Computational Thinking and Programming - 1

Working with Modules

XI

Python Modules

A Python module is a file which contains some variables and constants, some functions, objects defined in it, which can be used in other Python programs. In order to use a module, it needs to be first imported and then the module functions, variables, constants and other objects can be used in the program file.

math module

math	Description
pi	Value of the constant pie
e	Value of the constant exponent
sqrt	Returns the square root
fabs	Returns the absolute value (float)
ceil	Returns the ceiling value(the largest integer)
floor	Returns the floor value(the smallest integer)
pow	Return raised to power
sin	Returns the value of sine
cos	Returns the value of cosine
tan	Returns the value of tan

Importing Modules - Method 1

# Importing a module with all the variables/objects/constants

# functions and classes

import <ModuleName>

to import the entire module

CODE:

```
import math
print(math.pow(5,2))
A = -1.5
print(A,math.fabs(A))
```

OUTPUT:

```
25.0
-1.5 1.5
```

Importing Modules - Method 2

# Importing a module with some specific variables/objects/constants

# functions and classes. No need to prefix the module name for accessing

# those particular variables/objects/constants/functions and classes.

from <ModuleName> import <object>,<functions>,<classes>

to import selected objects from a module

CODE:

```
from math import pow,fabs
print(pow(5,2))
A = -1.5
print(A,fabs(A))
```

OUTPUT:

```
25.0
-1.5 1.5
```

Importing Modules - Method 3

# Importing module along with all

# variables/objects/functions and classes

import <ModuleName> as <identifierName>

The import statement can be used to import the entire module.

CODE:

```
import math as mt
print(mt.pow(5,2))
A = -1.5
print(A,mt.fabs(A))
```

OUTPUT:

```
25.0
-1.5 1.5
```

Importing Modules - Method 4

# Importing a module with **ALL** specific variables/objects/constants

# functions and classes. No need to prefix the module name for accessing

# those particular variables/objects/constants/functions and classes.

from <ModuleName> import *

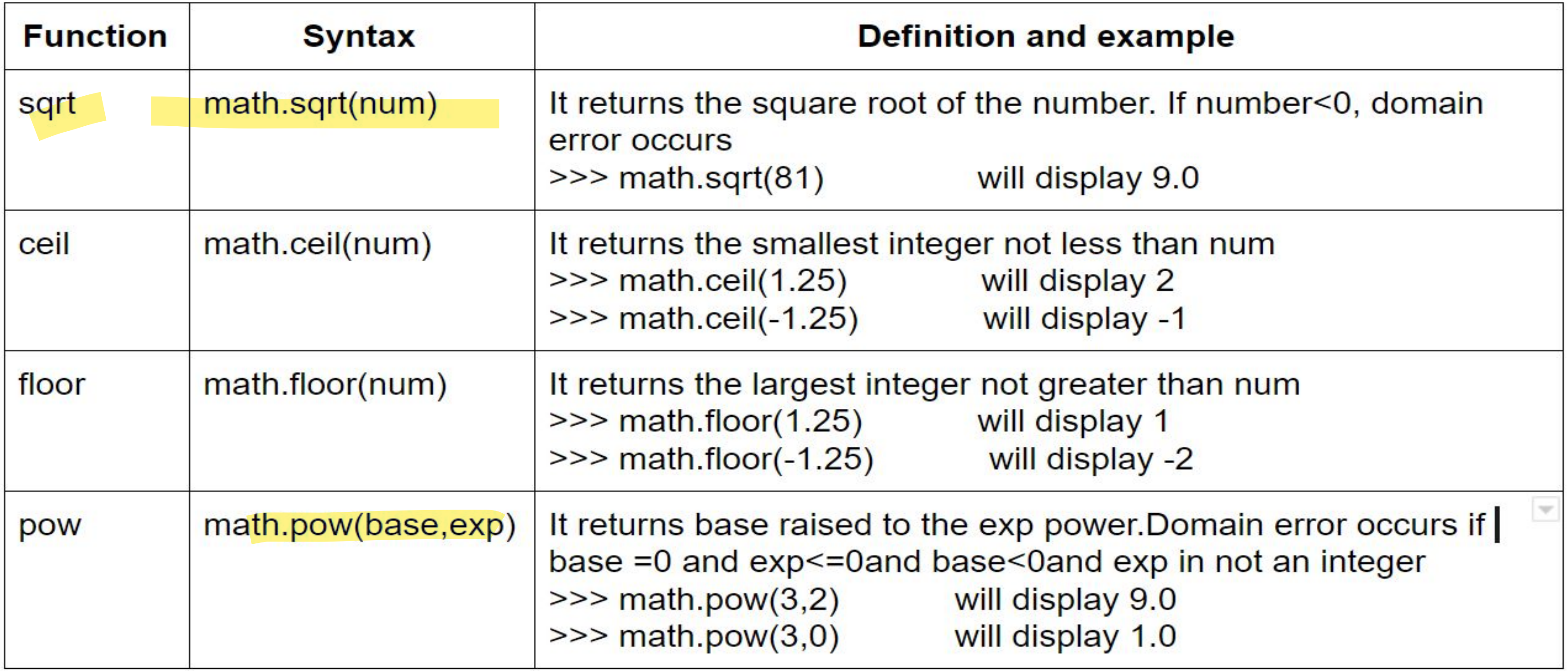
CODE:

```
from math import *
print(pow(5,2))
A = -1.5
print(A,fabs(A))
```

