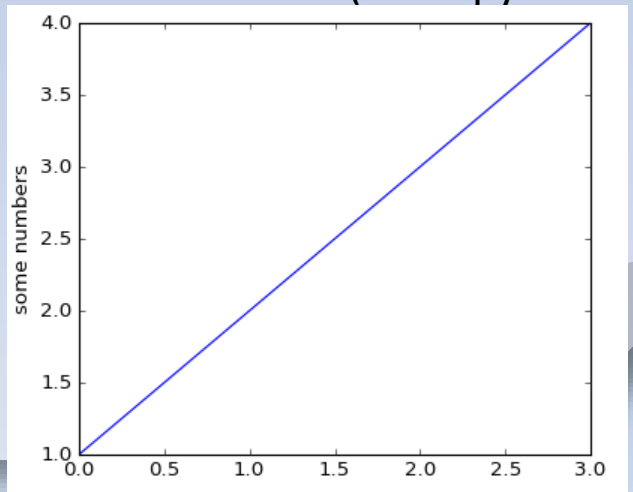
matplotib basics

- matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4])
plt.ylabel('some numbers')
plt.show()
```

- If you provide a single list or array to the plot() command, matplotlib assumes it is a sequence of y values
- It automatically generates the x values for you
  - x vector is made the same length as y vector buts starts at 0 (since python ranges start with 0)
- To plot x vs y

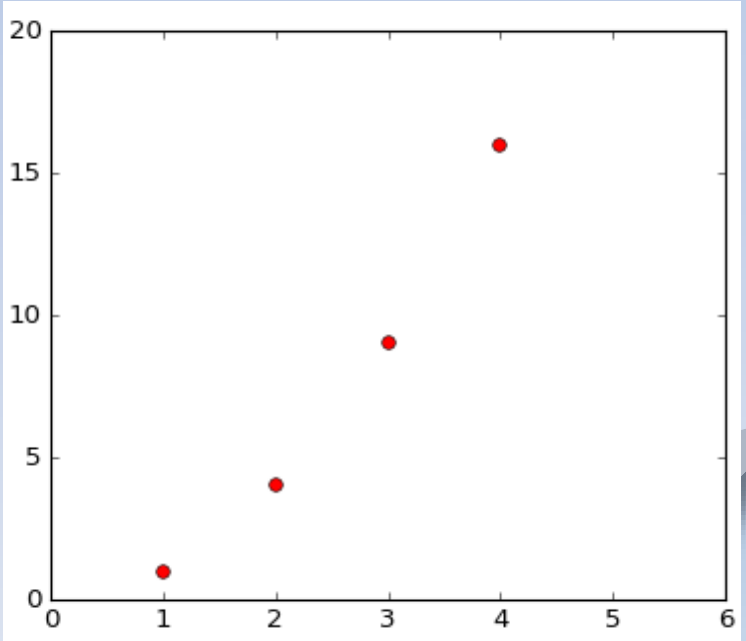
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])



matplotlib basics

- For every x, y pair of arguments, there is an optional third argument which is the format string
  - indicates the color and line type of the plot
  - you concatenate a color string with a line style string
  - default format string is ‘b-‘, which is a solid blue line
  - For example, to plot the above with red circles, you would issue ‘ro’
  - The axis() command here takes a list of [xmin, xmax, ymin, ymax] and specifies the viewport of the axes.

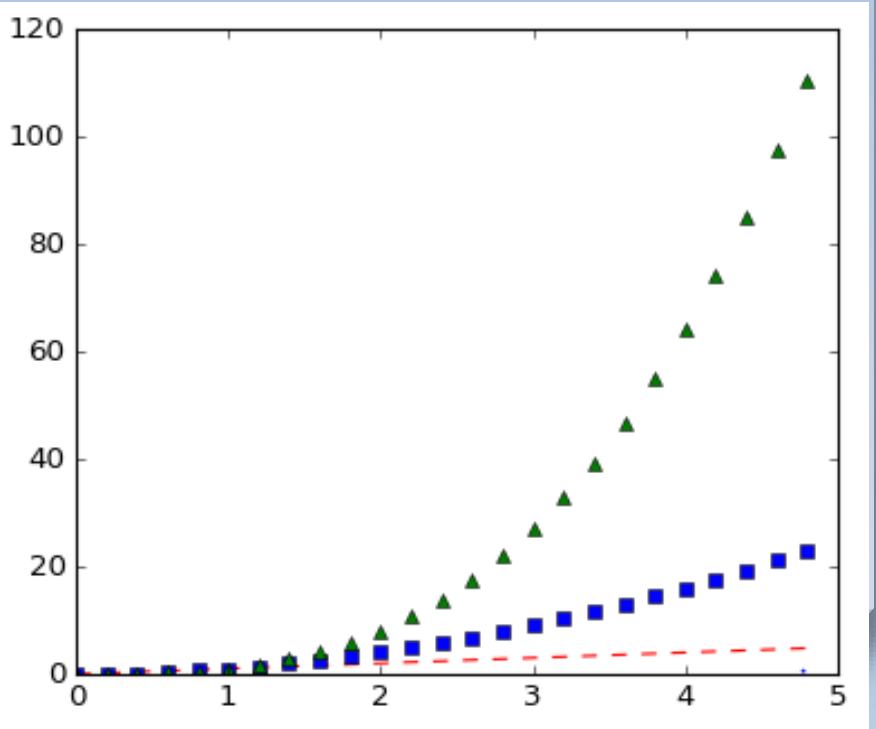
```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4], [1,4,9,16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```



matplotlib basics

- Use numPy arrays and you can specify multiple plot lines with a single plot() command

