import any module for them. In other words, the built-in functions are part of current namespace of Python interpreter. So, you use built-in functions of Python directly as:

<function-name>()

For example, the functions that you have worked with until now such as input ( ) , int( ), float( ) type( ), len( ) etc. Are all built in functions, that is why you never prefixed them with any module name.

Python's built – in Mathematical Functions

Python provides many mathematical built-in functions.

len()	divmod()
pow()	sum()
str()	max()
int()	min()
float()	oct()
range()	hex()
type()	abs()

Python's built-in String Functions

Let us now use some string functions. Although you have worked with many string functions in your previous class, let us use three new string-based functions. These are

- <Str>.join (<string iterable>) – joins a string or character (i.e., <str>) after each member of the string iterator i.e., a string-based sequence.
- <Str> .split(<string/char>)splits a string(i.e., <str) based on given string or character (i.e., <string/char>) and returns a list containing split strings as members.
- <Str>.replace(<word to be replaced>, <replace word>) – replaces a word or part a of the string with another in the given string <str>.

Working with Some Standard Library Modulus.

Other than built-in functions, standard library also provides some modules having functionality for specialized actions. Let us learn to use some such modules. In the following lines we shall talk about how to use some useful functions of random and string modules of Python's standard library.

Using Random Module

Python has a module namely random that provides random-number generators. A random number in simple words means – a number generated by chance, i.e., randomly.

To use random number generators in your Python program, you first need to import module random using any import command, e.g.,

import random

Some most common random number generator functions in random module are:

random( ): It returns a random floating point number N in the range 0.0 to 1.0 i.e., 0.0 ≤ N < 1.0. Notice that the number generated with random( ) will always be less than 1.0. (only lower range – limit is inclusive). Remember, it generates a floating-point number

randdint(a, b): It returns a random integer N in the range (a, b), i.e., a ≤N ≤ b (both range-limits are inclusive). Remember, it generates an integer.

random.uniform(a,b) : It returns a random floating point number N such that

a < = N < =b for a <= b and

b < = N < = a for b < a

random.randrange(stop): It returns a randomly selected element from range(start,stop,step) random.randrange(start,stop[,step])

Let us consider some examples. In the following lines we are giving some sample codes along with their output.

1. To generate a random floating-point number between 0.0 to 1.0 simply use random( ):

>>> import random

>>> print (random.random()) The on put generated is between range [0.0.1.0)

0.022353193431

2. To generate a random floating-point number between range lower to upper using random():

(a) multiply random() with difference of upper limit with lower limit, i.e, (upper-lower)

(b) add to it lower limit

For example, to generate between 15 to 35, you may write :

>>> import random

>>> print(random.random( )* (35-15) + 15)

28.307187234 →The output generated is floating point number between range 15 to 35.

3. To generate a random integer number in range 15 to 35 using randint( ), write:

>>>print (random.randint(15,35))

16 →The output generated is integer between range 15 to 35

4. To generate a floating-point random number in the ranges 11...55 or 111...55, after importing random module using import statement, you may write:

>>> random.uniform(11,55)

41.3451898131735

>>> random.uniform(111,55)

60.03906551659219

5. To generate a random integer in the ranges 23.47 with a step 3 or 0.235 after importing random module using import statement, you may write:

>>> random.randrange(23,47,3)

38

>>> random.randrange(235)

126

6. Given the following Python code, which is repeated four times. What could be the possible set of outputs out of given four set (dddd represent any combination of digits)?

import random

print(15+random.random()*5)

(i) 17.dddd,19.dddd,20.dddd,15.ddd (ii) 15.dddd,17.dddd,19.dddd,18.dddd

(iii) 14.dddd, 16.dddd,18.dddd,20.dddd (iv) 15.dddd, 15.dddd, 15.dddd,15.dddd

Solution : Option (ii) and (iv) are the correct possible outputs because:

(a) random( ) generates number N between range 0.0 < = N < 1.0

(b) when it is multiplied with 5, the range becomes 0.0 to < 5

(c) when 15 is added to it, the range becomes 15 to <20.

Only option (ii) and (iv) fulfil the condition of range 15 to <20.

7. What could be the minimum possible and maximum possible numbers by following code?

import random

print(random.randit(3,10)-3

Solution : Minimum possible number = 0

Maximum possible number = 7

Using String Module

Python has a module by the name string that comes with many constants and classes. It also offers a utility function capwords( ). Let us talk about some useful constants defined in the string module.

Please not that like other modules, before you can use any of the constants/ functions defined in the string module, you must import it using an import statement.

import string.

Some useful constants defined in the string module are being listed below:

string.ascii_letters:	It returns a string containing all the collection of ASCII letters.
string.ascii_lowercase:	It returns a string containing all the lowercase ASCII letters, i.e., 'abcdefghijklmnopqrstuvwxyz'.
string.ascii_uppercase:	It returns all the uppercase ASCII letters, i.e., 'ABCDEFCHIJKLMNOPQRSTUVWXYZ'.
string_digits :	It returns it string containing all the digits Python allows, i..e, the string, '0123456789'.
string.hexdigits :	It returns a string containing all the hexadecimal digits Python allows, i.e., the string '0123456789abcdefABCDEF'.
string_octdigits :	It returns a string containing all the octal digits Python allows, i.e., the string '01234567'.
string punctuation :	It returns a string of ASCII characters which are considered punctuation characters, i.e., the string
The string modules also offers a utility function capwords( ):	
capwords(<str>, [sepNone]):	It splits the specified string <Str> into words using <Str> split( ). Then it capitalizes each word using <Str> capitalize( ) function. Finally. It joints the capitalized words using <Str>joint( ). If the optional second argument scp is absent or is None, it will remove leading and trailing whitespaces and all inside whitespace characters are replaced by a single space.

You can obtain the value of the constants defined in string module by simply giving name with the string module name after importing string module, e.g.,

```
>>> import string
>>> string.ascii_letters
'abcdefghijklmonpqurstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>> import string
'0123456789'
>>> string.ascii_uppercase
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>>string.punctuation
\!"#$%&\'()*+,-./: ; ?@[\\]^_'{ |}~'
```

You can use capwords() using the string module name and passing the string name as its argument, e.g.,

```
>>> import string
>>> line = "this is a simple line\n New line"
>>>string.capwords(line)
'This Is A Simple Line New Line'
```

◆◆◆