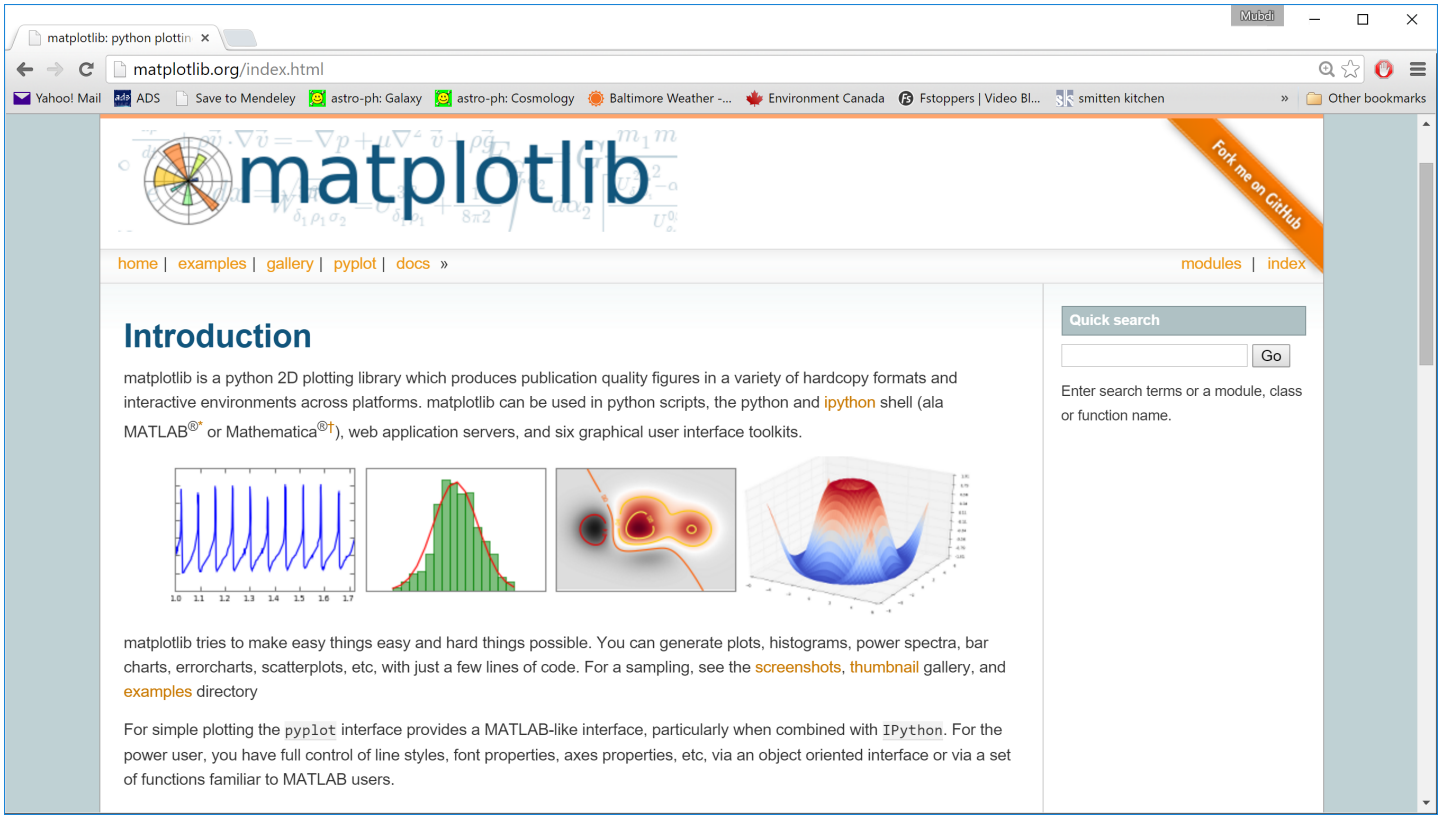
BASIC PLOTTING

Python Crash Course
Part 7

INTRODUCING MATPLOTLIB!



Very powerful plotting package.
The Docs: http://matplotlib.org

GETTING STARTED WITH MATPLOTLIB

Matplotlib has multiple ways of interfacing with it, as well as a large number of additional modules and patches that extend its capabilities significantly. The main interface we'll be using for this work is the **pyplot** (MATLAB-like) interface:

```
import matplotlib.pyplot as plt
```

You can choose to run matplotlib either **interactively** or **non-interactively**. For the interactive mode, the plot gets updated as you go along. For non-interactive, the plot doesn't show up until you've finished everything. To switch between the two:

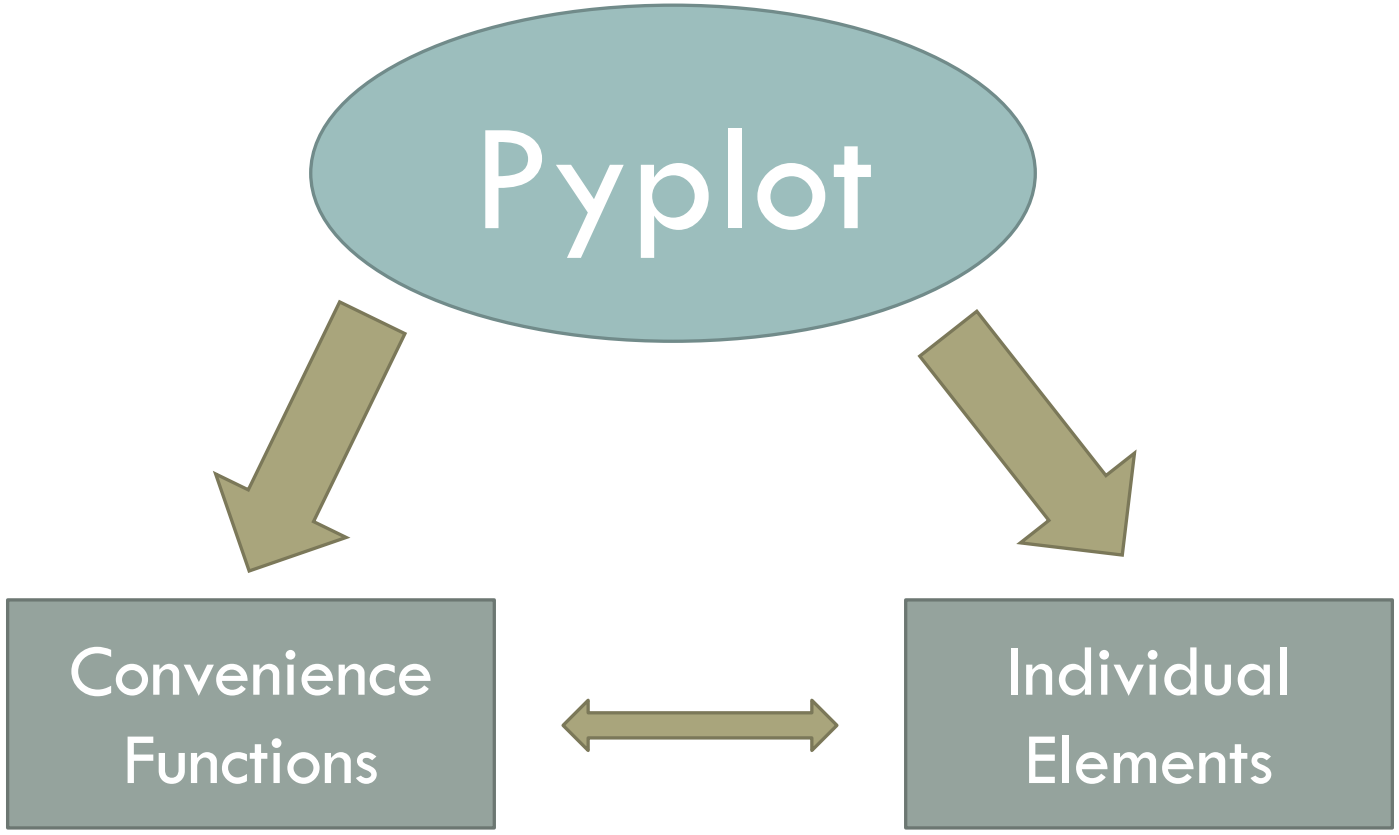
```
plt.ion() # Turn interactive mode on
plt.ioff() # Turn interactive mode off
plt.show() # Show the plot when interactive mode off
```

IMPORTANT NOTE

If you are in a situation where you can't display a plot or don't have the ability (i.e., ssh-ing without Xforwarding, running on a webserver), do the following before importing pyplot:

```
import matplotlib
matplotlib.use('Agg')
```

CHOOSE YOUR OWN ADVENTURE!



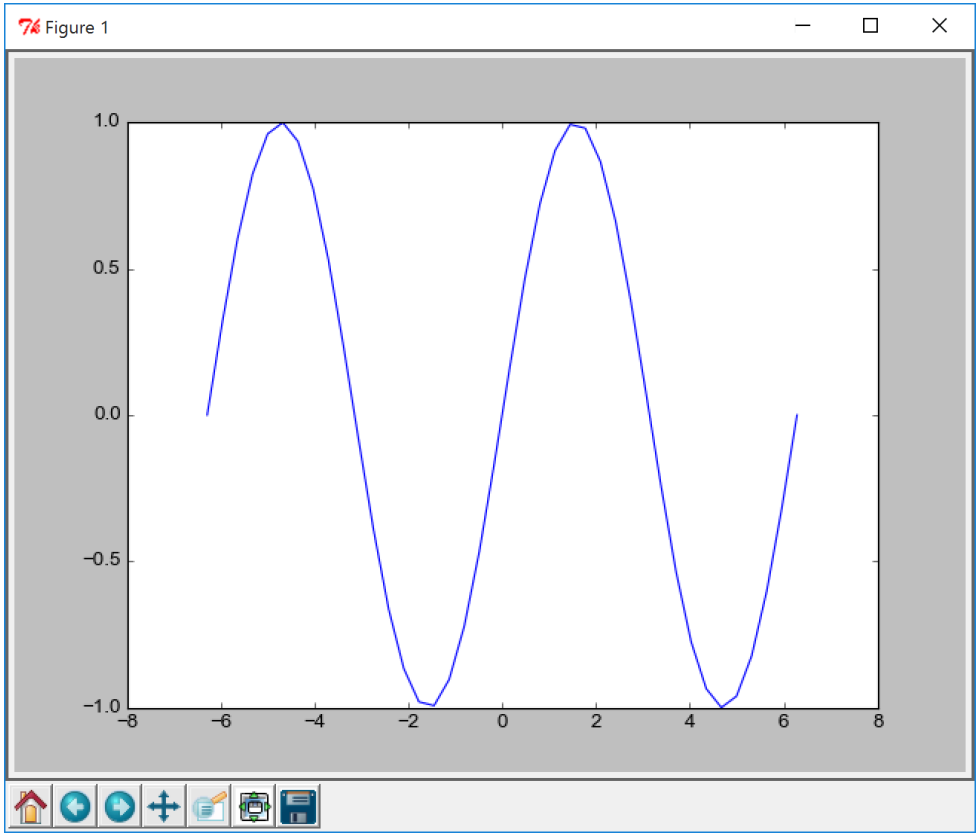
- Really simple to start
- Not as much flexibility

- Requires more coding
- Can plot anything!