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

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```



matplotlib lines

- Lines have many attributes that you can set: linewidth, dash style, antialiased, etc;
  - Use keyword args:

    plt.plot(x, y, linewidth=2.0)
  - Use the setter methods of a Line2D instance

    plot returns a list of Line2D objects; e.g., line1, line2 = plot(x1, y1, x2, y2).
    If we have only one line the list returned is of length 1
    We use tuple unpacking with line, to get the first element of that list:

    line, = plt.plot(x, y, '-')
    line.set_antialiased(False) # turn off antialising
  - Use the setp() command

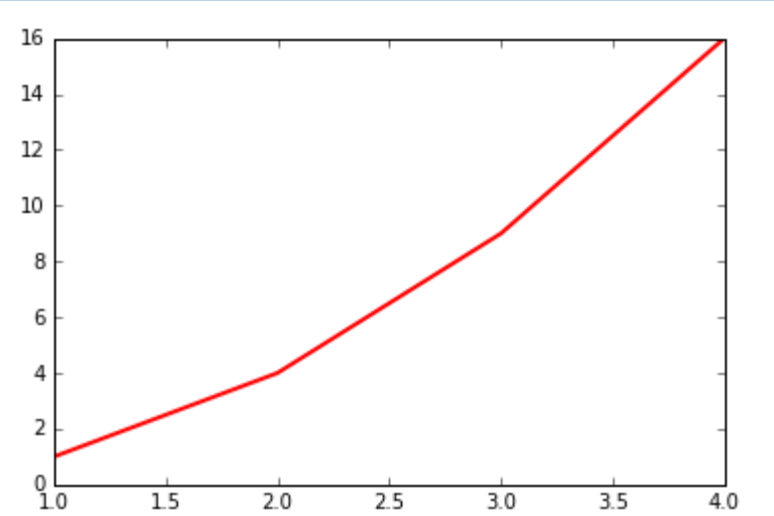
    The example below uses a MATLAB-style command to set multiple properties on a list of lines

    lines = plt.plot(x1, y1, x2, y2)
    # use keyword args
    plt.setp(lines, color='r', linewidth=2.0)
    # or MATLAB style string value pairs
    plt.setp(lines, 'color', 'r', 'linewidth', 2.0)

matplotlib lines

- Use lines

```
def line_chart_with_lineobject():
    line, = plt.plot([1,2,3,4], [1,4,9,16])
    #plot returns a tuple of line2D objects
    #use setp to change characteristics of the line object
    plt.setp(line, color='r', linewidth=2.0)
    #display the plot on the window

    plt.show()

    print("x points:", line.get_xdata(), "y points:", line.get_ydata())
    return()
```



