PYTHON FUNCTION CLASS XII

FUNCTION

A GROUP OF STATEMENTS WITHIN A PROGRAM THAT PERFORM AS SPECIFIC TASK.
USUALLY ONE TASK OF A LARGE PROGRAM.
FUNCTIONS CAN BE EXECUTED IN ORDER TO PERFORM OVERALL PROGRAM TASK
KNOWN AS DIVIDE AND CONQUER APPROACH

FUNCTION DEFINITION

- A function is a named sequence of statement(s) that performs a computation. It contains
- line of code(s) that are executed sequentially from top to bottom by Python interpreter.
- They are the most important building blocks for any software in Python.

TYPES

Functions can be categorized as -

i. Modules

ii. Built in

iii. User Defined

MODULE

- A module is a file containing Python definitions (i.e. functions) and statements.
- 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.

HOW TO IMPORT MODULE?

- There are many ways to import a module in your program, the one's which you should know are:
- Import
- From

Import

- It is simplest and most common way to use modules in our code.

- Its syntax is:
- import modulename1 [,modulename2, ---------]
- Example
- >>> import math
- To use/ access/invoke a function, you will specify the module name and name of the
- function- separated by dot (.). This format is also known as *dot notation*.
- **Example**
- >>> value= math.sqrt (25) # dot notation

From Statement

- It is used to get a specific function in the code instead of the complete module file. If we know beforehand which function(s), we will be needing, then we may use from. For modules having large no. of functions, it is recommended to use from instead of import.

Its syntax is:

```
>>> from modulename import functionname [, functionname…..]
>>>from modulename import * ( Import everything from the file)
```