SIMPLE PLOTTING BASICS

Much of your power is in the plot command:

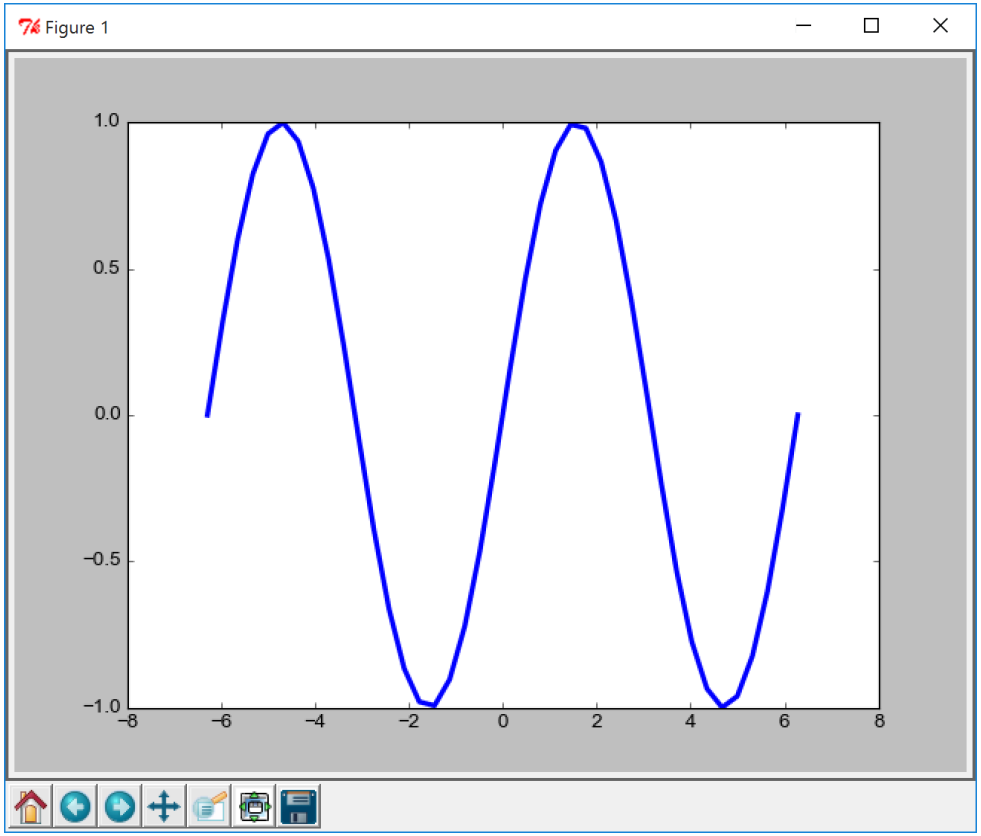
```
# The simplest of plots
import numpy as np
from numpy import pi
x = np.arange(-2*pi,2*pi,pi/8)
y = np.sin(x)
plt.ion()
plt.plot(x, y)
```



SIMPLE PLOTTING BASICS

Change the line width:

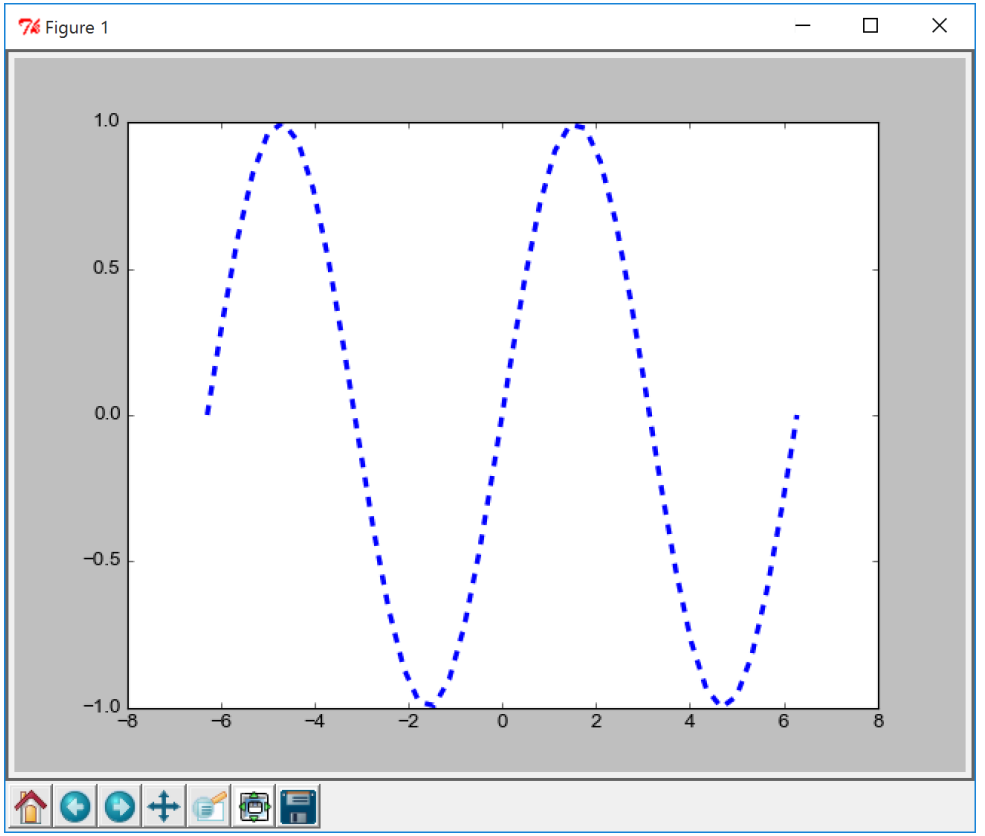
```
plt.plot(x, y, linewidth=3)
```



SIMPLE PLOTTING BASICS

Change the line style:

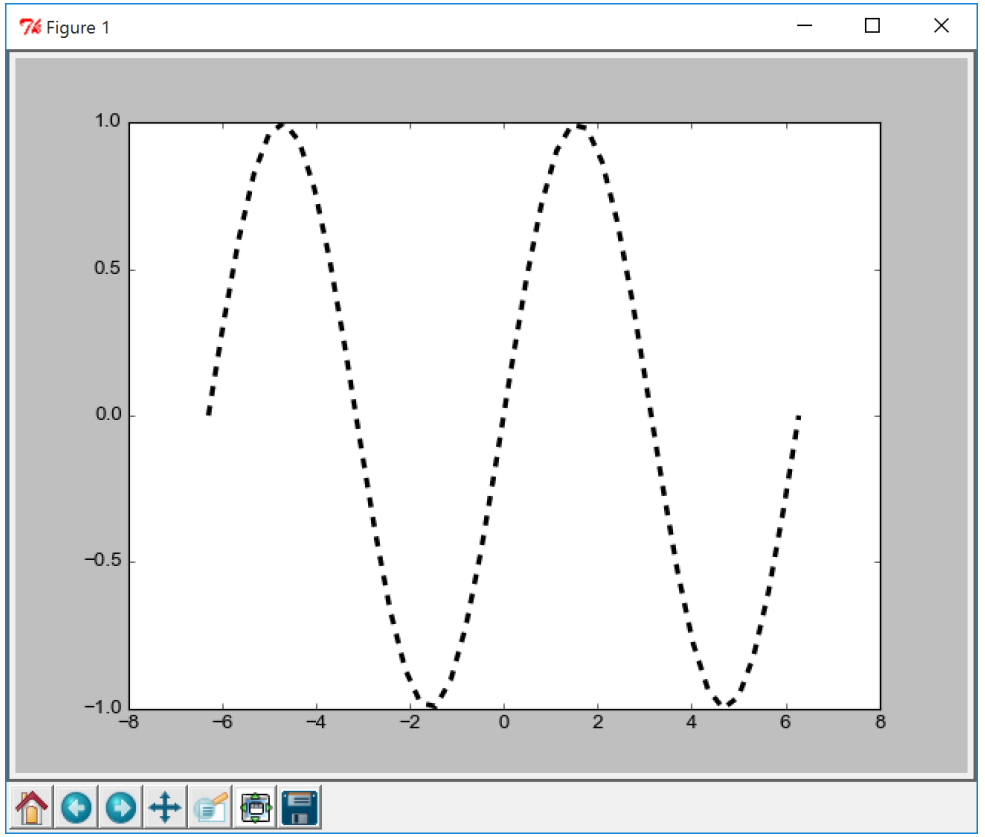
```
plt.plot(x, y,
linewidth=3,
linestyle='dashed')
```



SIMPLE PLOTTING BASICS

Change the line color:

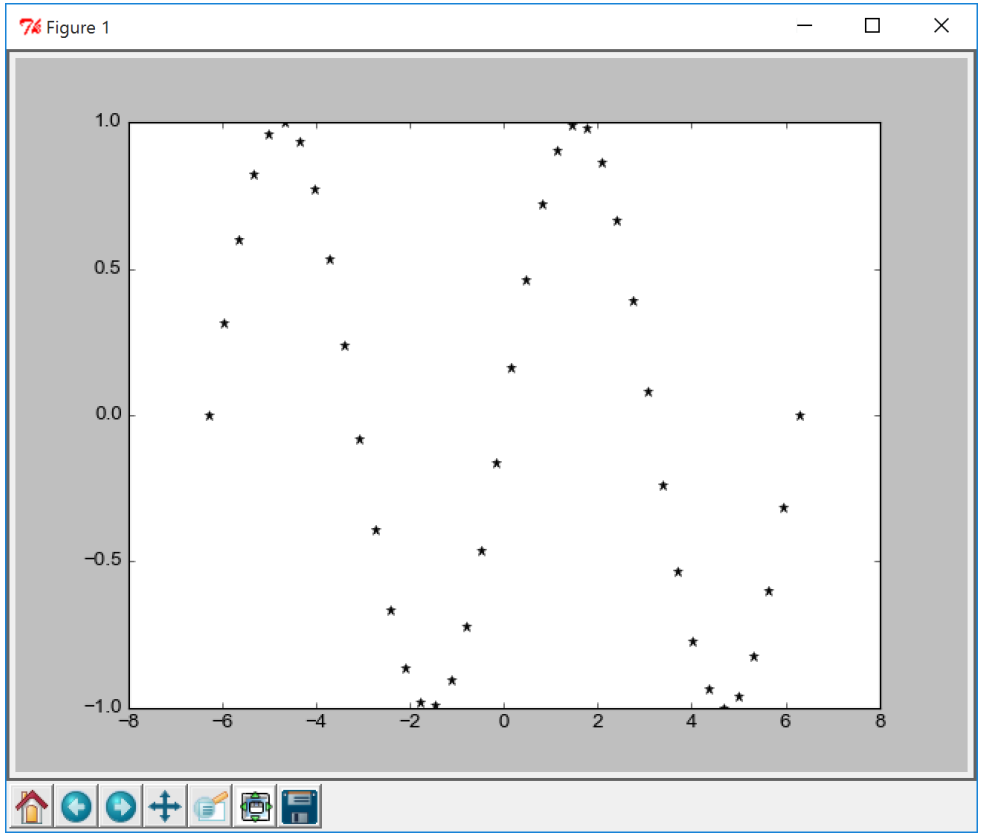
```
plt.plot(x, y,
linewidth=3,
linestyle='dashed',
color='k')
```



SIMPLE PLOTTING BASICS

Add point markers:

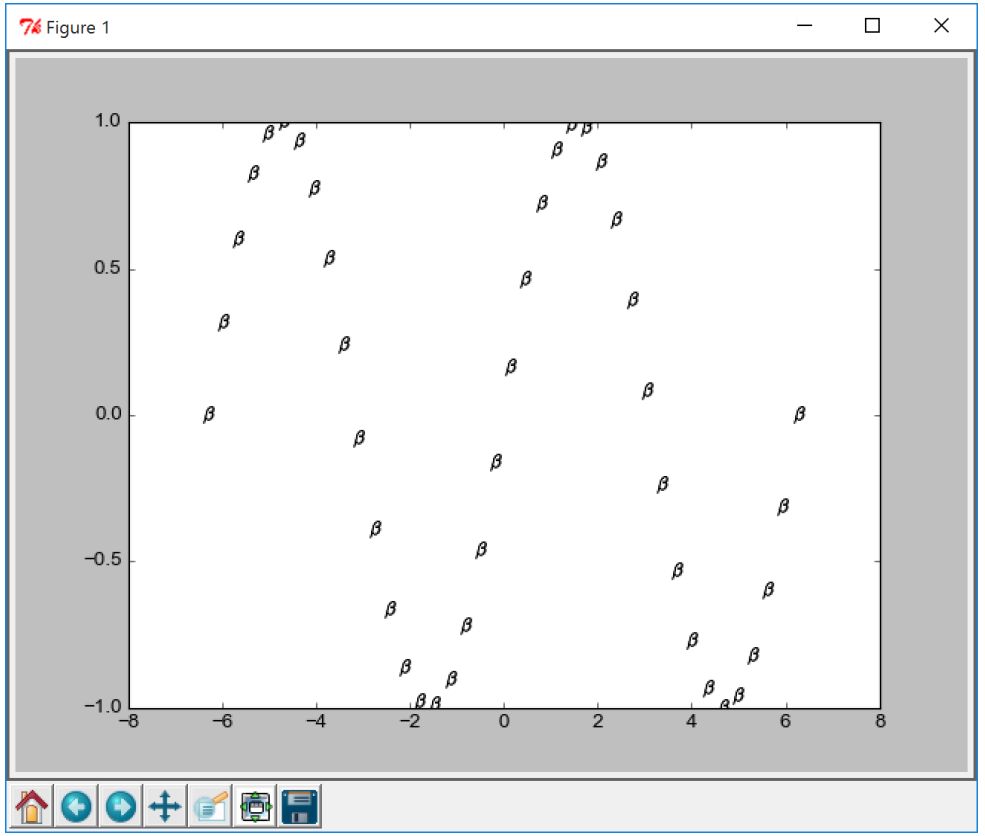
```
plt.plot(x, y,
linestyle='none',
color='k',
marker='*')
```



SIMPLE PLOTTING BASICS

Can use LaTeX symbols:

```
plt.plot(x, y,
linestyle='none',
color='k',
marker='$\\beta$',
markersize=10)
```



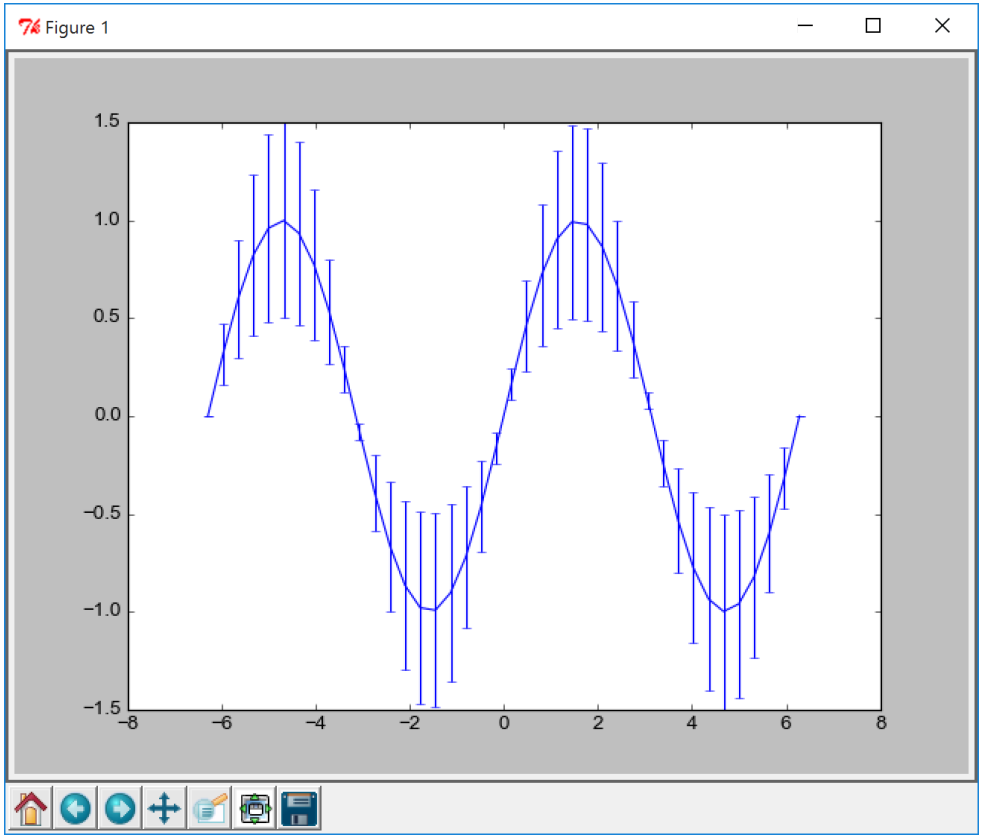
PRO TIP:

For a scatter plot, use plt.scatter() instead

SIMPLE PLOTTING BASICS

Creating error bars:

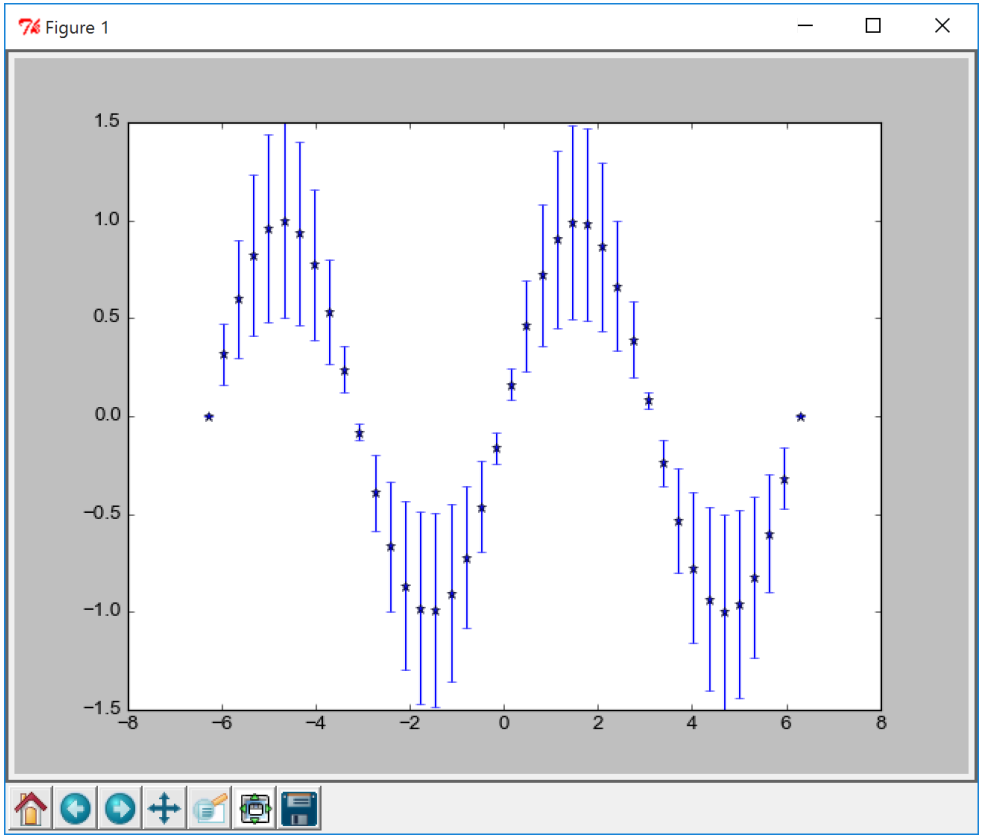
```
n = len(x)
yerr=np.random.uniform(0,.5,n)
plt.errorbar(x, y, yerr=yerr)
```



SIMPLE PLOTTING BASICS

Creating error bars without lines:

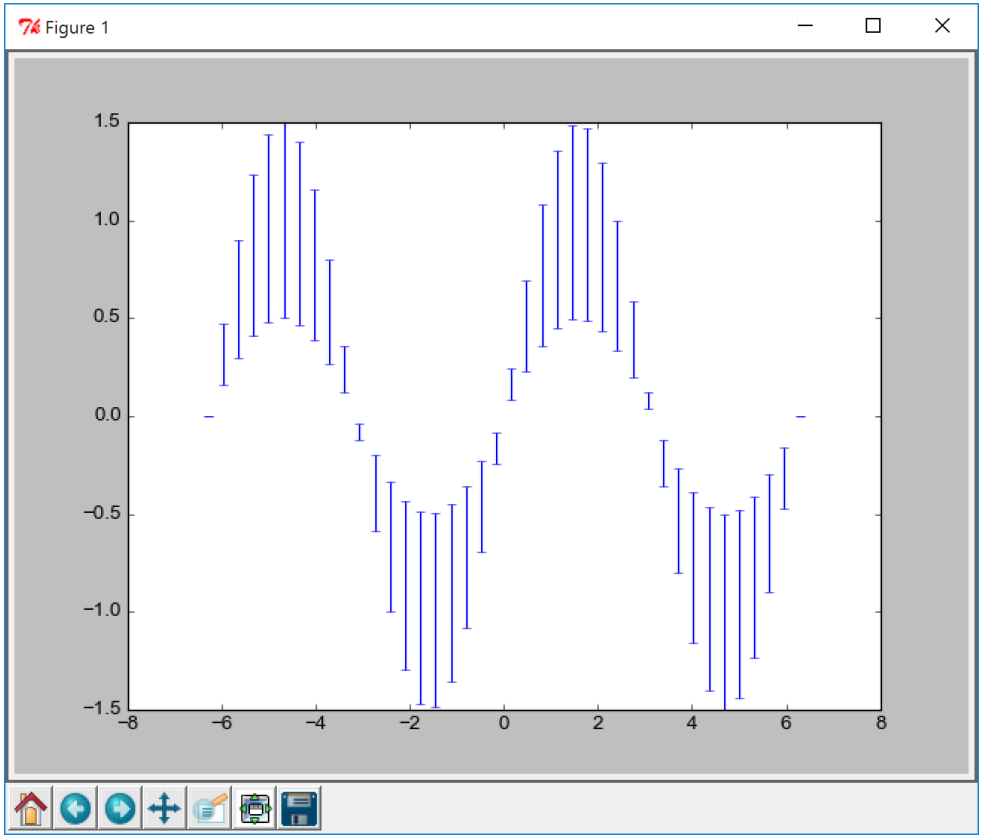
```
plt.errorbar(x, y,
yerr=yerr, fmt='*')
```



SIMPLE PLOTTING BASICS

Just error bars:

```
plt.errorbar(x, y,
yerr=yerr,
fmt='none')
```



PRO TIP:

All of these functions have **many** more options. Check the docs.