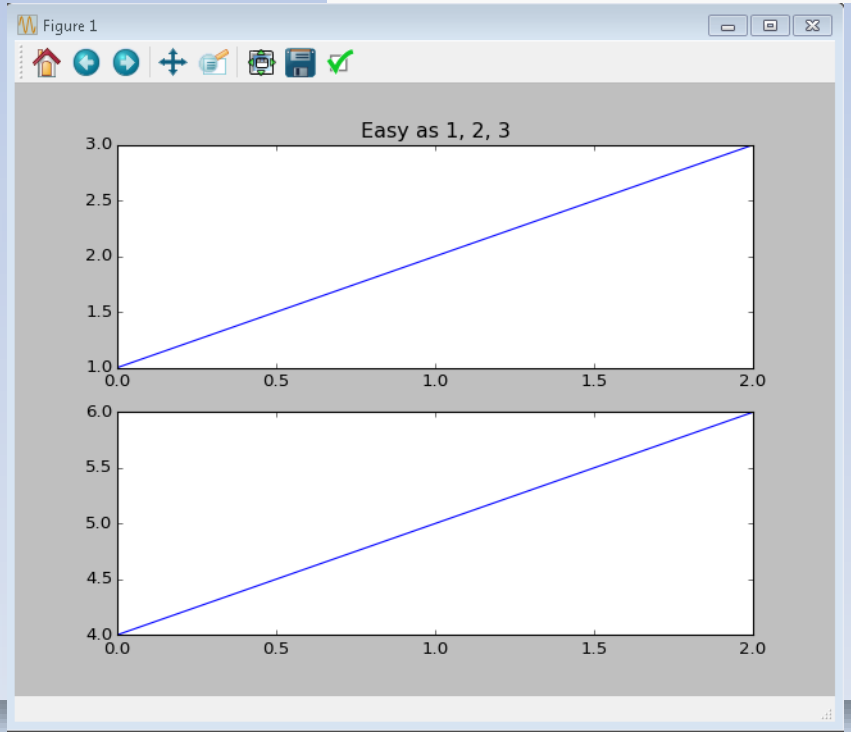
Figures and Axes

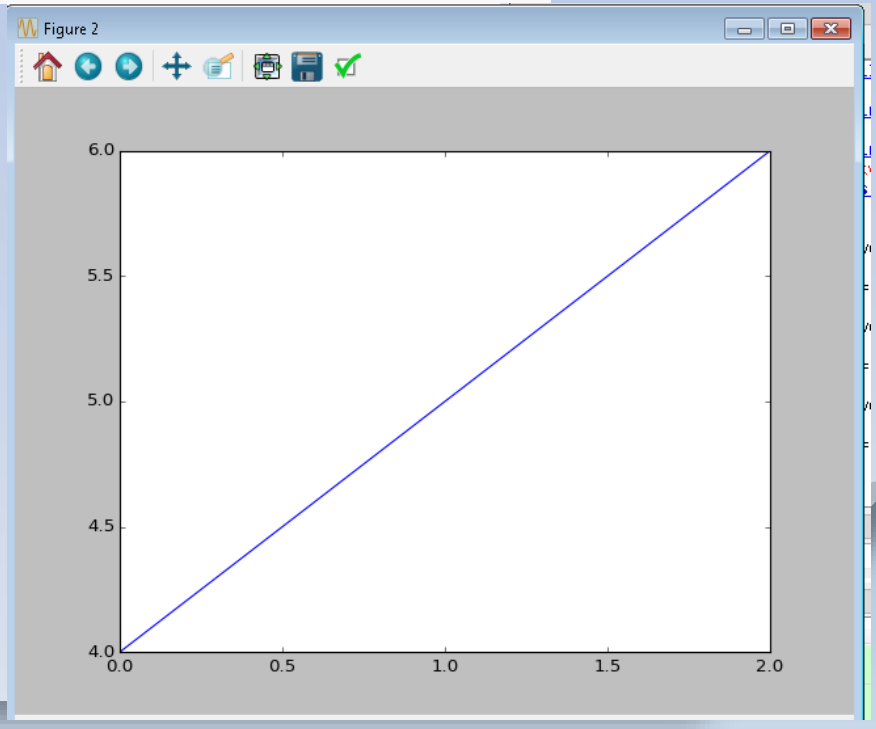
- pyplot much like MATLAB has the concept of the current figure and the current axes
- By default all plots are drawn to a default (or current) figure and axes
- If you call plot successively without referring to a new figure or axes, the plot will be drawn on the current figure & axes
- If you want to create separate multiple plots within one figure (window) use subplot
- The subplot() specifies numrows, numcols, fignum where fignum ranges from 1 to numrows*numcols *(fignum indicates which subplot is being refered to)*

  *The commas in the subplot command are optional if numrows*numcols<10. So subplot(211) is identical to subplot(2, 1, 1)*

Figures and Axes

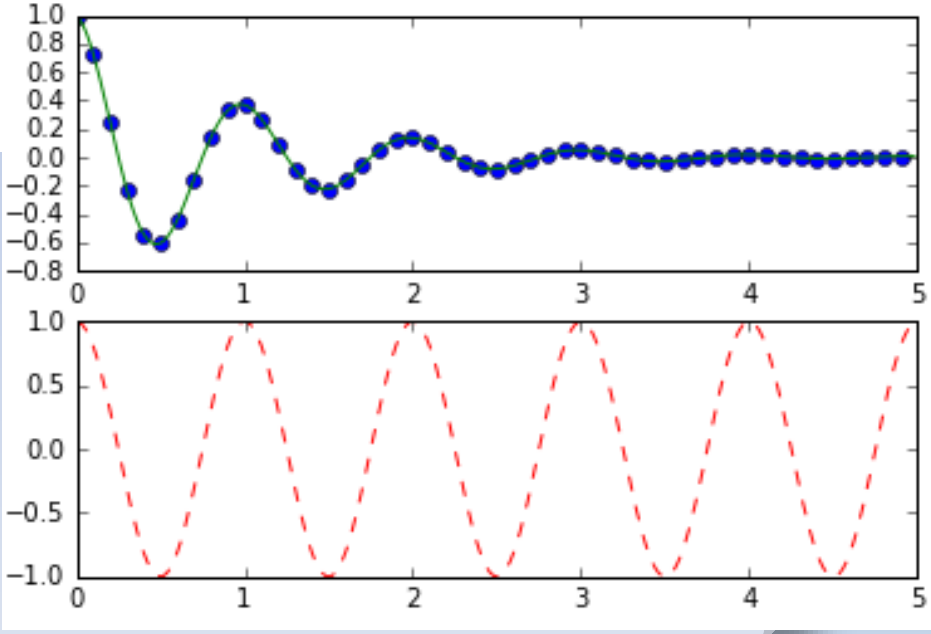
```
import matplotlib.pyplot as plt
plt.figure(1)                # the first figure
plt.subplot(211)             # the first subplot in the first figure
plt.plot([1, 2, 3])
plt.subplot(212)             # the second subplot in the first figure
plt.plot([4, 5, 6])


plt.figure(2)                # a second figure
plt.plot([4, 5, 6])          # creates a subplot(111) by default

plt.figure(1)                # figure 1 current; subplot(212) still current
plt.subplot(211)             # make subplot(211) in figure1 current
plt.title('Easy as 1, 2, 3') # subplot 211 title
```





