3 Using Python Libraries

3.1 INTRODUCTION

As our programs become larger and more complex, the need to organize our code becomes greater. We have already learnt in Chapter 2 that a large and complex program should be divided into functions. As we write more and more functions in a program, we should consider organizing the functions by storing them in modules.

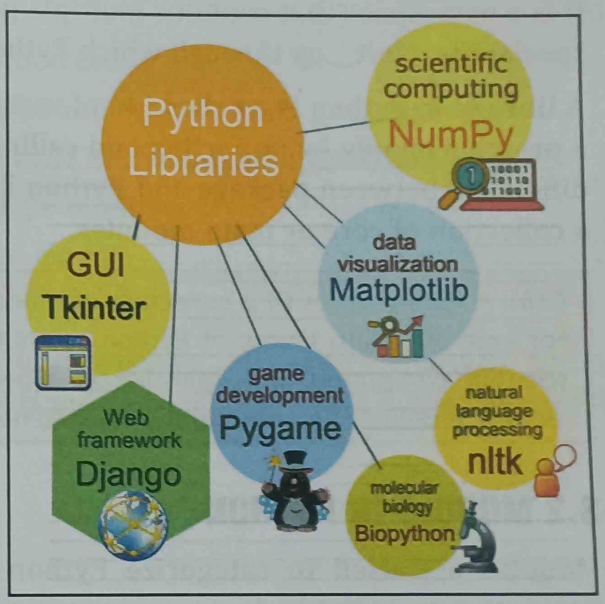


Fig. 3.1: Python Libraries

A module is simply a file that contains Python code. When we break a program into modules, each module should contain functions that perform related tasks. For example, suppose we are writing an accounting system. We would store all of the account receivable functions in their own module, all of the account payable functions in their own module, and all of the payroll functions in their own module. This approach, which is called **modularization**, makes the program easier to understand, test and maintain.

Even tiny real-world applications contain thousands of lines of code. In fact, applications that contain millions of lines of code are somewhat common. Imagine trying to work with a file large enough to contain millions of lines of code—you'd never find anything easily. In short, we need some method to organize code into small pieces that are easier to manage, much like the examples in this book. The Python solution is to place code in separate code groupings called **modules**.

Commonly-used modules that contain source code for generic needs are called **libraries**.

Therefore, when we speak about working with libraries in Python, we are, in fact, working with modules that are created inside the package(s) or, say, library. Thus, a Python program comprises three main components:

- Library or Package
- Module
- Functions/Sub-modules

Relation between a module, package and library in Python:

A **module** is a file containing Python definitions, functions, variables, classes and statements with .py extension. A module is a logical organization of Python code. Related codes are grouped into a module which makes the code easier to understand and use. Any Python module is an object with different attributes that we can bind and reference.

Simply, it is an executable file containing a set of functions which can be included in our application for performing a specific task.

On the other hand, a **Python package** is simply a directory of Python module(s).

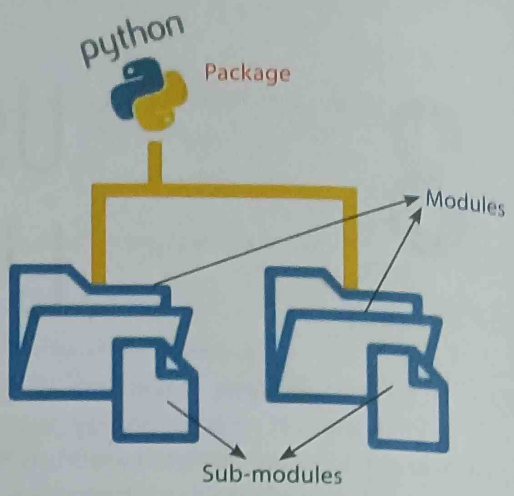


Fig. 3.2: Relation between a Package and Modules

It is a namespace that contains multiple packages or modules. It is a directory which contains a special file __init__.py through which Python interpreter interprets it as a package.

A **library** in Python is a collection of various packages. It is a reusable code that can be used in a program simply by importing and calling the method of that library. Conceptually, there is no difference between package and Python library. In Python, a library is used to loosely describe a collection of core or main modules.

CTM: A library refers to a collection of modules that together cater to specific needs or applications. For example, Math library of Python caters to mathematical computations, NumPy library deals with scientific computing, and Urllib library allows to access websites via your program by sending http request. It defines functions and classes to help in URL actions.

3.2 MODULE IN PYTHON

Modules are used to categorize Python code into smaller parts. The act of partitioning a program into individual components (modules) is called **modularity**. A module is a separate unit in itself. In other words, a module is simply a Python file where statements, classes, objects, functions, constants and variables are defined. Grouping similar code into a single file makes it easy to access. For example, if the content of a book is not indexed or categorized into individual chapters, the book will become boring and complicated. Hence, dividing the book into chapters makes it easy to understand. In the same way, Python modules are files which have a similar code. Thus, a module simplifies a Python code where classes, variables and functions are defined. The standard distribution of Python contains a large number of built-in modules, generally known as Standard library modules. For example, random, math, time, os, sys, etc.

CTM: A module is a file consisting of Python code that can define functions, classes and variables related to a particular task. A module allows us to organize our code by grouping related code, which makes the code easier to understand and use.

Python module may contain the following objects:

Objects	Description
docstring	String literals enclosed with triple quotes that appear right after the definition of a function, class, method or module. It is mainly used for documentation and can be accessed using: (__doc__) attribute.
Variables and constants	For storing values
Classes	A blueprint to create an object
Objects	Object is an instance of class. It represents class in real world.
Statements	Instructions
Functions	Group of statements that performs a specific task

Advantages of Python Modules

Python modules have the following advantages:

1. Putting code into modules is useful because of the ability to **import** the module's functionality.
2. A module can be used in some other Python code. Hence, it provides the facility of code reusability.
3. A module allows us to logically organize our Python code.
4. It creates numbers of well-defined, documented boundaries within a program.
5. Grouping related code into a module makes the code easier to understand and use.
6. Similar types of attributes can be placed in a single module.
7. It reduces the complexity of a program to a large extent.

Python Module Names

A module name is the file name with the .py extension. When we have a file named 'empty.py' empty is the module name. The __name__ is a variable that holds the name of the module being referenced.

For example,

```
>>> import random
>>> random.__name__
'random'
```

The current module, the module being executed (also called the main module), has a special name: '__main__'. With this name, it can be referenced from the Python code.

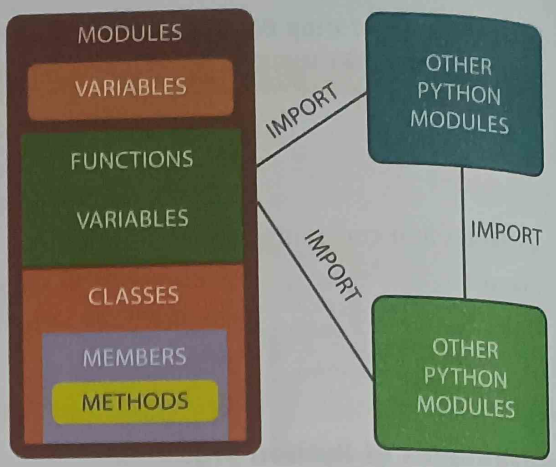
A module's file name should end in .py. If the module's file name does not end with .py as the extension, we will not be able to import it into other programs. A module's name cannot be the same as a Python keyword. Naming a module as 'for', 'while', etc., would result in an error.

CTM: The name of a module is stored inside a constant __ name __.

3.3 IMPORTING PYTHON MODULES

The most common way to create a module is to define a separate file containing the code that we want to group separately from the rest of the application. For example, we might want to create a print routine that an application uses in a number of places. The print routine isn't designed to work on its own but is part of the application as a whole. We want to separate it because the application uses it at numerous places and we could potentially use the same code in another application. The ability to reuse code ranks high on the list of reasons to create modules.

Modules also make it easier to reuse the same code in more than one program. If we have written a set of functions that is needed in several different programs, we can place those functions in a module. Then, we can import the module in each program that needs to call one of the functions. There are different ways by which we can import a module. We can use any Python source file as a module by executing an **import** statement.



Standard library of Python is extended as module(s) to a programmer. Definitions from the module can be used within the code of a program. To use these modules in the program, a programmer needs to import the module into other modules or into the main module using any of the following three constructs:

(i) **import** statement can be used to import a module. It is the simplest and most common way to use modules in our code. It gives access to all attributes (like variables, constants, etc.) and methods or functions present in the module. The syntax to import module in our program is:

```
import <module_name>
```

To import multiple modules, it can be done as:

```
import module1[, module2 [, ...moduleN]
```

To access particular methods from any module, the statement shall be:

```
<module_name>.<function_name>
```

Note: To use/access/invoke a function, we specify the module name and function name separated by a . (dot) . This format is also known as **dot notation**.

(ii) Python's **from** statement lets us import specific attributes or individual objects from a module into the current (calling) module or namespace.

```
from <module_name> import <function_name(s)>
```

Or

```
from <module_name> import function_name1
[,...function_name2[,....function_nameN]]
```

(iii) **from <module name> import*** statement can be used to import all names from a module into the current calling (namespace) program.

```
from <module_name> import*
```

Note: When importing using **from** keyword, do not refer to the module name when referring to the methods or functions. For example,

```
from math import pow, factorial
print(pow(2,3))              // this is correct
print(math.factorial(5)) //this is incorrect, will give an error
```

Let us look at a few simple examples.

Example 1: Let us say we have a Python file called **area.py** which has a few functions in it for calculating the area of a circle, square and rectangle. Use those functions again in some other programs.

area.py - C:/Users/preeti/AppData/Local/Programs/Python/Python...

File Edit Format Run Options Window Help

```
#To develop area module- "area.py"
import math   #importing math library

def circle_area(radius):
    #pi*r*r
    return math.pi*radius*radius

def square_area(side):
    #side * side
    return side*side

def rectangle_area(length,breadth):
    #length * breadth
    return length*breadth
```

Ln: 18 Col: 0

Fig. 3.3(a): Area Module

We have created the "area" module (Fig. 3.3(a)) which is a user-created module, not from the Python library.

However, this program uses the standard math module for using pi() function that belongs to it.

Now, we shall call these modules in another program (Fig. 3.3(b)) named "math1.py".

Note: Always remember that the module area.py should be in the same default root folder where math1.py also resides.

math1.py - C:/Users/preeti/AppData/Local/Programs/Python/Python36-32/math1.py (3.6.5)

File Edit Format Run Options Window Help

```
#Importing the module area in math1.py module

import area

print(area.circle_area(5))          #calling the submodule circle_area from module area
print(area.square_area(10))
print(area.rectangle_area(20,40))
```

Ln: 10 Col: 0

The `import` statement gives access to all attributes and methods present in module.

