

## Chapter-4 Using Python Libraries

### Introduction

**As our program become larger and more complex the need to organize our code becomes greater. We have already learnt in *Function* that large and complex program should be divided into functions that perform a specific task. As we write more and more functions in a program, we should consider organizing of functions by storing them in modules.**

**In this tutorial we will discuss the following topics**

S. No.	Topics
1	Python Library
2	Python Module
3	Creating user defined Module
4	Import Modules in Python
5	Python Package
6	Importing a Module from a Package
7	To use package from any location

## Chapter-4 Using Python Libraries

### Common Used Python Libraries



### What is a Python Library?

Python library is a collection of functions and methods that allows you to perform many actions without writing your code.

For example, the Python Numpy, is a one of the core libraries for numerical operations in Python.

Each library in Python contains a huge number of useful modules that you can import for your every day programming.

### What is a Python Module?

A **Python module** is a Python file containing a set of functions and variables to be used in an application. The variables can be of any type (arrays, dictionaries, objects, etc.).

Modules are usually stored in files with **.py** extension

## Chapter-4 Using Python Libraries

Some of the advantages of using modules in your Python code are:

- they enable you to organize your code into smaller pieces that are easier to manage.
- the code in the modules can be reloaded and rerun as many times as needed, which enables code reuse.
- modules are self-contained – you can never see a name in another file, unless you explicitly import it. This helps you to avoid name clashes across your programs.

### How to Creating user defined Module?

The most common way to create a module is to define a file with the **.py** extension that will contain the code you want to group separately from the rest of the application.

In our example we will create a module with two simple functions to display the area and perimeter of rectangle.

We will then import and use these functions in another file.

Let us create a module. Type the following and save it as **rectangle.py**.

```
def Area_rec(s1,s2):  
    area = s1*s2  
    return area  
  
def Perimeter_rec(s1,s2):  
    peri=2*(s1+s2)  
    return peri
```

## Chapter-4 Using Python Libraries

Here, we have defined two function `Area_rec(s1,s2)` and `Perimeter_rec(s1,s2)` inside a module named **rectangle**.

The function takes in two sides of rectangle and returns the area and perimeter of rectangle.

If you follow along, make sure to save the module in the same directory that will also contain the top-level file.

### How to import modules in Python?

There are two ways to import a module:

- using the **import** statement – imports the entire module.
- using the **from...import** statement – imports only individual module attributes.

#### 1) using the **import** statement

Here is how we can import a module using the **import statement**:

1. **import** rectangle
2. `a=int (input ("Enter side1: "))`
3. `b=int (input ("Enter side2: "))`
4. `print ("the area of rectangle is :", rectangle.Area_rec(a,b))`
5. `print ("the perimeter of rectangle is :", rectangle.Perimeter_rec(a,b))`

This does not import the names of the functions defined in **rectangle** directly in the current symbol table. It only imports the module name **rectangle** there.

Using the module name we can access the function using the dot (.) operator.

## Chapter-4 Using Python Libraries

The first line (**import** rectangle) imports the content of the file rectangle.py.

The second and third line calls the function Area\_rec ( ), Perimeter\_rec ( ) and print the area and perimeter of rectangle respectively.

**The result of the code above is:**

```
===== RESTART: D:/Amjad_CS/usingModule.py
Enter side1: 10
Enter side2: 20
the area of rectangle is : 200
the area of rectangle is : 60
```

### 2) using the *from...import* statement

We can import specific names from a module without importing the module as a whole.

When we import attributes or call the function using the **from...import** statement, we don't need to precede the attribute name with the name of the module.

Here is how it can be done:

1. **from** rectangle **import** Area\_re
2. a=int (input ("Enter side1: "))
3. b=int (input ("Enter side2: "))
4. print ("the area of rectangle is :", rectangle.Area\_rec(a,b))

