

Introduction to Python Modules

(New Chapter)

INTRODUCTION

As our programs become larger and more complex, the need to organize our code becomes greater. A large and complex program should be divided into files/functions that perform a specific task. As we write more and more functions in a program, we should consider organizing the functions by storing them in modules.

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 the account receivable functions in their own module, all the account payable functions in their own module, and all 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. In short, we need some method to organize the code into small pieces that are easier to manage, much like the examples in this book. The Python solution is to place the 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

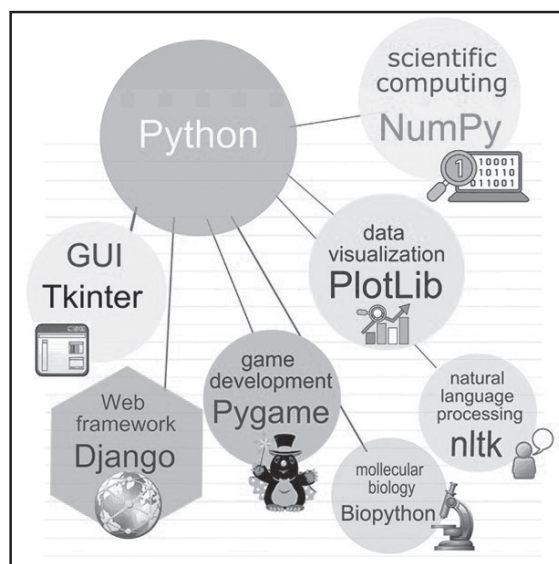


Fig. 1: Python Libraries

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.

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

A **library** in Python is a collection of various packages. 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.

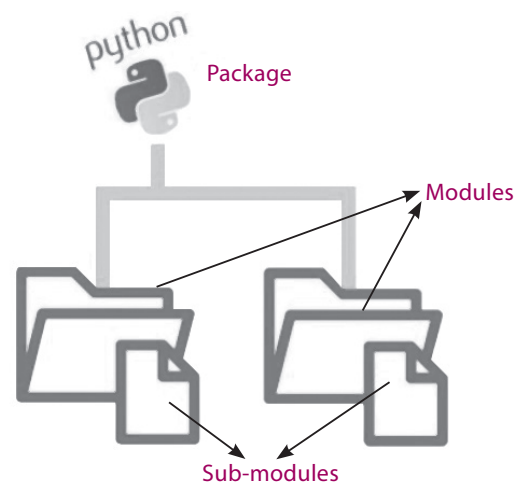


Fig. 2: Relation between a Package and Modules

MODULE IN PYTHON

Modules are used to categorize Python code into smaller parts. 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.

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.

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 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. Grouping related code into a module makes the code easier to understand and use.
5. Similar types of attributes can be categorized and placed in a single module.

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.

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.

IMPORTING PYTHON MODULES

The most common way to create a module is to define a separate file containing the code 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 a 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 present in the module.

```
import <module_name>
```

In case of accessing specific modules, it can be modified as:

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

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

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

- (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]]
```

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

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

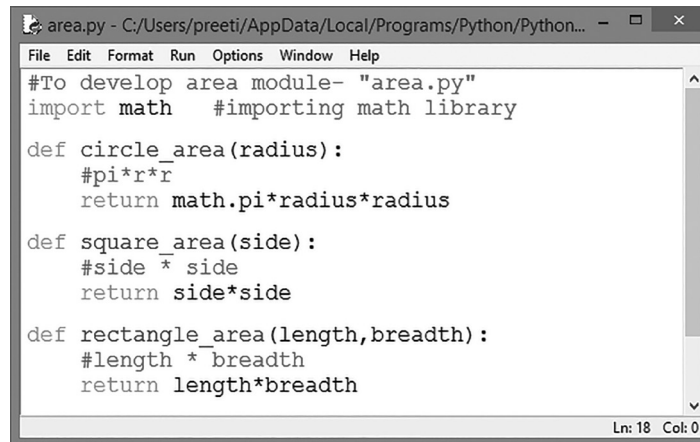
- (iii) **import *** statement can be used to import all names from a module into the current calling (namespace) module.

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

Note: We can access any function which is inside a module by module name and function name separated by a dot, *i.e.*, using dot notation.

Let's look at a few simple examples.

Example 1: Let's 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
```

