# Week 6: Python Modules, Packages and Plotting

## Chapter 0: Recap

- Data types (*int, float, string, complex etc*) are implicitly declared by Python
- **if**..**else** statements are used to execute a block of statements if the condition is true, and another block of statements when the condition is false**.**
- `While`-loop and `For` loop to repeatedly execute a set of statements.
- `List, Set` and `Dictionary` are data types to store data.
- Files Input/Output

## Modules in Python

There are several common functions which have already been written in Python and are available for you to use. These functions are grouped into Modules or Libraries. In other words, Python Modules are a collection of built-in functions you can use.

To view all the available modules, type

```
>>> help('modules')
```

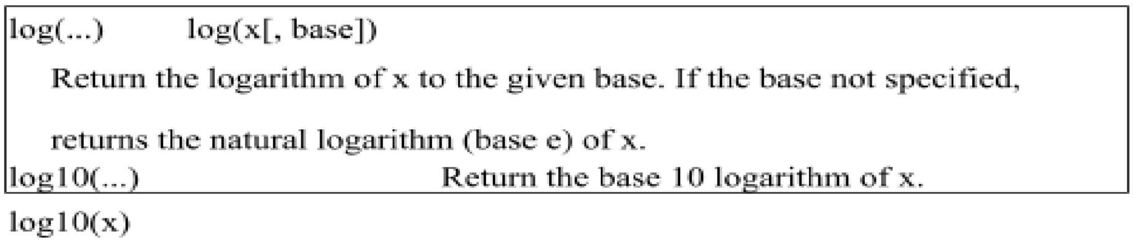
To view all the available functions with a specific module, say *math* module,:

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

http://docs.python.org/2/py-modindex.html → has a list of many python modules.

# Chapter I: Using Module - math

We want to compute *$log_{10}2$.* Now, if you scroll up, you can find the description of log functions available in **math** module:



```
log(...)        log(x[, base])
    Return the logarithm of x to the given base. If the base not specified,
    returns the natural logarithm (base e) of x.
log10(...)                      Return the base 10 logarithm of x.
log10(x)
```

So, let's try:

```
>>> log10(2)
```

You should get an ERROR message as output, saying "*name log10 is not defined*"!

To use the functions in the **math** module, we need to TELL python to USE the math module. To do that, type:

```
>>> import math
>>> math.log10(2)
```

# Chapter II: Using Packages

Packages are larger modules written in python and available for your use. The list of all the packages are available at https://pypi.python.org/pypi

If you want any of the packages, you need to install them. We shall learn about packages that are very useful for scientific computing: **numpy, scipy** and **matplotlib.** The **numpy** package can be used for array processing of numbers, strings and objects. The **matplotlib** package can be used for plotting.

## 1. Using matplotlib

Let's plot $x$ vs. $x^2$, for $x$ from 2 to 10.

`>>> x =[2,3,4,5,6,7,8,9,10]` → list $x$

`>>> y =[4,9,16,25,36,49,64,81,100]` → list $x^2$,

`>>> import matplotlib.pyplot as plt` → imports subpackage *pyplot* from *matplotlib* package, which we can simply refer to in the code as *plt.*

```
>>> plt.plot(x,y,'.')
>>> plt.show()
```

You should be able to see a plot of $x$ vs. $y$. To view all the options with the plot function, type:

```
>>> help('matplotlib.pyplot.plot')
```

Let's change the display of the plot, to use solid lines, with red color line and a * marker at the data points, and the axis are to be labelled.

```
>>> plt.plot(x,y,'r*-')
>>> plt.title('My First Graph', fontsize=12)
>>> plt.xlabel(r'$x$', fontsize=14)
>>> plt.ylabel(r'$x^2$', fontsize=14)
>>> plt.show()
```

## 2. Using numpy package

`>>> help('numpy')` → lists all numpy functions

`>>> import numpy as np` → imports *numpy* package, which we can simply refer in the code as *np*

`>>> p = [2.3,4.5,6.7,8.9]` → define a list

`>>> np.log(p)` → will display an *array* where every element is the natural logarithm of the corresponding element of $x$.

Note: `math.log(p)` will give an error since the math module cannot handle lists.

What is array? Array is a data structure similar to lists. Numpy uses a multi-dimensional array structure.

## 3. The `linpsace` function

The linspace function can be used to create a sequence of numbers within a range. Some examples:

`>>> np.linspace(0,20,5)` → Creates an array (list) of 5 evenly spaced numbers between 0 and 20, that includes 0 and 20.