OUTPUT:

```
25.0
-1.5 1.5
```

math module



Function	Syntax	Definition and example
sqrt	math.sqrt(num)	It returns the square root of the number. If number<0, domain error occurs >>> math.sqrt(81) will display 9.0

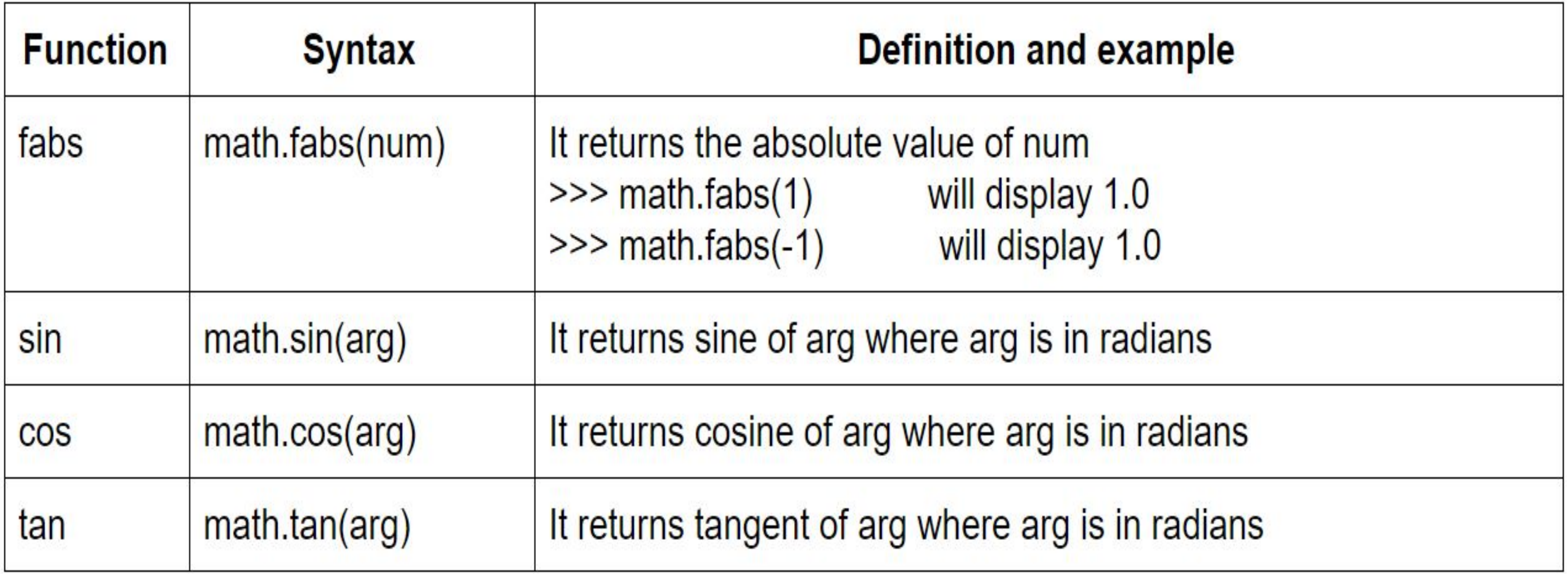
ceil	math.ceil(num)	It returns the smallest integer not less than num
		>>> math.ceil(1.25) will display 2
		>>> math.ceil(-1.25) will display -1

floor	math.floor(num)	It returns the largest integer not greater than num
		>>> math.floor(1.25) will display 1
		>>> math.floor(-1.25) will display -2

pow	math.pow(base,exp)	It returns base raised to the exp power.Domain error occurs if base =0 and exp<=0and base<0and exp in not an integer
		>>> math.pow(3,2) will display 9.0
		>>> math.pow(3,0) will display 1.0

fabs	abs	
In the math module	Built in function	
The value returned is always a float	The value returned depends on the argument passed.	

math module (contd.)