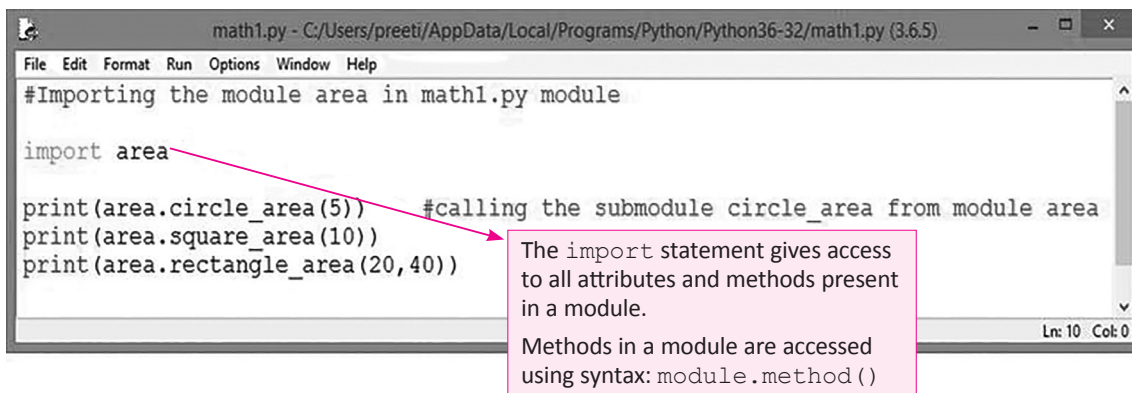
Fig. 3(a): Area Module

We have created the “area” module (Fig. 3(a)) which is a user-created module, not from the Python library.

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

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

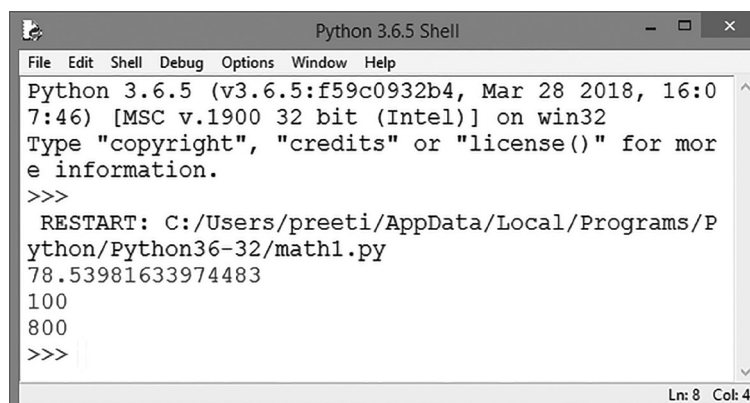
Now, we shall call these modules in another program (Fig. 3(b)) named “math1.py”.



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

Fig. 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, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32  
Type "copyright", "credits" or "license()" for more information.  
>>>  
RESTART: C:/Users/preeti/AppData/Local/Programs/Python/Python36-32/math1.py  
78.53981633974483  
100  
800  
>>>  
Ln: 8 Col: 4
```

Fig. 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:

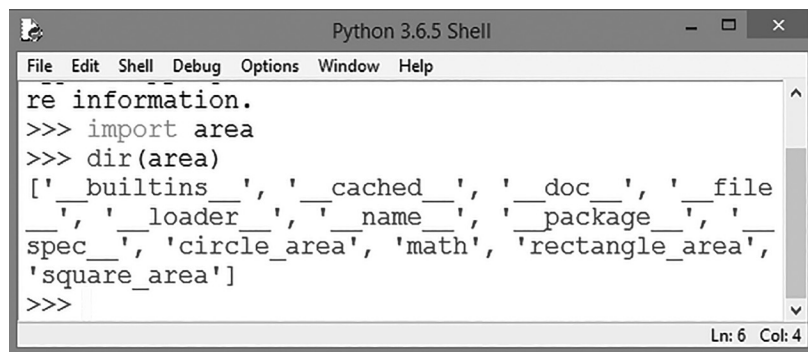
- `area.circle_area()`
- `area.square_area()`
- `area.rectangle_area()`

They shall generate the output as shown in Fig. 3(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”.



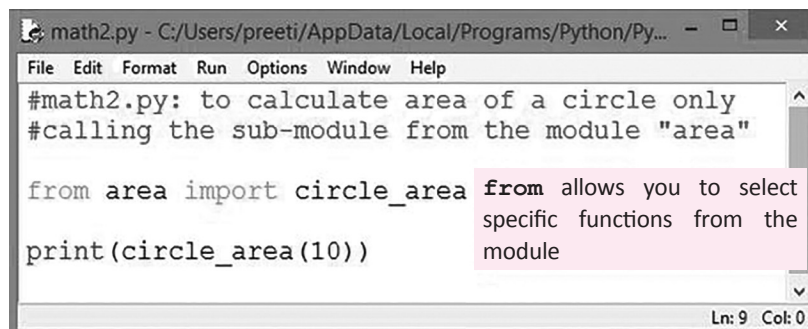
```