Figures and Axes

```
def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

def figures_and_axes_2():
    t1 = np.arange(0.0, 5.0, 0.1)
    t2 = np.arange(0.0, 5.0, 0.02)

    plt.figure(1)
    plt.subplot(211)
    plt.plot(t1, f(t1), 'bo', t2, f(t2), 'g')

    plt.subplot(212)
    plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
    plt.show()
    return()

figures_and_axes_2()
```



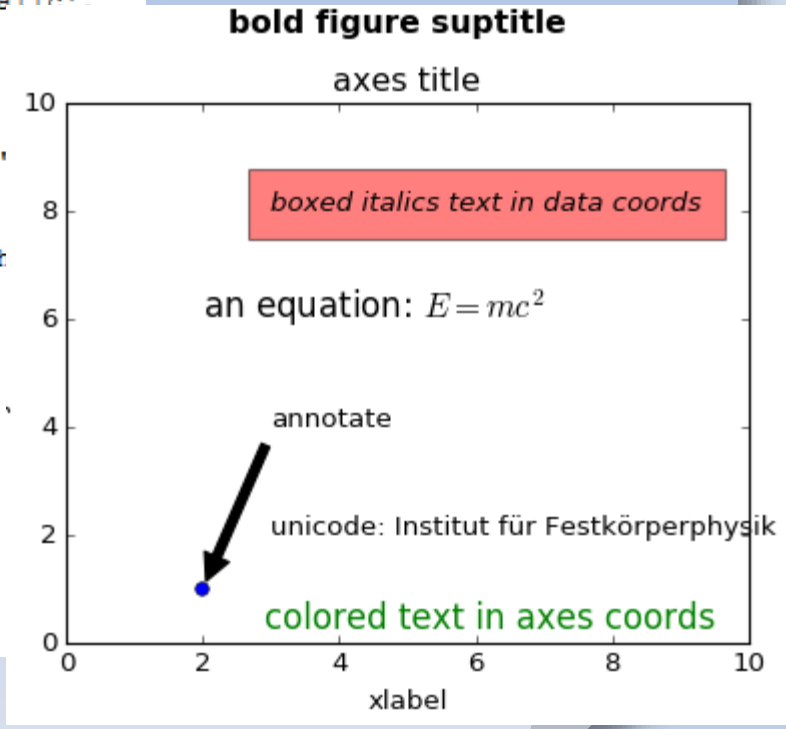
Text

- The text() command can be used to add text in an arbitrary location, and the xlabel(), ylabel() and title() are used to add text in the indicated locations

- `text()` - add text at an arbitrary location to the `Axes`; `matplotlib.axes.Axes.text()` in the API.
- `xlabel()` - add an axis label to the x-axis; `matplotlib.axes.Axes.set_xlabel()` in the API.
- `ylabel()` - add an axis label to the y-axis; `matplotlib.axes.Axes.set_ylabel()` in the API.
- `title()` - add a title to the `Axes`; `matplotlib.axes.Axes.set_title()` in the API.
- `figtext()` - add text at an arbitrary location to the `Figure`; `matplotlib.figure.Figure.text()` in the API.
- `suptitle()` - add a title to the `Figure`; `matplotlib.figure.Figure.suptitle()` in the API.
- `annotate()` - add an annotation, with optional arrow, to the `Axes` ; `matplotlib.axes.Axes.annotate()` in the API.

Text

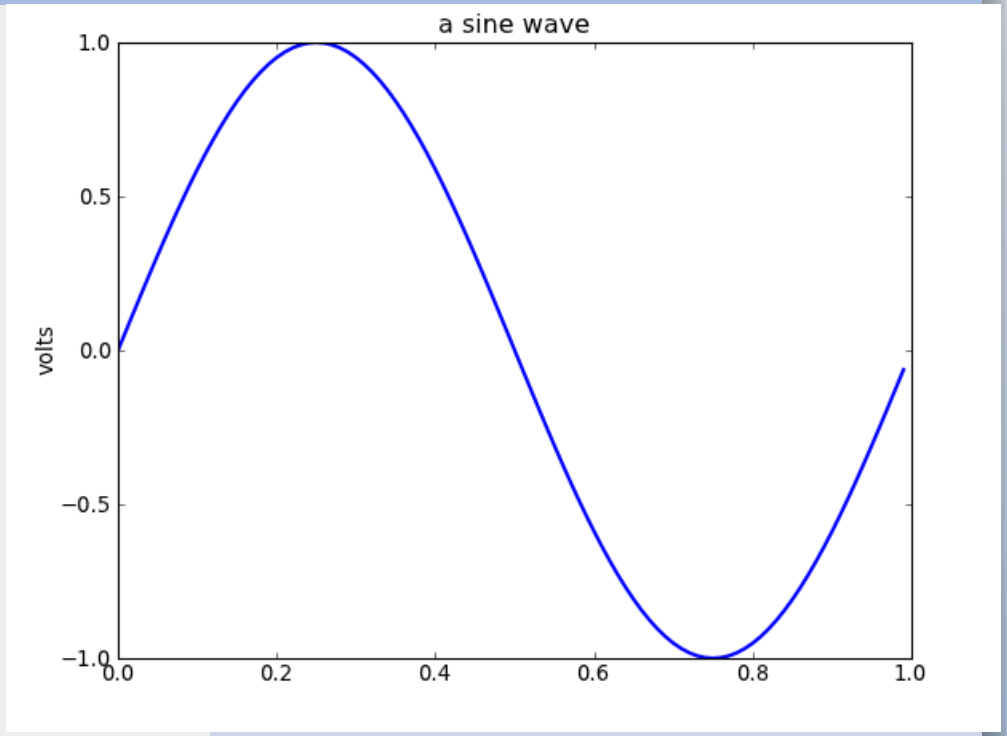
```
import matplotlib.pyplot as plt

fig = plt.figure()
fig.suptitle('bold figure suptitle', fontsize=14, fontweight='bold')

ax = fig.add_subplot(111)
fig.subplots_adjust(top=0.85)
ax.set_title('axes title')

ax.set_xlabel('xlabel')
ax.set_ylabel('ylabel')

ax.text(3, 8, 'boxed italics text in data coords', style='italic'
        bbox={'facecolor':'red', 'alpha':0.5, 'pad':10})

ax.text(2, 6, r'an equation: $E=mc^2$', fontsize=15)

ax.text(3, 2, u'unicode: Institut f\374r Festk\366rperphysik'

ax.text(0.95, 0.01, 'colored text in axes coords',
        verticalalignment='bottom', horizontalalignment='righ
        transform=ax.transAxes,
        color='green', fontsize=15)


ax.plot([2], [1], 'o')
ax.annotate('annotate', xy=(2, 1), xytext=(3, 4),
            arrowprops=dict(facecolor='black', shrink=0.05))

ax.axis([0, 10, 0, 10])

plt.show()
```