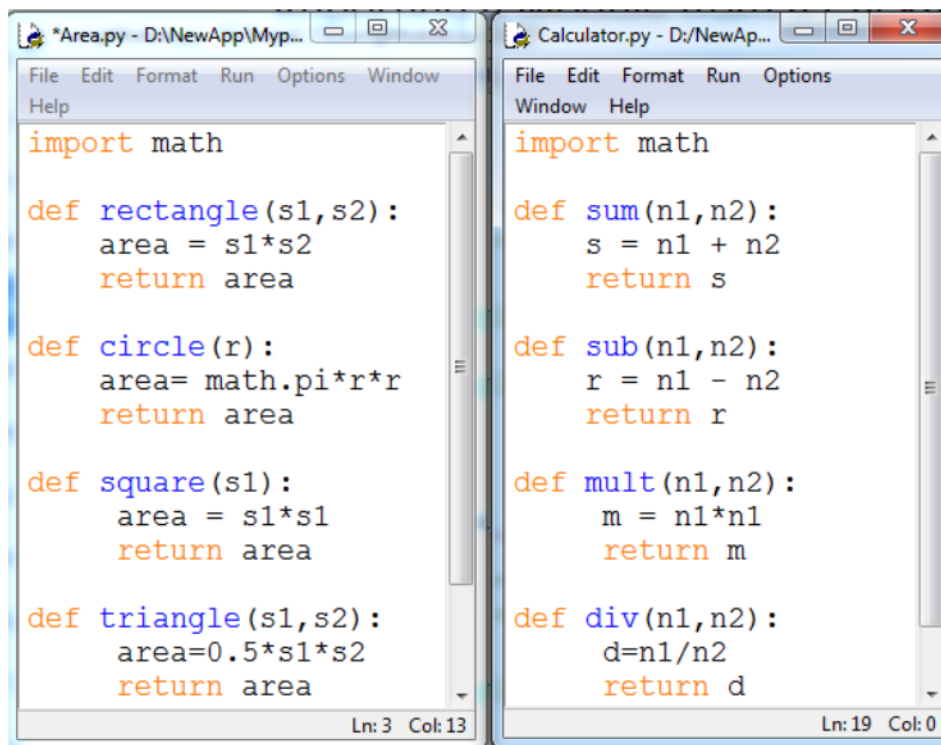
## Chapter-4 Using Python Libraries

### What is a Python Package?

Packages are namespaces which contain multiple packages and modules themselves. They are simply directories.

Let's create a package named **Mypackage**, using the following steps:

- Create a new folder named **NewApp** in D drive (**D:\NewApp**)
- Inside **NewApp**, create a subfolder with the name '**Mypackage**'.
- Create an empty **\_\_init\_\_.py** file in the **Mypackage** folder.
- Create modules **Area.py** and **Calculator.py** with following code:



The image shows two side-by-side screenshots of a Python IDE. The left window is titled '\*Area.py - D:\NewApp\Myp...' and contains the following code:

```
import math

def rectangle(s1,s2):
    area = s1*s2
    return area

def circle(r):
    area= math.pi*r*r
    return area

def square(s1):
    area = s1*s1
    return area

def triangle(s1,s2):
    area=0.5*s1*s2
    return area
```