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
re information.
>>> import area
>>> dir(area)
['_builtins_', '_cached_', '_doc_', '_file_', '_loader_', '_name_', '_package_', '_spec_', 'circle_area', 'math', 'rectangle_area', 'square_area']
>>>
Ln: 6 Col: 4

```

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.

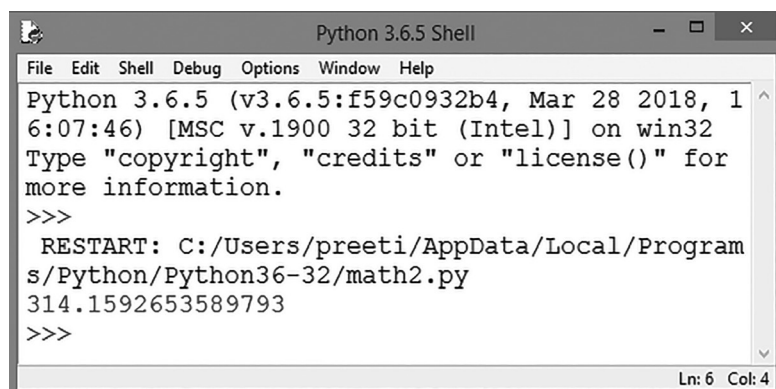


```

math2.py - C:/Users/preeti/AppData/Local/Programs/Python/Py...
File Edit Format Run Options Window Help
#math2.py: to calculate area of a circle only
#calling the sub-module from the module "area"

from area import circle_area
print(circle_area(10))
from allows you to select
specific functions from the
module
Ln: 9 Col: 0

```



```

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:/Users/preeti/AppData/Local/Programs/Python/Python36-32/math2.py
314.1592653589793
>>>
Ln: 6 Col: 4

```




from...import statement is used to import a particular attribute (sub-routine) from a module. In case you do not want the whole of the module to be imported, then you can use from...? import 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**.

Now let's see the third format of importing a module using **from...import***.

The import* statement allows a user to import everything from the file or module. This construct will import all Python definitions into the namespace of another module.

However, this construct is not recommended to be used much as it may result in namespace pollution. We may have several objects of the same name and their definitions can be overridden.

Example 4: Modification of Example 3—to import all the sub-modules from module “area”.

The screenshot shows a Python IDE window titled 'math3.py - C:/Users/preeti/AppData/Local/Programs/P...'. The code in the editor is as follows:

```
# Use of import * statement

from area import *

print(circle_area(12))
print(square_area(20))
print(rectangle_area(25,15))
```

A tooltip on the right explains: 'import * gives access to all functions in area module. You can use all the functions in area module directly.'

Below the editor is a 'Python 3.7.0 Shell' window showing the execution output:

```
RESTART: C:\Users\preeti\AppData\Local\Programs\Python\Python37-32\ma
th3.py
452.3893421169302
400
375
>>>
```

Thus, from all of the above examples, we have observed that **import** statement in Python can be represented in any of the following for importing a module in some other program:

- (a) **import test:** imports the module “test” and creates a reference to that module in the **current calling program (or namespace)**. After executing this statement, you can use test.name to refer to the objects defined in module ‘test’.
- (b) **import test1, test2, test3:** This statement shall import three modules, namely **test1**, **test2** and **test3** in **current namespace**. As a result, all the definitions of these modules are available and can be used as:

<module_name>.<name>
 ↘
 sub-module/any object



- (c) **from test import * :** When you use import * statement, it shall import all the sub-modules or objects defined in the module 'test' into the current namespace. So, after executing this statement, you are required to simply give the name of the object to be referred to, defined in the module test. In other words, <name> would now be used as name only, without specifying module's name (*i.e.*, **test.name** is not required to be given). If **name** was already defined, it is replaced by the new version, the one imported from the module.
- (d) **from test import t1,t2,t3:** This statement shall import the module test and create references in the current namespace to the given objects. In other words, you can now use t1, t2, and t3 in your program without specifying their module's name along with them.

Learning Tip: Whenever we are using a module in our program, for that module, its compiled file will be generated and stored in the hard disk permanently.

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)
```

```
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
>>>
```



Practical Implementation-2

Modify the above program 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

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

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**.

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.

```
prog_test3.py - C:/Users/preeti/AppData/Local/...
File Edit Format Run Options Window Help
# Member Aliasing

from test import x as p, add as sum
print(p)
sum(100,200)

>>>
RESTART: C:/Users/preeti/AppData/Local/Programs/Python/Python36-32/prog_test3
.py
100
The sum is: 300
>>>
```

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.

```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help

RESTART: C:/Users/preeti/AppData/Local/Programs/Python/Python36-32/prog_test3
.py
100
The sum is: 300
>>> from test import x as p
>>> print(x)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
>>>
```


Practical Implementation-4

Write a program in Python that calculates the following:

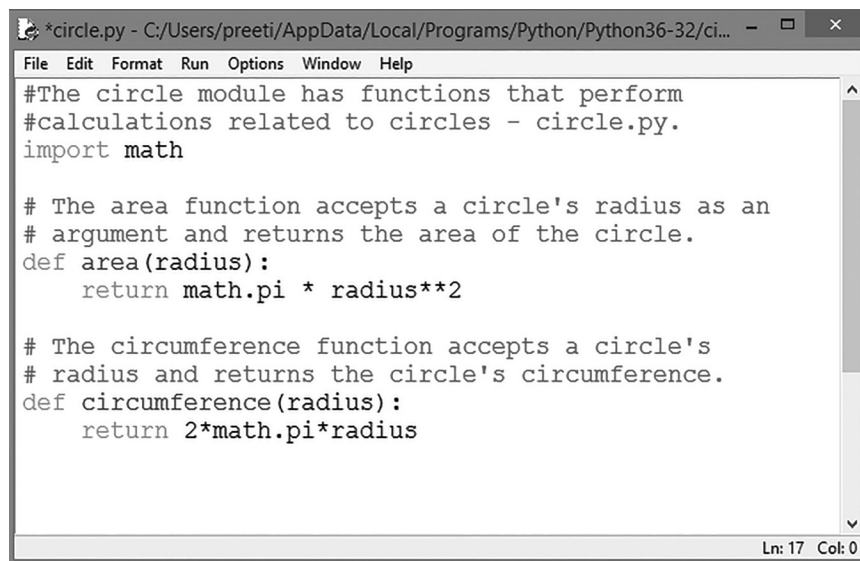
- Area of a circle
- Circumference of a circle
- Area of a rectangle
- Perimeter of a rectangle



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



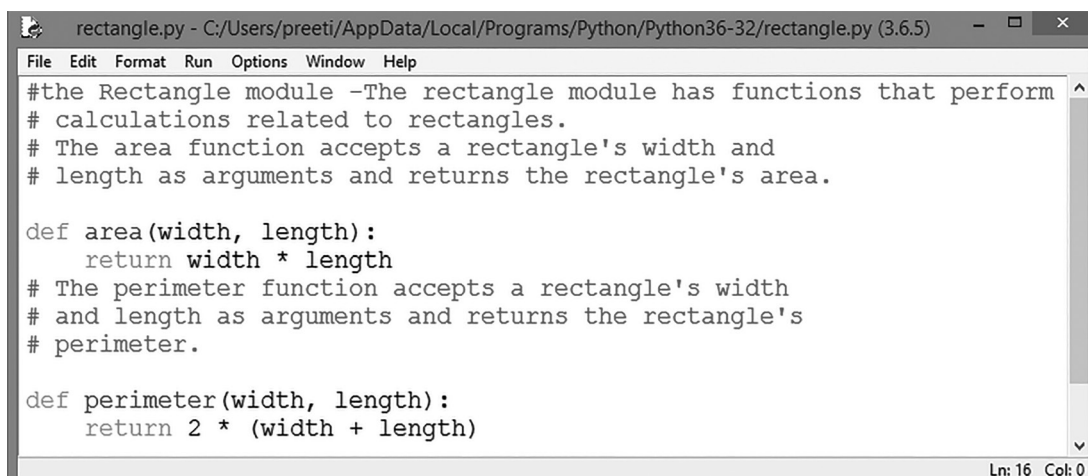
```
*circle.py - C:/Users/preeti/AppData/Local/Programs/Python/Python36-32/ci...
File Edit Format Run Options Window Help
#The circle module has functions that perform
#calculations related to circles - circle.py.
import math

# 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.
#def displaymenu():
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

```
*Python 3.7.0 Shell*
File Edit Shell Debug Options Window Help
RESTART: C:/Users/preeti/AppData/Local/Pro
grams/Python/Python37-32/prog_mod_alt1.py
MENU
1. Area of a circle
2. Circumference of a circle
3. Area of a rectangle
4. Perimeter of a rectangle
5. Quit
Enter your choice :2
Enter the circle's radius: 4
The circumference is 25.132741228718345
MENU
1. Area of a circle
2. Circumference of a circle
3. Area of a rectangle
4. Perimeter of a rectangle
5. Quit
Enter your choice :3
Enter the rectangle's width: 55
Enter the rectangle's length: 66
The area is 3630
```

Ln: 46 Col: 22