COLO(U)RS IN MATPLOTLIB

In matplotlib, colours can be specified in a number of ways:

Basic Colours

Most basic (primary and secondary) colours can be quoted by their first letter:

'b' – blue
'r' – red
'g' – green
'y' – yellow
'w' – white
'k' – black

HTML Colours

Any defined HTML colour name is a valid colour:

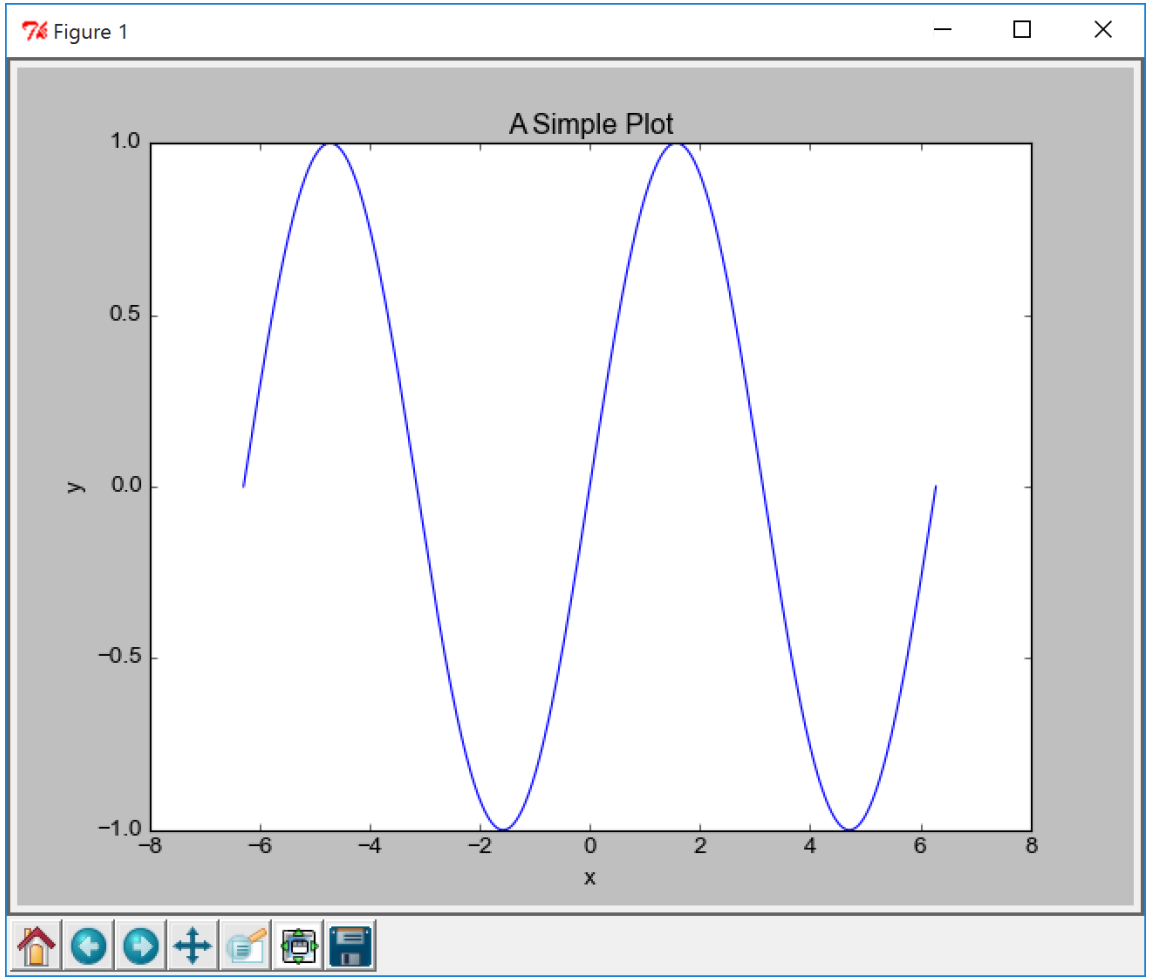
"deeppink"
"slateblue"
"ivory"
"lemonchiffon"

Hex code

Any string of hex codes in the form of "#rrggbb" where each pair goes from 00 to ff:

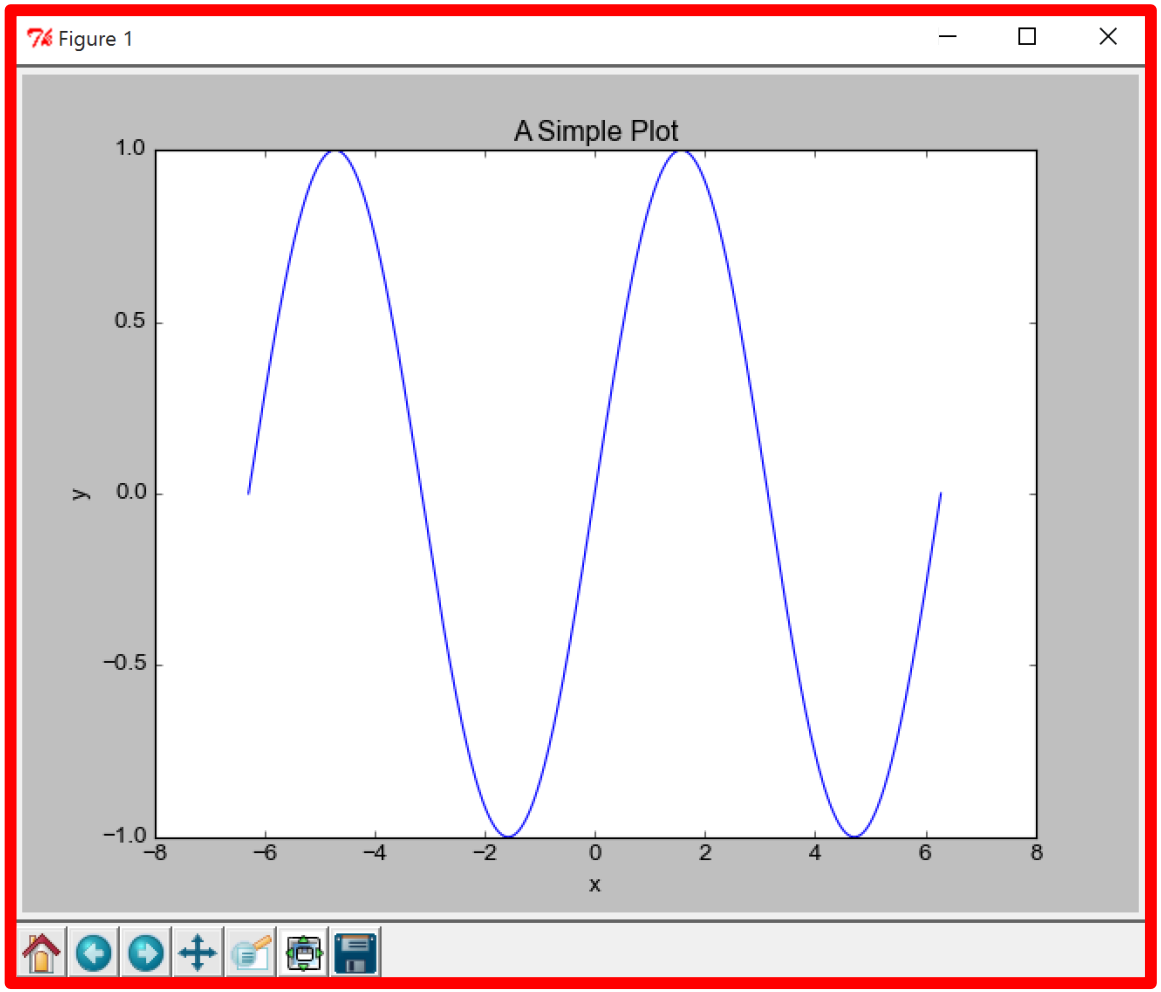
"#ffffff"
"#000000"
"#ff0000"
"#ff00ff"