http://matplotlib.org/users/mathtext.html

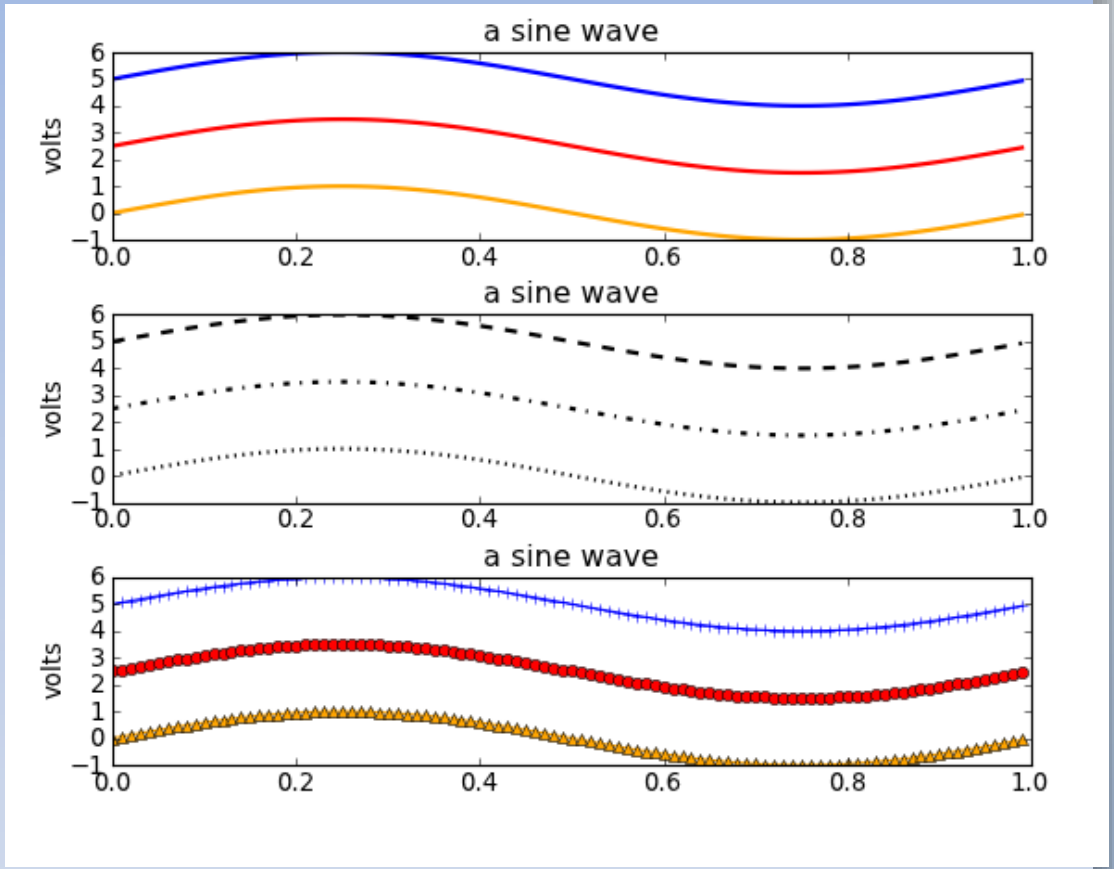
Simple Line Plot

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

## initialize the axes
fig = plt.figure()
ax = fig.add_subplot(111)

## format axes
ax.set_ylabel('volts')
ax.set_title('a sine wave')

t = np.arange(0.0, 1.0, 0.01)
s = np.sin(2*np.pi*t)
line, = ax.plot(t, s, color='blue', lw=2)
```



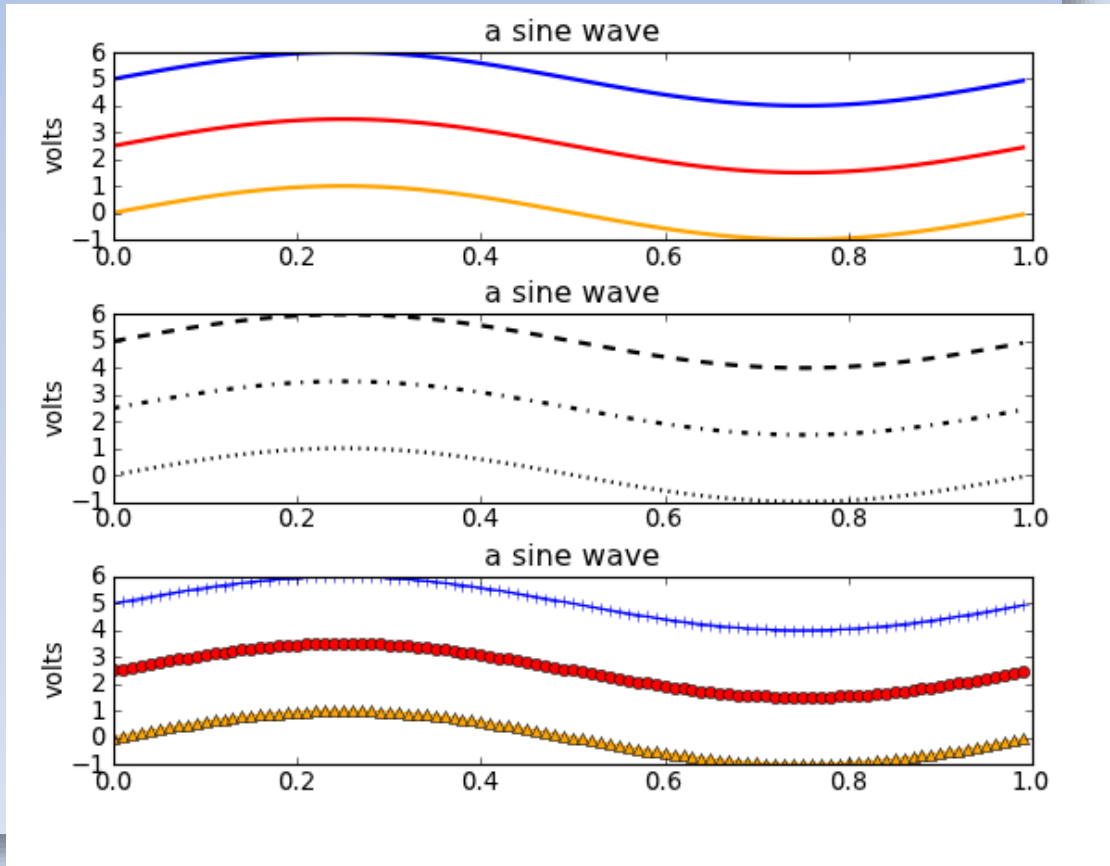
Plots with line markers

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

## initialize the figure
fig = plt.figure()


## the data
t = np.arange(0.0, 1.0, 0.01)
s = np.sin(2*np.pi*t)

## the top axes
ax1 = fig.add_subplot(3,1,1)
ax1.set_ylabel('volts')
ax1.set_title('a sine wave')

line1 = ax1.plot(t, s+5.0, color='blue', lw=2)
line2 = ax1.plot(t, s+2.5, color='red', lw=2)
line3 = ax1.plot(t, s, color='orange', lw=2)

## the middle axes
ax2 = fig.add_subplot(3,1,2)
ax2.set_ylabel('volts')
ax2.set_title('a sine wave')

line1 = ax2.plot(t, s+5.0, color='black', lw=2,linestyle="--")
line2 = ax2.plot(t, s+2.5, color='black', lw=2,linestyle="-.")
line3 = ax2.plot(t, s, color='#000000', lw=2,linestyle=":")
```