Methods in a module are accessed using syntax: `module.method()`

Fig. 3.3(b): "math1.py" Module

Python 3.6.5 Shell

File Edit Shell Debug Options Window Help

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:0
7:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for mor
e information.
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs/P
ython/Python36-32/math1.py
78.53981633974483
100
800
>>>
```

Ln: 8 Col: 4

Fig. 3.3(c): Output of Module 'math1.py'

When we will run this program, the sub-modules, namely circle_area(), square_area() and rectangle_area(), shall be called from module area using the statements:

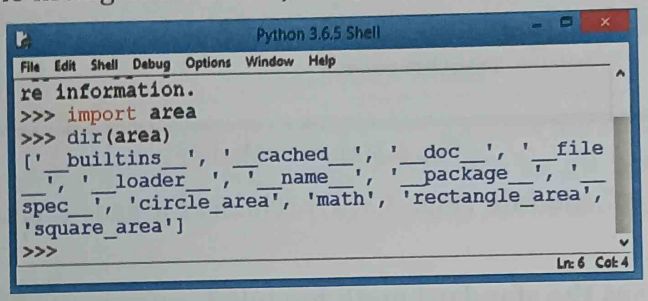
- area.circle_area()
- area.square_area()
- area.rectangle_area()

They shall generate the output as shown in Fig. 3.3(c)).

Retrieving objects from a Module

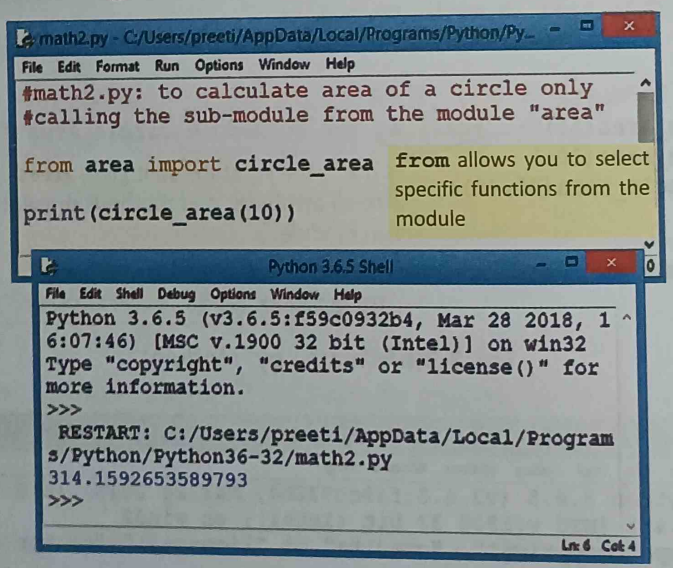
There is another function **dir()** which, when applied to a module, gives you the names of all that is defined inside the module.

Example 2: To retrieve the listing of all the objects present inside the user-created module "area".



As shown above, in order to obtain information about a module (area in this case), the usage of dir() function, with name of the module to be retrieved as the parameter, shall result in the display of all its respective objects, *viz.* name of the sub-modules, variables and constants (if any) present in the module.

Example 3: Modification of Example 1—to calculate the area of the circle only. This can be done using the "from" statement along with the name of the sub-module to be called specifically.



`from <module_name> import <function_name(s)>/<object>` is used to import a particular attribute (sub-routine) from a module. In case you do not want whole of the module to be imported, then you can use `from <module_name> import <function_name(s)>` statement as we have done in the above example. In order to calculate area of the circle, circle_area() function is being called from module "area".

CTM: For modules having a large number of functions, it is recommended to use **from** instead of **import**.

3.4 NAMESPACES IN PYTHON

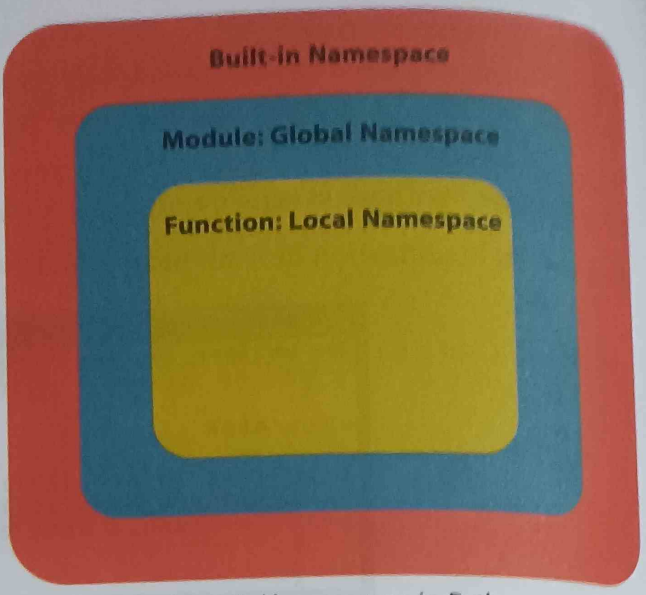


Fig. 3.4: Namespaces in Python

In all of the given statements, we have imported a module into another module or program, which we had referred to as a namespace. Before defining a namespace, it becomes essential to define a name. A Python name is an identifier—something we use to access a **Python object** and, in Python, everything is an object. Thus, a name can hold any kind of value. Namespaces are used to distinguish between different sections of a program. The body of a section may consist of a method, or a function, or all the methods of a class. So, a namespace is a practical approach to define the scope and it helps to avoid name conflicts. A namespace in Python is a collection of names. So, a namespace is essentially a mapping of names to corresponding objects. It ensures that names are unique and won't lead to any conflict.

Also, Python implements namespaces in the form of dictionaries. It maintains a name-to-object mapping where names act as keys and objects as values. Multiple namespaces may have the same name but point to a different variable.

There are three types of Python namespaces—**global**, **local** and **built-in**. It is the same with a variable scope in Python. When Python interpreter starts up, the namespace containing the built-in names is created and is never deleted as long as the interpreter quits. Due to this, Python functions like print() and id() are always available. Also, each module creates its own global namespace in Python.

When we call a function, a local Python namespace is created for all the names in it. A module has a global namespace. The built-in namespace encloses this.

Thus, if we consider examples 1, 3, and 4, where import and from...import statements have been used, it will differ in the allocated namespaces in each case as shown in Fig. 3.5.

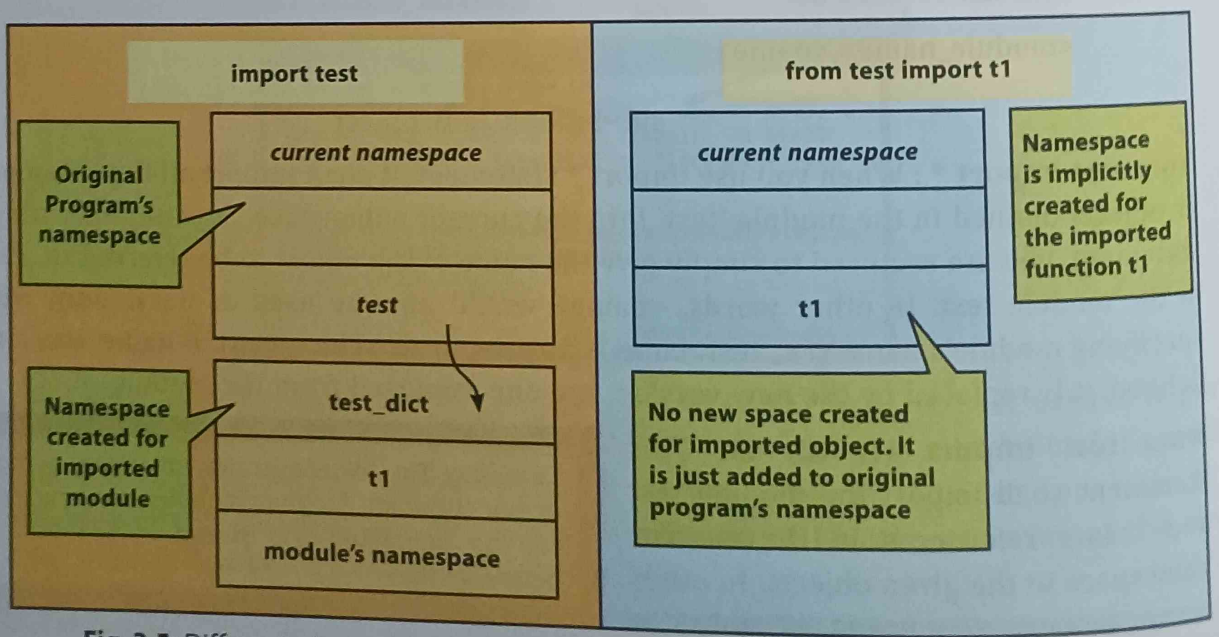


Fig. 3.5: Difference between import <module> and from <module> import commands

POINTS TO REMEMBER

(a) Python uses namespaces to keep track of variables.

(b) A namespace is just like a dictionary where the keys are names of variables and the dictionary values are the values of those variables.

3.5 NAME RESOLUTION (RESOLVING SCOPE OF A NAME)

We have already read in Chapter 2 about the scope of variables in a program, broadly classified as global and local. The entire Python program revolves around the variable scope and their names.

For every name reference within a program, *i.e.*, when you access a variable from within a program or function, Python follows name resolution rule, also known as **LEGB rule**.

That is, for every name reference, Python does the following to resolve it:

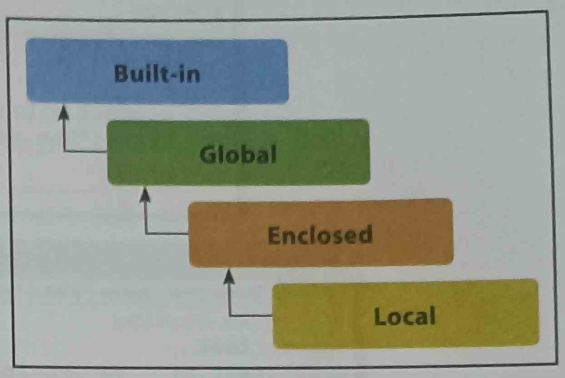


Fig. 3.6: LEGB rule for name resolution

(i) It checks within its **Local** environment (LEGB) or local namespace, if it has a variable with the same name; if yes, Python uses its value. If not, then it moves to step (ii).

(ii) Python now checks the **Enclosing** environment (LEGB) (*e.g.*, whether there is a variable with the same name); if yes, Python uses its value. If the variable is not found in the current environment, Python repeats this step to higher-level enclosing environments, if any. If not, then it moves to step (iii).

(iii) Python now checks the **Global** environment (LEGB) whether there is a variable with the same name; if yes, Python uses its value. If not, then it moves to step (iv).

(iv) Python checks its **Built-in** environment (LEGB), which contains all built-in variables and functions of Python, if there is a variable with the same name; if yes, Python uses its value.

(v) Otherwise, Python would generate the error:

name < variable > is not defined.

Thus, in a nutshell, LEGB rule means checking in the order of Local, Enclosing, Global, Built-in environments (namespaces) to resolve a name reference.

3.6 MODULE ALIASING

You can create an alias when you import a module by using the **as** keyword. Thus, Python provides the salient feature of renaming a module at the time of import.

Syntax:

```
import <module_name> as <alias_name>
```

Practical Implementation–1

Develop a program calculator for performing addition and multiplication operations using modules and importing these modules in the program.

*test.py - C:/Users/preeti/AppData/Local/Programs/...

File Edit Format Run Options Window Help

```
# Program for calculator - test.py

x = 100
def add(a,b):    #Module for addition
    print("The sum is:", a+b)

def product(a,b):  #Module for product
    print("The Product is :",a*b)
```

Ln: 11 Col: 0

prog_test1.py - C:/Users/preeti/AppData/Local/Programs/Python/Pyt...

File Edit Format Run Options Window Help

```
# Program testing
import test
print(test.x)    #accessing the value of variable x
test.add(10,20)
test.product(10,20)
```

```
>>>
 RESTART: C:/Users/preeti/AppData/L
ocal/Programs/Python/Python36-32/pr
og_test1.py
100
The sum is: 30
The Product is : 200
>>>
```

Ln: 8 Col: 4

Practical Implementation–2

Modify the program in Practical Implementation–1 by using alias name for the module "test".

prog_test2.py - C:/Users/preeti/AppData/Local/Programs/Python/Python36-32/prog_test2.py (3.6.5)

File Edit Format Run Options Window Help

```
# Program testing using alias module name

import test as m   # m is the alias name for module test
print(m.x)   #using alias name for accessing the value of variable x
m.add(10,20)
m.product(10,20)   #alias name for calling the functions of the module
```

Ln: 11 Col: 0

```
>>>
 RESTART: C:/Users/preeti/AppData/Local
/Programs/Python/Python36-32/prog_test2
.py
100
The sum is: 30
The Product is : 200
>>>
```

Ln: 13 Col: 4

As is evident from the same output obtained from the above two programs, we can say that alias name works well with Python modules.

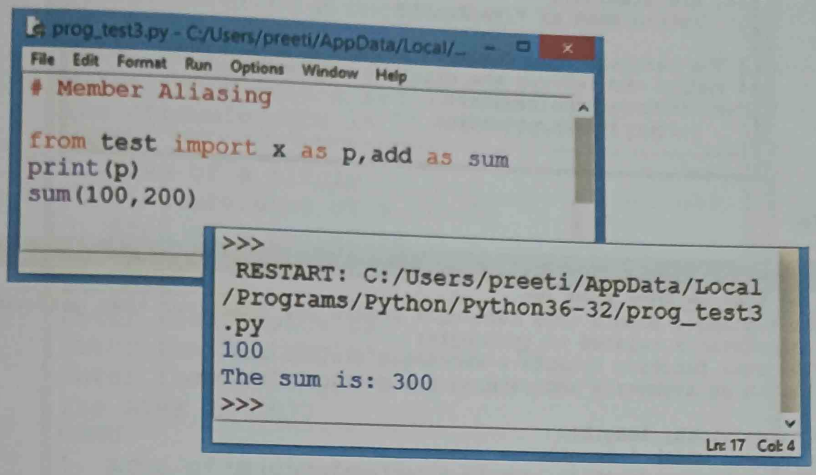
Here, **test** is the original module name and **m** is the alias name. We can easily access members of the main module by using the alias name **m**.

3.7 MEMBER ALIASING

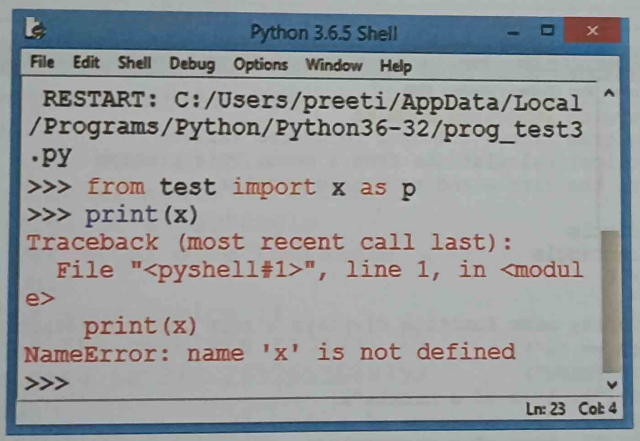
Like module aliasing, member aliasing is also supported and implemented in Python modules.

Practical Implementation–3

Modify the previous program for implementing aliasing with the members of the module.



Once we have defined an alias name, we should use alias name only and use of original module name should be avoided as it may create ambiguity and may result in an unexpected output or sometimes even an error, as shown below.



Practical Implementation–4

Write a program in Python that calculates the following:

- Area of a circle
- Circumference of a circle
- Area of a rectangle
- Perimeter of a rectangle

Create respective modules for each of the operations and call them separately using a menu-driven program.

There are obviously two categories of calculations required in this program: those related to circles, and those related to rectangles. You could write all circle-related functions in one module and all rectangle-related functions in another module.

Circle Module

*circle.py - C:/Users/preeti/AppData/Local/Programs/Python/Python36-32/ci...

File Edit Format Run Options Window Help