ANATOMY OF A PLOT WINDOW



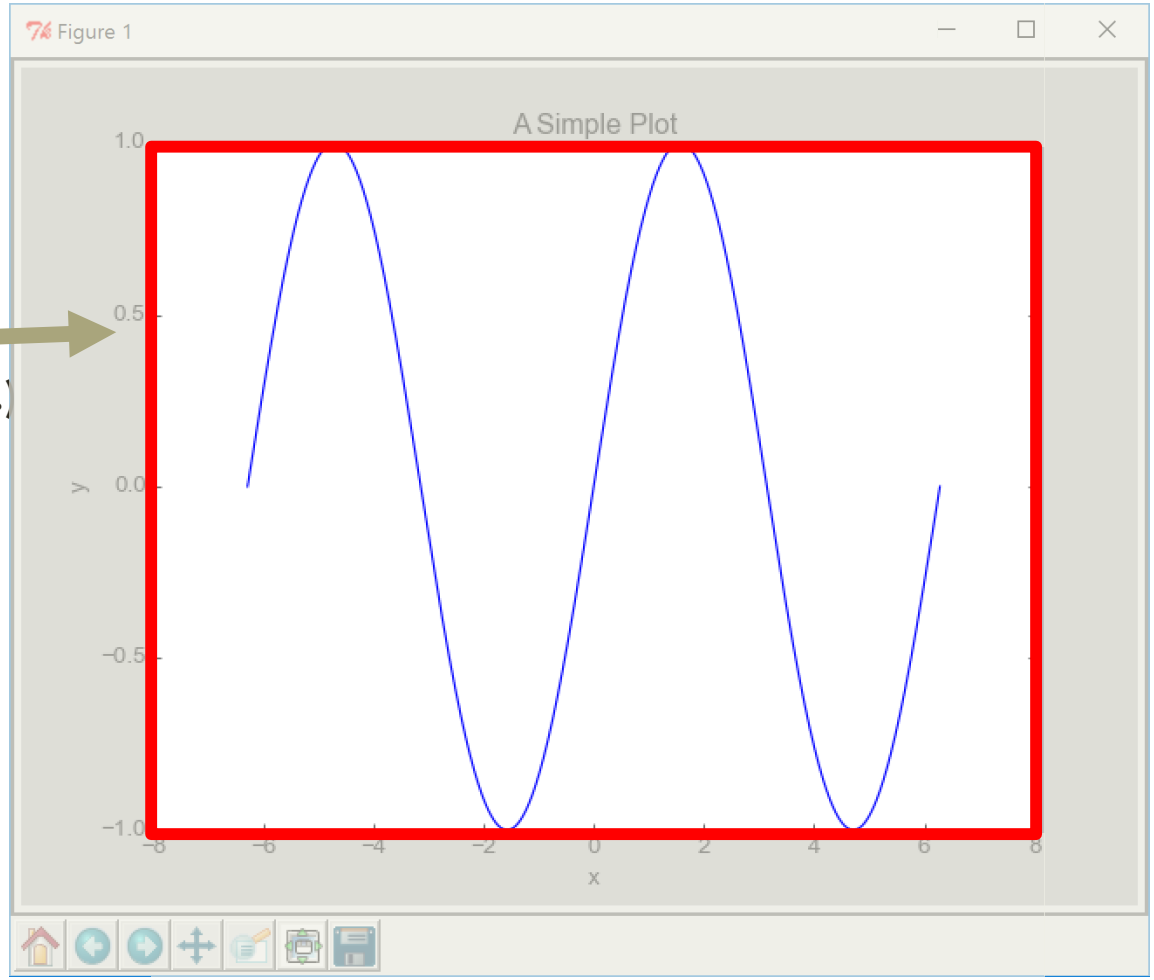
ANATOMY OF A PLOT WINDOW

Figure

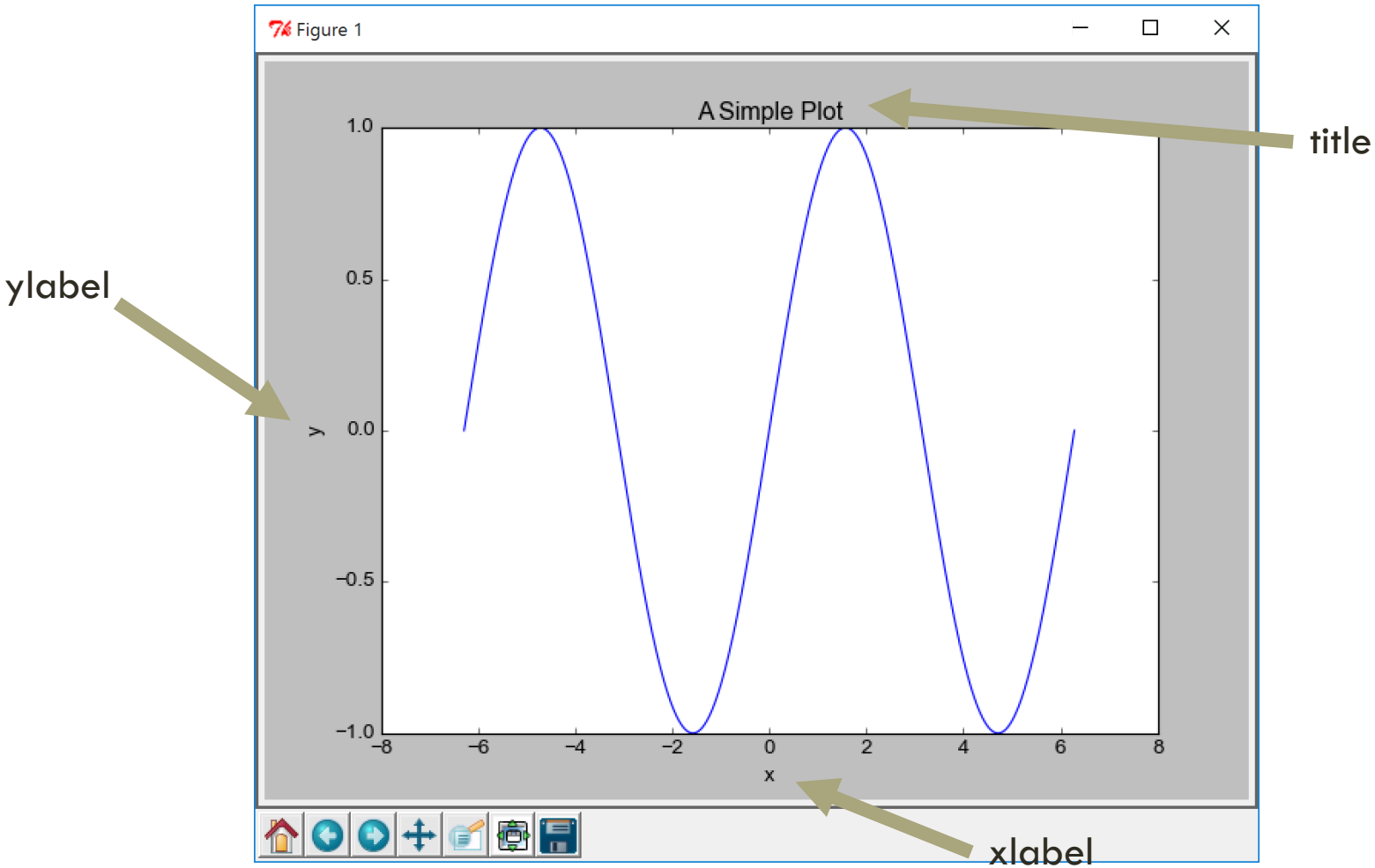


ANATOMY OF A PLOT WINDOW

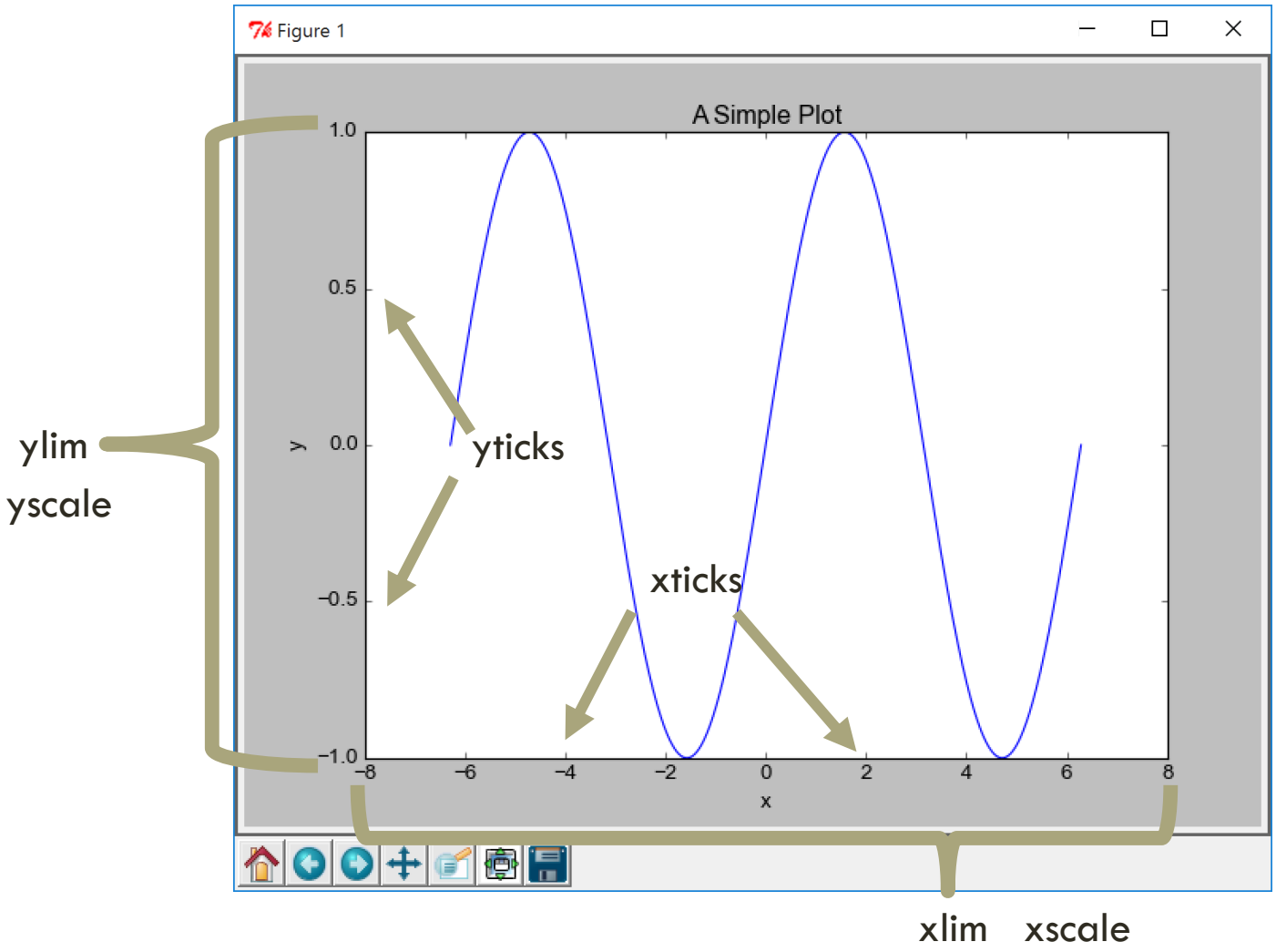
Axis
(confusingly...)



ANATOMY OF A PLOT WINDOW



ANATOMY OF A PLOT WINDOW



HOUSEKEEPING FUNCTIONS

To deal with the various figures and axes that there can be, you have the following housekeeping functions:

```
# Clearing Plots
plt.cla() # Clear Current Axis
plt.clf() # Clear Current Figure

# Getting active objects
ax1 = plt.gca() # Get Current Axis
fig1 = plt.gcf() # Get Current Figure

# Make new figure
plt.figure() # Make new figure (with defaults)
plt.figure(figsize=(6,8)) # Make new figure (6”x8”)
```

SETTING AXIS PROPERTIES

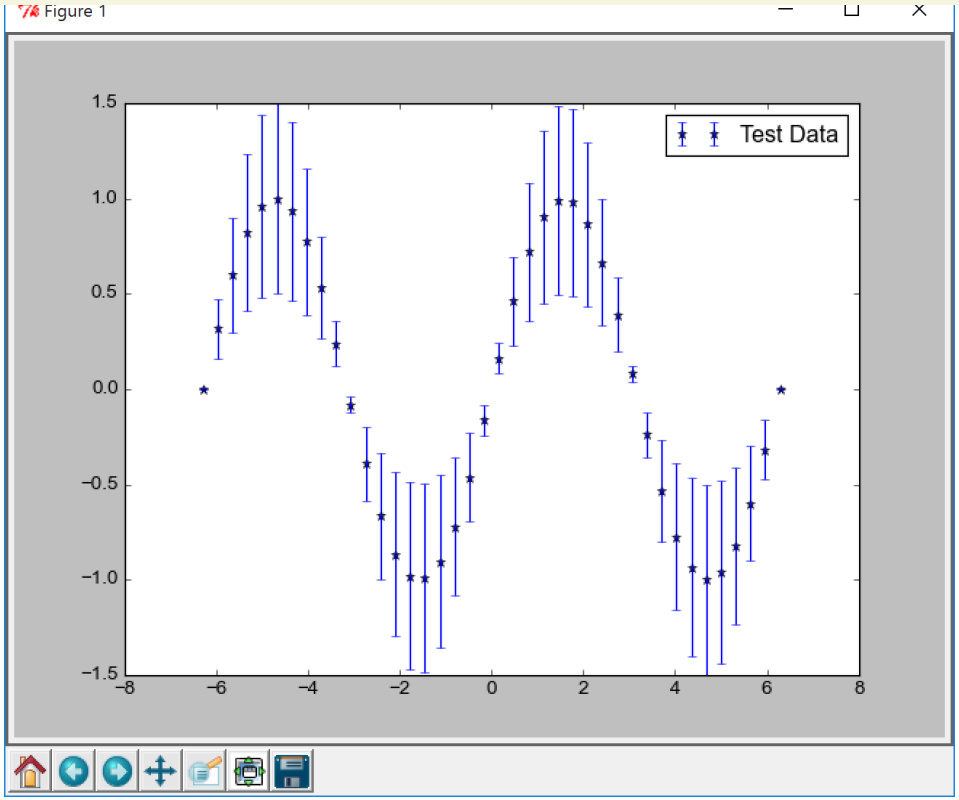
You can (at any time in the plotting) change the range (lim), scale (log or linear), labels or ticks on a plot. Replace x with y (or vice versa) when necessary:

```
# Limits and Scale
plt.xlim([0, 5]) # Set x-limits to 0 -> 5
plt.yscale('log') # Set y-axis to logarithmic

# Setting Labels
plt.xlabel('X-axis') # Label the X-axis
plt.title("Title") # Set the Axis title

# Setting Ticks
plt.xticks([0, 0.5, 3, 4.9]) # Location of x-ticks
```

LABELS AND LEGENDS

You can use “labels” on any plot object to automatically populate a legend:

```
plt.errorbar(x, y, yerr=yerr,color''k',fmt=‘*’,label='Test Data')
plt.legend(frameon=True)
```