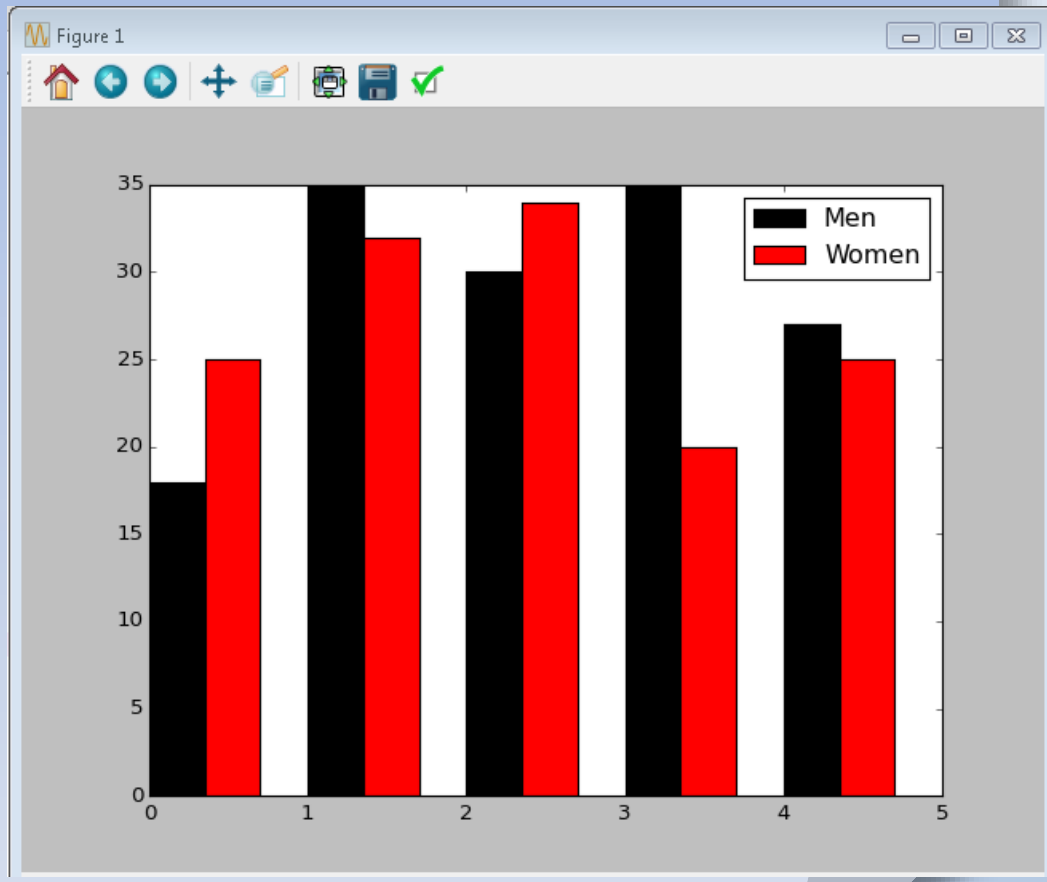
Plots with line markers (contd.)

```
## the third axes
ax3 = fig.add_subplot(3,1,3)
ax3.set_ylabel('volts')
ax3.set_title('a sine wave')

line1 = ax3.plot(t,s+5.0, color='blue', marker="+")
line2 = ax3.plot(t,s+2.5, color='red', marker="o")
line3 = ax3.plot(t,s, color='orange', marker="^")

## adjust the space between plots
plt.subplots_adjust(wspace=0.2,hspace=.4)
```