```
#The circle module has functions that perform
#calculations related to circles - circle.py.
import math

# The area function accepts a circle's radius as an
# argument and returns the area of the circle.
def area(radius):
    return math.pi * radius**2

# The circumference function accepts a circle's
# radius and returns the circle's circumference.
def circumference(radius):
    return 2*math.pi*radius
```

Ln: 17 Col: 0

Rectangle Module

rectangle.py - C:/Users/preeti/AppData/Local/Programs/Python/Python36-32/rectangle.py (3.6.5)

File Edit Format Run Options Window Help

```
#the Rectangle module -The rectangle module has functions that perform
# calculations related to rectangles.
# The area function accepts a rectangle's width and
# length as arguments and returns the rectangle's area.

def area(width, length):
    return width * length
# The perimeter function accepts a rectangle's width
# and length as arguments and returns the rectangle's
# perimeter.

def perimeter(width, length):
    return 2 * (width + length)
```

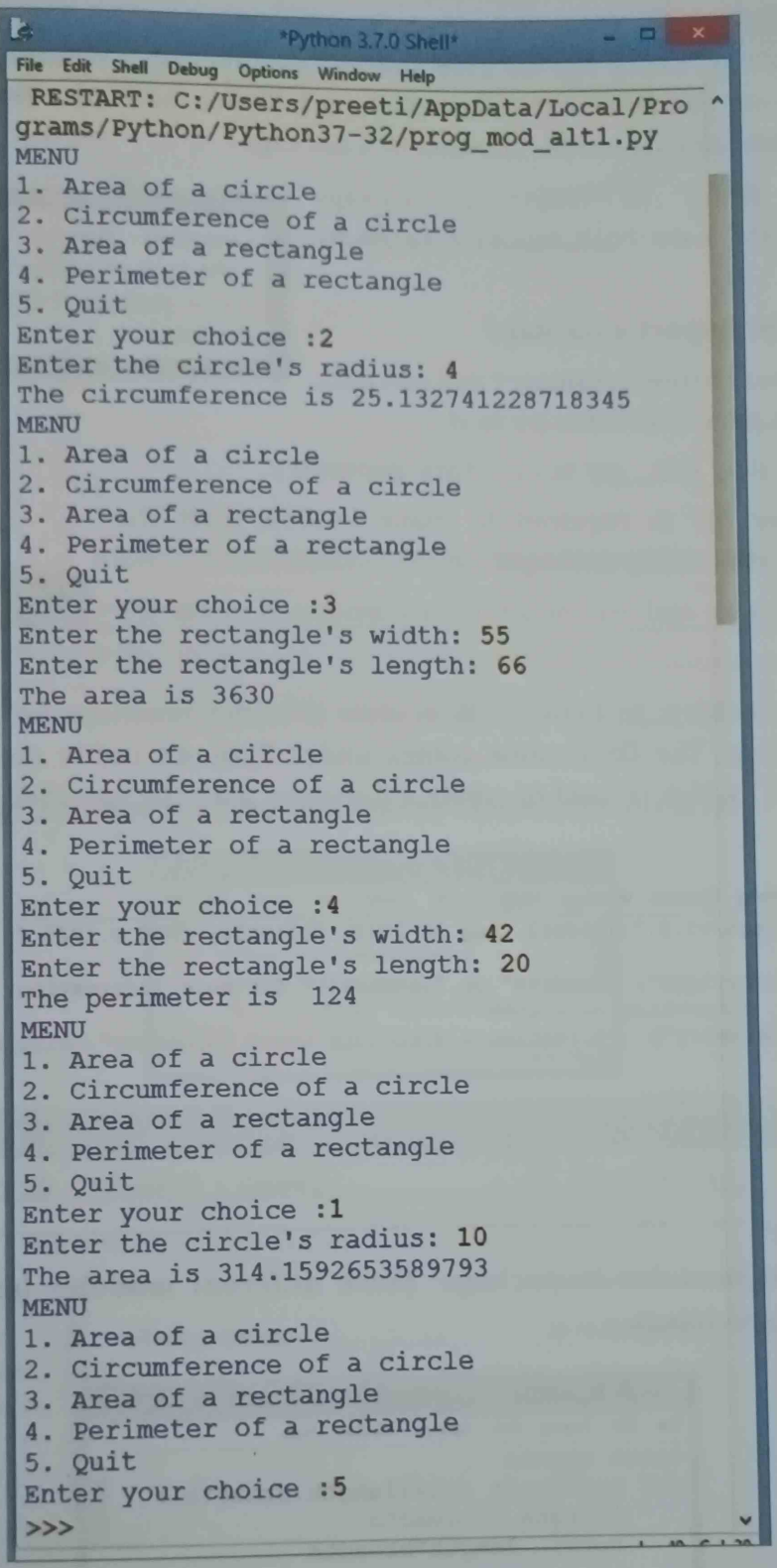
Ln: 16 Col: 0

prog_mod_alt1.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_mo...

File Edit Format Run Options Window Help

```
#Main Program importing modules
#This program allows the user to choose various
# geometrical calculations from a menu. This program
# imports the circle and rectangle modules.

import circle
import rectangle
choice = 0
ch = "y"
# The display-menu function displays a menu upon execution.
while (ch == "y"):
    print("MENU")
    print("1. Area of a circle")
    print("2. Circumference of a circle")
    print("3. Area of a rectangle")
    print("4. Perimeter of a rectangle")
    print("5. Quit")
    choice = int(input ("Enter your choice :"))
    if (choice == 1):
        radius = int(input("Enter the circle's radius: "))
        print("The area is",circle.area(radius))
    elif (choice == 2):
        radius = int(input("Enter the circle's radius: "))
        print("The circumference is",circle.circumference(radius)
    elif (choice == 3):
        width = int(input("Enter the rectangle's width: "))
        length = int(input("Enter the rectangle's length: "))
        print('The area is',rectangle.area(width, length))
    elif (choice == 4):
        width = int(input("Enter the rectangle's width: "))
        length = int(input("Enter the rectangle's length: "))
        print('The perimeter is ',rectangle.perimeter(width, length))
    elif (choice == 5):
        print('Exiting the program ...')
    else:
        print('Error: invalid selection.')
```

Ln: 40 Col: 4



3.8 PACKAGE/LIBRARY

Python package is a collection of related modules. You can either import a built-in package or create your own.

CTM: The main difference between a module and a package is that a package is a collection of modules and has an __init__.py file.

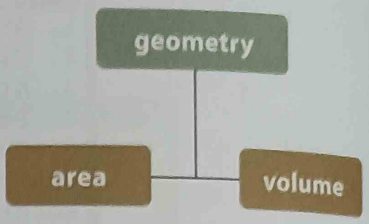
A Python package is simply a directory of Python module(s). In Example 1, we have created one module named **area.py**. Let us say we have another module called **volumes.py** which has a few functions to compute volumes of the various shapes. Hence, **area.py** and **volume.py** are both related to each other and should be bundled in a package.

Thus, we are going to create a package **geometry** which will have both modules **(area.py and volume.py)**.

> **Learning Tip:** A package is simply a collection of similar modules, sub-packages, etc. A package and a Library are technically the same.

Steps to create and import a package

1. Create a directory named **geometry** and add both modules **area.py** and **volume.py** in it.
2. Also create a file **__init__.py** in directory **geometry**.
3. The **__init__.py** file is required to make Python treat the directories as containing packages.
4. Import the package and use the attributes contained inside the package.

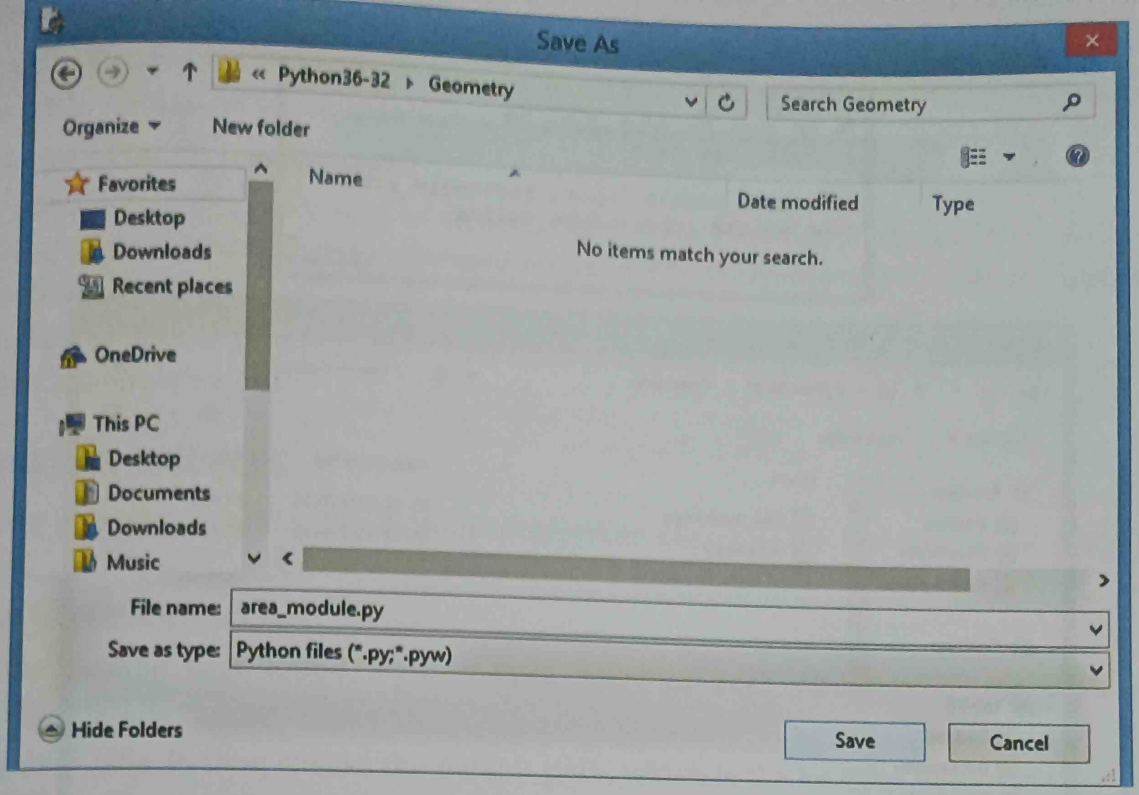


1. **Create the directory:** In Python, OS module provides functions for interacting with the operating system. The OS module comes under Python's utility modules. It provides a method mkdir() which is used to create a directory with the specified name.

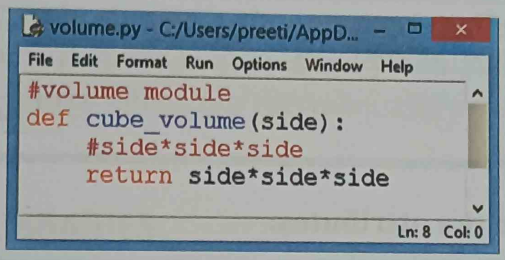
```
IDLE Shell 3.9.7
File Edit Shell Debug Options Window Help
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on w
in32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import os   #importing os module
>>> os.mkdir("Geometry")   #creating a directory named "Geometry" using mkdir() function
>>>
                                                                    Ln: 5  Col: 4
```

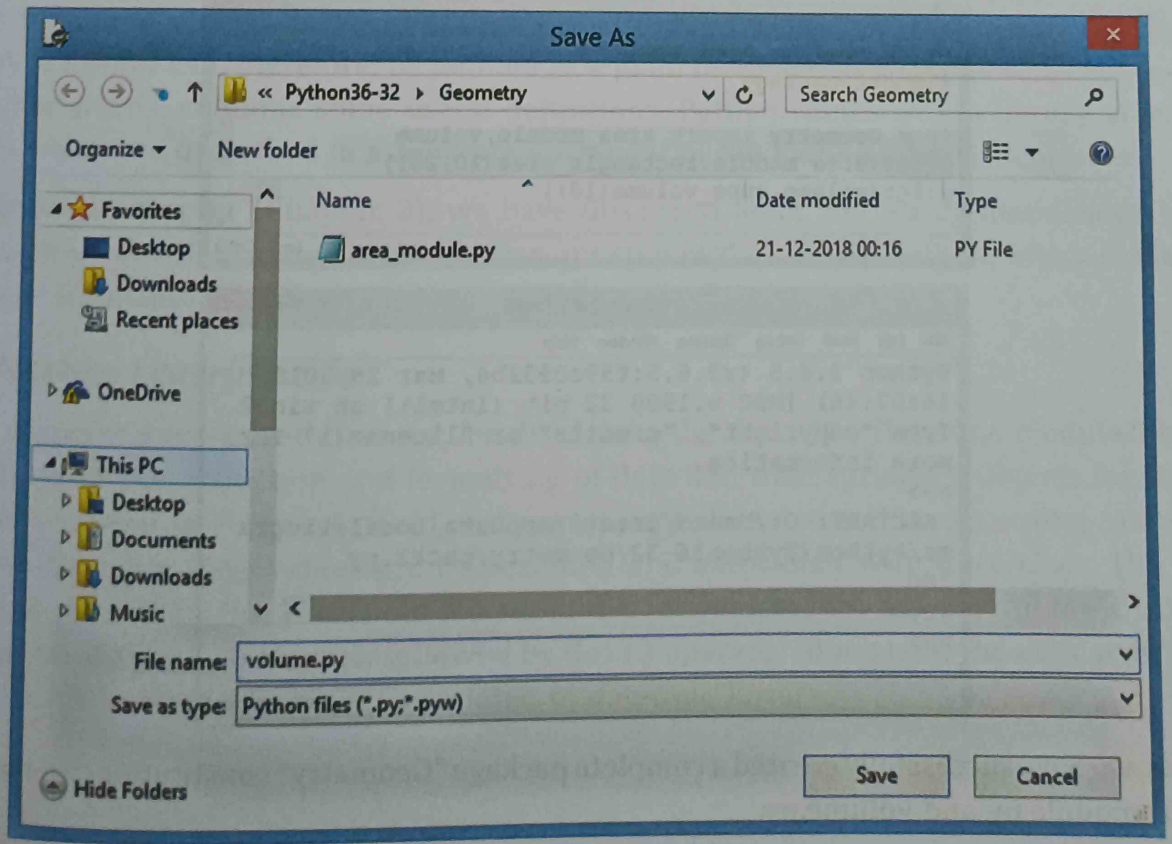
2. **Place different modules in package: (Save different modules inside the Geometry package)** area_module.py

