Functions:

1. Using Math functions:

We have been using functions in Python since the beginning of the course. These functions include print, input, int, range, and type etc. The Python standard library includes many other functions useful for common programming tasks.

In mathematics, a function computes a result from a given value; eg, if f(x)=$\sqrt{(x+1)}$, then f(5)=$\sqrt{6} =$ 2.45. But Python interpreter is not automatically aware of the sqrt function. The sqrt function is not part of the standard functions (like type, int, str, and range) always available to Python programs. The sqrt function is part of *math module*. A module is a collection of Python code that can used in other programs. The statement

```
from math import sqrt
```

makes the sqrt function available for use in the program. Or simply

```
import math
```

to import complete math module.

The math module has many other mathematical functions. These include trigonometric, logarithmic, hyperbolic, and other mathematical functions.

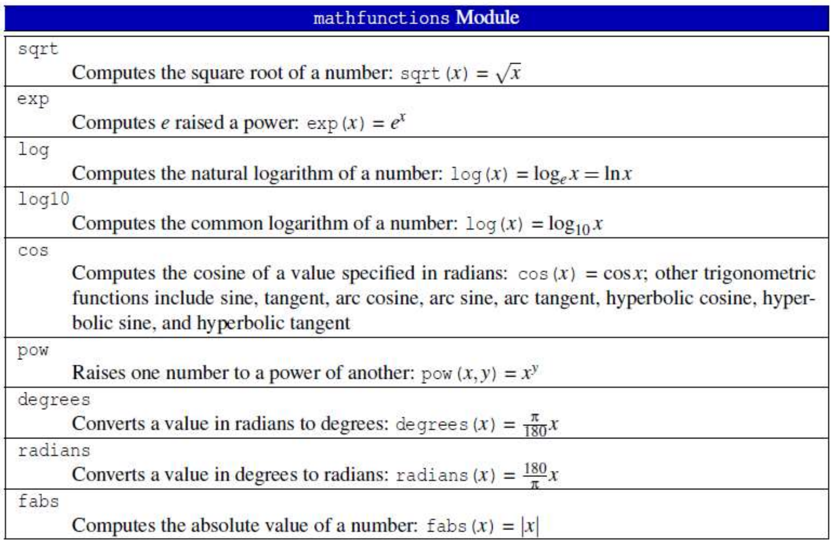
When calling a function, the function's name is followed by parentheses that contain the information to pass to the function so it can perform its task, eg. sqrt()

Eg. Write a program to perform square root or other mathematical functions

Program

```
import math             #or from math import sqrt     # the use of math functions from standard library
x=int(input("please enter a positive integer: "))
num=math.sqrt(x)          #using a method in Math.  i.e. use dot notation
print("The square root of num is ", sqrt(num)
```

The standard math module provides much of the functionality of a scientific calculator. The table below shows some of the available functions.



mathfunctions Module	
sqrt	Computes the square root of a number: sqrt $(x) = \sqrt{x}$
exp	Computes e raised a power: exp $(x) = e^x$
log	Computes the natural logarithm of a number: log $(x) = \log_e x = \ln x$
log10	Computes the common logarithm of a number: log $(x) = \log_{10} x$
cos	Computes the cosine of a value specified in radians: cos $(x) = \cos x$; other trigonometric functions include sine, tangent, arc cosine, arc sine, arc tangent, hyperbolic cosine, hyperbolic sine, and hyperbolic tangent
pow	Raises one number to a power of another: pow $(x, y) = x^y$
degrees	Converts a value in radians to degrees: degrees $(x) = \frac{\pi}{180}x$
radians	Converts a value in degrees to radians: radians $(x) = \frac{180}{\pi}x$
fabs	Computes the absolute value of a number: fabs $(x) = \lvert x \rvert$

Activity 1:

Write a program to compute the circumference and area of a circle given the radius input by the user.

2. Random Numbers

Random numbers are useful particularly in games and simulations. For example, many board games use a die to determine how many places a player is to advance. Or a player rolls a die and the value of a face after a roll is determined at random.

The random module provides functions that generate random numbers. The function random (ie. random()) returns a random *float between 0.0 and 1.0* (including 0.0 but not 1.0). To see a sample, run this loop:

Program	Remarks

import random
for i in range(10):
   x = random.random()
   print(x, end=" ")
	1. Observe its output. Briefly describe what it does?
	
	2. How can we modify the program so that it can print 10 integers?

To generate 10 integers from 1 to 10, try the following program:	Output

import random
for i in range (10):
   x = int(random.random()*10)
   print (x, end = " ")	

Activity 2:

Can you modify the above program so that it can generate 10 numbers from 1 to 100?

The function randint takes parameters *low* and *hig*h and returns an integer between The above method can be the method to generate positive integers. However, we are going to explore another method using randint which has the following format:

```
randint(x, y)
```

where x is the low value while y is the high value. Randint is used to generate an integer between the x and y (including both). For example

```
>>> random.randint(5, 10)
    5
>>> random.randint(5, 10)

    9
```

Activity 3:

Using randint, write a program to generate any 6 numbers from 1 to 44.

3 Adding new functions

So far, we have only been using the functions that come with Python, but it is also possible to *add new functions*. A **function definition** specifies the name of a new function and the sequence of statements that execute when the function is called.

For example:

Program	Explanation
def message():	The first line of the function definition is called the **header.** The empty parentheses after the function name indicate that this function doesn't take any arguments. *The header has to end with a colon and the **body** has to be indented.*
print ("Good day. I'm happy.")	By convention, the indentation is always *four spaces*. The body can contain any number of statements.
print ("Coding is fun.")	
message () *# for calling the function. This is # the main program*	The syntax for calling the new function is the same as for built-in functions: message()

Note that you have to create a function **before** you can execute it. In other words, the function definition has to be executed before the first time it is called (ie. The function must come before the main program.)

Activity 4:

Create a program, by using your own function, which is used to print out the product of three numbers entered by the user. Give the program the name factorial.py

4. Functions with parameters passing

The program on factorial.py can be further modified as follows:

Program	Explanation
def factorial(num):	The factorial function expects an integer parameter, called num. It returns an integer result called product to the main program.
product = 1	
for i in range(1, num+1): *#note that (1, 3) does not include 3. So we have to make it 3+1*	The function uses 3 local variables called num, product and i. Local variables have meaning only within the function. Two functions can have the same named variables. However, local variables cannot be used again in the main program.
product = i * product	
print(product)	
factorial(3) #this is the main body	From the main program, 3 is passed to the function; i.e. num = 3. The function then processes the factorial and print out product.
def factorial(num):	However, the result --- product can also be passed back to the main body. Consider the program from the left which is modified from the program above.
product = 1	
for i in range(1, num+1	
product = i * product	
return product	
print("The product of the numbers is ", factorial(3))	

More examples:

Listing 7.5: betterprompt.py

```
#  Definition of the prompt function
def prompt():
    value = int(input("Please enter an integer value: "))
    return value

print("This program adds together two integers.")
value1 = prompt()     #  Call the function
value2 = prompt()     #  Call the function again
sum = value1 + value2
print(value1, "+", value2, "=", sum)
```

The advantages of using functions

- Creating a new function gives you an opportunity to name a group of statements, which makes your program easier to read and debug.
- Functions can make a program smaller by eliminating repetitive code.
- Dividing a long program into functions allows you to debug the parts one at a time
- and then assemble them into a working whole.

Program Practice (for submission):

Modify your factorial.py so that the program can compute the product of n to m consecutive integers where n and m are keyboard entry.