`>>> np.linspace(1,20,10)` → Creates an array (list) of 10 evenly spaced numbers between 1 and 20, that includes 1 and 20.

## 4. Combined use of numpy and plotting

Let's write a code to plot the Sine wave. Write the following code in a new file.

```
import numpy as np
import matplotlib.pyplot as plt

x =  np.linspace(0,2*np.pi,10)

y = np.sin(x)
plt.plot(x,y)

plt.savefig('sine_plain.png')  → saves graph to file

plt.grid(True)          → shows gridlines

plt.show()
```

You should see a very jagged sine wave.

Make the following modifications to the above file:

- In the `linspace` command, change 10 to 100.
- Include command to give x-axis label as "*x*", y-axis label as "sin(*x*)", and graph title as "My Sine Wave"

Run the modified model and generate a new graph.

**5. Plotting multiple functions**

Let's write a code to plot the Sine wave and Cos wave on the same graph. Write the following code in a new line.

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0,2*np.pi,100)
f = np.sin(x)
g = np.cos(x)
plt.plot(x,f,label=r'$f(x) = \sin(x)$')
plt.plot(x,g,label=r'$g(x) = \cos(x)$')
plt.title('Sine and Cosine graph', fontsize=16)
plt.xlabel(r'$x$',fontsize=14)
plt.ylabel(r'Function of $x$', fontsize=14)
plt.legend(loc='lower left')
plt.savefig('sine_cos.png')
plt.show()
```

## 6. Matrix computation using NumPy

We can do basic linear algebra using NumPy.

```
>>> import numpy as np
```
→ imports *numpy* package, which we can simply refer in the code as *np*

```
>>> Amat = np.matrix('1 7 3; 4 0 6; 2 5 -1')
>>> print(Amat)
>>> Amat.T
```
→ Transpose of *Amat*

```
>>> Amat.I
```
→ Inverse of *Amat*

```
>>> Ymat = np.matrix('1 0 -1; -1 0 1; 0 -1 1')
>>> Amat*Ymat
```
→ matrix multiplication

```
>>> Cmat = np.matrix('2 3 7')
>>> bmat = Cmat.T
>>> Amat*bmat
```
→ matrix multiplication

There is a linear algebra module in numpy called *linalg* which can be used to do other linear algebra computations.

```
>>> np.linalg.det(Amat)
```
→ return determinant of *Amat*

```
>>> np.linalg.eig(Amat)
```
→ return eigenvalues of *Amat*

```
>>> np.linalg.solve(Amat, bmat)
```
→ *Solves Ax=b,*

## 7. Generating random Distributions

### Normal Distribution

Generate a random normal distribution of size 'l' with mean at 'mu' and standard deviation of 'sigma'

```
from numpy import random
mu,sigma,l =(0,1,10)
x=random.normal(mu,sigma,l)
print(x)
```

For given x, mu, sigma

We can find the F(x) i.e cdf of x
e.g.

```
from scipy.stats import norm
x , mu,sigma = (2,0,10)
F_x = norm.cdf(x,mu,sigma)
print(F_x)
```

Inverse of cdf – percentile

