3 Plotting using PyLab & Multithreading

TOPICS COVERED :

- Plotting using PyLab
- Extended examples
- Thread
- Starting a thread
- Threading module
- Synchronizing threads
- Multithreaded Priority

Plotting using PyLab:

- PyLab is a Python standard library module that provides many of the facilities of MATLAB, "a high- level technical computing language and interactive environment for algorithm development, data visualization, data analysis, and numeric computation
- PyLab is a procedural interface to the Matplotlib object-oriented plotting library.
- Matplotlib is the whole package; matplotlib.pyplot is a module in Matplotlib; and PyLab is a module that gets installed alongside Matplotlib.
- PyLab is a convenience module that bulk imports matplotlib.pyplot (for plotting) and NumPy (for Mathematics and working with arrays) in a single name space. Although many examples use PyLab, it is no longer recommended.

Matplotlib (Python Plotting Library):

- matplotlib.pyplot is a plotting library used for 2D graphics in python programming language.
- It can be used in python scripts, shell, web application servers and other graphical user interface toolkits.
- Matploitlib is a Python Library used for plotting, this python library provides and objected-oriented APIs for integrating plots into applications.
- Human minds are more adaptive for the visual representation of data rather than textual data.
- We can easily understand things when they are visualized.
- It is better to represent the data through the graph where we can analyze the data more efficiently and make the specific decision according to data analysis.
- Before learning the matplotlib, we need to understand data visualization and why data visualization is important.
- Matplotlib 2.0.x supports Python versions 2.7 to 3.6 till 23 June 2007.
- Python3 support started with Matplotlib 1.2. Matplotlib 1.4 is the last version that supports Python 2.6.

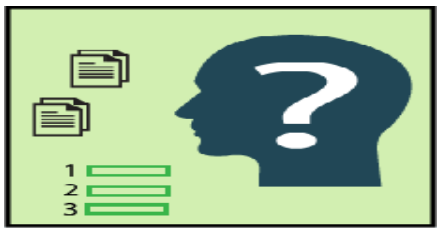
- There are various toolkits available that are used to enhance the functionality of the matplotlib.
- Some of these tools are downloaded separately, others can be shifted with the matplotlib source code but have external dependencies.

➢ **Bashmap:** It is a map plotting toolkit with several map projections, coastlines, andpolitical boundaries.
➢ **Cartopy:** It is a mapping library consisting of object-oriented map projection definitions,and arbitrary point, line, polygon, and image transformation abilities.
➢ **Excel tools:** Matplotlib provides the facility to utilities for exchanging data withMicrosoft Excel.
➢ **Mplot3d:** It is used for 3D plots.
➢ **Natgrid:** It is an interface to the Natgrid library for irregular gridding of the spaced data.

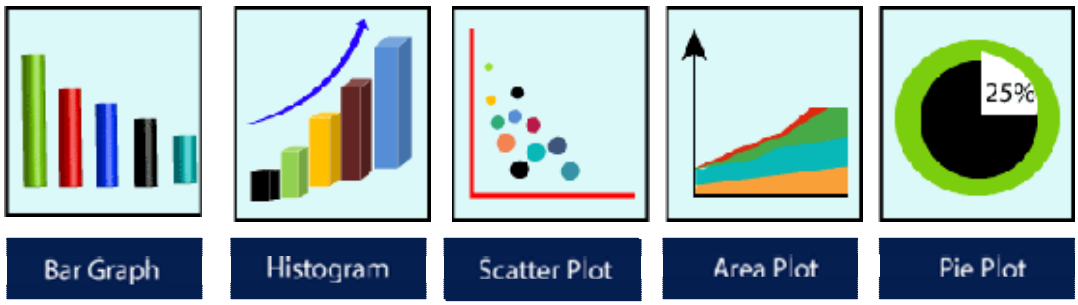
Python(Matplotlib) vs. MATLAB:

Python Programming	MATLAB
It is an open-source programming language, free to use.	MATLAB is a commercial platform. Hence, it is not free.
Matplotlib is more flexible and capable for plotting.	Plotting is comparatively not as flexible and capable as Python plotting.
Python provides a large number of libraries to work with.	It is tricky to add libraries and work with them in MATLAB.
Python is an easy-to-read and powerful programming language.	MATLAB is not as powerful as Python.
Matplotlib plotting is faster in Python.	Plotting of data in MATLAB requires time and effort.
Integrated development environment (IDE) needs to be added, additionally.	IDE will be provided within the MATLAB environment.
Code can be used in multiple systems. It is portable.	Code portability is restricted.
Namespace is supported in Python.	Core of MATLAB does not support namespace.

Data Visualization:



- Graphics provides an excellent approach for exploring the data, which is essential for presenting results.
- Data visualization is a new term.
- It expresses the idea that involves more than just representing data in the graphical form (instead of using textual form).
- This can be very helpful when discovering and getting to know a dataset and can help with classifying patterns, corrupt data, outliers, and much more.
- With a little domain knowledge, data visualizations can be used to express and demonstrate key relationships in plots and charts.
- The static does indeed focus on quantitative description and estimations of data.
- It provides an important set of tools for gaining a qualitative understanding.
- There are five key plots that are used for data visualization.



1. **Visualize:** We analyze the raw data, which means it makes complex data more accessible, understandable, and more usable. Tabular data representation is used where the user will look up a specific measurement, while the chart of several types is used toshow patterns or relationships in the data for one or more variables.
2. **Analysis:** Data analysis is defined as cleaning, inspecting, transforming, and modeling data to derive useful information. Whenever we make a decision for the business or in daily life, is by past experience. **What will happen to choose a particular decision**, it is nothing but analyzing our past. That may be affected in the future, so the proper analysisis necessary for better decisions for any business or organization.
3. **Document Insight:** Document insight is the process where the useful data or informationis organized in the document in the standard format.
4. **Transform Data Set:** Standard data is used to make the decision more effectively.

Why need data visualization?

- Data visualization can perform below tasks:
  - It identifies areas that need improvement and attention.
  - It clarifies the factors.
  - It helps to understand which product to place where.
  - Predict sales volumes.