```
area_module.py - C:/Users/preeti/AppData/Lo...
File Edit Format Run Options Window Help
#area module
def rectangle_area(length,breadth):
    #length * breadth
    return length*breadth
                                                                    Ln: 9  Col: 0
```

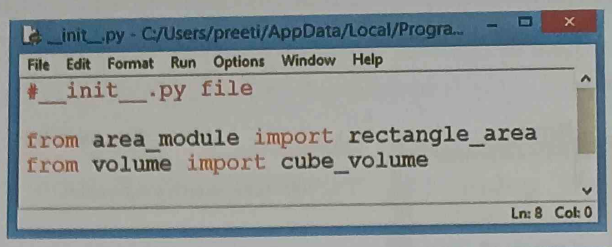


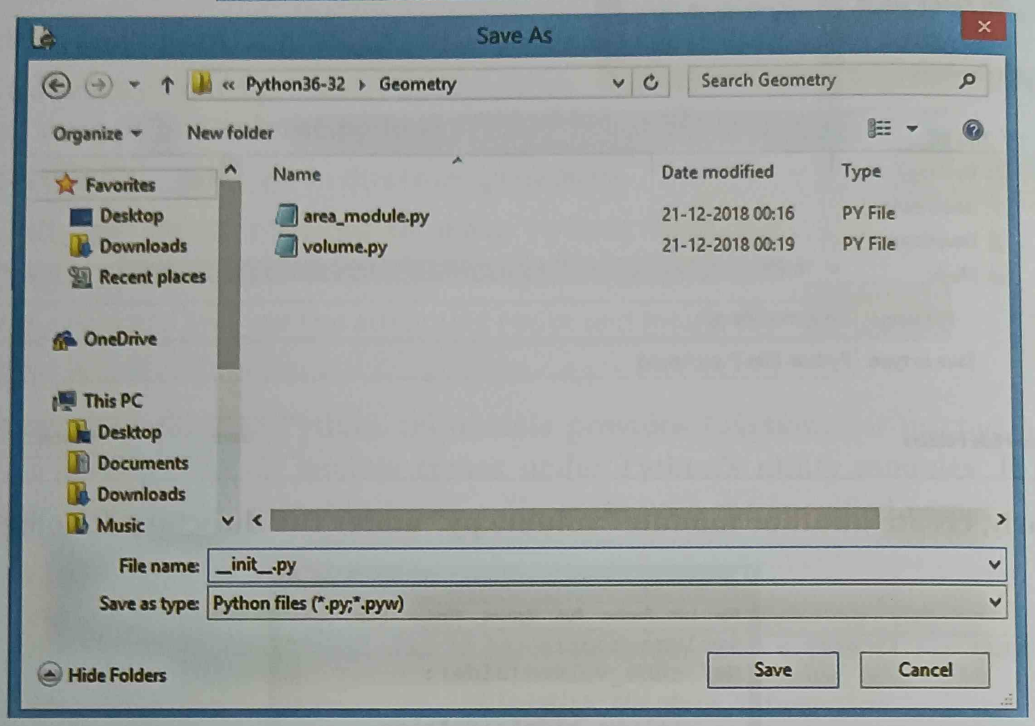
Similarly, create another module "volume.py" under the directory "Geometry".



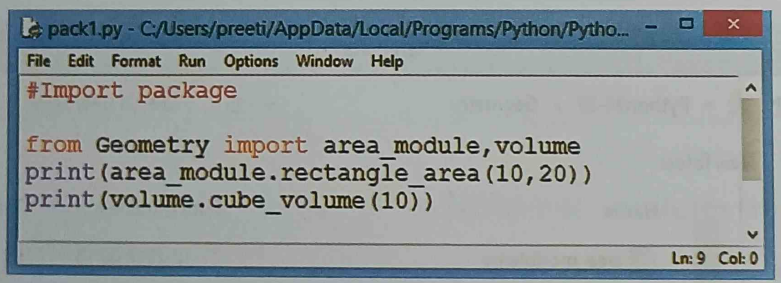


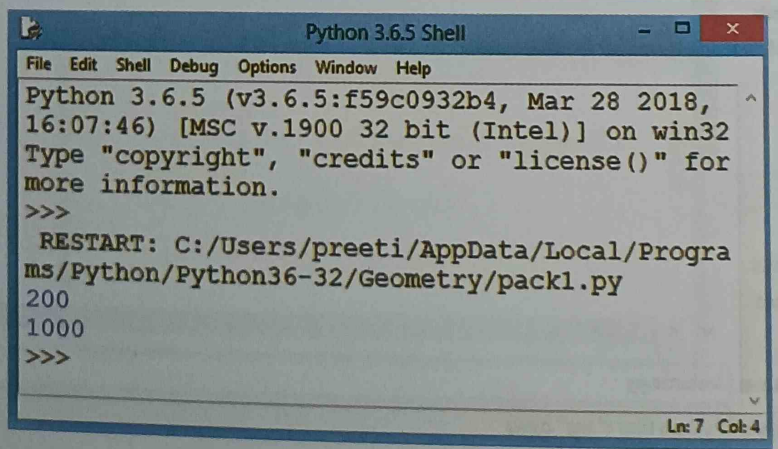
3. **Create __init__.py file:**





4. **Import package and use the attributes:**





Thus, we have successfully created a complete package "Geometry" constituting two modules—area_module.py and volume.py.

What is _init_.py file

init.py is simply a file that is used to consider the directories on the disk as package of Python. It is basically used to initialize the Python packages.

Advantages of Libraries/Packages

1. Libraries/Packages are a way of structuring Python's module namespace by using "dotted module names". For example, the module name X.Y designates a sub-module named Y in a package named X.
2. The use of dotted module names saves the programmers from having to worry about each other's module names.

3.9 LOCATING MODULES

When you import a module, the Python interpreter searches for the module in the following sequence:

- The current directory.
- If the module isn't found, Python then searches each directory in the shell variable PYTHONPATH.
- If all else fails, Python checks the default path, which is the location where Python has been installed on your computer system.

Thus, PYTHONPATH is the **variable** that tells the interpreter where to locate the module files imported into a program. Hence, it must include the Python source library directory and the directories containing Python source code. You can manually set PYTHONPATH, but usually the Python installer will preset it.

3.10 PYTHON STANDARD LIBRARIES

By now, it must be clear that Libraries or Packages in Python are the same entities constituting modules. A library can, therefore, be defined as a main module that contains some sub-modules or, in other words, contains some useful definitions. Python comes with a library of standard modules which are described in a separate document in the Python Library Reference.

In the previous chapter (Chapter 2), we have discussed Math and String modules or, in turn, libraries. Now we will discuss another important standard library of Python, which is **Datetime Library** or **Datetime Module**.

3.10.1 Datetime Library/Module

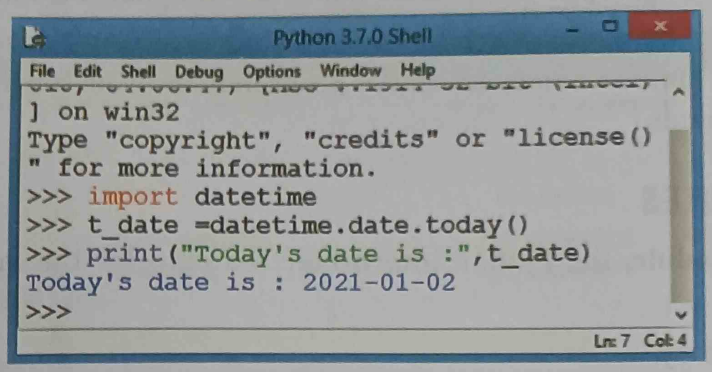
Python supports **yyyy-mm-dd** format for displaying the date. The Datetime module/library in Python handles the extraction and formatting of date and time variables. Objects for Datetime library are created in Python to hold and represent a date and a time according to a specific time zone. Calendar date values are represented and associated with **date** class. The objects (instances) created for this class hold the attributes for year, month and day. All these attributes are accessed using the class name followed by dot (.) operator along with the date object. Let us now discuss few important functions related to datetime library.

date class

- **today():** This method of date class with datetime module is used to create a date representing today's date or the current system date. The syntax is:

```
import datetime
t_date=datetime.date.today()
```

For example,



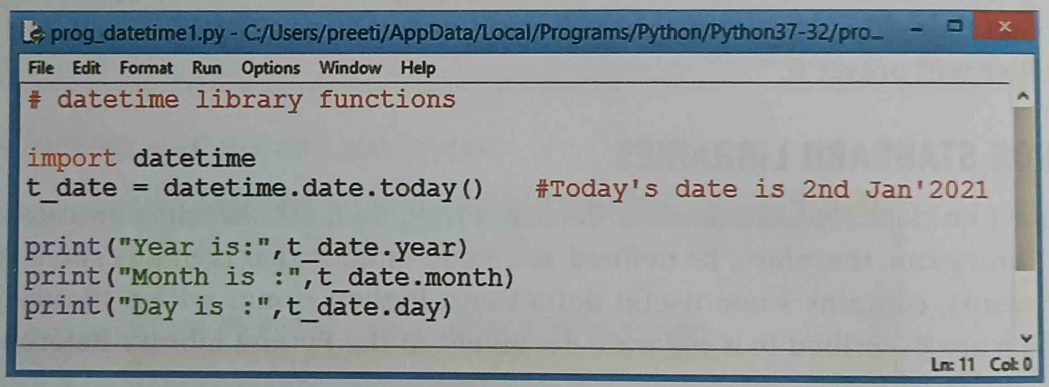
- **year:** This attribute returns and displays year from the datetime object.

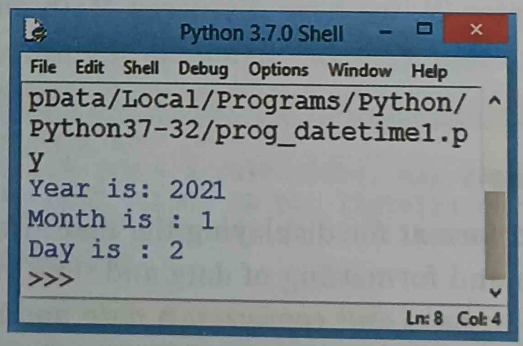
For example,