Function	Syntax	Definition and example

fabs	math.fabs(num)	It returns the absolute value of num
		>>> math.fabs(1) will display 1.0
		>>> math.fabs(-1) will display 1.0

sin	math.sin(arg)	It returns sine of arg where arg is in radians
cos	math.cos(arg)	It returns cosine of arg where arg is in radians
tan	math.tan(arg)	It returns tangent of arg where arg is in radians

math module (contd.)

The math module also makes available two useful constants namely pi and e which can be used as

math.pi gives the mathematical constant

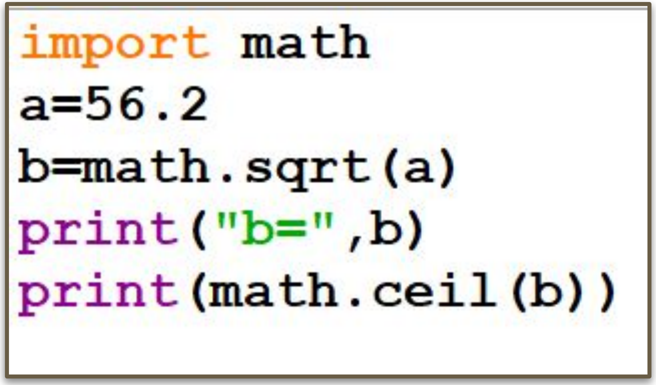
π= 3.141592…… to available precision

math.e gives the mathematical constant

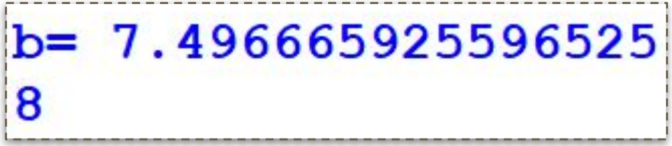
e= 2.718281…… to available precision

math module (contd.)

Example :



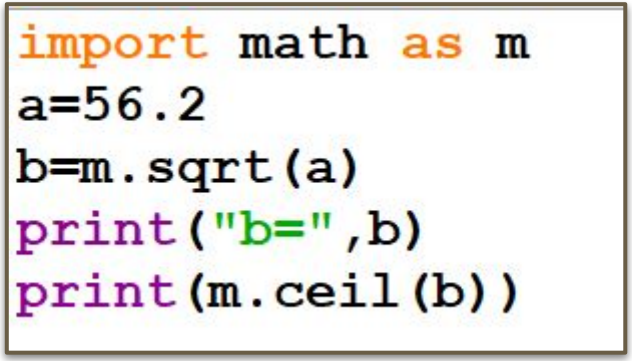
```
import math
a=56.2
b=math.sqrt(a)
print("b=",b)
print(math.ceil(b))
```



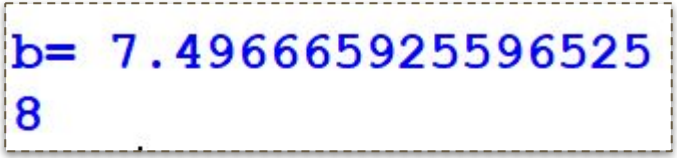
```
b= 7.496665925596525
8
```

Importing a module in the program and then calling its methods/objects by prefixing module name

math module (contd.)



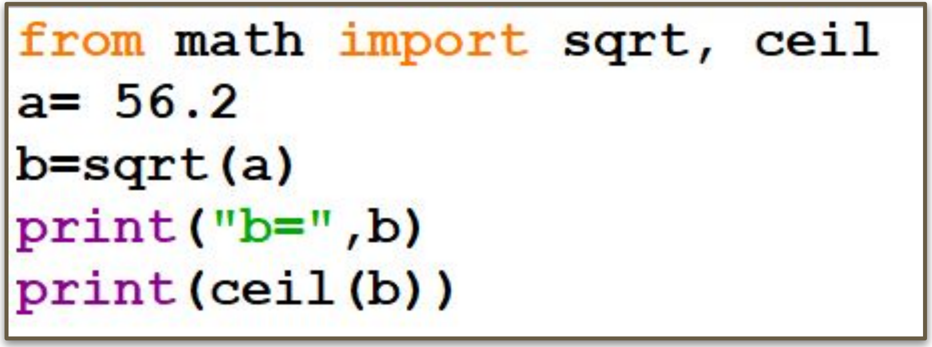
```
import math as m
a=56.2
b=m.sqrt(a)
print("b=",b)
print(m.ceil(b))
```