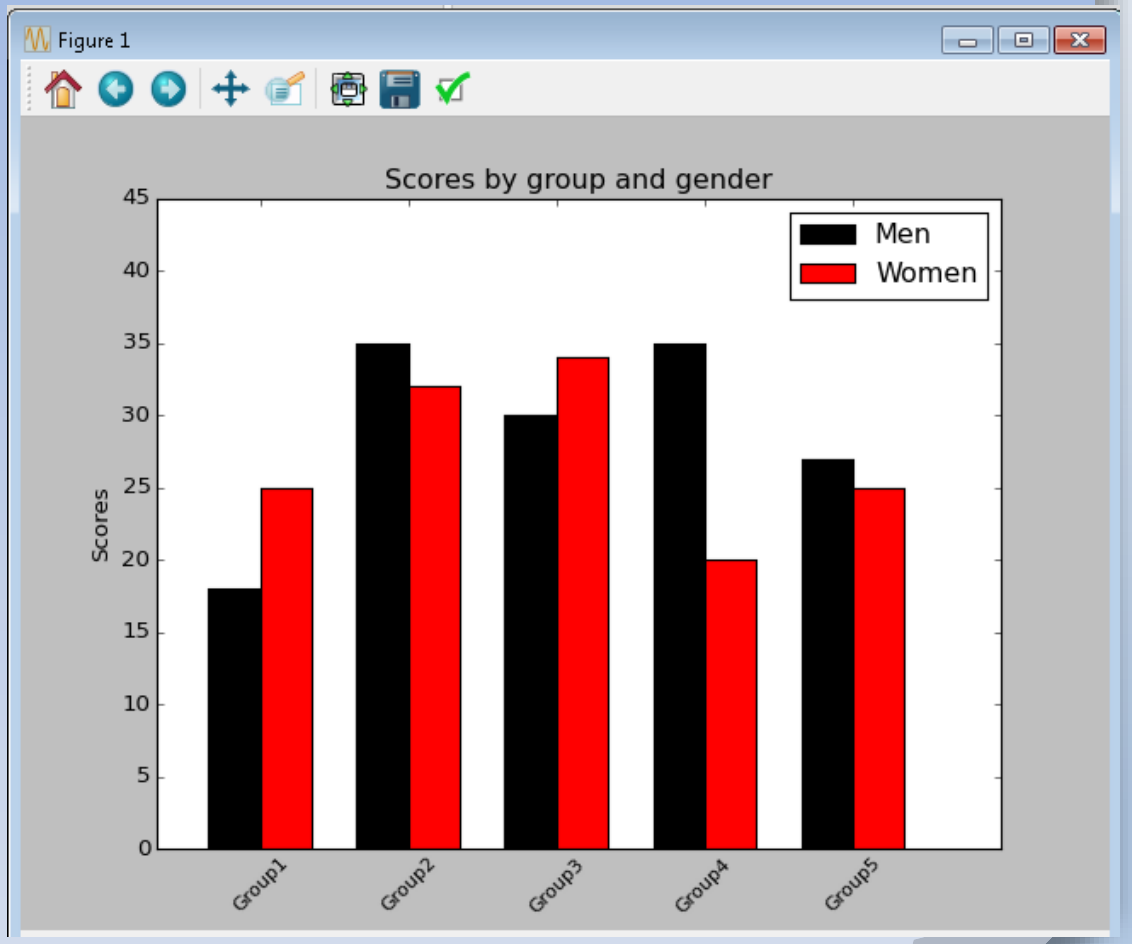
Simple Bar Plot

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

fig = plt.figure()
ax = fig.add_subplot(111)

## the data
N = 5
menMeans = [18, 35, 30, 35, 27]
womenMeans = [25, 32, 34, 20, 25]

## necessary variables
ind = np.arange(N)          # the x locations for the groups
width = 0.35                # the width of the bars

## the bars
rects1 = ax.bar(ind, menMeans, width,
            color='black',)

rects2 = ax.bar(ind+width, womenMeans, width,
                color='red')

## add a legend
ax.legend( (rects1[0], rects2[0]), ('Men', 'Women') )

plt.show()
```



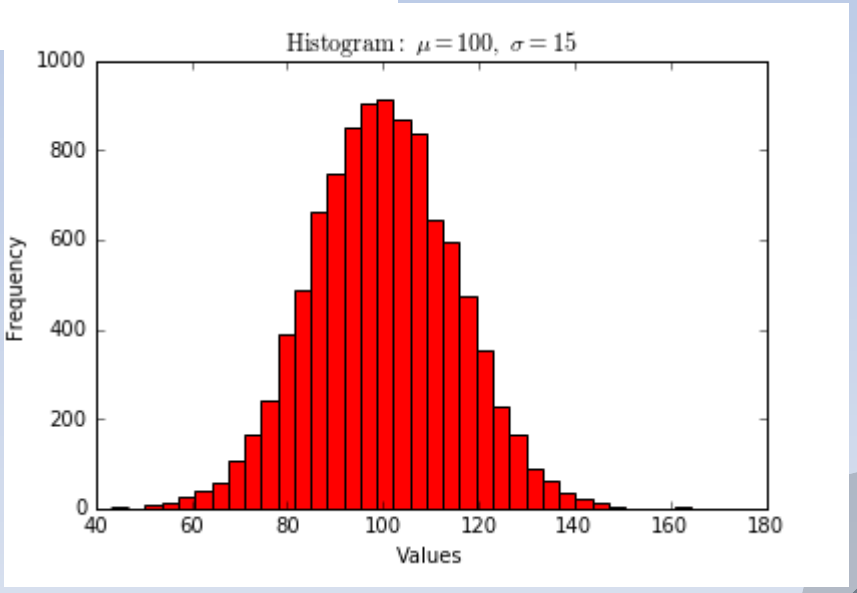
Simple Bar Plot

```
# axes and labels
ax.set_xlim(-width,len(ind)+width)
ax.set_ylim(0,45)
ax.set_ylabel('Scores')
ax.set_title('Scores by group and gender')
xTickMarks = ['Group'+str(i) for i in range(1,6)]
ax.set_xticks(ind+width)
xtickNames = ax.set_xticklabels(xTickMarks)
plt.setp(xtickNames, rotation=45, fontsize=10)

plt.show()
```



Histogram

```
def histogram():
    #typically meant to plot probably density function
    #for a set of values
    mu = 100
    sigma = 15
    x = np.random.normal(mu, sigma, 10000)
    print(len(x))