SAVING A FIGURE

Saving a figure is a one-line operation. Matplotlib will figure out what format you want by the extension of the filename:

```
plt.savefig(“myplot.pdf”) # Saving as a PDF
plt.savefig(“myplot.png”) # Saving as a PNG
plt.savefig(“myplot.eps”) # Saving as an EPS

# Can also determine what output DPI:
plt.savefig(“myplot.jpg”, dpi=300)
```

PRO TIP:

EPS files do not support transparency natively

BUILDING FROM THE GROUND UP

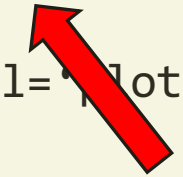
This method gives you a lot more flexibility. Instead of using convenience functions, you use methods on each of the objects:

```
fig1 = plt.figure()
ax1 = fig1.add_axes([0.1, 0.1, 0.8, 0.8]) # Set area

ax1.plot(x, y, marker='o', label='plotted line')
ax1.legend()

ax1.set_xlim([1, 8])
ax1.set_ylim([-2, 3])

ax1.set_xscale('log')
ax1.set_xlabel('X Label')
ax1.set_ylabel('Y Label')
fig1.savefig('plot2.png')
```

BUILDING FROM THE GROUND UP

This method gives you a lot more flexibility. Instead of using convenience functions, you use methods on each of the objects:

```
fig1 = plt.figure()
ax1 = fig1.add_axes([0.1, 0.1, 0.8, 0.8]) # Set area

ax1.plot(x, y, marker='o', label='plotted line')
ax1.legend()

ax1.set_xlim([1, 8])
ax1.set_ylim([-2, 3])

ax1.set_xscale('log')
ax1.set_xlabel('X Label')
ax1.set_ylabel('Y Label')
fig1.savefig('plot2.png')
```



This uses matplotlib's location format, which takes the format of:

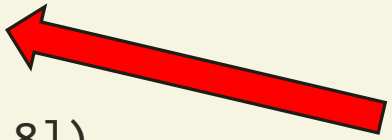
[left, bottom, width, height]

where each of the numbers are from 0 to 1 (in units of a fraction of the figure)

BUILDING FROM THE GROUND UP

This method gives you a lot more flexibility. Instead of using convenience functions, you use methods on each of the objects:

```
fig1 = plt.figure()
ax1 = fig1.add_axes([0.1, 0.1, 0.8, 0.8]) # Set area

ax1.plot(x, y, marker='o', label='plotted line')
ax1.legend()

ax1.set_xlim([1, 8])
ax1.set_ylim([-2, 3])

ax1.set_xscale('log')
ax1.set_xlabel('X Label')
ax1.set_ylabel('Y Label')
fig1.savefig('plot2.png')
```



All of those major plotting functions (i.e, plot, scatter, legend, *et cetera*) are now just methods on the axis.

BUILDING FROM THE GROUND UP

This method gives you a lot more flexibility. Instead of using convenience functions, you use methods on each of the objects:

```
fig1 = plt.figure()
ax1 = fig1.add_axes([0.1, 0.1, 0.8, 0.8]) # Set area

ax1.plot(x, y, marker='o', label='plotted line')
ax1.legend()

ax1.set_xlim([1, 8])
ax1.set_ylim([-2, 3])

ax1.set_xscale('log')
ax1.set_xlabel('X Label')
ax1.set_ylabel('Y Label')
fig1.savefig('plot2.png')
```

All axis properties (i.e., x/ylim, x/yscale) can be set by the methods axis.set_*property*. Also, you can get the current values for these by axis.get_*property*.

BUILDING FROM THE GROUND UP

This method gives you a lot more flexibility. Instead of using convenience functions, you use methods on each of the objects:

```
fig1 = plt.figure()
ax1 = fig1.add_axes([0.1, 0.1, 0.8, 0.8]) # Set area

ax1.plot(x, y, marker='o', label='plotted line')
ax1.legend()

ax1.set_xlim([1, 8])
ax1.set_ylim([-2, 3])

ax1.set_xscale('log')
ax1.set_xlabel('X Label')
ax1.set_ylabel('Y Label')
fig1.savefig('plot2.png')
```

Saving the figure is a method of the figure itself

This is particularly useful if you have multiple figures and axes.

CUSTOMIZING DEFAULTS

There's a lot of different parameters that matplotlib chooses by default, but you can set your own using a **matplotlibrc** file. This file will not exist by default, but you can download a sample one here:

http://matplotlib.org/_static/matplotlibrc

The place to put this file depends on your platform:

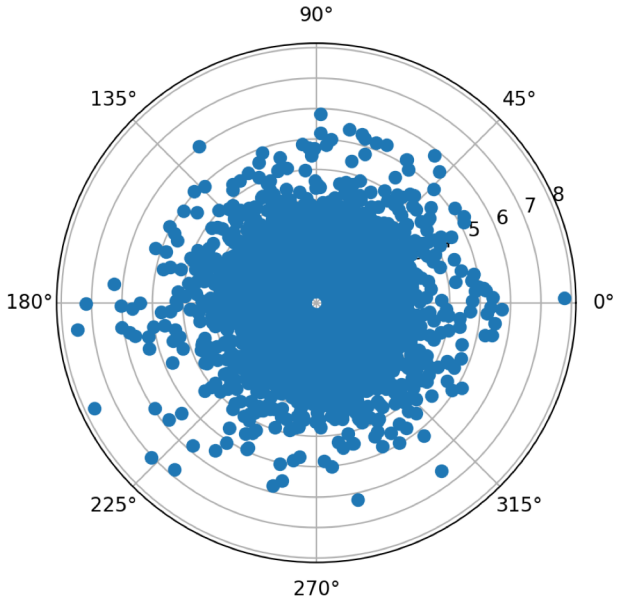
Windows: *UserDirectory*/.matplotlib/matplotlibrc
(i.e., C:/Users/*username*/.matplotlib/matplotlibrc)

MacOS: *UserDirectory*/.matplotlib/matplotlibrc
(i.e., Users/*username*/.matplotlib/matplotlibrc)

Linux: *UserDirectory*/.config/matplotlib/matplotlibrc
(i.e., /home/*username*/.config/matplotlib/matplotlibrc)

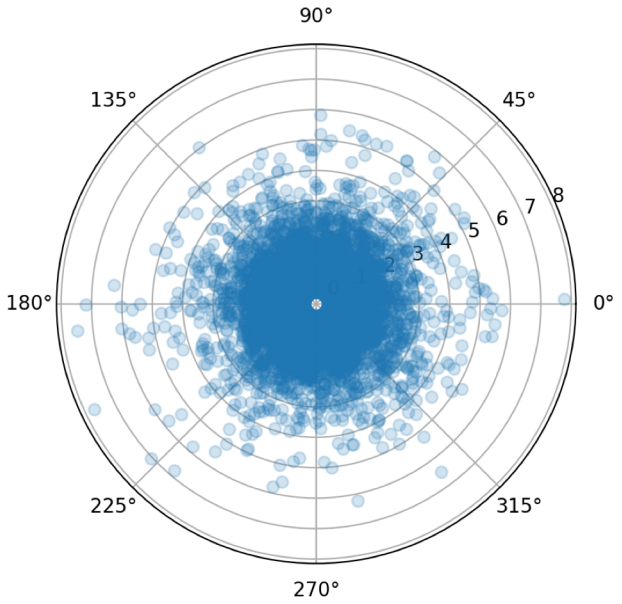
SOME ADVANCED FEATURES

ALPHA/TRANSPARENCY




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

# Cluster data
a = np.random.uniform(0, 2*np.pi, size=10000)
r = np.random.exponential(size=10000)

# Make a polar plot.
plt.polar(a, r, linestyle='none', marker='o')
```

ALPHA/TRANSPARENCY



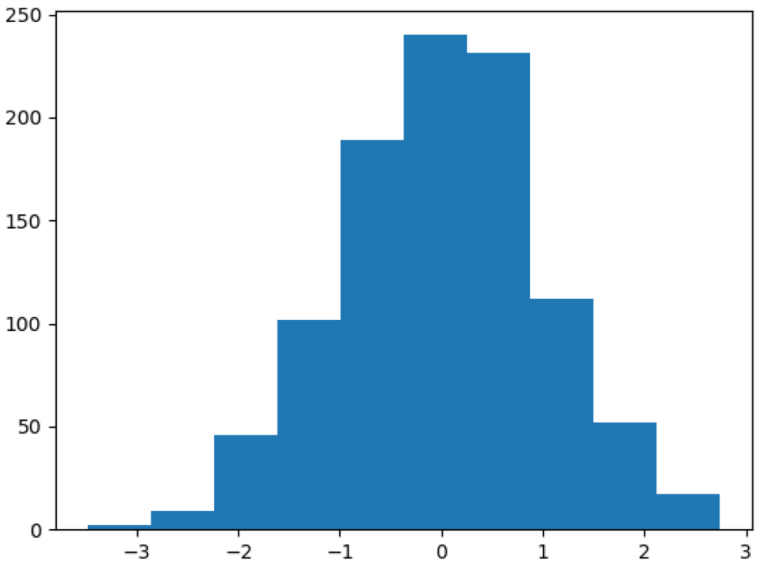
PRO TIP:

Saving to EPS doesn't support transparency.

```
# Alpha controls transparency
# 0 = transparent
# 1 = opaque

# Make markers transparent.
plt.clf()
plt.polar(a, r, linestyle='none', marker='o', alpha=0.2)
```

HISTOGRAMS

```
# Gaussian random data
y = np.random.normal(size=1000)

# Default histogram (10 bins)
plt.clf()
plt.hist(y)
```

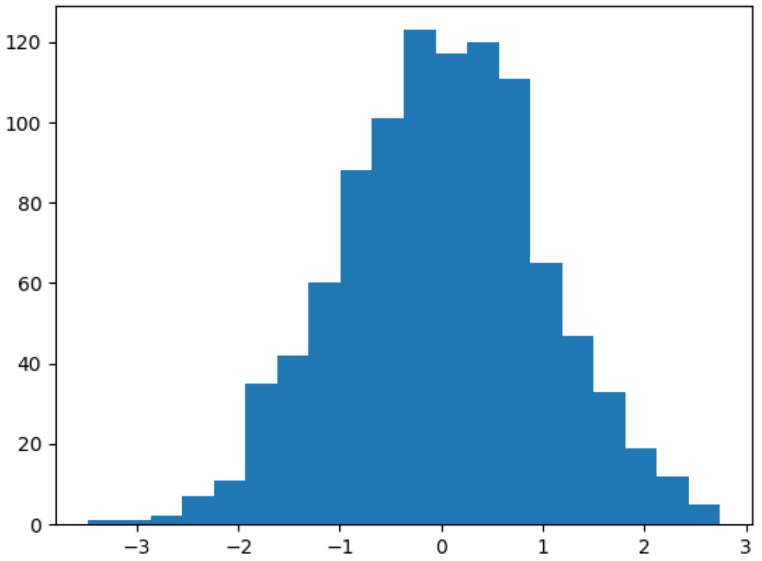


PRO TIP:

The histogram function takes a one-dimensional array. If it isn't already, flatten it!

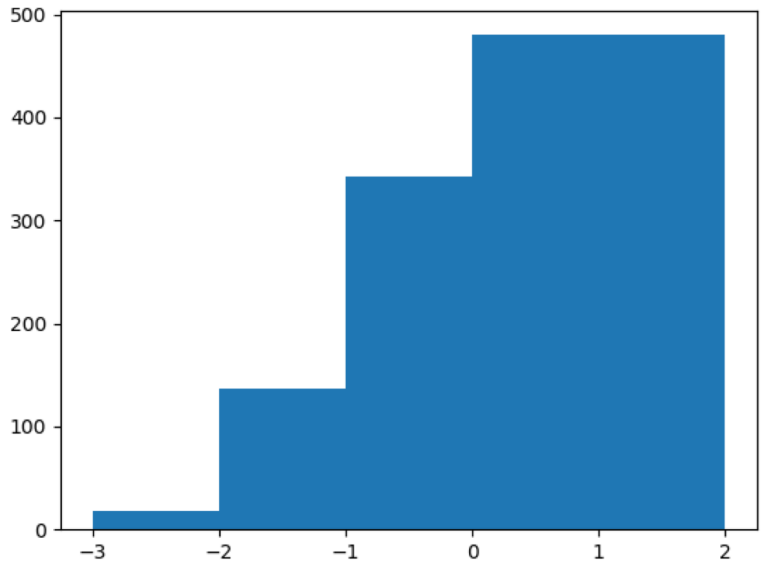
HISTOGRAMS