Benefit of Data Visualization:

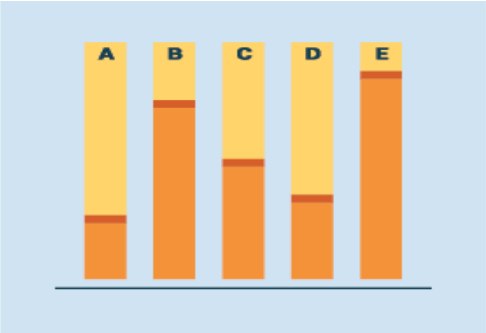
- Here are some benefits of the data visualization, which helps to make an effective decision for the organizations or business:

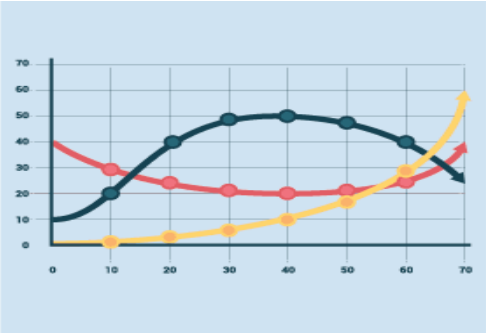
1. Building ways of absorbing information:

- Data visualization allows users to receive vast amounts of information regarding operational and business conditions.
- It helps decision-makers to see the relationship between multi-dimensional data sets.
- It offers new ways to analyses data through the use of maps, fever charts, and other rich graphical representations.
- Visual data discovery is more likely to find the information that the organization needs and then end up with being more productive than other competitive companies.

2. Visualize relationship and patterns in Businesses:

- The crucial advantage of data visualization is that it is essential to find the correlation between operating conditions and business performance in today's highly competitive business environment.





- The ability to make these types of correlations enables the executives to identify the root cause of the problem and act quickly to resolve it.
- Suppose a food company is looking their monthly customer data, and the data is presented with bar charts, which shows that the company's score has dropped by five points in the previous months in that particular region; the data suggest that there's a problem with customer satisfaction in this area.

3. Take action on the emerging trends faster:

- Data visualization allows the decision-maker to grasp shifts in customer behavior and market conditions across multiple data sets more efficiently.
- Having an idea about the customer's sentiments and other data discloses an emerging opportunity for the company to act on new business opportunities ahead of their competitor.

4. Geological based Visualization:

- Geo-spatial visualization is occurred due to many websites providing web-services, attracting visitor's interest.
- These types of websites are required to take benefit of location-specific information, which is already present in the customer details.

Matplotlib Architecture:

- There are three different layers in the architecture of the matplotlib which are the following:
  1. Backend Layer
  2. Artist layer
  3. Scripting layer

1. Backend layer:

- The backend layer is the bottom layer of the figure, which consists of the implementation of the various functions that are necessary for plotting.
- There are three essential classes from the backend layer FigureCanvas(The surface on which the figure will be drawn), Renderer(The class that takes care of the drawing on the surface), and Event(It handle the mouse and keyboard events).

2. Artist Layer:

- The artist layer is the second layer in the architecture.
- It is responsible for the various plotting functions, like axis, which coordinates on how to use the renderer on the figure canvas.

3. Scripting layer:

- The scripting layer is the topmost layer on which most of our code will run.
- The methods in the scripting layer, almost automatically take care of the other layers, and all we need to care about is the current state (figure & subplot).

The General Concept of Matplotlib:

- A Matplotlib figure can be categorized into various parts as below:
  1. Figure:
  - It is a whole figure which may hold one or more axes (plots).
  - We can think of aFigure as a canvas that holds plots.

2. **Axes:**
- A Figure can contain several Axes.
- It consists of two or three (in the case of 3D) Axis objects.
- Each Axes is comprised of a title, an x-label, and a y-label.

3. **Axis:**
- Axises are the number of line like objects and responsible for generating the graph limits.

4. **Artist:**
- An artist is the all which we see on the graph like Text objects, Line2D objects, and collection objects.
- Most Artists are tied to Axes.

Matplotlib Getting Started:

1. **Installation of Matplotlib:**
- If you have Python and PIP already installed on a system, then installation of Matplotlibis very easy.
- Install it using this command:
  C:\Users*Your Name*>pip install matplotlib

- If this command fails, then use a python distribution that already has Matplotlib installed, like Anaconda, Spyder etc.

2. **Import Matplotlib:**
- Once Matplotlib is installed, import it in your applications by adding the import module statement:
  import matplotlib
- Now Matplotlib is imported and ready to use:

3. **Checking Matplotlib Version:**
- The version string is stored under __version_____attribute.

```
import matplotlib
print(matplotlib.__vrsion )
```

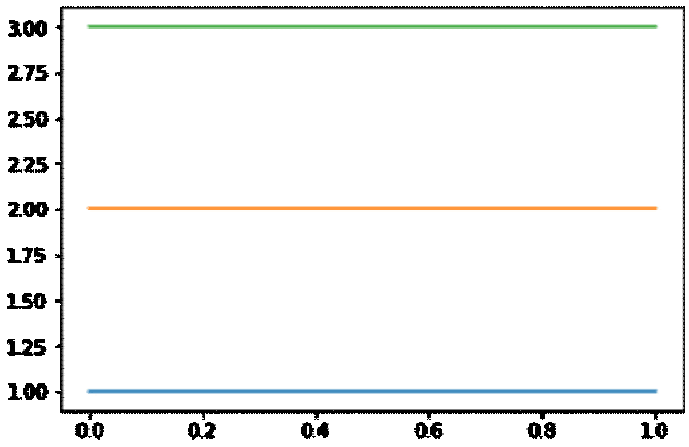
Note: two underscore character are used in __version

4. **Pyplot:**
- Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:
  import matplotlib.pyplot as plt

Python Matplotlib Example:

```
import matplotlib.pyplot as plt
plt.plot([1,1])
plt.plot([2,2])
plt.plot([3,3]) plt.show()
```

Output:



Plot:

```
import matplotlib.pyplot as plt
import numpy as np
plt.plot([1,1])
plt.show()
```

Output:

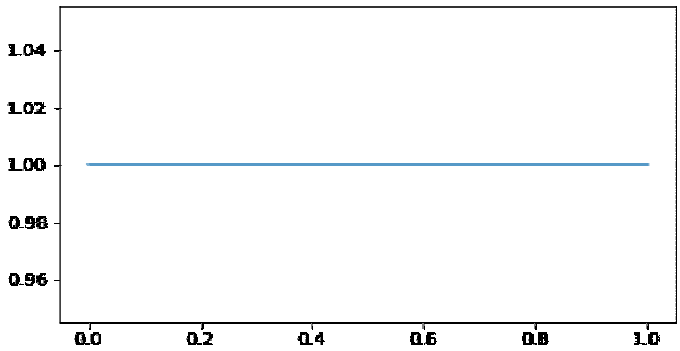
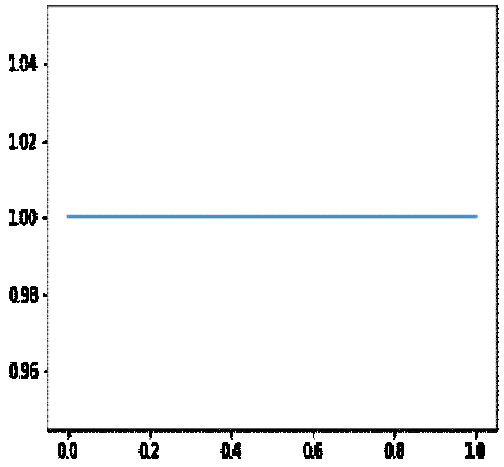