    fig = plt.figure()
    ax = fig.add_subplot(111)
    # the histogram of the data
    ax.hist(x, bins=35, color='r')
    ax.set_xlabel('Values')
    ax.set_ylabel('Frequency')
    ax.set_title(r'$\mathrm{Histogram:}\ \mu=%d,\ \sigma=%d$' % (mu,sigma))
    plt.show()
    return()
```



Scatter Plot

- A scatter plot is often used to identify potential association between two variables
- Often drawn before working on a fitting regression function
- Gives a good visual picture of the correlation, particularly for nonlinear relationships
- scatter() function is used to plot x versus y

Scatter Plot

```
def scatter_plot():
    # generate x values
    x = np.random.randn(1000)
    # random measurements, no correlation
    y1 = np.random.randn(len(x))

    plt.subplot(121)
    plt.scatter(x, y1, color='indigo', alpha=0.3, edgecolors='white',
    label='no correlation')
    plt.xlabel('no correlation')
    plt.grid(True)
    plt.legend()

    # strong correlation
    y2 = 1.2 + np.exp(x)
    plt.subplot(122)
    plt.scatter(x, y2, color='green', alpha=0.3, edgecolors='grey',
    label='correlation')
    plt.xlabel('strong correlation')
    plt.grid(True)
    plt.legend()
    plt.show()
    return()
```