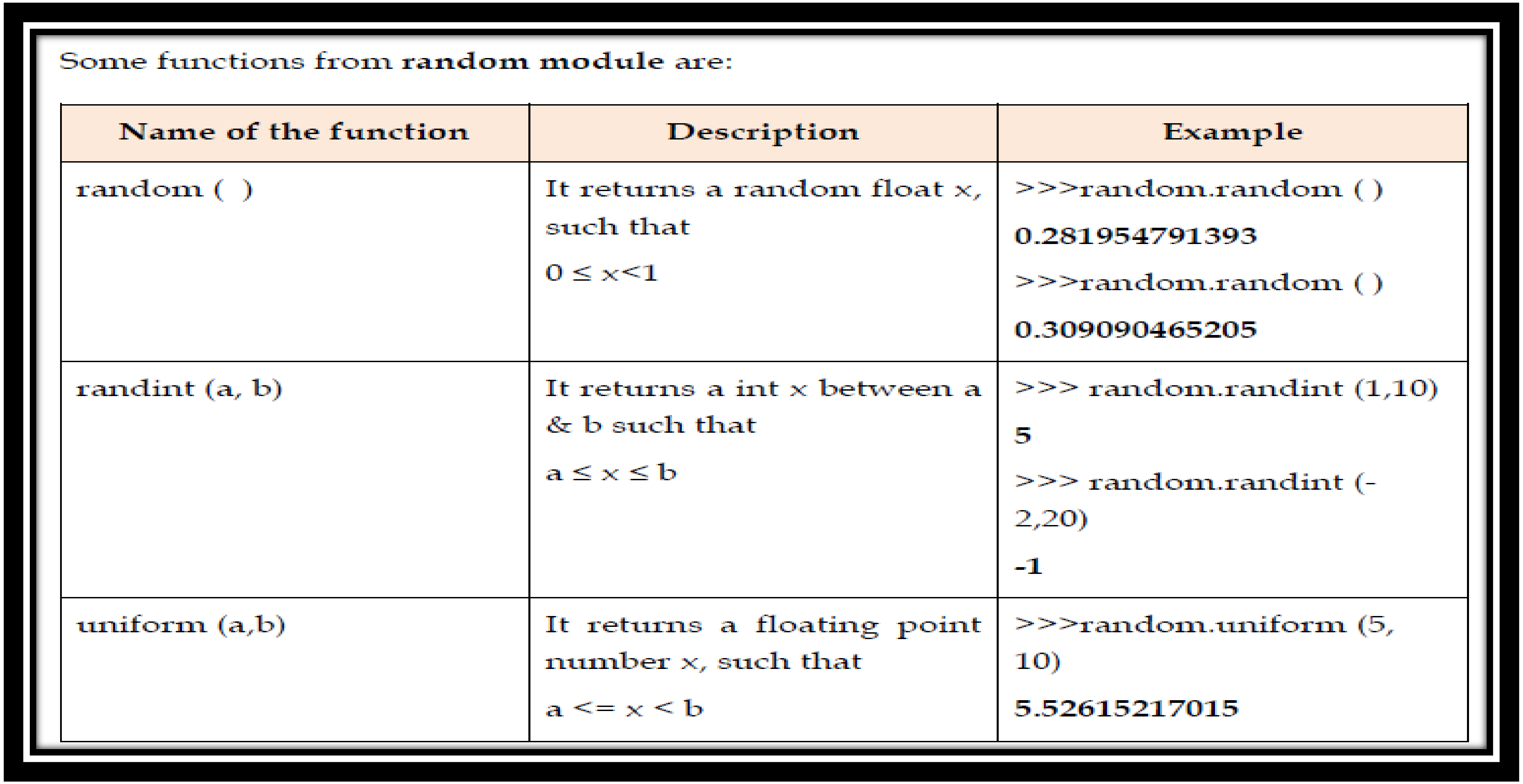
Example

- `>>> from math import sqrt`
- `value = sqrt (25)`

floor(x)	It returns the largest integer not greater than x, where *x is a numeric expression.*	math.floor(-45.17)
		-46.0
		math.floor(100.12)
		100.0
		math.floor(100.72)
		100.0

fabs(x)	It returns the absolute value of x, *where x is a numeric value.*	math.fabs(-45.17)
		45.17
		math.fabs(100.12)
		100.12
		math.fabs(100.72)
		100.72

exp(x)	It returns exponential of x: e^x, where x is a numeric expression.	math.exp(-45.17)
		2.41500621326e-20
		math.exp(100.12)
		3.03084361407e+43
		math.exp(100.72)
		5.52255713025e+43



Some functions from **random module** are:

Name of the function	**Description**	**Example**
random ( )	It returns a random float x, such that 0 ≤ x<1	>>>random.random ( ) **0.281954791393** >>>random.random ( ) **0.309090465205**

randint (a, b)	It returns a int x between a & b such that
	a ≤ x ≤ b
		>>> random.randint (1,10)
		5
		>>> random.randint (-2,20)
		-1

uniform (a,b)	It returns a floating point number x, such that a <= x < b	>>>random.uniform (5, 10) **5.52615217015**

HOW TO CREATE PYTHON MODULE ?

- Python modules are .py files that consist of Python code. Any Python file can be referenced as a module.
- Some modules are available through the Python Standard Library and are therefore installed with your Python installation. Others can be installed with Python’s package manager pip. Additionally, you can create your own Python modules since modules are comprised of Python .py files.

- Writing a module is just like writing any other Python file. Modules can contain definitions of functions, classes, and variables that can then be utilized in other Python programs.

To begin, we’ll create a function that prints Hello, World!:

hello.py

```
# Define a function
def world( ):
    print("Hello, World!")
```

If we run the program on the command line with python hello.py nothing will happen since we have not told the program to do anything.

- Let’s create a second file in the same directory called main_program.py so that we can import the module we just created, and then call the function. This file needs to be in the same directory so that Python knows where to find the module since it’s not a built-in module.

main_program.py

```
# Import hello module
import hello

# Call function
hello.world()

# or from hello import world
```

To see how we can use variables in a module, let's add a variable definition in our `hello.py` file:

hello.py

```
# Define a function
def world():
    print("Hello, World!")

# Define a variable
shark = "Sammy"
```

Next, we'll call the variable in a `print()` function within our `main_program.py` file:

main_program.py

```
# Import hello module
import hello


# Call function
hello.world()

# Print variable
print(hello.shark)
```

ACCESSING MODULES FROM ANOTHER DIRECTORY

MODULES MAY BE USEFUL FOR MORE THAN ONE PROGRAMMING PROJECT, AND IN THAT CASE IT MAKES LESS SENSE TO KEEP A MODULE IN A PARTICULAR DIRECTORY THAT'S TIED TO A SPECIFIC PROJECT.

APPENDING PATHS

- To append the path of a module to another programming file, you’ll start by importing the sys module alongside any other modules you wish to use in your main program file.

- The sys module is part of the Python Standard Library and provides system-specific parameters and functions that you can use in your program to set the path of the module you wish to implement.

- For example, let’s say we moved the hello.py file and it is now on the path /usr/sammy/ while the main_program.py file is in another directory.

- In our main_program.py file, we can still import the hello module by importing the sys module and then appending /usr/sammy/ to the path that Python checks for files.

main_program.py

- import sys
- sys.path.append('/user/sammy/')
- import hello

...

- As long as you correctly set the path for the hello.py file, you'll be able to run the main_program.py file without any errors and receive the same output as above when hello.py was in the same directory.

Built in Function

- Built in functions are the function(s) that are built into Python and can be accessed by a programmer.
- These are always available and for using them, we don’t have to import any module (file).