```
MENU
1. Area of a circle
2. Circumference of a circle
3. Area of a rectangle
4. Perimeter of a rectangle
5. Quit
Enter your choice :4
Enter the rectangle's width: 42
Enter the rectangle's length: 20
The perimeter is 124
MENU
1. Area of a circle
2. Circumference of a circle
3. Area of a rectangle
4. Perimeter of a rectangle
5. Quit
Enter your choice :1
Enter the circle's radius: 10
The area is 314.1592653589793
MENU
1. Area of a circle
2. Circumference of a circle
3. Area of a rectangle
4. Perimeter of a rectangle
5. Quit
Enter your choice :5
```

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.

STANDARD BUILT-IN PYTHON MODULES

In Python, we have worked with script mode in the previous chapters, which provides the capability to retain our work for future usage. For working in script mode, we need to write a module/function in Python and save it in the file having .py extension.

A Python module is written once and used/called as many times as required.

CTM: Modules are the most important building blocks for any application in Python and work on divide and conquer approach.

Modules can be categorized into the following two types:

- (i) Built-in Modules
- (ii) User-defined Modules

Let us discuss built-in modules in detail.

BUILT-IN FUNCTIONS

Built-in functions are the predefined functions that are already available in Python. Functions provide efficiency and structure to a programming language. Python has many useful built-in functions to make programming easier, faster, and more powerful. These are always available in the standard library and for using them, we don't have to import any module (file).

Let us discuss math and datetime modules which are one of the most commonly used built-in modules in Python.

To access/use any of the function present in the imported module, you have to specify the name of the module followed by the name of the function, separated by a dot (also known as a period). This format is called **dot notation**.

For example,

```
result = math.sqrt(64)           #Gives output as 8 which gets
                                stored in variable 'result'
```

sqrt() is one of the functions that belongs to math module. This function accepts an argument and returns the square root of the argument. Thus, the above statement calls the sqrt() passing 64 as an argument and returns the square root of the number passed as the argument, i.e., 8 which is then assigned to the variable 'result'.

math Module

Let us understand a few more functions from the math module.

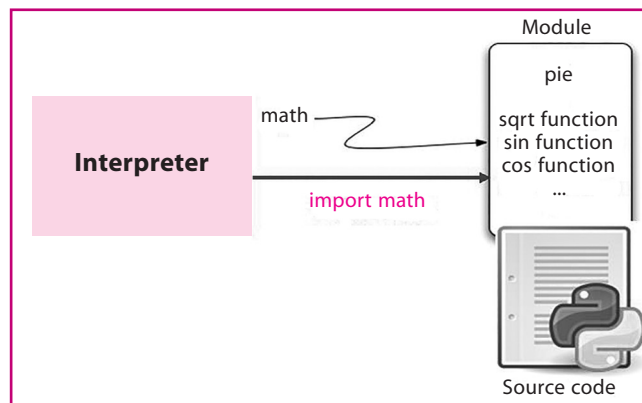


Fig. 4: math module in Python

```
import math #the very first statement to be given
```

☞ **ceil(x)**: Returns the smallest integer that is greater than or equal to x.

For example,

```
>>> print("math.ceil(3.4):", math.ceil(3.4))
```

Output: math.ceil(3.4): 4

☞ **floor(x)**: Returns the largest integer that is less than or equal to x.

```
>>> math.floor(-45.17)
```

```
-46
```

```
>>> math.floor(100.12)
```

```
100
```

```
>>> math.floor(100.72)
```

```
100
```



☞ **pow(x,y):** It returns the value of x^y , where x and y are numeric expressions.

```
>>> print("math.pow(3, 3):", math.pow(3, 3))
math.pow(3, 3): 27.0
>>> math.pow(100, -2)
0.0001
>>> math.pow(2, 4)
16.0
>>> math.pow(3, 0)
1.0
```

☞ **sqrt(x):** Returns the square root of x.

```
>>> print("math.sqrt(65):", math.sqrt(65))
math.sqrt(65): 8.06225774829855
>>> math.sqrt(100)
10.0
>>> math.sqrt(7)
2.6457513110645907
```

Alternatively,

```
>>> print("math.sqrt(65):", round(math.sqrt(65), 2))
math.sqrt(65): 8.06
```

☞ **fabs(x):** Returns the absolute value of x, represented as-

`math.fabs(x)`

where, x can be any numeric value.

For example,

```
>>> print(math.fabs(500.23))
500.23
>>> print(math.fabs(-200))
200
>>> print(math.fabs(-15.6343))
15.6343
```

☞ **help function**

If you forget how to use a standard function, Python's library utilities can help. In this case, Python's `help()` library functions is extremely relevant. It is used to know the purpose of a function and how it is used.

For example,

```
>>> import math
>>> print(help(math.cos))
Help on built-in function cos in module math:
cos(...)
None
cos(x)
Return the cosine of x (measured in radians).
```

random module (Generating Random Numbers)

The programming concepts we have learnt so far involved situations where we were aware of the definite output to be obtained when a particular task was to be executed, which is described as a deterministic approach. But there are certain situations/applications that involve games or simulations which work on a non-deterministic approach. In these types of situations, random numbers are extensively used such as:

1. Pseudorandom numbers on lottery scratch cards.

2. Recaptcha (like in login forms) uses a random number generator to define which image is to be shown to the user.
3. Computer games involving throwing of a dice, picking a number, or flipping a coin.
4. Shuffling deck of playing cards, etc.

In Python, random numbers are not enabled implicitly; therefore, we need to invoke random module explicitly or to invoke random code library in order to generate random numbers.

- import statement is the first statement to be given in a program for generating random numbers:

```
import random
```

The various functions associated with this module are explained as follows:

- ☞ **randrange():** This method generates an integer between its lower and upper argument. By default, the lower argument is 0.

For example,

```
ran_number = random.randrange(30)
```

Alternatively,

<pre>>>> ran_number = random.randrange(30) >>> print(ran_number) 1 >>></pre>	<pre>>>> import random >>> random.randrange(30) 13 >>></pre>
---	---

This line of code shall generate random numbers from 0 to 29. Let us see another example:

Example 1: To select a random subject from a list of subjects.

```
radn_demo.py - C:/Users/preeti/AppData/Local/Programs/Python/Python36-32/radn...
File Edit Format Run Options Window Help
#To demonstrate randrange()
#To select a random subject from a list of subjects

import random
subjects = ["Computer Science", "IP", "Physics", "Accountancy"]
ran_index = random.randrange(2)
print(subjects[ran_index])

>>>
RESTART: C:/Users/preeti/
rams/Python/Python36-32/ra
Computer Science
>>>
```

- ☞ **random()**

This function generates a random number from 0 to 1 such as 0.5643888123456754. This function can be used to generate pseudorandom floating point values. It takes no parameters and returns values uniformly distributed between 0 and 1 (including 0, but excluding 1). Let us take some examples to gain better clarity about this function:

Example 2:

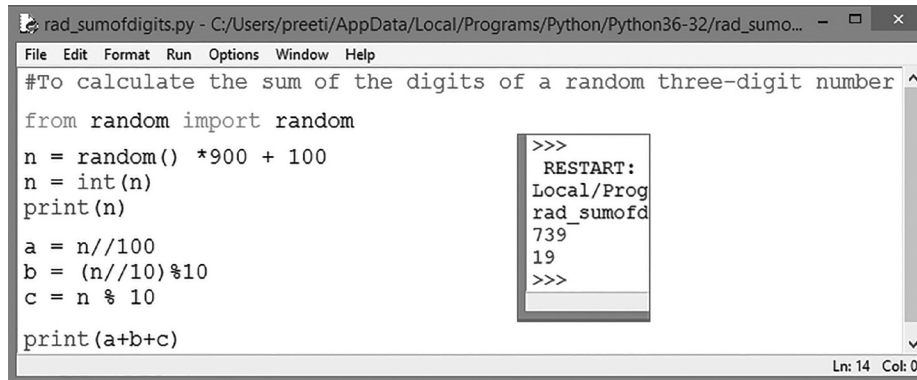
```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16
:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for m
ore information.
>>> from random import random
>>> random()
0.1353092847401708
>>> random()
0.9198861501740009
>>>
```

POINT TO REMEMBER

This is to be kept in mind that every time we are executing this program, it shall display a different value as random() method always picks up any value from the given range.

As is evident from the above statements, each time random() generates a different number, the random() function can also be written with the following alternative approach:

Example 3: Program to calculate the sum of the digits of a random three-digit number.



```
rad_sumofdigits.py - C:/Users/preeti/AppData/Local/Programs/Python/Python36-32/rad_sumo...
File Edit Format Run Options Window Help
#To calculate the sum of the digits of a random three-digit number

from random import random

n = random() * 900 + 100
n = int(n)
print(n)

a = n//100
b = (n//10)%10
c = n % 10

print(a+b+c)
```

Shell window output:

```
>>>
RESTART:
Local/Prog
rad_sumofd
739
19
>>>
```

Explanation:

The random() function generates a random fractional number between 0 and 1. When a random number generated using random() gets multiplied by 900, a random number is obtained between 0 and 899. When you add 100 to it, you get a number between 100 and 999.

Then, the fractional part gets discarded using int() and gets printed on the shell window. In the next statement, the first digit (the most significant) of the number is extracted by dividing it with 100 ($a = n//100$).

The last digit of the number is extracted by dividing it by 10. The remaining number then gets divided by 10 and the quotient is again mod with 10 to get ten's digit and the remainder is again mod with 10 and this process gets repeated till all the digits are extracted and finally added together. In the last statement, the sum of digits is calculated and displayed on the screen.

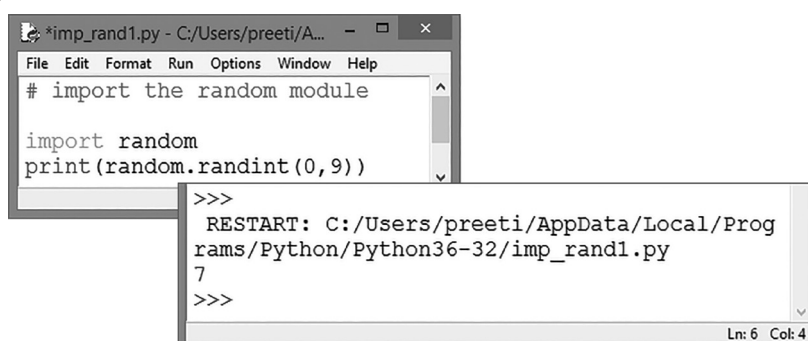
randint()

This function accepts two parameters, a and b, as the lowest and highest number; returns a number 'N' in the inclusive range [a,b], which signifies $a \leq N \leq b$, where the endpoints are included in the range. This function generates a random integer number between two given numbers. This function is best suited for guessing number applications. The syntax is:

```
random.randint(a,b)
```

Here, a is the lower bound and b is the upper bound. In case you input a number N, then the result generated will be $a \leq N \leq b$. Please make sure that the upper limit is included in randint() function.

Example 4: To generate a number between 0 and 9.



```
*imp_rand1.py - C:/Users/preeti/A...
File Edit Format Run Options Window Help
# import the random module

import random
print(random.randint(0,9))
```

Shell window output:

```
>>>
RESTART: C:/Users/preeti/AppData/Local/Prog
rams/Python/Python36-32/imp_rand1.py
7
>>>
```

Example 5: Write a function that fills a list with numbers. (Using randint())

```
*rand_list1.py - C:/Users/preeti/AppData/Local/Programs/Python/Python36-32/rand_list1.py (3.6.5)*
File Edit Format Run Options Window Help
# A function that fills a list with numbers

from random import randint

def fill_list(lst,limit_num,low,high):
    for i in range(limit_num):
        lst.append(random.randint(low,high))

minimum = int(input("Min : "))
maximum = int(input("Max : "))
n = int(input("Numbers limit :"))
a = []      #an empty list
fill_list(a,n,minimum,maximum)
print(a)    #Prints the newly created list with max and min value inputted

Ln: 19 Col: 0
```

```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:/Users/preeti/AppData/Local/Programs/Python/Python36-32/rand_list1.py
Min : 10
Max : 50
Numbers limit :5
[33, 25, 10, 13, 20]
>>>

Ln: 9 Col: 4
```

statistics module

The statistics module implements many common statistical formulas for efficient calculations using Python's various numerical types (int, float, Decimal, and Fraction).

It provides functions for calculating mathematical statistics of numeric (Real-valued) data. Before using the functions of this module, it is required to be imported first using the statement—`import statistics`.

The following popular statistical functions are defined in this module:

☞ **mean():** The mean() method calculates the arithmetic mean of the numbers in a list.

For example,

```
>>> import statistics
>>> statistics.mean([4, 6, 7, 8, 10, 45])
13.333333333333334
```

☞ **median():** The median() method returns the middle value of numeric data in a list. This function finds the centre value, and if the data set has an even number of values, it averages the two middle items.

For example,

```
>>> import statistics
>>> statistics.median([1, 2, 3, 8, 9])
3
>>> statistics.median([1, 2, 3, 7, 8, 9])
5.0
```





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`. This does not apply to modules.
- To import everything from a module, we use the wildcard `*`. But this does not work with packages.
- 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 the `.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.

SOLVED QUESTIONS

1. Python has certain functions that you can readily use without having to write any special code. What types of functions are these?
Ans. The predefined functions that are always available for use are known as Python's built-in functions. Examples of some built-in functions are: `len()`, `type()`, `int()`, `input()`, etc.
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 that is the first thing in a function body/a module/a class. When executing a function body (or a module/class), the docstring doesn't do anything like comment, but Python stores it as part of the function documentation. This documentation can later be displayed using the `help()` function.
 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 an argument? Give an example.
Ans. An argument is data passed to a function through function call statement. *For example*, in the statement `print(math.sqrt(25))`, the integer 25 is an argument.



6. What is the utility of built-in function help()?

Ans. Python's built-in function help() is very useful. When it is provided with a program name or a module name or a function name as an argument, it displays the documentation of the argument as help. It also displays the docstring within its passed-argument's definition.

For example,

```
>>>help(math)
```

will display the documentation related to module math.

It can even take function name as argument,

For example,