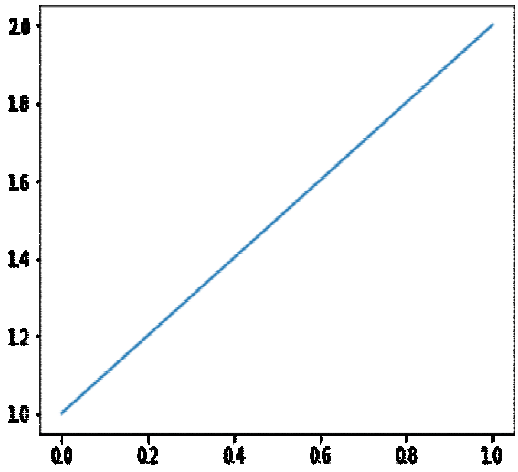
Figure:

- It is a diagram or a shape that can be formed by a collection of plots in different dimensions.

```
import matplotlib.pyplot as plt
import numpy as np
plt.figure(1)
plt.plot([1,1])
plt.figure(2)
plt.plot([1,2])
plt.show()
```

Output:



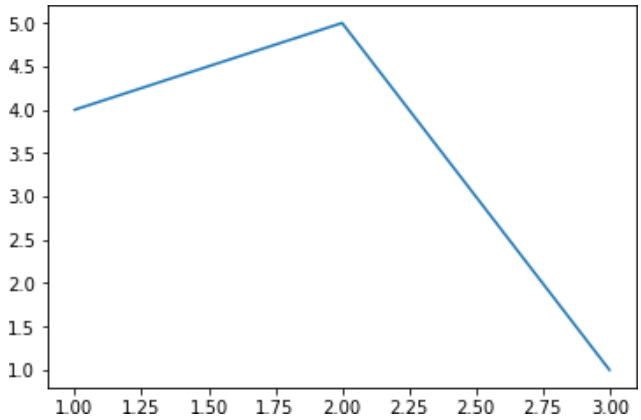


Basic Example of plotting:

Graph:

```
from matplotlib import pyplot as plt
#ploting our canvas
plt.plot([1,2,3],[4,5,1])
#display the graph
plt.show()
```

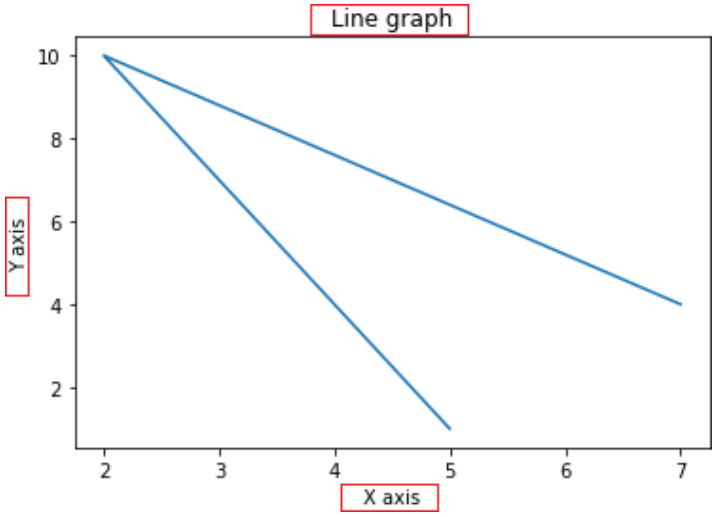
Output:



- It takes only three lines to plot a simple graph using the Python matplotlib.
- We can add titles, labels to our chart which are created by Python matplotlib library to make it more meaningful.
- The example is the following:

```
from matplotlib import pyplot as plt
x = [5, 2, 7]
y = [1, 10, 4]
plt.plot(x, y)
plt.title('Line graph')
plt.ylabel('Y axis')
plt.xlabel('X axis')
plt.show()
```

Output:

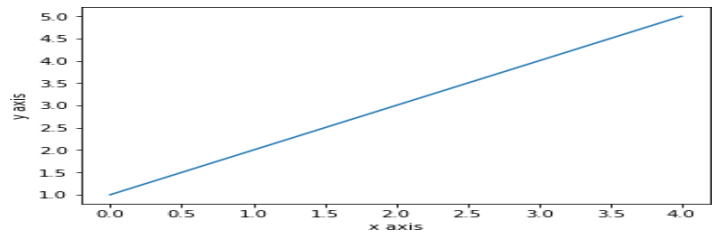


Working with Pyplot:

- The matplotlib.pyplot is the collection command style functions that make matplotlib feel like working with MATLAB.
- The pyplot functions are used to make some changes to figure such as create a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot including labels, etc.
- It is good to use when we want to plot something quickly without instantiating any figure or Axes.
- While working with matplotlib.pyplot, some states are stored across function calls so that it keeps track of the things like current figure and plotting area, and these plotting functions are directed to the current axes.
- The pyplot module provide the plot() function which is frequently use to plot a graph. Let's have a look on the simple example:

```
from matplotlib import pyplot as plt
plt.plot([1,2,3,4,5])
plt.ylabel("y axis")
plt.xlabel('x axis')
plt.show()
```

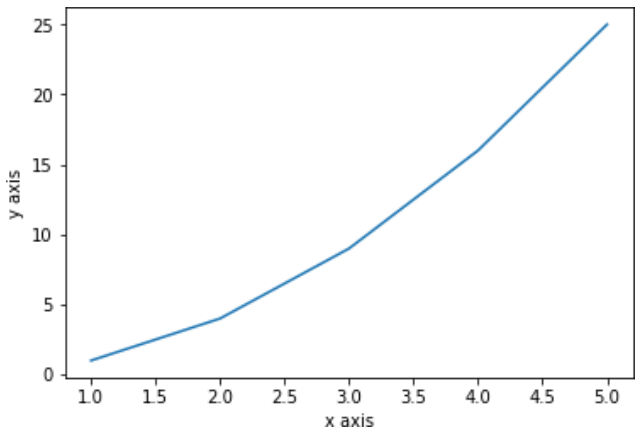
Output:



- In the above program, it plots the graph x-axis ranges from 0-4 and the y-axis from 1-5.
- If we provide a single list to the plot(), matplotlib assumes it is a sequence of y values, and automatically generates the x values.
- Since we know that python index starts at 0, the default x vector has the same length as y but starts at 0.
- Hence the x data are [0, 1, 2, 3, 4].
- We can pass the arbitrary number of arguments to the plot().
- For example, to plot x versus y, we can do this following way:

```
from matplotlib import pyplot as plt
plt.plot([1,2,3,4,5],[1,4,9,16,25])
plt.ylabel("y axis")
plt.xlabel('x axis')
plt.show()
```

Output:

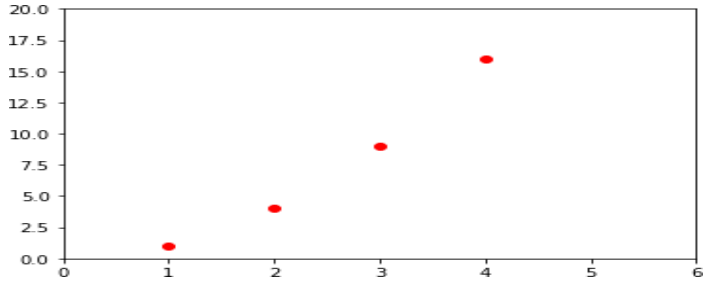


Formatting the style of the plot:

- There is an optional third argument, which is a format string that indicates the color and line type of the plot.
- The default format string is 'b-'which is the solid blue as you can observe in the above plotted graph.
- Let's consider the following example where we plot the graph with the red circle.

```
from matplotlib import pyplot as plt
plt.plot([1, 2, 3, 4,5], [1, 4, 9, 16,25],'ro')
plt.axis([0, 6, 0,20])
plt.show()
```

Output:



Example format String:

'b'	Using for the blue marker with default shape.
'ro'	Red circle
'-g'	Green solid line
'--'	A dashed line with the default color
'^k:'	Black triangle up markers connected by a dotted line

- The matplotlib supports the following color abbreviation:

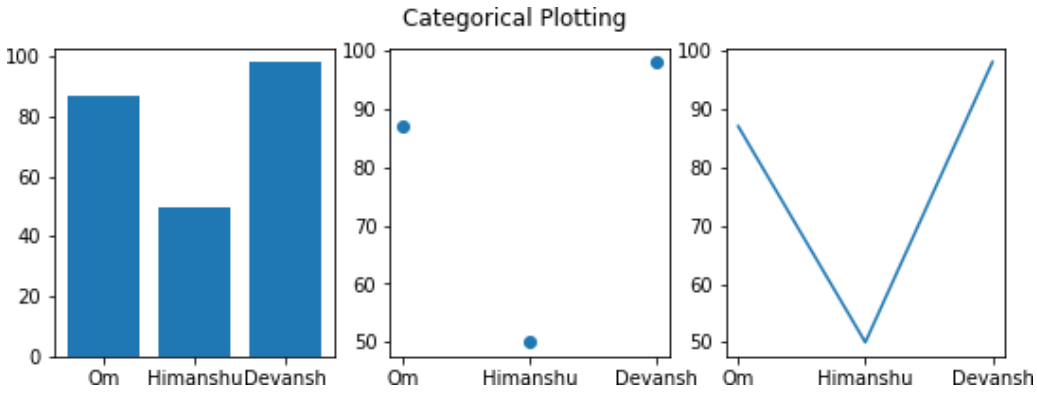
Character	Color
'b'	Blue
'g'	Green
'r'	Red
'c'	Cyan
'm'	Magenta
'y'	Yellow
'k'	Black
'w'	White

Plotting with categorical variables:

- Matplotlib allows us to pass categorical variables directly to many plotting functions: consider the following example

```
from matplotlib import pyplot
names = ['Abhishek', 'Himanshu', 'Devansh']
marks= [87,50,98]
plt.figure(figsize=(9,3))
plt.subplot(131)
plt.bar(names, marks)
plt.subplot(132)
plt.scatter(names, marks)
plt.subplot(133)
plt.plot(names, marks)
plt.suptitle('Categorical Plotting')
plt.show()
```

Output:



- In the above program, we have plotted the categorical graph using the subplot() function. Let's a have a look on the subplot() function.

What is subplot():

- The Matplotlib subplot() function is defined as to plot two or more plots in one figure.
- We can use this method to separate two graphs which plotted in the same axis Matplotlib supports all kinds of subplots, including 2x1 vertical, 2x1 horizontal, or a 2x2 grid.
- It accepts the three arguments: they are nrows, ncols, and index.
- It denote the number of rows, number of columns and the index.
- The subplot() function can be called in the following way:

```
subplot(nrows,ncols,index,**kwargs)
subplot(pos,**kwargs)
subplot(ax)
```

Parameters:

1. ***args:**

- Three separate integers or three-digit integer describes the position of the subplot.
- If the three integers are nrows, ncols, and index in order, the subplot will take the index position on a grid with nrows row and ncol column.
- The argument pos are a three-digit integer, where the first digit is denoted the number of rows, the second digit denoted the number of columns, and the third represents the index of the subplot.
- For example, subplot (1, 3, 2) is the same as the subplot (132).

Note: Passes integer must be less than 10

2. ****kwargs:**

- The subplot() function also accepts the keyword arguments for the returned axes base class.

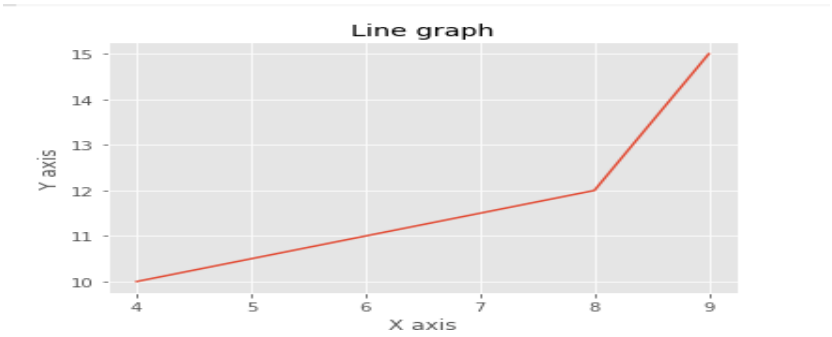
Creating different types of graph:

1. **Line graph:**

- The line graph is one of charts which shows information as a series of the line.
- The graph is plotted by the plot() function. The line graph is simple to plot; let's consider the following example:

```
from matplotlib import pyplot as plt
x = [4,8,9]
y = [10,12,15]
plt.plot(x,y)
plt.title("Line graph")
plt.ylabel('Y axis')
 plt.xlabel('X axis')
plt.show()
```

Output:

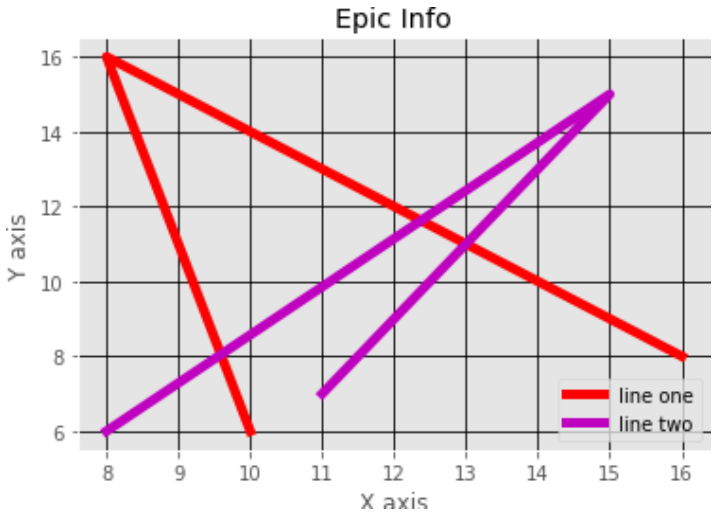


- We can customize the graph by importing the style module.
- The style module will be built into a matplotlib installation.
- It contains the various functions to make the plot more attractive.
- In the below program, we are using the style module:

```
from matplotlib import pyplot as plt
from matplotlib import style
style.use('ggplot')
x = [16, 8, 10]
y = [8, 16, 6]
x2 = [8, 15, 11]
y2 = [6, 15, 7]
```

```
plt.plot(x, y, 'r', label='line one', linewidth=5)
plt.plot(x2, y2, 'm', label='line two', linewidth=5)
plt.title('Epic Info')
fig = plt.figure()
plt.ylabel('Y axis')
plt.xlabel('X axis')
plt.legend()
plt.grid(True, color='k')
plt.show()
```

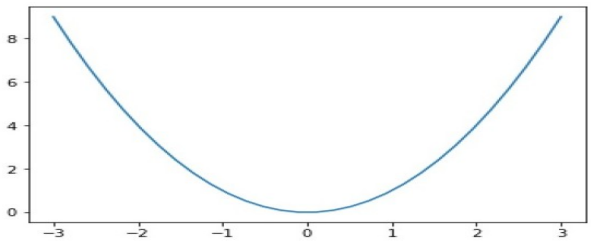
Output:



- Plotting curves is done with the plot command. It takes a pair of same-length arrays (or sequences) –

```
from numpy import *
from pylab import *
x = linspace(-3, 3, 30)
 y = x**2
plot(x, y)
show()
```

Output:

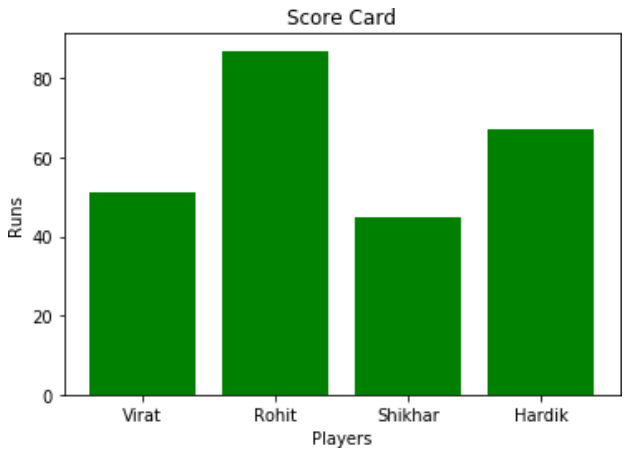


2. Bar graphs:

- Bar graphs are one of the most common types of graphs and are used to show data associated with the categorical variables.
- Matplotlib provides a bar() to make bar graphs which accepts arguments such as: categorical variables, their value and color.

```
from matplotlib import pyplot as plt
players = ['Virat','Rohit','Shikhar','Hardik']
runs = [51,87,45,67]
plt.bar(players,runs,color = 'green')
plt.title('Score Card')
plt.xlabel('Players')
plt.ylabel('Runs')
plt.show()
```

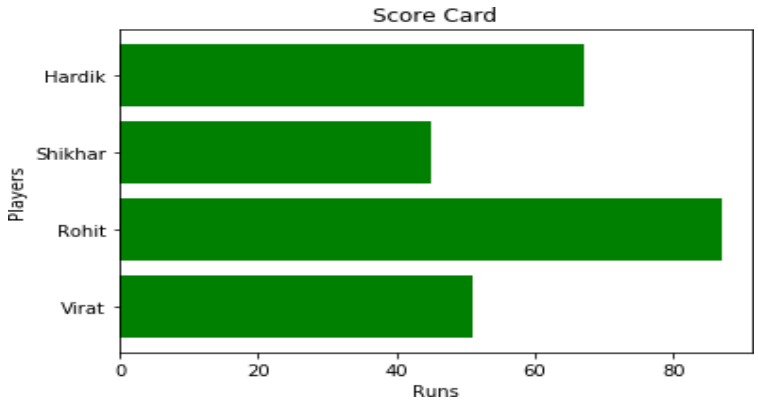
Output:



- Another function barh() is used to make horizontal bar graphs.
- It accepts xerr or yerr as arguments (in case of vertical graphs) to depict the variance in our data as follows:

```
from matplotlib import pyplot as plt
players = ['Virat','Rohit','Shikhar','Hardik']
runs = [51,87,45,67]
plt.barh(players,runs, color = 'green')
plt.title('Score Card')
plt.xlabel('Players')
plt.ylabel('Runs')
plt.show()
```

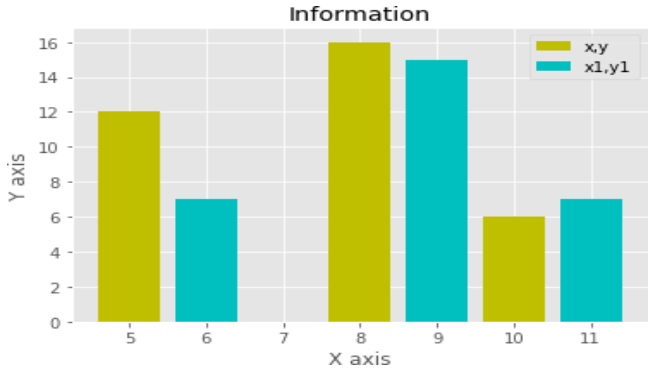
Output:



- Let's have a look on the other example using the style() function:

```
from matplotlib import pyplot as plt
from matplotlib import style
style.use('ggplot')
x = [5,8,10]
y = [12,16,6]
x2 = [6,9,11]
y2 = [7,15,7]
plt.bar(x, y, color = 'y', align='center')
plt.bar(x2, y2, color='c', align='center')
plt.title('Information')
plt.ylabel('Y axis')
plt.xlabel('X axis')
```

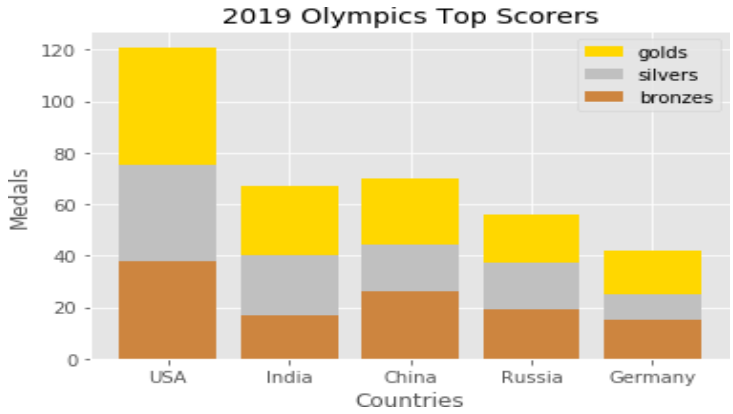
Output:



- Similarly to vertical stack, the bar graph together by using the bottom argument and define the bar graph, which we want to stack below and its value.

```
from matplotlib import pyplot as plt
import numpy as np
countries = ['USA', 'India', 'China', 'Russia', 'Germany']
bronzes = np.array([38, 17, 26, 19, 15])
silvers = np.array([37, 23, 18, 18, 10])
golds = np.array([46, 27, 26, 19, 17])
ind = [x for x, _ in enumerate(countries)]
plt.bar(ind, golds, width=0.5, label='golds', color='gold', bottom=silvers+bronzes)
plt.bar(ind, silvers, width=0.5, label='silvers', color='silver', bottom=bronzes)
plt.bar(ind, bronzes, width=0.5, label='bronzes', color='#CD853F')
plt.xticks(ind, countries)
plt.ylabel("Medals")
plt.xlabel("Countries")
plt.legend(loc="upper right")
plt.title("2019 Olympics Top Scorers")
```

Output:

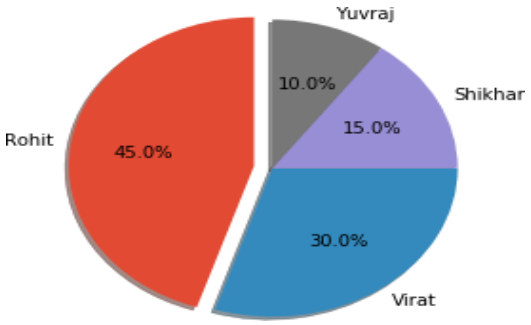


3. Pie Chart:

- A pie chart is a circular graph that is broken down in the segment or slices of pie.
- It is generally used to represent the percentage or proportional data where each slice of pie represents a particular category.
- Let's have a look at the below example:

```
from matplotlib import pyplot as plt
# Pie chart, where the slices will be ordered and plotted counter-clockwise:
Players = 'Rohit', 'Virat', 'Shikhar', 'Yuvraj'
Runs = [45, 30, 15, 10]
explode = (0.1, 0, 0, 0) # it "explode" the 1st slice
fig1, ax1 = plt.subplots()
ax1.pie(Runs, explode=explode, labels=Players, autopct='%1.1f%%', shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```

Output:

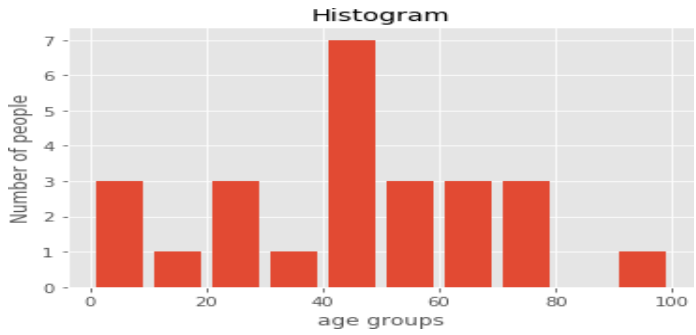


4. Histogram:

- First, we need to understand the difference between the bar graph and histogram.
- A histogram is used for the distribution, whereas a bar chart is used to compare different entities.
- A histogram is a type of bar plot that shows the frequency of a number of values compared to a set of values ranges.
- For example we take the data of the different age group of the people and plot a histogram with respect to the bin. Now, bin represents the range of values that are divided into series of intervals. Bins are generally created of the same size.

```
from matplotlib import pyplot as plt
from matplotlib import pyplot as plt
population_age = [21,53,60,49,25,27,30,42,40,1,2,102,95,8,15,105,70,65,55,70,75,60,52,
44,43,42,45]
bins = [0,10,20,30,40,50,60,70,80,90,100]
plt.hist(population_age, bins, histtype='bar', rwidth=0.8)
plt.xlabel('age groups')
plt.ylabel('Number of people')
plt.title('Histogram')
plt.show()
```

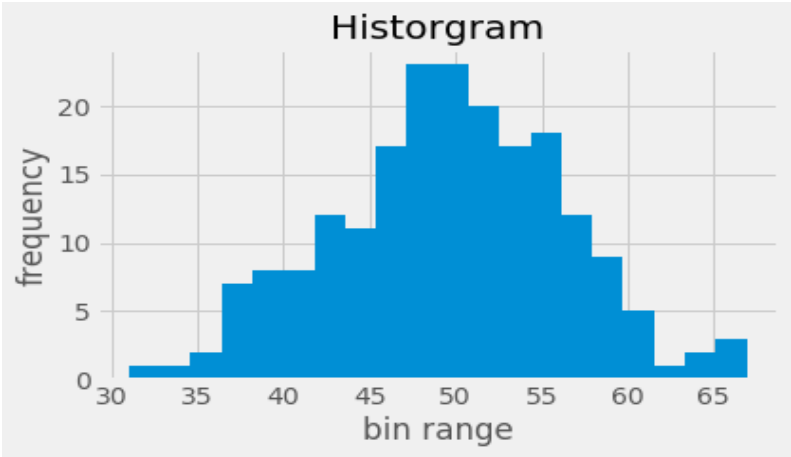
Output:



- Let's consider the another example of plotting histogram:

```
from matplotlib import pyplot as plt
# Importing Numpy Library
import numpy as np
plt.style.use('fivethirtyeight')
mu = 50
sigma = 7
x = np.random.normal(mu, sigma, size=200)
fig, ax = plt.subplots()
ax.hist(x, 20)
ax.set_title('Historgram')
ax.set_xlabel('bin range')
ax.set_ylabel('frequency')
fig.tight_layout()
plt.show()
```

Output:



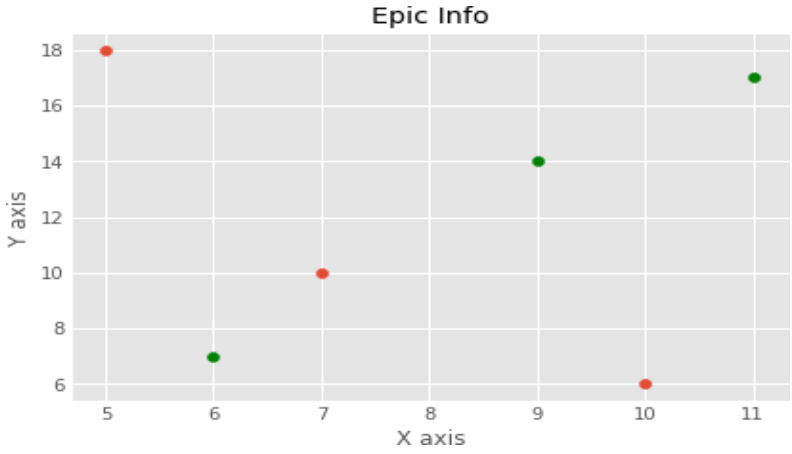
5. Scatter plot:

- The scatter plots are mostly used for comparing variables when we need to define how much one variable is affected by another variable.
- The data is displayed as a collection of points.
- Each point has the value of one variable, which defines the position on the horizontal axes, and the value of other variable represents the position on the vertical axis.

Example-1:

```
from matplotlib import pyplot as plt
from matplotlib import style
style.use('ggplot')
x = [5,7,10]
y = [18,10,6]
x2 = [6,9,11]
y2 = [7,14,17]
plt.scatter(x, y)
plt.scatter(x2, y2, color='g')
plt.title('Epic Info')
plt.ylabel('Y axis')
plt.xlabel('X axis')
plt.show()
```

Output:

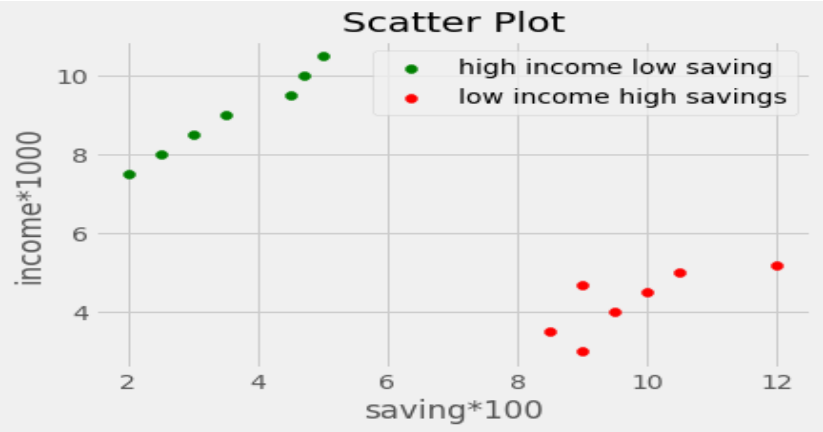


Example-2:

```
import matplotlib.pyplot as plt
x = [2, 2.5, 3, 3.5, 4.5, 4.7, 5.0]
y = [7.5, 8, 8.5, 9, 9.5, 10, 10.5]
x1 = [9, 8.5, 9, 9.5, 10, 10.5, 12]
y1 = [3, 3.5, 4.7, 4, 4.5, 5, 5.2]
plt.scatter(x, y, label='high income low saving', color='g')
plt.scatter(x1, y1, label='low income high savings', color='r')
plt.xlabel('saving*100')
plt.ylabel('income*1000')
plt.title('Scatter Plot')
```

```
plt.legend()
plt.show()
```

Output:



Thread:

- Running several threads is similar to running several different programs concurrently, but with the following benefits –
- Multiple threads within a process share the same data space with the main thread and can therefore share information or communicate with each other more easily than if they were separate processes.
- Threads sometimes called light-weight processes and they do not require much memory overhead; they are cheaper than processes.
- A thread has a beginning, an execution sequence, and a conclusion. It has an instruction pointer that keeps track of where within its context it is currently running.
- It can be pre-empted (interrupted)
- It can temporarily be put on hold (also known as sleeping) while other threads are running - this is called yielding.

Starting a thread:

- To spawn another thread, you need to call following method available in thread module – **thread.start_new_thread (function, args[, kwargs])**
- This method call enables a fast and efficient way to create new threads in both Linux and Windows.
- The method call returns immediately and the child thread starts and calls function with the passed list of args.
- When function returns, the thread terminates.
- Here, args is a tuple of arguments; use an empty tuple to call function without passing any arguments. kwargs is an optional dictionary of keyword arguments.

Example:

```
#!/usr/bin/python

import thread
import time

# Define a function for the thread
def print_time( threadName, delay):
    count = 0
    while count < 5:
        time.sleep(delay)
        count += 1
        print "%s: %s" % ( threadName, time.ctime(time.time()) )

# Create two threads as follows try:
thread.start_new_thread( print_time, ("Thread-1", 2, ) )
thread.start_new_thread( print_time, ("Thread-2", 4, ) )
except:
    print "Error: unable to start thread"

while 1:
pass
```

Output:

```
Thread-1: Thu Jan 22 15:42:17 2009
Thread-1: Thu Jan 22 15:42:19 2009
Thread-2: Thu Jan 22 15:42:19 2009
Thread-1: Thu Jan 22 15:42:21 2009
Thread-2: Thu Jan 22 15:42:23 2009
Thread-1: Thu Jan 22 15:42:23 2009
Thread-1: Thu Jan 22 15:42:25 2009
Thread-2: Thu Jan 22 15:42:27 2009
Thread-2: Thu Jan 22 15:42:31 2009
Thread-2: Thu Jan 22 15:42:35 2009
```

- Although it is very effective for low-level threading, but the thread module is very limited compared to the newer threading module.

Threading module:

- The newer threading module included with Python 2.4 provides much more powerful, high-level support for threads than the thread module discussed in the previous section.
- The threading module exposes all the methods of the thread module and provides some additional methods –

➢ **threading.activeCount()** – Returns the number of thread objects that are active.
➢ **threading.currentThread()** – Returns the number of thread objects in the caller's thread

control.

- **threading.enumerate()** – Returns a list of all thread objects that are currently active.
- In addition to the methods, the threading module has the Thread class that implements threading. The methods provided by the Thread class are as follows –
- **run()** – The run() method is the entry point for a thread.
- **start()** – The start() method starts a thread by calling the run method.
- **join([time])** – The join() waits for threads to terminate.
- **isAlive()** – The isAlive() method checks whether a thread is still executing.
- **getName()** – The getName() method returns the name of a thread.
- **setName()** – The setName() method sets the name of a thread.

Creating Thread Using Threading Module:

- To implement a new thread using the threading module, you have to do the following –
- Define a new subclass of the Thread class.
- Override the init (self [,args]) method to add additional arguments.
- Then, override the run(self [,args]) method to implement what the thread should do when started.
- Once you have created the new Thread subclass, you can create an instance of it and then start a new thread by invoking the start(), which in turn calls run() method.

Example:

```
#!/usr/bin/python

import threading
import time

exitFlag = 0

class myThread (threading.Thread):
    def __init (self, threadID, name, counter):
            threading.Thread. init__(self)
            self.threadID = threadID
            self.name = name
            self.counter = counter
    def run(self):
            print "Starting " + self.name
            print_time(self.name, 5, self.counter)
            print "Exiting " + self.name

    def print_time(threadName, counter, delay):
            while counter:
                    if exitFlag:
                            threadName.exit()
                     time.sleep(delay)
                    print "%s: %s" % (threadName, time.ctime(time.time()))
```

```
                counter -= 1

    # Create new threads
    thread1 = myThread(1, "Thread-1", 1)
    thread2 = myThread(2, "Thread-2", 2)

    # Start new Threads
    thread1.start()
    thread2.start()

    print "Exiting Main Thread"
```

Output:

```
Starting Thread-1
Starting Thread-2
Exiting Main Thread
Thread-1: Thu Mar 21 09:10:03 2013
Thread-1: Thu Mar 21 09:10:04 2013
Thread-2: Thu Mar 21 09:10:04 2013
Thread-1: Thu Mar 21 09:10:05 2013
Thread-1: Thu Mar 21 09:10:06 2013
Thread-2: Thu Mar 21 09:10:06 2013
Thread-1: Thu Mar 21 09:10:07 2013
Exiting Thread-1
Thread-2: Thu Mar 21 09:10:08 2013
Thread-2: Thu Mar 21 09:10:10 2013
Thread-2: Thu Mar 21 09:10:12 2013
Exiting Thread-2
```

Synchronizing threads:

- The threading module provided with Python includes a simple-to-implement locking mechanism that allows you to synchronize threads.
- A new lock is created by calling the Lock() method, which returns the new lock.
- The acquire(blocking) method of the new lock object is used to force threads to run synchronously.
- The optional blocking parameter enables you to control whether the thread waits to acquire the lock.
- If blocking is set to 0, the thread returns immediately with a 0 value if the lock cannot be acquired and with a 1 if the lock was acquired.
- If blocking is set to 1, the threads blocks and wait for the lock to be released.
- The release () method of the new lock object is used to release the lock when it is no longer required.

Example:

```
#!/usr/bin/python

import threading
import time

class myThread (threading.Thread):
    def __init (self, threadID, name, counter):
        threading.Thread. init__(self)
        self.threadID = threadID
        self.name = name
        self.counter = counter
    def run(self):
        print "Starting " + self.name
        # Get lock to synchronize threads
        threadLock.acquire()
        print_time(self.name, self.counter, 3)
        # Free lock to release next thread
        threadLock.release()

    def print_time(threadName, delay, counter):
        while counter:
            time.sleep(delay)
            print "%s: %s" % (threadName, time.ctime(time.time()))
            counter -= 1

        threadLock = threading.Lock()
        threads = []

        # Create new threads
        thread1 = myThread(1, "Thread-1", 1)
        thread2 = myThread(2, "Thread-2", 2)

        # Start new Threads
        thread1.start()
        thread2.start()

        # Add threads to thread list
        threads.append(thread1)
        threads.append(thread2)

        # Wait for all threads to complete
        for t in threads:
            t.join()
        print "Exiting Main Thread"
```

Output:

```
Starting Thread-1
Starting Thread-2
Thread-1: Thu Mar 21 09:11:28 2013
Thread-1: Thu Mar 21 09:11:29 2013
Thread-1: Thu Mar 21 09:11:30 2013
Thread-2: Thu Mar 21 09:11:32 2013
Thread-2: Thu Mar 21 09:11:34 2013
Thread-2: Thu Mar 21 09:11:36 2013
Exiting Main Thread
```

Multithreaded Priority:

- The Queue module allows you to create a new queue object that can hold a specific number of items.
- There are following methods to control the Queue –

- **get()** – The get() removes and returns an item from the queue.
- **put()** – The put adds item to a queue.
- **qsize()** – The qsize() returns the number of items that are currently in the queue.
- **empty()** – The empty( ) returns True if queue is empty; otherwise, False.
- **full()** – the full() returns True if queue is full; otherwise, False.

Example:

```
#!/usr/bin/python

import Queue
import threading
import time

exitFlag = 0

class myThread (threading.Thread):
    def __init (self, threadID, name, q):
            threading.Thread. init__(self)
            self.threadID = threadID
            self.name = name
            self.q = q
    def run(self):
            print "Starting " + self.name
            process_data(self.name, self.q)
            print "Exiting " + self.name

    def process_data(threadName, q):
            while not exitFlag: queueLock.acquire()
                    if not workQueue.empty():
                            data = q.get()
```

```
                        queueLock.release()
                        print "%s processing %s" % (threadName, data)
                    else:
                        queueLock.release()
                        time.sleep(1)

threadList = ["Thread-1", "Thread-2", "Thread-3"]
nameList = ["One", "Two", "Three", "Four", "Five"]
queueLock = threading.Lock()
workQueue = Queue.Queue(10)
threads = []
threadID = 1

# Create new threads
for tName in threadList:
    thread = myThread(threadID, tName, workQueue)
    thread.start()
    threads.append(thread)
    threadID += 1

# Fill the queue
queueLock.acquire()
for word in nameList:
    workQueue.put(word)
queueLock.release()

# Wait for queue to empty
while not workQueue.empty():
    pass

# Notify threads it's time to exit
exitFlag = 1

# Wait for all threads to complete
for t in threads:
    t.join()
print "Exiting Main Thread"
```

Output:

```
Starting Thread-1
Starting Thread-2
Starting Thread-3
Thread-1 processing One
Thread-2 processing Two
Thread-3 processing Three
Thread-1 processing Four
Thread-2 processing Five
```

```
Exiting Thread-3
Exiting Thread-1
Exiting Thread-2
Exiting Main Thread
```