The right window is titled 'Calculator.py - D:\NewAp...' and contains the following code:

```
import math

def sum(n1,n2):
    s = n1 + n2
    return s

def sub(n1,n2):
    r = n1 - n2
    return r

def mult(n1,n2):
    m = n1*n1
    return m

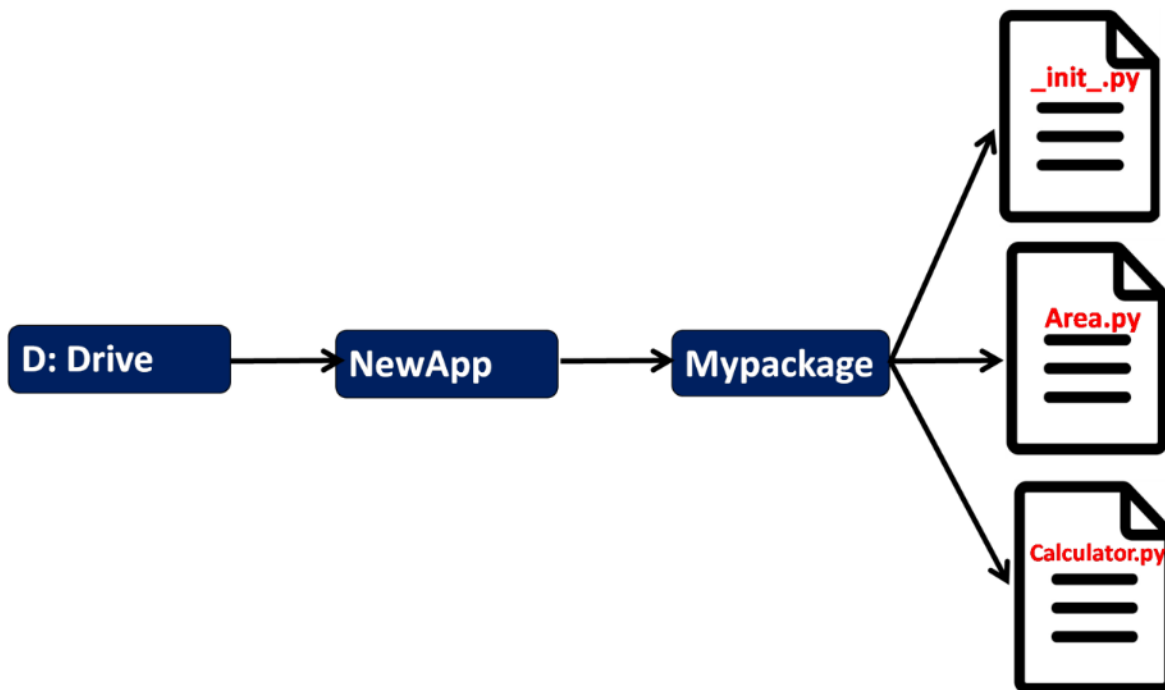
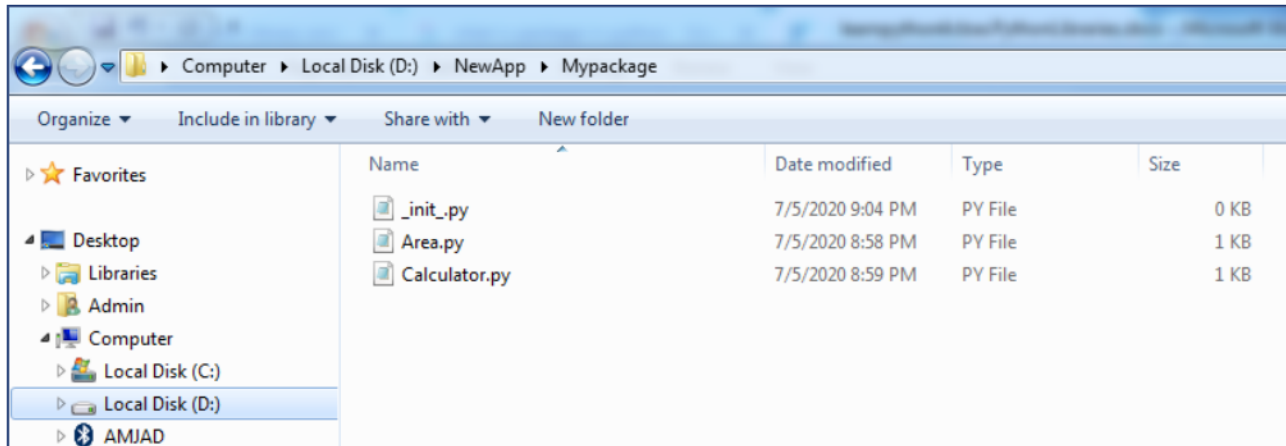
def div(n1,n2):
    d=n1/n2
    return d
```

### **\_\_init\_\_.py**

Each package in Python is a directory which **MUST** contain a special file called **\_\_init\_\_.py**. This file can be empty, and it indicates that the directory it contains is a Python package, so it can be imported the same way a module can be imported.

## Chapter-4 Using Python Libraries

We have created our package called Mypackage. The following is a folder structure:



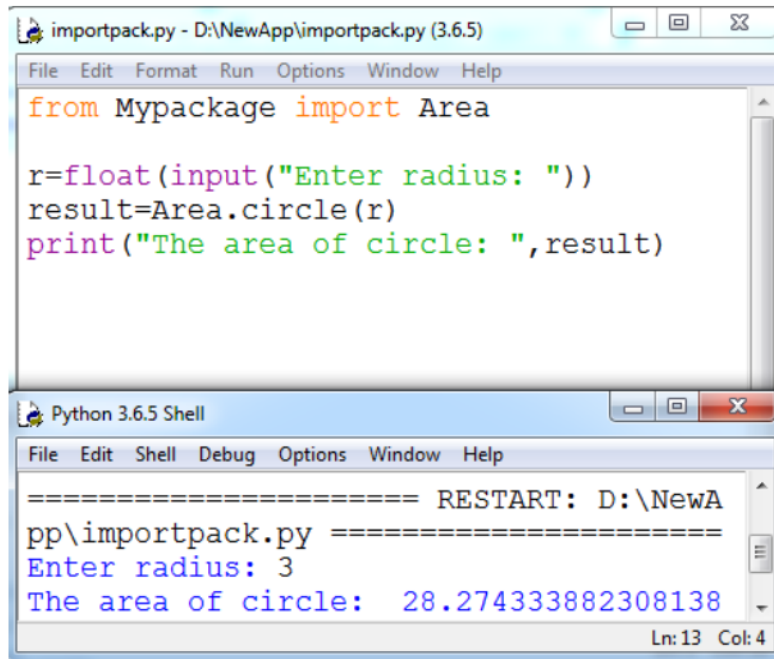
### Importing a Module from a Package

Now, to test our package, invoke the Python prompt from the NewApp folder.



## Chapter-4 Using Python Libraries

**Example:** Import the Area module from the Mypackage package and call its circle () function. (Save this file in NewAPP folder with .py and run.)

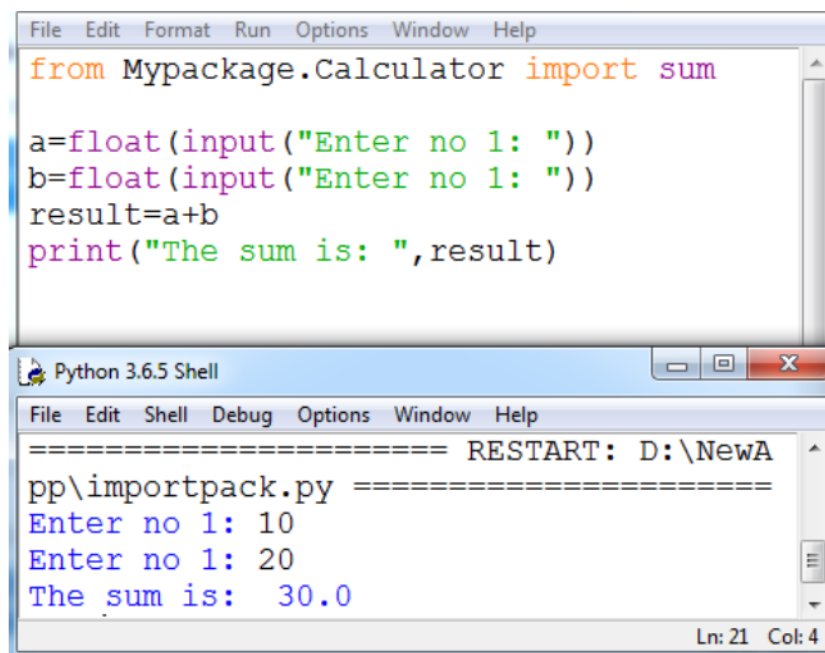


```
importpack.py - D:\NewApp\importpack.py (3.6.5)
File Edit Format Run Options Window Help
from Mypackage import Area

r=float(input("Enter radius: "))
result=Area.circle(r)
print("The area of circle: ",result)

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
===== RESTART: D:\NewA
pp\importpack.py =====
Enter radius: 3
The area of circle:  28.274333882308138
Ln: 13 Col: 4
```

**Example:** It is also possible to import specific functions from a module in the package



```
File Edit Format Run Options Window Help
from Mypackage.Calculator import sum

a=float(input("Enter no 1: "))
b=float(input("Enter no 1: "))
result=a+b
print("The sum is: ",result)

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
===== RESTART: D:\NewA
pp\importpack.py =====
Enter no 1: 10
Enter no 1: 20
The sum is:  30.0
Ln: 21 Col: 4
```



## Chapter-4 Using Python Libraries

### To use package from any location

- At this time, the package ***“Mypackage”*** will be accessible from its location only i.e. the file which wants to import this package must be in same folder/drive where ***“Mypackage”*** is.
- To enable ***“Mypackage”*** to be used from any location, Copy this Mypackage to Python’s **site-packages** folder inside **Lib** folder of Python’s installation folder. (*Try with copying to Lib folder also*)
- Path of site-packages is :  
C:\Users\AppData\Local\Programs\Python\Python36-32\Lib\site-packages
- After this you can import this package ***“Mypackage”*** in any python file.