```
import datetime
t_date = datetime.date.today()
print("Year is:",t_date.year)
```

Output:

```
Year is: 2021
```





- **month:** This attribute returns and displays month from the datetime object.

For example,

```
print("Month is:",t_date.month)
```

Output:

```
Month is: 1
```

- **day:** This attribute returns and displays day from the datetime object.

  For example,

```
print("Day is:",t_date.day)
```

  Output:

```
Day is: 2
```

  This was all about datetime library functions for date class. Now, we will discuss time values which are represented with the **datetime** and **time** class.

time class

time class holds attributes for hour, minute, second and microsecond, and can be accessed through dot (.) operator along with **time** object. It can also include time zone information for each associated zone respectively.

- **now():** This method returns the current date and time using datetime library/module. The syntax is:

```
import datetime
t_date = datetime.datetime.now()
```

  For example,

```
========== RESTART: Shell ====
==========
>>> import datetime
>>> t_date= datetime.datetime.now()
>>> print("Today is:",t_date)
Today is: 2021-01-02 11:47:23.302276
>>>
                                Ln: 14  Col: 4
```

- **hour:** This attribute returns and displays hours from the datetime object. For example,

```
print("Hours are:",t_date.hour)
```

  Output: Hours are: 11

```
prog_datetime2.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_da...
File  Edit  Format  Run  Options  Window  Help
#time class

import datetime
t_date= datetime.datetime.now()  #Right now the time is: 11:58:18
print("Hours are :",t_date.hour)
print("Minutes are :",t_date.minute)
print("Seconds are :",t_date.second)
                                Ln: 11  Col: 0
```

```
Python 3.7.0 Shell
File  Edit  Shell  Debug  Options  Window  Help
/AppData/Local/Programs/P
ython/Python37-32/prog_da
tetime2.py
Hours are : 11
Minutes are : 58
Seconds are : 18
>>>
                                Ln: 8  Col: 4
```

- **minute:** This attribute returns and displays minutes from the datetime object.

  For example,

```
print("Minutes are:",t_date.minute)
```

  Output:

```
Minutes are: 58
```

- **second:** This attribute returns and displays seconds from the datetime object.

  For example,

```
print("Seconds are:",t_date.second)
```

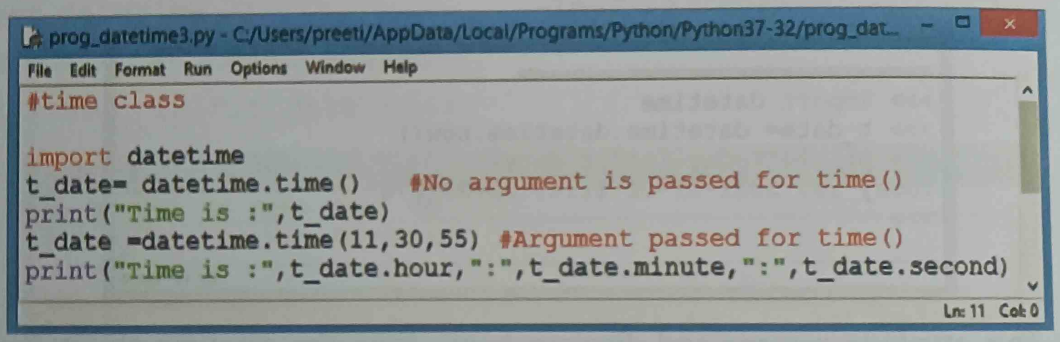
  Output:

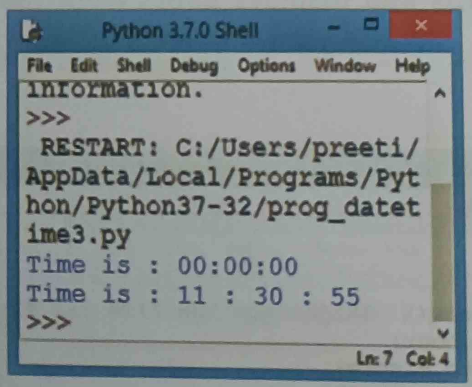
```
Seconds are: 18
```

Passing arguments to time class

We can also apply this class for user-defined arguments. If no arguments are passed, the output shall display the value as 0 for time() function; otherwise it will display the value for hours, minutes and seconds from the arguments respectively.

For example,





3.11 USING PYTHON STANDARD LIBRARY'S FUNCTIONS AND MODULES

Python's standard library is very extensive, offering a wide range of modules and functions. The library contains built-in modules (written in C) that provide access to system functionality such as file I/O, that would otherwise be inaccessible to Python programmers, as well as modules written in Python that provide standardized solutions for many problems that occur in everyday programming. Some of these important modules are explicitly designed to encourage and enhance the portability of Python programs, such as **math** module for mathematical functions; **random** module for pseudo-random number generation; and **string** module for string/text-related functions.

We have already discussed important math and string modules and their functions.

Let us talk about some more built-in mathematical functions.

3.11.1 Python's built-in Mathematical Functions

We have already discussed major functions using **math** module. Python provides many other mathematical built-in functions as well. These include trigonometric functions, representation functions, logarithmic functions, angle conversion functions, etc. In addition, two mathematical constants are also defined in this module.

Pie (π): Pie (π) is a well-known mathematical constant, which is defined as the ratio of the circumference to the diameter of a circle and its value is 3.141592653589793. To import this single object (pie) from the **math** module:

```
>>> import math
>>> math.pi
3.141592653589793
```

Alternatively, it can be done without prefixing the module name and writing the name of the object directly after keyword **import** as shown below:

```
>>> from math import pi
>>> print(pi)
3.141592653589793
```

Similarly, another well-known mathematical constant defined in the math module is **e**. It is called **Euler's number** and it is a base of the natural logarithm. Its value is 2.718281828459045.

```
>>> math.e
2.718281828459045
```

Hence, these constants can be directly called and used from math module.

We should remember not to 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.

The math module contains functions for calculating various trigonometric ratios for a given angle. The functions (sin, cos, tan, etc.), which we have already discussed, need the angle in radians as an argument. We, on the other hand, are used to express the angle in degrees.

The math module presents two-angle conversion functions — **degrees()** and **radians()** — to convert the angle from degrees to radians and vice versa.

degrees() converts a value in radians to degrees using the formula: $\text{degrees}(x) = \frac{\pi}{180} x$

On the contrary, **radians()** converts a value in degrees to radians: $\text{radians}(x) = \frac{180}{\pi} x$

POINT TO REMEMBER

π radian is equivalent to 180 degrees.

For example,

To convert an imputed angle of 50 degrees to radians and vice versa.

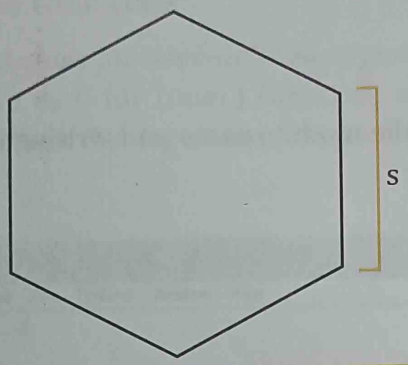
```
prog_rad_degree1.py - C:\Users\preeti\AppData\Local\Programs\Python\Pytho...
>>>
 RESTART: C:\Users\preeti\AppData\Local\Programs\Python\Python37-32\
ree1.py
Conversion of degree to radians 0.8726646259971648
Conversion of radians to degree 2864.7889756541163
                                                              Ln: 6  Col: 0
```

Practical Implementation–5

Write a Python program to calculate the area of a regular polygon.

For solving this problem, we have to understand the formula to be used first, which is given as:

Area of a Regular Polygon



n = number of sides

s = length of a side

$$\text{Area} = \frac{ns^2}{4 \tan\left(\frac{\pi}{n}\right)}$$

```
prog_area_polygon.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_area...
File  Edit  Format  Run  Options  Window  Help
#Python program to calculate the area of a regular polygon

from math import tan, pi
no_sides = int(input("Input number of sides: "))
side_length = float(input("Input the length of a side: "))
poly_area = no_sides * (side_length ** 2) / (4 * tan(pi / no_sides))
print("The area of the polygon is: ",poly_area)
                                                              Ln: 16  Col: 0
```

```
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs/Python
lygon.py
Input number of sides: 6
Input the length of a side: 10
The area of the polygon is:  259.8076211353316
```

Practical Implementation–6

Write a Python program to find the roots of a quadratic function.

A quadratic equation is represented as: $ax^2 + bx + c = 0$

Here, a, b and c are numbers, where $a \neq 0$; and

x is to be calculated.

The **roots** of any **quadratic equation** are given by the formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

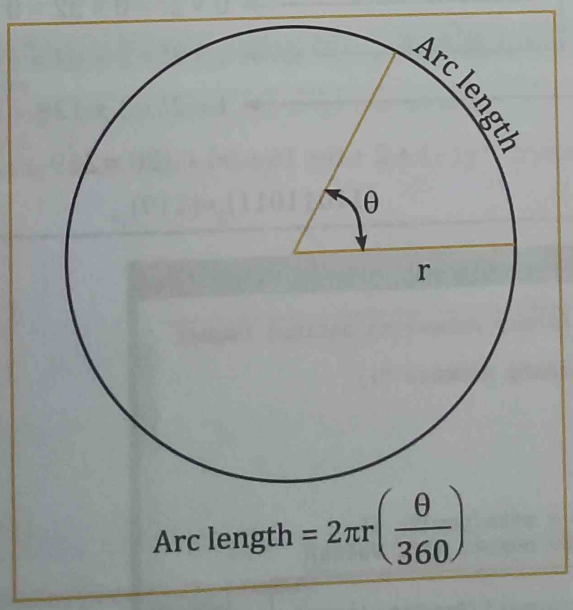
```
prog_quad_root.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_quad_ro
File Edit Format Run Options Window Help
#Program to find the roots of a quadratic function
from math import sqrt

print("Quadratic equation is : (a * x^2) + b*x + c")
a = float(input("a: "))
b = float(input("b: "))
c = float(input("c: "))

r = b**2 - 4*a*c

if r > 0:
    num_roots = 2
    x1 = (((-b) + sqrt(r))/(2*a))
    x2 = (((-b) - sqrt(r))/(2*a))
    print("The two roots are: ", x1)
    print("and",x2)
elif r == 0:
    num_roots = 1
    x = (-b) / 2*a
    print("There is one root: ", x)
else:
    num_roots = 0
    print("No roots, discriminant < 0.")
```

```
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs/Python
ot.py
Quadratic equation is : (a * x^2) + b*x + c
a: 6
b: 25
c: 15
The two roots are:  -0.7267649503250245
and -3.4399017163416423
```

Practical Implementation–7

Write a Python program to calculate arc length of an angle.

In planar geometry, an angle is a figure formed by two rays called the sides of the angle, sharing a common endpoint known as the vertex of the angle. Angles formed by two rays lie in a plane, but this plane does not have to be a Euclidean plane.



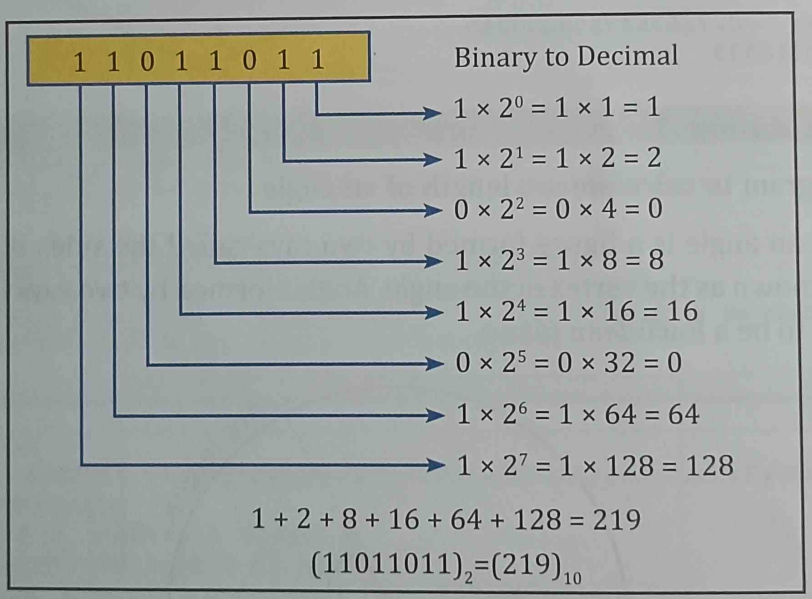
```
prog_arc_angle1.py - C:/Users/preeti/AppData/Local/Programs/Python/...
File Edit Format Run Options Window Help
#Python program to calculate arc length of an angle

import math
def arclength():    #user-defined function
    diameter = float(input('Diameter of circle: '))
    angle = float(input('angle measure: '))
    if angle >= 360:
        print("Angle is not possible")
        return
    arc_length = 2 * (math.pi*diameter/2) * (angle/360)
    print("Arc Length is: ", arc_length)

arclength()
```

```
 RESTART: C:/Users/preeti/AppData/Local/Programs/Pytho
le1.py
Diameter of circle: 20
angle measure: 45
Arc Length is:  7.853981633974483
```

Practical Implementation–8

Write a Python program to convert a binary number to decimal number.

A binary number is a number which is represented in the form of 0 and 1 (binary digits) only and decimal number system, the contemporary number system used mathematically, constitutes numbers from 0 to 9.Also, a binary number system has 2 as its base while the decimal system has 10 as its base. The method of conversion of a binary number to its decimal equivalent is:




```
prog_bin_decimal.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32...
File Edit Format Run Options Window Help
#Python program to convert a binary number to decimal number
import math
b_num = list(input("Input a binary number: "))
value = 0

for i in range(len(b_num)):
        digit = b_num.pop()
        if digit == '1':
                value = value + math.pow(2, i)
print("The decimal value of the number is", value)
```

```
>>>
 RESTART: C:/Users/preeti/AppData/Local/
/Python/Python37-32/prog_bin_decimal.py
Input a binary number: 11011011
The decimal value of the number is 219
```

In the above program, the input taken from the user is converted to a list using the function list() and gets stored inside variable b_num. The loop gets executed up to the length of the list, *i.e.*, the number of list elements. With every iteration, each digit is extracted or popped from the list using function pop(). This extracted digit is checked for character as 1 and is raised to the power of 2. Finally, the value gets printed which is in decimal format. In case the digit is 0, no execution takes place since the product of any value with 0 results in 0 itself. Hence, the output is obtained as 219.

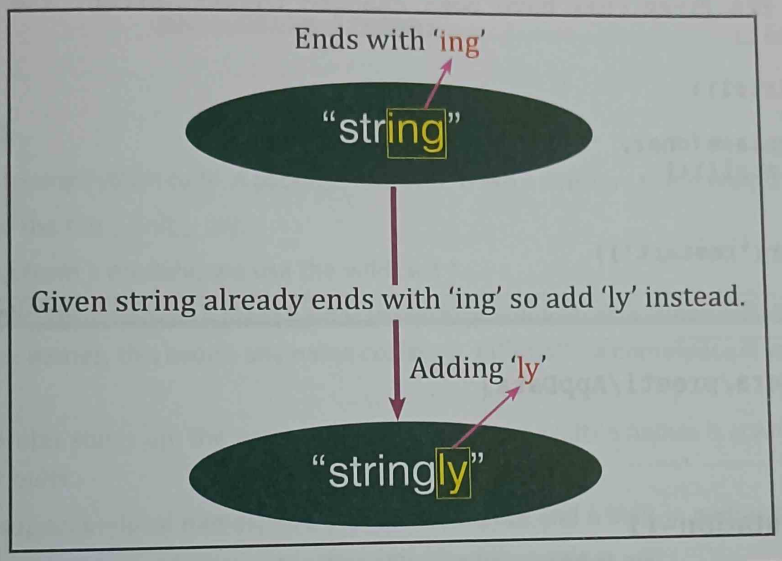
3.11.2 Python's built-in String Functions

We have already discussed built-in String functions in the previous chapters.

Let us now see some more practical implementations of most commonly used inbuilt string functions.

Practical Implementation–9

Write a Python function to add 'ing' at the end of a given string (length should be at least 3). If the given string already ends with 'ing' then add 'ly' instead. If the string length of the given string is less than 3, leave it unchanged.



prog_ch3_12cs1.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_ch3_12cs1....

File Edit Format Run Options Window Help

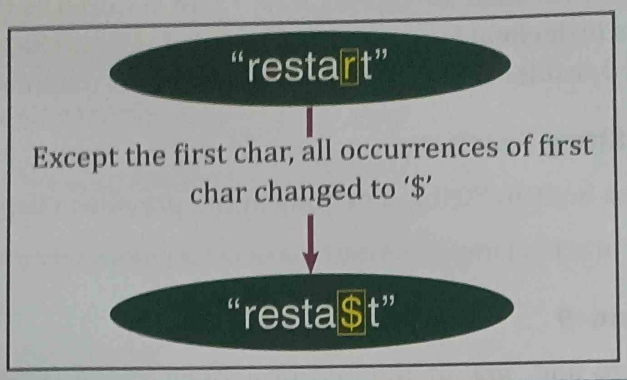
```
#Python function to add 'ing' at the end of a given string
#(length should be at least 3)
#If the given string already ends with 'ing' then add 'ly' instead.
def add_string(str1):
  length = len(str1)

  if length > 2:
    if str1[-3:] == 'ing':
      str1 += 'ly'
    else:
      str1 += 'ing'

  return str1
print(add_string('xy'))
print(add_string('xyz'))
print(add_string('string'))
```

Ln: 21 Col: 0

```
>>>
 RESTART: C:/Users/preeti/AppData
s1.py
xy
xyzing
stringly
```

Practical Implementation–10

Write a Python function to get a string from a given string where all occurrences of its first char have been changed to '$', except the first char itself.




```
prog_11c_strreplce1.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_11c_strre...
File Edit Format Run Options Window Help
#Write a Python function to get a string from a given string where all
#occurrences of its first char have been changed to '$', except the
#first char itself.

def change_char(str1):
  char = str1[0]
  str1 = str1.replace(char, '$')
  str1 = char + str1[1:]
  return str1

print(change_char('restart'))
                                                                Ln: 16 Col: 0
```

```
>>>
 RESTART: C:/Users/preeti/AppData/
replce1.py
resta$t
```

Practical Implementation–11

Write a Python program to capitalize first and last letters of each word of a given string.

```
prog_capitalize2.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_capitalize2.p...
File Edit Format Run Options Window Help
#Python program to capitalize first and last letters of
#each word of a given string
def capitalize_first_last_letters(str1):
     str1 = result = str1.title()
     result = ""
     for word in str1.split():
        result += word[:-1] + word[-1].upper() + " "
     return result[:-1]

print(capitalize_first_last_letters("python string library functions"))
print(capitalize_first_last_letters("implementation"))
                                                                Ln: 14 Col: 0
```

```
>>>
 RESTART: C:/Users/preeti/AppData/Local
ize2.py
PythoN StrinG LibrarY FunctionS
ImplementatioN
```

Practical Implementation–12

Write a Python program to swap each character of a given string from lowercase to uppercase and vice versa.

```
#program to swap each character of a given string
#from lowercase to uppercase and vice-versa
def swap_case_string(str1):
    result_str = ""
    for item in str1:
        if item.isupper():
            result_str += item.lower()
        else:
            result_str += item.upper()
    return result_str
print(swap_case_string("String Module in"))
print(swap_case_string("Python"))
print(swap_case_string("Matplotlib Library"))
```

```
>>>
 RESTART: C:/Users/preeti/AppData
se.py
sTRING mODULE IN
pYTHON
mATPLOTLIB lIBRARY
```

MEMORY BYTES

- A module is a file containing Python code. A package, however, is like a directory that holds sub-packages and modules.
- A package must hold the file __init__.py.
- To import everything from a module, we use the wildcard *.
- A namespace is a collection of names. It maps names to corresponding objects. When different namespaces contain objects with the same names, this avoids any name collisions. Internally, a namespace is implemented as a **Python dictionary**.
- When Python interpreter starts up, the namespace containing the built-in names is created and is never deleted until the interpreter quits.
- We have three namespaces—local namespace, global namespace and a built-in namespace.
- A module is a separately saved unit whose functionality can be reused at will.
- A Python module can contain objects like docstrings, variables, constants, classes, objects, statements, functions and has .py extension.
- A Python module can be imported in a program using import statement.
- There are two forms of import statements:
  - `import <modulename>`
  - `from <module> import <object>`
- The random module of Python provides random number generation functionality.
- A math module of Python provides math functionality.
- By default, Python names the segment with top-level statements (main program) as __main__.
- Python resolves the scope of a name using LEGB rule, *i.e.*, it checks environments in the order: Local, Enclosing, Global and Built-in.
- A library refers to a collection of modules that together cater to a specific type of needs or applications.
- The **urllib** module lets you send and receive http request and their results and web browser lets you open a web page from within a Python program.
- A package installed or attached to site-packages folder of Python installation can easily be imported using import command.

OBJECTIVE TYPE QUESTIONS

1. Fill in the blanks.

(a) Commonly-used modules that contain source code for generic needs are called ..... .

(b) A Python ..... is a directory of Python module(s).

(c) We can use any Python source file as a module by executing an ..... statement.

(d) ..... function, when applied to a module, gives you the names of all the components (variables, constants, functions and sub-modules) defined inside the module.

(e) The ..... is a variable that holds the name of the module being referenced.

(f) A ..... is a separately saved unit whose functionality can be reused at will.

(g) The random module of Python provides ..... functionality.

(h) ..... are used to distinguish between different sections of a program.

(i) Python provides three types of namespaces—....., ... and ..... .

(j) Python follows name resolution rule, also known as ..... rule.

(k) ..... function is used to get all the information about a module, *i.e.*, name of all functions, variables, etc., available in that module.

2. State whether the following statements are True or False.

(a) A library in Python is a collection of various packages.

(b) Modules are used to categorize Python code into smaller parts.

(c) The docstrings are useful for compilation of the code.

(d) Python uses namespaces for documentation.

(e) Module aliasing is carried out by using the as keyword.

(f) Python does not allow creating own packages.

(g) A library can have multiple modules in it.

(h) Modularity increases the complexity of a Python program.

(i) A package in Python must hold the file __init__.py.

(j) A package and a library are technically not the same.

3. Multiple Choice Questions (MCQs)

(a) What will be the output of the following Python code?

```
from math import factorial
print(math.factorial(5))
```

(i) 120

(ii) Nothing is printed

(iii) Error, method factorial doesn't exist in math module

(iv) Error, the statement should be: print(factorial(5))

(b) What is the order of namespaces in which Python looks for an identifier?

(i) Python first searches the global namespace, then the local namespace and finally the built-in namespace

(ii) Python first searches the local namespace, then the global namespace and finally the built-in namespace

(iii) Python first searches the built-in namespace, then the global namespace and finally the local namespace

(iv) Python first searches the built-in namespace, then the local namespace and finally the global namespace

(c) Which of the following is not a valid namespace?

(i) Global namespace

(ii) Public namespace

(iii) Built-in namespace

(iv) Local namespace

(d) What will be the output of the following Python code?

```
#module1
def change(a):
    b=[x*2 for x in a]
    print(b)
#module2
def change(a):
    b=[x*x for x in a]
    print(b)
from module1 import change
from module2 import change
#main
s=[1,2,3]
change(s)
```

(i) [2,4,6]
(ii) [1,4,9]
(iii) [2,4,6]
[1,4,9]
(iv) There is a name clash

(e) What is the output of the function shown below (random module has already been imported)?
random.choice('sun')
(i) sun (ii) u (iii) either s, u or n (iv) error

(f) What possible output(s) are expected to be displayed at the time of execution of the following code?

```
import random
AR=[20,30,40,50,60,70]
FROM=random.randint(1,3)
TO=random.randint(2,4)
for K in range(FROM,TO+1):
  print(AR[K],end="#")
```

(i) 20#40#70# (ii) 30#40#50# (iii) 50#60#70# (iv) 40#50#70#

(g) Which of the statements is used to import all names from a module into the current calling module?
(i) import (ii) from (iii) import * (iv) dir()

(h) Which of the variables tells the interpreter where to locate the module files imported into a program?
(i) local (ii) import variable (iii) PYTHONPATH (iv) current

(i) Which of the following date class function returns the current system date?
(i) day() (ii) today() (iii) month() (iv) year()

(j) What is the range of values that random.random() can return?
(i) [0.0, 1.0] (ii) (0.0, 1.0] (iii) (0.0, 1.0) (iv) [0.0, 1.0)

(k) What will be the output of the following code?

```
value = 50
def display(N):
    global value
    value = 25
    if N%7==0:
      value = value + N
    else:
      value = value - N
print(value, end="#")
display(20)
print(value)
```

(i) 50#50 (ii) 50#5 (iii) 50#30 (iv) 25#25

(l) What will be the output of the following code?