```
>>>help(math.sqrt())
```

The above code will list the documentation of math.sqrt() function only.

7. Write a function that:

- (i) Asks the user to input a diameter of a sphere (centimetres, inches, etc.)
- (ii) Sets a variable called radius to one-half of that number.
- (iii) Calculates the volume of a sphere using this radius and the formula:
 $\text{Volume} = \frac{4}{3} \pi r^3$
- (iv) Prints a statement estimating that this is the volume of the sphere; include the appropriate unit information in litres or quarts. Note that there are 1000 cubic cm in a litre and 57.75 cubic inches in a quart.
- (v) Returns this same amount as the output of the function.

Ans. `def compute_volume():`

```
import math
diameter = float(input("Please enter the diameter in inches:"))
radius = diameter/2
Volume_in_cubic_inches = ((4/3) * math.pi) * (radius ** 3)
Volume_in_quarts = Volume_in_cubic_inches/57.75
print("The sphere contains", Volume_in_quarts, 'quarts')
return(Volume_in_quarts)
```

8. What will be the output of the following code?

```
def interest(prnc, time=2, rate = 0.10):
    return (prnc * time * rate)
print(interest(6100, 1))
print(interest(5000, rate =0.05))
print(interest(5000, 3, 0.12))
print(interest(time = 4, prnc = 5000))
```

Ans. 610.0
500.0
1800.0
2000.0

9. What is the utility of Python standard library's math module and random module?

Ans. (i) Math module: The math module is used for math-related functions that work with all number types except for complex numbers.
(ii) Random module: The random module is used for different random number generator functions.

10. What is a module list? 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


```
# the math_operation module
```

Fill in the blanks for the following code:

- Ans.**
- | | |
|-----------------------|---------------------------|
| 1. input | 2. math_operation__name__ |
| 3. math_operation.add | |

```
# import the subtract function
# from the math operation module
```

- Ans.** 1. `from math_operation import subtract` 2. `subtract`
3. `from math_operation import *`

Ans. While using import all statements, you can overwrite functions and this can be dangerous, especially if you don't maintain that module.

Ans. A **module** is a script in Python that defines import statements, functions, classes, and variables. It also holds executable Python code. ZIP files and DLL files can be modules too. The module holds its name as a string that is in a global variable.

[HOTS]

16. Why is a banner saying “RESTART” always displayed in Python module/program execution?

30



- 



- 



- 





Ans. # To calculate the average height of the students in a class
`import statistics`
`heights = [5.9, 5.5, 6.1, 6.0, 7.2]` # the height data
`avg = statistics.mean(heights)`
`print("Average height in the class:", avg)`
`median_value = statistics.median(heights)`
`print("Median value for heights:", median_value)`

Output:

Average height in the class: 6.14

Median value for heights: 6.0

21. Write a Python program to guess a number between 1 and 9.

Ans. `import random`
`target_num, guess_num = random.randint(1, 10), 0`
`while target_num != guess_num:`
 `guess_num = int(input("Guess a number between 1 and 10 \`
 `until you get it right:"))`

 `print(target_num)`
 `target_num = random.randint(1, 10)`
`print('Well guessed!')`

Explanation: In the above code, the user is asked to guess a number, which is then fetched and gets stored inside the variable "guess_num". If the user guesses wrong then the prompt appears again and the user continues to input another number repetitively until the guess is correct. On successful guess, the user will get a "Well guessed!" message, and the program will exit.

UNSOLVED QUESTIONS

1. What is a Python library? Explain with example.
2. 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.
3. What is a module? What is the file extension of a Python module?
4. How do you reuse the functions defined in a module in your program?
5. In how many ways can you import objects from a module?
6. How are the following two statements different from each other?
 - (i) `Import math`
 - (ii) `From math import *`
7. What is the utility of math module?
8. How does Python resolve the scope of a name or identifier?
9. A program having multiple functions is considered better designed than a program without any functions. Why?
10. Write a module called `calculate_area()` that takes base and height as an input argument and returns an area of a triangle as an output. The formula used is,
Triangle Area = $\frac{1}{2} \times \text{base} \times \text{height}$
11. 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 ()
```

12. What is the difference between import statement and from import statement?

13. Why is from import * statement not recommended for importing Python modules in an external program?
14. Write a function to calculate volume of a box with appropriate default values for its parameters. Your function should have the following input parameters:
- (a) Length of box
 - (b) Width of box
 - (c) Height of box
- Test it by writing complete program to invoke it.

15. Consider the temperature given below for the month of June in North India. Calculate the average temperature and median value. This temperature gets dipped with a variation of 20° C in the month of December. Write a Python program to calculate the changed median value and average temperature.

Location	Temperature (in °C)
Delhi	41
Shimla	32
Chandigarh	43
Rohtak	40
Srinagar	28
SriGanganagar	45

16. Consider the amount of donations received by a charitable organisation given as under:
- donations = [100, 60, 70, 900, 100, 200, 500, 500, 503, 600, 1000, 1200]
- Now write a Python program to calculate the average amount obtained and median of the above data.
17. Write a Python program to generate a random number between 0 and 9.