```
# Change the number of bins.
plt.clf()
plt.hist(y, bins=20)
```



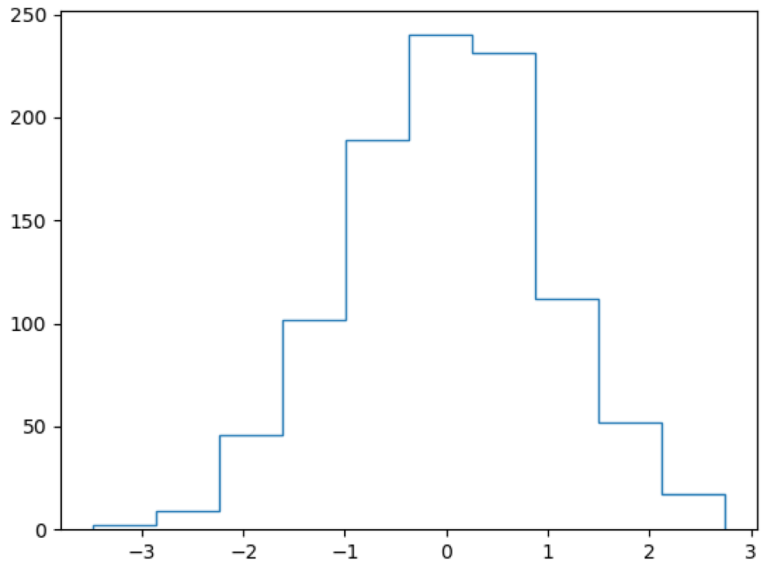
HISTOGRAMS

```
# Change the bin edges.
plt.clf()
edges=[-3, -2, -1, 0, 2]
plt.hist(y, bins=edges)
```



HISTOGRAMS

```
# Just show the lines.
plt.clf()
plt.hist(y, histtype='step')
```



PRO TIP:

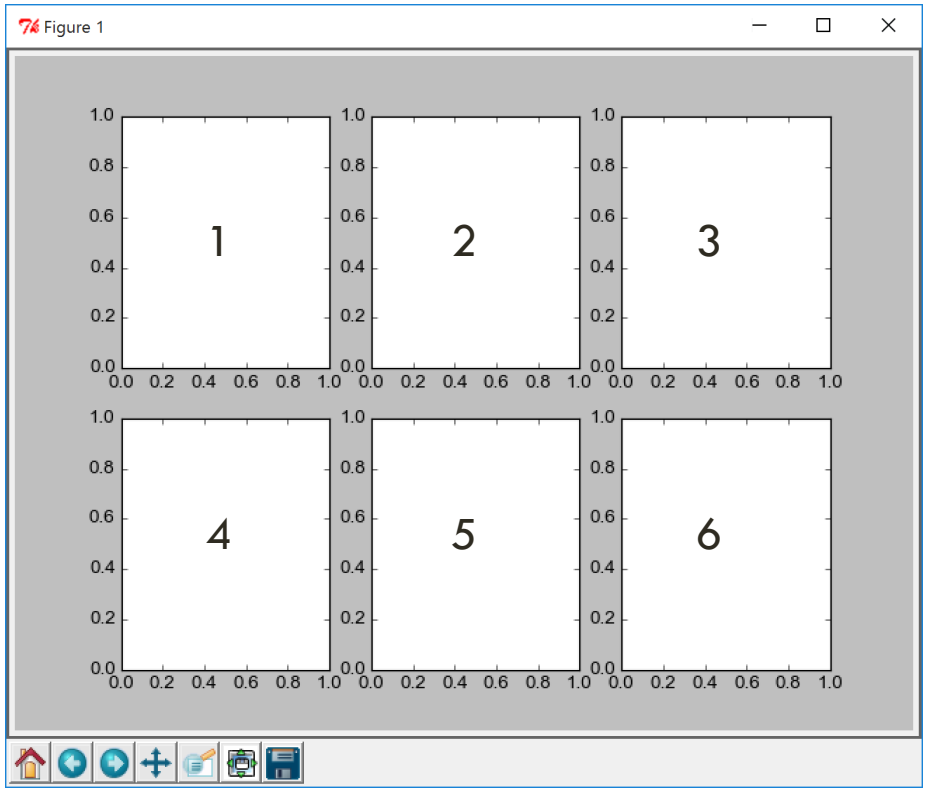
If you just want an array of histogram values, check out the numpy functions histogram, histogram2d, and histogramdd

SUBPLOTS/MULTIPLE PLOTS

Making subplots are quite easy using the convenience function "subplot":

```
ax1 = plt.subplot(
  nrows, ncols,
  plotnum
)
```

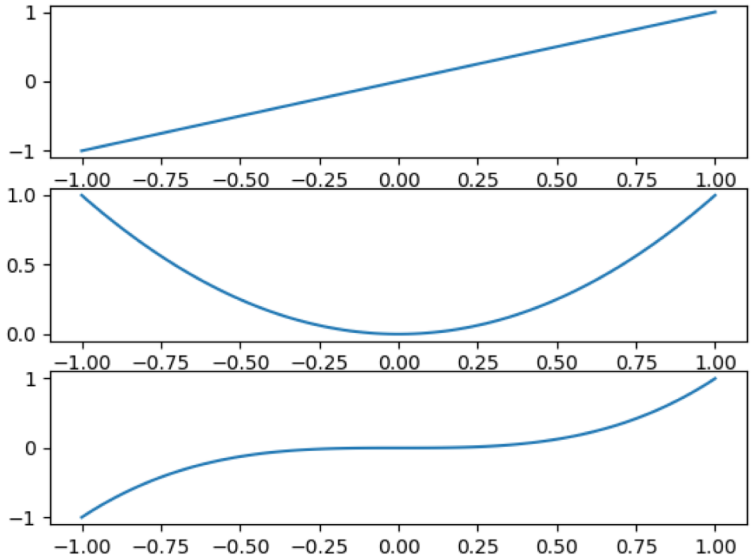
plotnum starts at 1.




```
# Create a 2x3 grid of plots.
ax1 = plt.subplots(3, 2)
```

SUBPLOTS/MULTIPLE PLOTS

```
# 3x1 grid of plots.
plt.clf()
x = np.linspace(-1, 1, 100)
y1 = x
y2 = x**2
y3 = x**3

# Make the plots one at a time.
ax1 = plt.subplot(3, 1, 1)
ax1.plot(x, y1)
ax2 = plt.subplot(3, 1, 2)
ax2.plot(x, y2)
ax3 = plt.subplot(3, 1, 3)
ax3.plot(x, y3)
```



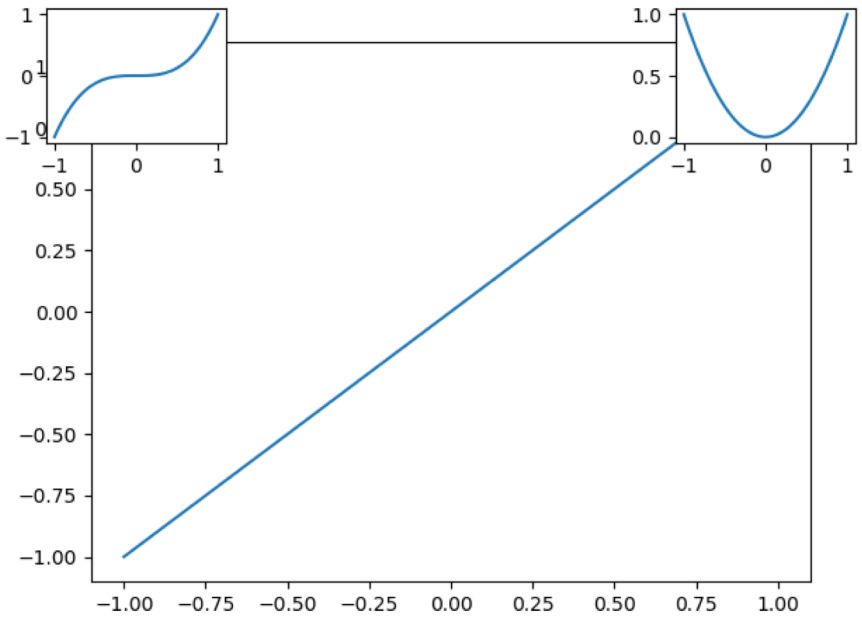
For simple, small numbers of subplots, you can use an alternate argument for the call:

```
plt.subplot(321)
```

where this axis is the first in a grid of 3 rows and 2 columns.

SUBPLOTS/MULTIPLE PLOTS

```
# Arbitrary subplot placement.
plt.clf()

# Make the plots one at a time.
ax1 = plt.axes([0.1, 0.1, 0.8, 0.8])
ax1.plot(x, y1)
ax2 = plt.axes([0.75, 0.75, 0.2, 0.2])
ax2.plot(x, y2)
ax3 = plt.axes([0.05, 0.75, 0.2, 0.2])
ax3.plot(x, y3)
```



ANNOTATIONS: TEXT

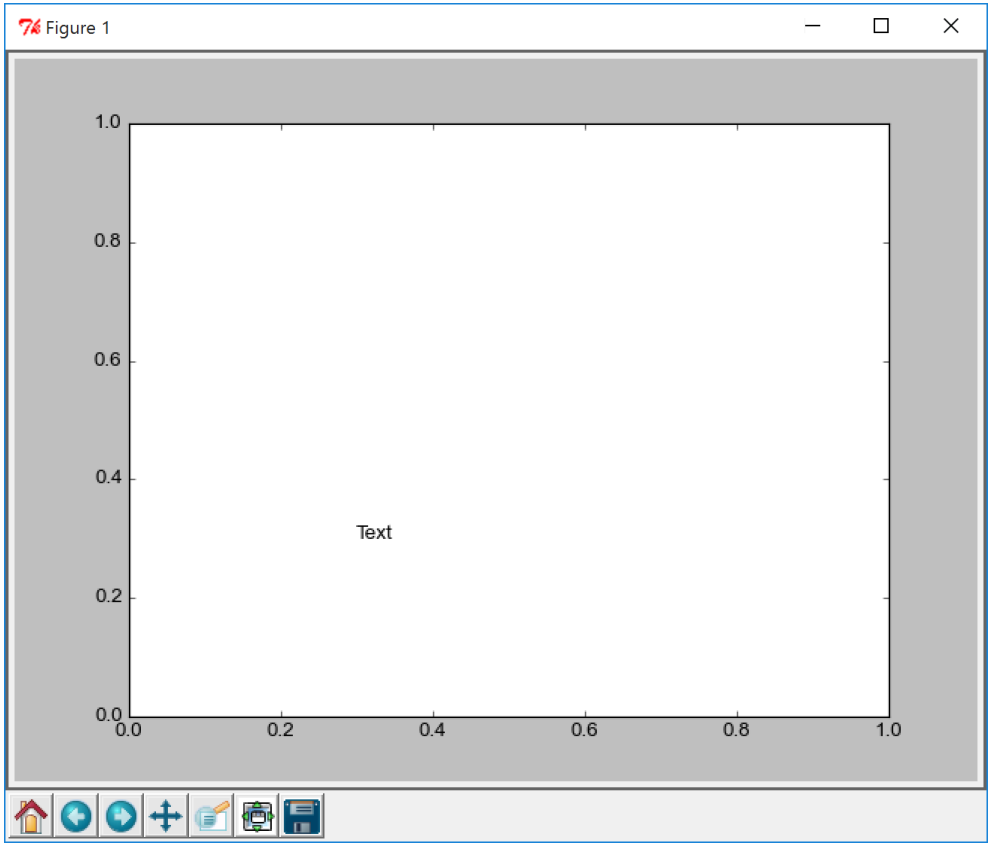
Adding text to axes is simple using the "text" command:

```
plt.text(
  x, y, "Text"
)
```

Or if adding to the figure:

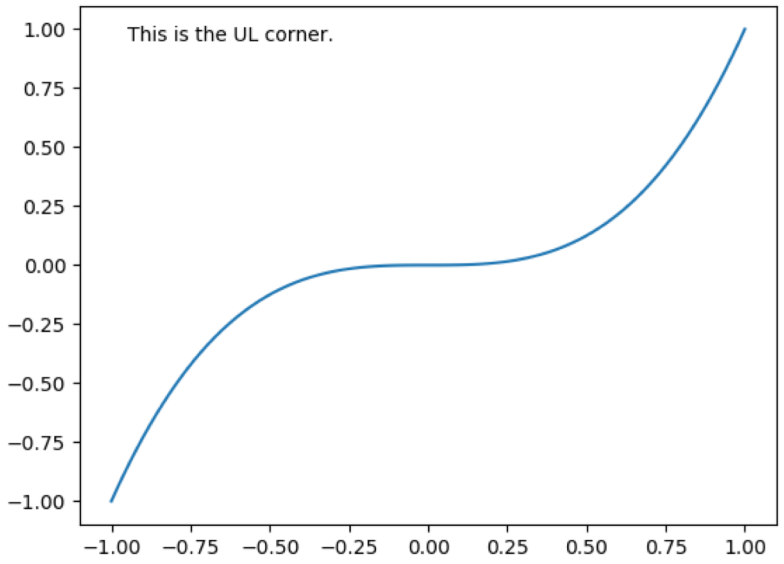
```
plt.figtext(
  x, y, "Text"
)
```

Where these coordinate go from 0 to 1 in fractions of the figure.



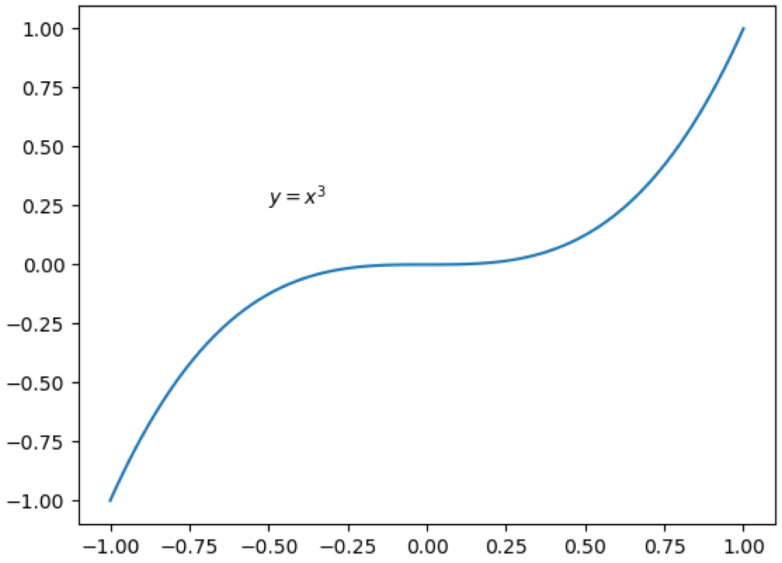
ANNOTATIONS: TEXT

```
# Just specify location and text.
# Location in axis units!
plt.clf()
plt.plot(x, y3)
plt.text(-0.95, 0.95, 'This is the UL corner.')
```



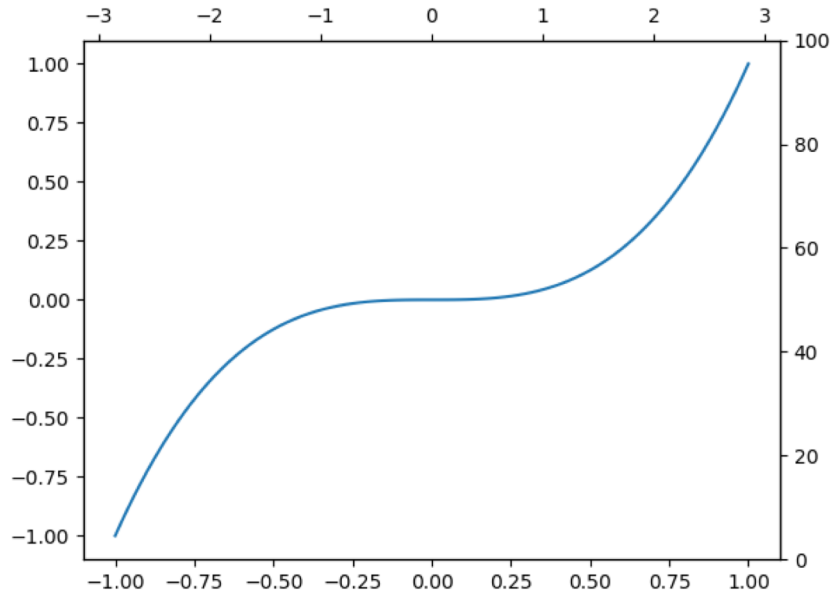
ANNOTATIONS: TEXT

```
# You can use Latex!
plt.clf()
plt.plot(x, y3)
plt.text(-0.5, 0.25, '$y=x^3$')
```



MULTIPLE AXES ON A SINGLE PLOT

```
# Plot the data.
plt.clf()
plt.plot(x, y3)

# Add extra x and y axes.
# WATCH CAREFULLY!
xaxis2 = plt.gca().twiny()
yaxis2 = plt.gca().twinx()

# Set new values for new axes.
xaxis2.set_xlim(-np.pi, np.pi)
yaxis2.set_ylim(0,100)
```



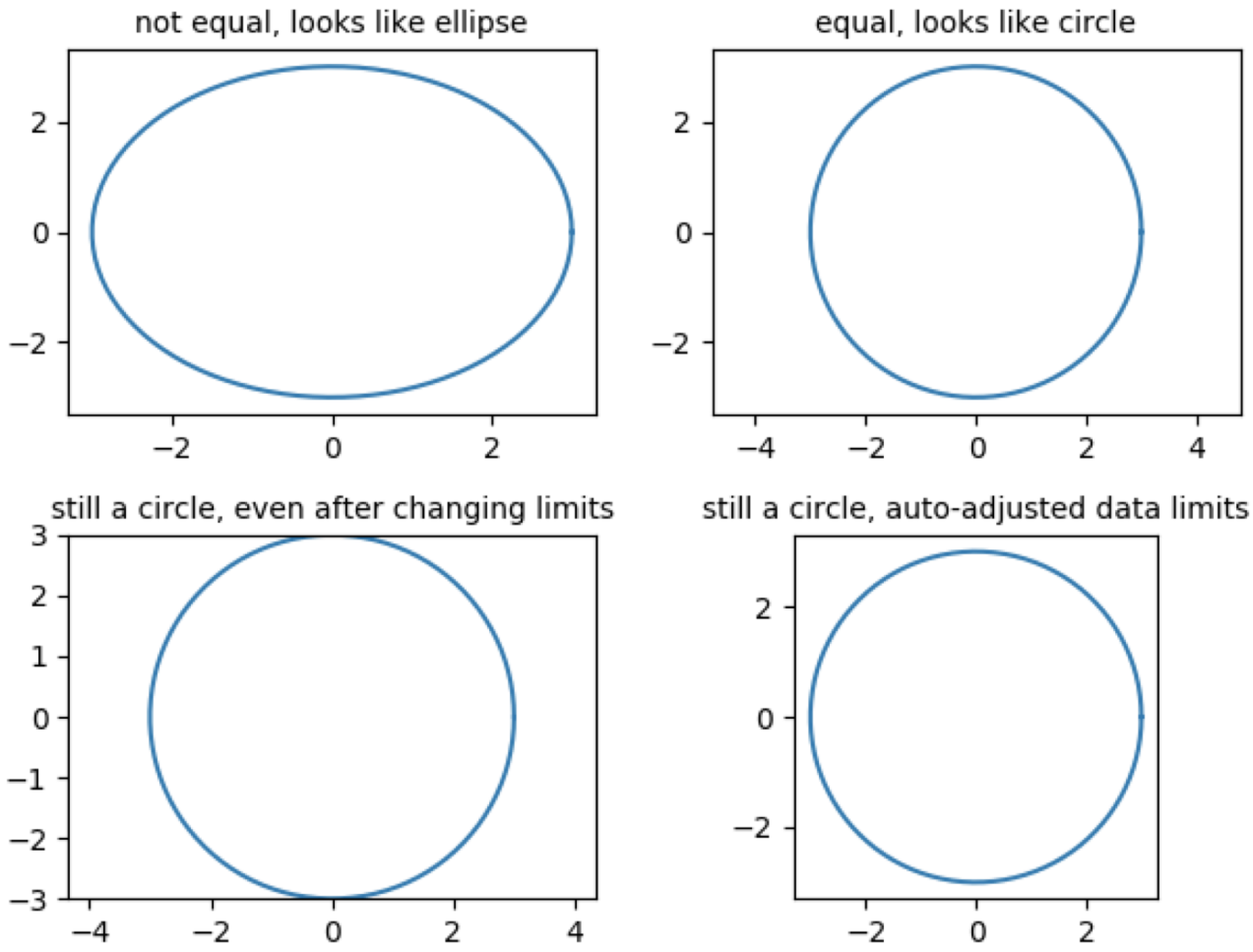
VERY ADVANCED PLOTTING! (1 OF 3)

```
# EQUAL AXES
import matplotlib.pyplot as plt
import numpy as np

# Let's create a circle of radius 3.
an = np.linspace(0, 2 * np.pi, 100)
x = 3 * np.cos(an)
y = 3 * np.sin(an)
```

VERY ADVANCED PLOTTING! (2 OF 3)

```
fig, axs = plt.subplots(2, 2)

axs[0, 0].plot(x, y)
axs[0, 0].set_title('not equal, looks like ellipse', fontsize=10)

axs[0, 1].plot(x, y)
axs[0, 1].axis('equal')
axs[0, 1].set_title('equal, looks like circle', fontsize=10)

axs[1, 0].plot(x, y)
axs[1, 0].axis('equal')
axs[1, 0].axis([-3, 3, -3, 3])
axs[1, 0].set_title('still a circle, even after changing limits', fontsize=10)

axs[1, 1].plot(x, y)
axs[1, 1].set_aspect('equal', 'box')
axs[1, 1].set_title('still a circle, auto-adjusted data limits', fontsize=10)

fig.tight_layout()

plt.show()
```

VERY ADVANCED PLOTTING! (3 OF 3)



EXERCISE TIME!