```
import random
List=["Delhi","Mumbai","Chennai","Kolkata"]
for y in range(4):
    x = random.randint(1,3)
    print(List[x],end="#")
```

(i) Delhi#Mumbai#Chennai#Kolkata#
(ii) Mumbai#Chennai#Kolkata#Mumbai#
(iii) Mumbai# Mumbai #Mumbai # Delhi#
(iv) Mumbai# Mumbai #Chennai # Mumbai

SOLVED QUESTIONS

1. Define a library in Python. What is the relation between a library, module and package in Python?

Ans. A Python library is a reusable chunk of code that you may require to include in your programs/projects. Here, a 'library' loosely describes a collection of core modules. We may say that a library is a collection of modules. A module is a script file with some Python code with .py extension. A package is a library that can be installed using a package manager. A package must contain __init __.py file.

2. What is a Python module? What is its significance?

Ans. A "module" is a chunk of Python code that exists in its own (.py) file and is intended to be used by Python code outside itself. Modules allow one to bundle together code in a form in which it can be easily used later. Modules can be "imported" in other programs so that the functions and other definitions in imported modules become available to code that imports them.

3. What are docstrings? How are they useful?

Ans. A docstring is just a regular Python triple-quoted string literal that appears right after function body/module/class. When executing a function body (or a module/class), the docstring doesn't do anything like comments but Python stores it as part of the function documentation. This documentation can later be displayed using the help() function or (__doc__) attribute. For example,

```
def add(a,b):
'''Takes two variables and returns the sum'''
    return a+b
```

The above docstring can be accessed using (__doc__) attribute or help() function. For example,

```
print(add.__doc__)
```

So, even though docstrings appear like comments (no execution), they are different from comments.

4. What happens when Python encounters an import statement in a program? What would happen if there is one more import statement for the same module, already imported in the same program?

Ans. When Python encounters an import statement, it does the following:

(i) Code of the imported module is interpreted and executed.
(ii) Defined functions and variables created in the module are now available to the program that imported the module.
(iii) For imported module, a new namespace is set up with the same name as that of the module.

Any duplicate import statement for the same module in the same program is ignored by Python.

5. What is a Python module? Give at least two reasons why we need modules.

Ans. A module is a file containing Python definitions and statements. We need modules because of their following salient features:

1. A module allows code reuse.
2. It makes the "main" program shorter and more readable.
3. It enables clearer code organization.

6. Rohit is a Python programmer and has imported the module **math**. From this math module, he wants to use the function sqrt() to calculate the square root of n. But he has forgotten how to use a function from imported module. Help Rohit use the function sqrt().

```
import math
n = int(input("Enter number "))
num =                          # statement to call sqrt() function for value n
print(num)
```

Ans. `num = math.sqrt(n)`

7. Name any two built-in modules along with two functions belonging to each category.

Ans. Built-in modules:

1. math: sqrt(), pow()
2. statistics: mean(), median()

8. Answer the following questions based on the given code:

```
from math import sqrt, pow
print()                          # Statement 1
print()                          # Statement 2
print(math.sqrt(121))            # Statement 3
```

(i) Identify the suitable code for statement 1 to calculate square root of 144.

(ii) Identify the suitable code for statement 2 to calculate power of $(3)^7$.

(iii) Will statement 3 give any output? If not, give reason.

Ans. (i) `print(sqrt(144))`

(ii) `print(pow(3,7))`

(iii) On execution, statement 3 shall result in an error because sqrt() method is imported in the current namespace, so sqrt() can be referred directly without having to specify the module name. The correct syntax to get the output is:

```
print(sqrt(121))
```

9. Observe the following code and answer the questions based on it.

```
# the math_operation module
def add(a,b):
    return a + b
def subtract(a,b):
    return a - b
```

Fill in the blanks for the following code:

```
1. _____math_operation   # import the math_operation module in a program
2. print(_________)      # get the name of the module output: math_operation
3. print(_________(1, 2))# Add 1 and 2
   # output 3
```

Ans.

1. import
2. math_operation.__name__
3. math_operation.add

10. Consider the code given in Q.9 and, on the basis of it, complete the code given below:

```
# import the subtract and add functions from the math_operation module
1. _________________
2. print(_______(2, 1))          # subtract 1 from 2    output : 1
3. print(_______(1, 1))          # Add 1 and 1          output : 2
```

Ans.

1. from math_operation import subtract, add
2. subtract
3. add

11. How can you locate environment variable for Python to locate the module files imported into a program?

Ans. PYTHONPATH variable is used for the same. It has a role similar to PATH. This variable tells the Python interpreter where to locate the module files imported into a program. It should include the Python source library directory and the directory containing Python source code.

12. Why is the use of import* statement not recommended?

Ans. Using import* statement will import all the functions and classes into the currently-executing program namespace, which may overwrite functions. This can be dangerous, especially if we do not maintain that module, and becomes very difficult to identify from which library does a particular function come from. This situation is also called polluting the namespace.

13. What is PYTHONPATH variable?

Ans. PYTHONPATH is the variable that tells the interpreter where to locate the module files imported into a program. Hence, it must include the Python source library directory and the directories containing Python source code. You can manually set PYTHONPATH, but usually, the Python installer will preset it.

14. How do you create your own package in Python?

Ans. We know that a package may contain sub-packages and modules. A module is nothing but Python code. To create a package of our own, we create a **directory** using the command—

```
import os
os.mkdir("directory-name")
```

followed by creating a file __init__.py in it. We leave it empty. Then, in that package, we create a **module**(s) with whatever code we want.

15. How will you share global variables across modules? [HOTS]

Ans. To share global variables across modules within a single program, we create a special module called config module or (cfg), then import it in all modules of our application. This lets the module be global to all the modules.

16. What is a Python module?

Ans. A module is a script file in Python with .py extension. It contains Python statements, functions, classes and variables. It also holds executable Python code. ZIP files and DLL files can be modules too. The modules can be imported into other modules or programs so the functions and other definitions become available to the program that imports them.

17. What is namespace in Python?

Ans. A namespace is a system that has a unique name for each and every object in Python. An object might be a variable or a method. In Python, every name has a place where it resides and can be looked for. This is known as namespace. Python implements namespaces as dictionary, where a variable name is mapped to the object placed. Whenever the variable is searched, the dictionary key will be searched to retrieve the corresponding object.

18. What are the rules for local and global variables in Python? [HOTS]

Ans. In Python, variables that are only referenced and not defined inside any function are called global variables. If a variable is ever assigned a new value inside the function, the variable is implicitly local. Thus, the global variables are accessible throughout the program and inside any function by being explicitly declared as global with the 'global' keyword whereas the local variables are accessible only inside a function.

However, if we assign another value to a globally declared variable inside the function, a new local variable is created in the function's namespace. This assignment will not alter the value of the global variable.

19. Define 'module' and 'package'. [HOTS]

Ans. Each Python program file is a 'module' which imports other modules like 'objects' and 'attributes'.
A Python program folder is a 'package' of 'modules'. A package can have 'modules' or 'sub-folders'.

20. Why is a banner saying "RESTART" always displayed in Python module/program execution?

Ans. When you execute a program from the IDLE Editor, the interpreter gives a banner saying "RESTART", meaning that all the things you defined in any shell session so far are wiped clean and the program you are running starts afresh.

21. Which of the following isn't true about main modules?

(a) When a Python file is directly executed, it is considered main module of a program.
(b) Main modules may import any number of modules.
(c) Special name given to main modules is: __main__.
(d) Other main modules can import main modules.

Ans. (d)

Explanation: Main modules are not meant to be imported into other modules.

22. Which of the following is not a valid namespace?

(a) Global namespace (b) Public namespace
(c) Built-in namespace (d) Local namespace

Ans. (b)

Explanation: During a Python program execution, there are as many as three namespaces—built-in namespace, global namespace and local namespace.

23. Which of the following is false about "import modulename" form of import?
    (a) The namespace of imported module becomes part of importing module.
    (b) This form of import prevents name clash.
    (c) The namespace of imported module becomes available to importing module.
    (d) The identifiers in module are accessed as: modulename.identifier

Ans. (a)
Explanation: When we import a module using "import modulename", a separate namespace of imported module becomes available to us in the current program.

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

```
from math import factorial
print(math.factorial(5))
```

    (a) 120
    (b) Nothing is printed
    (c) Error, method factorial doesn't exist in math module
    (d) Error, the statement should be: print(factorial(5))

Ans. (d)
Explanation: In the "from-import" form of import, we can refer to the functions by name [in this case factorial ()] rather than through dot notation.

25. What is the order of namespaces in which Python looks for an identifier?
    (a) Python first searches the global namespace, then the local namespace and finally the built-in namespace.
    (b) Python first searches the local namespace, then the global namespace and finally the built-in namespace.
    (c) Python first searches the built-in namespace, then the global namespace and finally the local namespace.
    (d) Python first searches the built-in namespace, then the local namespace and finally the global namespace.

Ans. (b)
Explanation: Python first searches for the local, then the global and finally the built-in namespace.

26. How can you generate random numbers in Python?

Ans. random module is the standard module used to generate random numbers. The methods available in Python for the same are:

1. random(): The random() method in random module generates a float number between 0 and 1, by importing it first using the following statements:

```
import random
random.random()
```

The statement random.random() method returns the floating point number that is in the range of (0 and <1). The function generates the random float numbers. The methods that are used with the random class are the bound methods of the hidden instances. The instances of the Random can be done to show the multi-threading programs that create different instances of individual threads. The other random generators that are used in this are:

2. randrange(a, b): It chooses an integer and defines the range in-between (a, b). It returns the elements by selecting it randomly from the range that is specified. It doesn't build a range object.
3. uniform(a, b): It chooses a floating point number that is defined in the range of (a, b). It returns the floating point number.

27. Which of these definitions correctly describes a module?
    (a) Denoted by triple quotes for providing the specification of certain program elements
    (b) Design and implementation of specific functionality to be incorporated into a program
    (c) Defines the specification of how it is to be used
    (d) Any program that reuses code

Ans. (b)
Explanation: The term "module" refers to the implementation of specific functionality to be incorporated into a program.

28. Which of the following is not an advantage of using modules?
    (a) Provides a means of reusing program code
    (b) Provides a means of dividing up tasks
    (c) Provides a means of reducing the size of the program
    (d) Provides a means of testing individual parts of the program

Ans. (c)
Explanation: The total size of the program remains the same regardless of whether modules are used or not. Modules simply divide the program.

29. What is a module, package and a library?

Ans. **Module:** A module is a file with some Python code and is saved with .py extension.
Package: A package is a directory that contains sub-packages and modules in it along with some special files such as __init__.py.
Library: A Python library is a reusable chunk of code that is used in program/script using import command. A package is a library if it is installable or gets attached to site-packages folder of Python installation. The line between a package and a Python library is quite blurred and both these terms are often used interchangeably.

30. What happens when Python encounters an import statement in a program? What would happen if there is one more import statement for the same module, already imported in the same program?

Ans. When Python encounters an import statement, it does the following:
- The code of imported module is interpreted and executed.
- Defined functions and variables created in a module are available to the program that imported the module.
- For imported module, a new namespace is set up with the same name as that of the module.
- Any duplicate import statement for the same module in the same program is ignored by Python.

31. What possible outputs(s) are expected to be displayed on screen at the time of execution of the program from the following code? Also specify the maximum values that can be assigned to each of the variables FROM and TO.

```
import random
AR=[20,30,40,50,60,70]
FROM=random.randint(1,3)
TO=random.randint(2,4)
for K in range(FROM,TO+1):
    print(AR[K],end="#")
```

(a) 10#40#70# (b) 30#40#50# (c) 50#60#70# (d) 40#50#70#

Ans. (b) 30#40#50#
Maximum value FROM, TO is (3, 4)

32. Consider a module 'greeting' given below:

```
#module greeting.py
def greet_msg():
    """Welcome to module1"""
    print("Module1")
def new_msg():
    """Significance of using Modules"""
    print("Efficient")
count = 100
print("Modules are must for a good program")
```

Another program 'test.py' imports this module. The code inside test.py is:

```
# test.py
import greeting
print(greeting.count)
```

What would be the output produced, if we run the program test.py? Justify your answer.

Ans. The output produced would be:

```
Modules are must for a good program
100
```

The reason is that import module's main block is executed upon import, so its import statement causes it to print:

```
Modules are must for a good program
```

And print(greeting,count) statement causes output's next line, *i.e.*,

```
100
```

33. What is the procedure to create own library/package in Python?

Ans. To create own library or package in Python, we should do the following:

(a) Create a package folder having the name of the package/library using dir() method.
(b) Add module files (.py files containing actual code functions) to this package folder.
(c) Add a special file __init__.py to it (even if the file is empty).
(d) Attach this package folder to site-package folder of Python installation.

34. Why is the package attached to site-package folder of Python installation? Can't we use it without doing so?

Ans. In order to import a package using import command, the packages and their contents must be attached to site-package folder of Python installation as this is the default place from where Python interpreter imports Python library and packages. So, in order to import our package with import command in our programs, we must attach it to site-package folder of Python installation.

35. Find and write the output of the following Python code: [CBSE Sample Paper 2019-20]

```
a=10
def call():
    global a
    a=15
    b=20
    print(a)
call()
```

Ans. 15

If there is a local and global variable with the same name, then the priority is given to the local variable.

36. What do you understand by local and global scope of variables? How can you access a global variable inside the function, if function has a variable with the same name?

Ans. A global variable is a variable that is accessible globally. A local variable is one that is only accessible to the current scope, such as temporary variables used in a single function definition.

A variable declared outside of the function or in global scope is known as global variable. This means that global variable can be accessed inside or outside of the function whereas local variable can be used only inside of the function. We can access a global variable inside a function by declaring variable explicitly using the keyword 'global', like **global A**.

37. When is a global statement used? Why is its use not recommended?

Ans. A global statement is used when the mentioned variable to be used is from global scope. The use of global statement is always discouraged as programmers tend to lose control over variables and their scope.

38. Write a method in Python to find and display the prime numbers between 2 to N. Pass N as argument to the method. [Delhi 2016]

Ans.