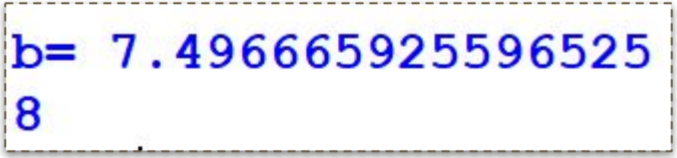
```
b= 7.496665925596525
8
```

Importing a module in the program and providing its alias to prefix a short name while calling its methods/objects

math module (contd.)



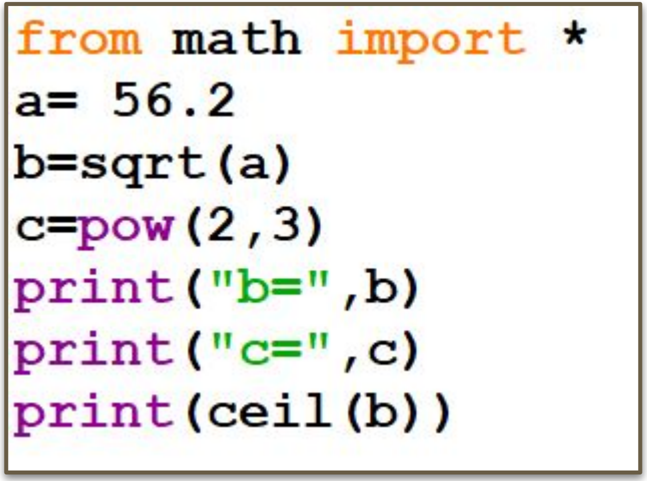
```
from math import sqrt, ceil
a= 56.2
b=sqrt(a)
print("b=",b)
print(ceil(b))
```



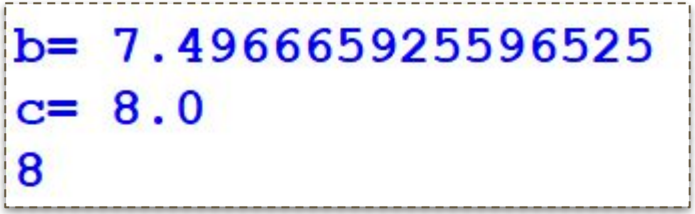
```
b= 7.496665925596525
8
```

Importing specific methods/objects from a module in the program and calling them without a prefix

math module (contd.)



```
from math import *
a= 56.2
b=sqrt(a)
c=pow(2,3)
print("b=",b)
print("c=",c)
print(ceil(b))
```



```
b= 7.496665925596525
c= 8.0
8
```

This statement will help in importing all methods/ objects of a module. So there will be no need to prefix module name while calling functions

random module

This module provides random number generators. To use random numbers, firstly import the random module as :

```
import random
```

There are three most common random number generators functions in random modules are :

- random()
- randint()
- randrange()

random module

Function Name	Syntax	Description
random	random()	returns a random floating point number N in the range [0.0, 1.0).
randint	randint(<Start>,<End>)	it returns a random integer >=Start Value and <= End Value
randrange	OPTIONAL OPTIONAL randrange(<Start>,<End>,<Step>)	Returns a random integer >=Start Value (Optional) and < End Value (Required) with a Step (Optional) value. By default Step is 1.

random module (contd.)

random( )- it returns a random floating point number N in the range [0.0, 1.0).

To generate a random floating point number between 0.0 to 1.0, simply use :

```
>>>import random
>>>print(random.random())
    0.02235193431
```

To generate a random floating point number between range lower to upper :

```
>>>import random
>>>print(random.random()*(upper-lower)+lower)
```

random module - random()

FUNCTION	EXAMPLE	OUTPUT	DESCRIPTION
random()	`print(random.random())`	0.85961520150273 0.15450708551458736	Returns a random floating point number N in the range [0.0, 1.0).

random()	`lower=10`
	`upper=20`
	`RV=random.random()*(upper-lower)+lower`
	`print(RV)`
		19.56150850699983	To generate a random floating point number >= lower value and < upper value.

random module - randint() and randrange()