Name	Description	Example

abs (x)	It returns distance between x and zero, where *x is a numeric expression.*	>>>abs(-45)
		45
		>>>abs(119L)
		119

max(x, y, z, ...)	It returns the largest of its arguments: where x, y and z are numeric variable/expression.	>>>max(80, 100, 1000)
		1000
		>>>max(-80, -20, -10)
		-10

min(x, y, z, ...)	It returns the smallest of its arguments; where x, y, and z are numeric variable/expression.	>>> min(80, 100, 1000)
		80
		>>> min(-80, -20, -10)
		-80

cmp(x, y)	It returns the sign of the difference of two numbers: -1 if x < y, 0 if x == y, or 1 if x > y, *where x and y are numeric variable/expression.*	>>>cmp(80, 100)
		-1
		>>>cmp(180, 100)
		1

divmod (x,y)	Returns both quotient and remainder by division through a tuple, when x is divided by y; where x & y are variable/expression.	>>> divmod (14,5)
		(2,4)
		>>> divmod (2.7, 1.5)
		(1.0, 1.20000)

len (s)	Return the length (the number of items) of an object. The argument may be a sequence (string, tuple or list) or a mapping (dictionary).	>>> a= [1,2,3]
		>>>len (a)
		3
		>>> b= 'Hello'
		>>> len (b)
		5

range (*start*, *stop*[, *step*])	This is a versatile function to create lists containing arithmetic progressions. It is most often used in for loops. The arguments must be plain integers. If the *step* argument is omitted, it defaults to 1. If the *start* argument is omitted, it defaults to 0. The full form returns a list of plain integers [start, start + step, start + 2 * step, ...]. If *step* is positive, the last element is the largest start + i * step less than *stop*; if *step* is negative, the last element is the smallest start + i * step greater than *stop*. *step* must not be zero (or else Value Error is raised).	>>> range(10)
		[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
		>>> range(1, 11)
		[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
		>>> range(0, 30, 5)
		[0, 5, 10, 15, 20, 25]
		>>> range(0, 10, 3)
		[0, 3, 6, 9]
		>>> range(0, -10, -1)
		[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
		>>> range(0)
		[]
		>>> range(1, 0)
		[]

round(x [, n])	It returns float x rounded to n digits from the decimal point, *where x and n are numeric expressions.* If n is not provided then x is rounded to 0 decimal digits.	>>>round(80.23456, 2) **80.23** >>>round(-100.000056, 3) **-100.0** >>> round (80.23456) **80.0**

Apart from these functions, you have already seen the use of the following functions:

bool ( ), chr ( ), float ( ), int ( ), long (), str ( ), type ( ), id ( ), tuple ( )

USER DEFINED FUNCTIONS

- To define a function keyword def is used
- After the keyword comes an identifier i.e. name of the function, followed by parenthesized list of parameters and the colon which ends up the line.
- Next follows the block of statement(s) that are the part of function.

EXAMPLE

```
def sayHello ( ):   #  Header
        print “Hello World!”
```

- *Example-*

```
def area (radius):
    a = 3.14*radius**2
    return a
  Function call
  >>> print area (5)
```

SCOPE OF VARIABLES

The part of the program where a variable can be used is known as Scope of variable

Two types of scopes :

- Global Scope
- Local Scope

GLOBAL SCOPE

- With global scope, variable can be used anywhere in the program

eg:

```
x=50
def test ( ):
    print(“inside test x is “, x)
print(“value of x is “, x)
```

Output:

```
inside test x is 50
value of x is 50
```

LOCAL SCOPE

With local scope, variable can be used only within the function / block that it is created .

Eg:

```
X=50
def test ( ):
    y = 20
    print(‘value of x is ’, X, ‘ y is ’ , y)
print(‘value of x is ’, X, ‘ y is ‘ , y)
```

On executing the code we will get
Value of x is 50 y is 20

The next print statement will produce an error, because the variable y is not accessible outside the def()

MORE ON SCOPE OF VARIABLES

To access global variable inside the function prefix keyword global with the variable

Eg:

```
x=50
def test ( ):
    global x =5
    y =2
    print(‘value of x & y inside the function are ‘ , x , y)
Print(‘value of x outside function is ‘ ‘, )
```

This code will produce following output:

```
Value of x & y inside the function are 5  2
Value of x outside the function is 5
```

DEFAULT ARGUMENT

A default argument is a function parameter that has a default value provided to it. If the user does not supply a value for this parameter, the default value will be used. If the user does supply a value for the default parameter, the user-supplied value is used.

Eg.

```
def greet (message, times=1):
      print message * times

>>> greet (‘Welcome’)            #  function call with one argument value
>>> greet (‘Hello’, 2)       # function call with both the argument values.
```

Output:

```
Welcome
HelloHello
```

QUESTION BASED ON FUNCTIONS

- What is the difference between methods, functions & user defined functions.
- Open help for math module

i. How many functions are there in the module?

ii. Describe how square root of a value may be calculated without using a math module

iii. What are the two data constants available in math module.

- Create a python module to find the sum and product of digits (separately) and imports in another program.
- Create a python function to find the a year is leap year of not a leap year
- What is local and global variable? Is global is keyword in python?
- Create a python module to find pow(x,n) and import in another program
- Write a function roll_D ( ), that takes 2 parameters- the no. of sides (with default

value 6) of a dice, and the number of dice to roll-and generate random roll values for each dice rolled. Print out each roll and then return one string "That's all".

Example roll_D (6, 3)

- 4
- 1
- 6