```
#Module to find the prime numbers between 2 and N
def check_primeAll(N):
    print("Prime nos. between 2 and N")
    for num in range(2, N+1):      #to iterate between 2 to N+1
        for i in range(2, num):    #to iterate on the factors of the number
            if num % i == 0:       #to determine the first factor
                j=num/i            #to calculate the second factor
                break              #to move to the next number
            else:                  #else part of the loop
            #loop fall through without finding a factor
                print(num, end="\n")
```

39. Write definition of a method ZeroEnding(SCORES) to add all those values in the list of SCORES, which are ending with zero (0) and display the sum. [Delhi 2018]

For example,
If the SCORES contain [200,456,300,100,234,678]
The sum should be displayed as 600

Ans.
```
def ZeroEnding(SCORES):
    SZero=0
    for i in SCORES:
        if i%10==0:
            SZero=SZero+i
    print("Sum of numbers ending with zero:", SZero)
```

40. Write a program that performs the following operations on a string:

(a) Prompt the user to input a string.

(b) Extract all the digits from the string, if there are digits in the inputted string.

(c) Calculate and display the sum of the digits. Also display:

- The original string
- The digits
- The sum of the digits

(d) If there are no digits:

- Display the original string along with the appropriate message: "No Digits are present"

Ans.
```
str1 = input("Enter the string: ")
sum = 0
num = 0
if str1.isalpha() == False:
    for i in str1:
        if i.isdigit() == True:
            num =num*10 + int(i)
            sum += int(i)
    print("Original String: ",str1)
    print("Digits: ",num)
    print("Sum of digits is: ",sum)
else:
    print("Original String: ", str1, "has no digit")
```

41. Write a program with a user-defined function with string as a parameter which replaces all vowels in the string with '*'.

Ans.
```
#Function to replace all vowels in the string with '*'
def replaceVowel(st):
    #Create an empty string
    newstr = ''
    for character in st:
        #Check if next character is a vowel
        if character in 'aeiouAEIOU':
            #Replace vowel with *
            newstr += '*'
        else:
            newstr += character
    return newstr
    #End of function
st = input("Enter a String: ")
st1 = replaceVowel(st)
print("The original String is:", st)
print("The modified String is:", st1)
```

42. Find and write the output of the following Python code: [CBSE Sample Paper 2020-21]

```
def Display(str):
    m=""
    for i in range(0,len(str)):
      if(str[i].isupper()):
          m=m+str[i].lower()
      elif str[i].islower():
        m=m+str[i].upper()
      else:
        if i%2==0:
          m=m+str[i-1]
        else:
          m=m+"#"
          print(m)
Display('Fun@Python3.0')
```

Ans. fUN#pYTHONn#.

UNSOLVED QUESTIONS

1. What is Python library? Explain with example.
2. Write a program to calculate the following using modules:
   (a) Energy = m * g * h (b) distance = $ut + \frac{1}{2}at^2$
   (c) Speed = distance / time
3. Write a module to input total number of days and find the total number of months and remaining days after months, and display it in another program.
4. Write a program to calculate the volume and area of a sphere inside separate modules and import them in one complete package.
   Volume = $\frac{4}{3}\pi r^3$ and Surface Area = $4\pi r^2$
5. What is a module? What is the file extension of a Python module?
6. How do you reuse the functions defined in a module in your program?
7. In how many ways can you import objects from a module?
8. How are the following two statements different from one another?
   (a) import math (b) from math import *
9. What is the utility of math module?
10. How does Python resolve the scope of a name or identifier?
11. Find the output of the following:

```
import math
print(math.floor(5.5))
```

12. Rewrite the following Python code after removing all syntax error(s). Underline the corrections done. [CBSE Sample Paper 2014-15]

```
def main():
    r = input('enter any radius:')
    A =pi * maths.pow(r,2)
    Print("Area = "+a)
       Main ()
```

13. What is the significance of assigning namespace to Python modules?
14. How are packages and modules related to each other?
15. What is the difference between import statement and from import statement?
16. Why is from import * statement not recommended for importing Python modules in an external program?
17. What is a library?
18. What is the importance of site-package folder of Python installation?

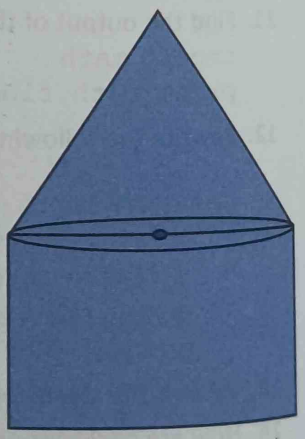
19. How are the following import statements different?
    (a) import X (b) from X import* (c) from X import a, b, c
20. Name the Python library modules which need to be imported to invoke the following functions: [CBSE D 2016]
    (a) log() (b) pow()
21. What is dot notation of referring to objects inside a module?
22. Why should the from <module> import <object> statement be avoided to import objects?
23. What do you understand by standard library of Python?
24. Explain the difference between import <module> and from <module> import statements, with examples.
25. Create a module lengthconversion.py that stores functions for various lengths conversions, *e.g.*,
    - Miletokm() # to convert miles into kilometres
    - Kmtomile() # to convert kilometres into miles
    - Feettoinches() # to convert feet into inches
    - Inchestofeet() # to convert inches into feet

    It should also store constant values such as value of (mile in kilometre and vice versa)
    [1 mile = 1.609344 kilometre; 1 feet = 12 inches]
    help() function should display proper information.
26. Create a module MassConversion.py that stores function for mass conversion, *e.g.*,
    - Kgtotonne() # to convert kg into tonnes
    - Tonnetokg() # to convert tonne into kg
    - Kgtopound() # to convert kg into pound
    - Poundtokg() # to convert pound into kg

    (Also store constants 1 kg = 0.001 tonne, 1 kg = 2.20462 pound)
    help() function should give proper information about the module.
27. Write a user-defined function to calculate the area of a triangle.
28. Write a user-defined function to convert a string with more than one word into title case string where string is passed as parameter. (Title case means that the first letter of each word is capitalized.)
29. Write a function deleteChar() which takes two parameters—one is a string and the other is a character. The function should create a new string after deleting all occurrences of the character from the string and return the new string.
30. Write a program to print the number of occurrences of a substring into a line using built-in string function find().

CASE-BASED/SOURCE-BASED INTEGRATED QUESTIONS

1. Ceremony Tent House manufactures tents as per the user's requirements. The shape of the tent is cylindrical surmounted by a conical top.



The company performs the following tasks to fix the selling price of each tent:
   (a) Accept user requirements for the tent, such as—
      - height
      - radius
      - slant height of the conical part
   (b) Calculate the area of the canvas used.
   (c) Calculate the cost of the canvas used for making the tent.
   (d) Calculate the net amount payable by the customer that is inclusive of 18% tax.

The company has developed a computerized solution for a quick and accurate calculation of the payable amount. Write a Python program to calculate the cost of tent function definition.

Ans.

```
#Program to calculate the cost of tent
#function definition
from math import pi
#function definition for cylindrical part
def cyl(h,r):
    area_cyl = 2*pi*r*h #Area of cylindrical part
    return(area_cyl)

#function definition for conical part
def con(l,r):
    area_con = pi*r*l #Area of conical part
    return(area_con)

#function definition for computing total amount
def post_tax_price(cost): #compute payable amount for the tent
    tax = 0.18 * cost;
    net_price = cost + tax
    return(net_price)
```

```
print("Enter values of cylindrical part of the tent in meters:")
h = float(input("Height: "))
r = float(input("Radius: "))
csa_cyl = cyl(h,r) #function call
l = float(input("Enter slant height of the conical area in meters: "))
csa_con = con(l,r) #function call

#Calculate area of the canvas used for making the tent
canvas_area = csa_cyl + csa_con
print("Area of canvas = ",canvas_area," m^2")

#Calculate cost of canvas
unit_price = float(input("Enter cost of 1 m^2 canvas in rupees: "))
total_cost = unit_price * canvas_area
print("Total cost of canvas before tax = ",total_cost)
print("Net amount payable (including tax) = ",post_tax_price(total_cost))
```

OUTPUT:

```
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/
nvas.py
Enter values of cylindrical part of the tent in meters:
Height: 20
Radius: 4
Enter slant height of the conical area in meters: 12
Area of canvas =  653.451271946677  m^2
Enter cost of 1 m^2 canvas in rupees: 200
Total cost of canvas before tax =  130690.2543893354
Net amount payable (including tax) =  154214.50017941577
```

2. (a) Gurukul Academy uses "Student Management Information System" (SMIS) to manage student-related data. This system provides facilities for:
   - recording and maintaining personal details of students,
   - maintaining marks scored in assessments and computing results of students,
   - keeping track of student attendance, and
   - managing many other student-related data.

   Let us automate this process step by step.

   Identify the personal details of students from your school identity card and write a program using a user-defined function to accept these details for all students of your school and display them in the following format:

School Name	
Name: ABC	Roll No: 25
Age: 16	Class: XII
Address: Address line1	State: Delhi
Pin Code: 999999	

Ans.

```
File Edit Format Run Options Window Help
def personal_details():
    name= input("Enter student Name:")
    roll_no= int(input("Enter the Roll no:"))
    age = int(input("Enter the age:"))
    Class =input("Enter student's class:")
    address =input("Enter the address line1:")
    state = input("Enter the state:")
    pin_code= input("Enter the address pincode:")
    print("\t\t ABC Public School")
    print("Name: {}\t\t Roll No: {}\nAge: {}\t\t\tClass: {}\nAddress: {}\tState: {}\nPin code: {}"
          .format(name,roll_no,age,Class,address,state,pin_code))

personal_details()
```

```
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs
thon/Python37-32/prog_stud_details.py
Enter student Name:Teena
Enter the Roll no:3
Enter the age:19
Enter student's class:XII
Enter the address line1:Ashok Vihar
Enter the state:Delhi
Enter the address pincode:110052
                ABC Public School
Name: Teena              Roll No: 3
Age: 19                  Class: XII
Address: Ashok Vihar     State: Delhi
Pin code: 110052
```

(b) On the basis of the above scenario, write a user-defined function to:

- Accept the marks of the student in five major subjects in Class XII and display the same.
- Calculate the sum of the marks of all subjects.
- Divide total marks by number of subjects, *i.e.*, 5, and calculate and display the percentage (percentage = total marks/5).
- Find the grade of the student as per the following criteria:

Criteria	Grade
percentage > 90	A
percentage < 90 and >= 80	B
percentage < 80 and >= 70	C
percentage < 70 and >= 60	D
percentage < 60 and >= 40	E
percentage < 40	Retest

Ans.

```
File Edit Format Run Options Window Help
def cal_tot_per():
    english = float(input(" Please enter English Marks: "))
    math = float(input(" Please enter Math score: "))
    computers = float(input(" Please enter Computer Marks: "))
    physics = float(input(" Please enter Physics Marks: "))
    chemistry = float(input(" Please enter Chemistry Marks: "))
    total = english + math + computers + physics + chemistry
    percentage = (total / 500) * 100

    print("Total Marks = " ,total)
    print("Marks Percentage = ",percentage)

    if(percentage >= 90):
        print("A Grade")
    elif(percentage >= 80):
        print("B Grade")
    elif(percentage >= 70):
        print("C Grade")
    elif(percentage >= 60):
        print("D Grade")
    elif(percentage >= 40):
        print("E Grade")
    else:
        print("Retest")

cal_tot_per()
```

```
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs/Python
.py
 Please enter English Marks: 70
 Please enter Math score: 80
 Please enter Computer Marks: 60
 Please enter Physics Marks: 100
 Please enter Chemistry Marks: 90
Total Marks =  400.0
Marks Percentage =  80.0
B Grade
```