FUNCTION	EXAMPLE	OUTPUT	DESCRIPTION
randint()	print(random.randint(15, 35)) print(random.randint(15, 35))	35	Prints a random integer >=15 &<= 35
randrange()	print(random.randrange(45)) print(random.randrange(45))	13 23	Prints a random number >=0 & <45 By default, start =0, Step = 1
randrange()	print(random.randrange(10,45)	43	Prints a random number >=10 & < 45 By default, step = 1
randrange()	print(random.randrange(11,45,4)) print(random.randrange(11,45,4)) print(random.randrange(11,45,4))	35 39 11	generate a random number between 11 and 45 with a step value of 4

dir() method

In Python, there is a **dir()** method which can list all functions and attributes of a module.

```
>>> import math
>>> print(dir(math))
```

would print the list of functions and attributes of the math module.

statistics module

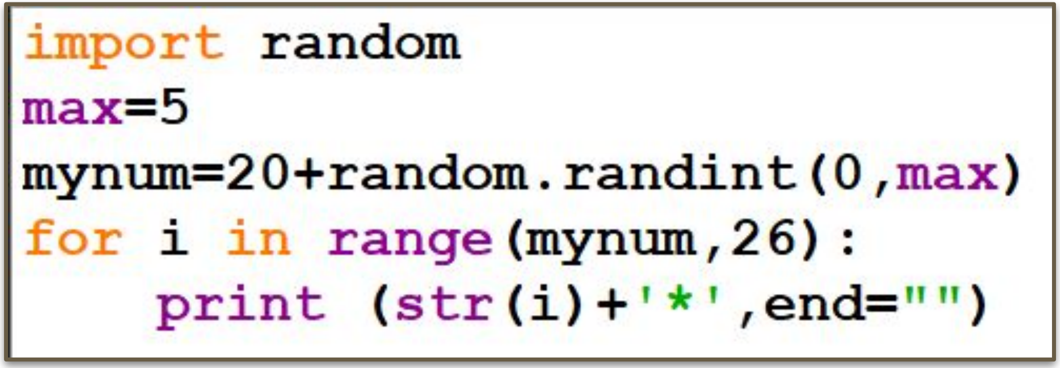
statistics	Example	Output	Description

mean	score = [10,20,20,30,40,40,50]
	from statistics import *
	print("Mean : ",mean(score))
	print("Mode : ", mode(score))
	print("Median : ",median(score))
	print("Using FOR loop")
	s=0
	for i in score:
	    s+=i
	print("Mean :",s/len(score))
	print("Another way")
	print(sum(score)/len(score))
		Mean : 30
		Mode : 20
		Median : 30
		Using FOR loop
		Mean : 30.0
		Another way
		30.0
			Return the mean of the collection.

median			Returns the middle value of the collection
mode			Returns the most often repeated value of the collection.

QUESTIONS

What are the possible outcome(s) expected from the following python code? Also specify maximum and minimum value, which we can have for the variable mynum.



```
import random
max=5
mynum=20+random.randint(0,max)
for i in range(mynum,26):
    print (str(i)+'*',end="")
```

i) 20*21*22*23*24*25
ii) 22*23*24*25*
iii) 23*24

iv)21*22*23*24*25
v) None of the above

Difference between pow(), math.pow() and **

pow()	math.pow()	**
Built in function	Belongs to math module	Operator
Returns integer result if both arguments are integer.	Always returns a float even if both the arguments are integer	Returns integer result if both are integer, otherwise returns float.

```
import math
print(math.pow(2,3))                            8.0
print(math.pow(2.0,3))                          8.0
print(pow(2,3))                                 8
print(pow(2.0,3))                               8.0
print(2**3)                                     8
print(2.0**3)                                   8.0
```

pow() function - 3 parameters

pow(x,y[,z]) x, y, z may be integer or floating point number

It results in x^y (x raised to the power y) if z is not provided

if z is provided, then: (x^y) % z

CODE:

```
print(pow(5,2))
print(pow(5.0,2))
print(pow(3,3,4))
```

OUTPUT:

```
25
25.0
3
```

SOLUTION

Maximum value assigned to the variable mynum - **25**
Minimum value assigned to the variable mynum - **20**

Correct possible outcomes:
ii) 22*23*24*25*

Incorrect outcomes:
i) 20*21*22*23*24*25 - does not terminate with *
iii) 23*24 - does not terminate with *
iv)21*22*23*24*25 - does not terminate with *

THANK YOU!

DEPARTMENT OF COMPUTER SCIENCE