```
Inv_F_x = norm.ppf(F_x,0,10)
print(Inv_F_x)
```

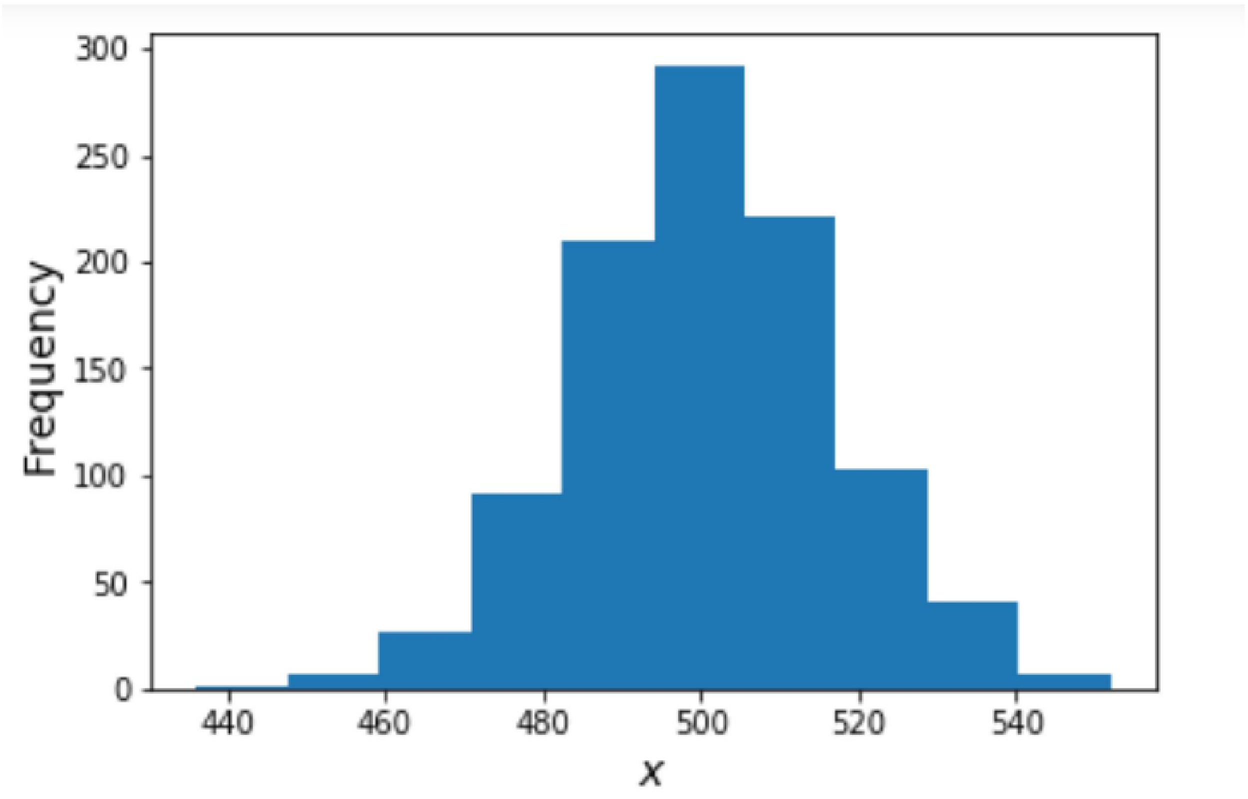
**Binomial Distribution**

e.g.
Given 10 trials for coin toss generate 10 data points

```
x = random.binomial(n=10, p=0.5, size=10)
print(x)
```

To see the behavior of the distribution we can plot a histogram
To plot a histogram we use 'hist' from matplotlib

```
x = random.binomial(n=100, p=0.5, size=100)
plt.hist(x)
plt.xlabel(r'$x$',fontsize=14)
plt.ylabel(r'Frequency', fontsize=14)
plt.show()
```



We can see that as p = 0.5, the coin is fair and the number of heads or tails should be equal i.e close to 500 which we can see from the above graph.

https://docs.scipy.org/doc/scipy/reference/stats.html

https://www.w3schools.com/python/numpy_random_normal.asp

In above two links you can find scipy and numpy libraries for different distribution functions.

Exercises

1. Using the appropriate function from the **math** module, compute the following:

   $e^x$ , $|x|$, $\ln x$, $x^y$, fractional and integer parts of $x$, for values of $x = 2.34$, and $x = -5.67$, and $y = 1.5$; cos(π), tan(π/4), π/4 in degrees.

2. Do the following:

   Create a list or array of 20 numbers between 3 and 4.

   Create a list or array of 50 numbers between -π and π.

3. Create a plot of function $e^{-x}\cos(6\pi x)$; where x takes on 200 data points over interval (0,2). Bound the plot by the functions $e^{-x}$ and $-e^{-x}$. Give different colors/patterns for each curve.

4. Generate 10 random values which follow the given distribution, and also find the cdf for each value

   Poisson distribution (parameter, mu = 0.6)

   Exponential distribution(parameter, lambda =0.6)

5. Compute the following (if they exist)

   (a) Determinant

   (b) Inverse

   (c) Eigen values of the product D=ABC.

$$\text{Let } A = \begin{pmatrix} 2 & 4 & 1 & 0 \\ -1 & 2 & 3 & 1 \\ 2 & 5 & -1 & 2 \end{pmatrix} \quad B = \begin{pmatrix} 2 & -2 \\ -1 & 0 \\ 4 & 1 \\ -3 & 2 \end{pmatrix} \quad C = \begin{pmatrix} 1 & -3 & 2 \\ 2 & 0 & 2 \end{pmatrix}$$