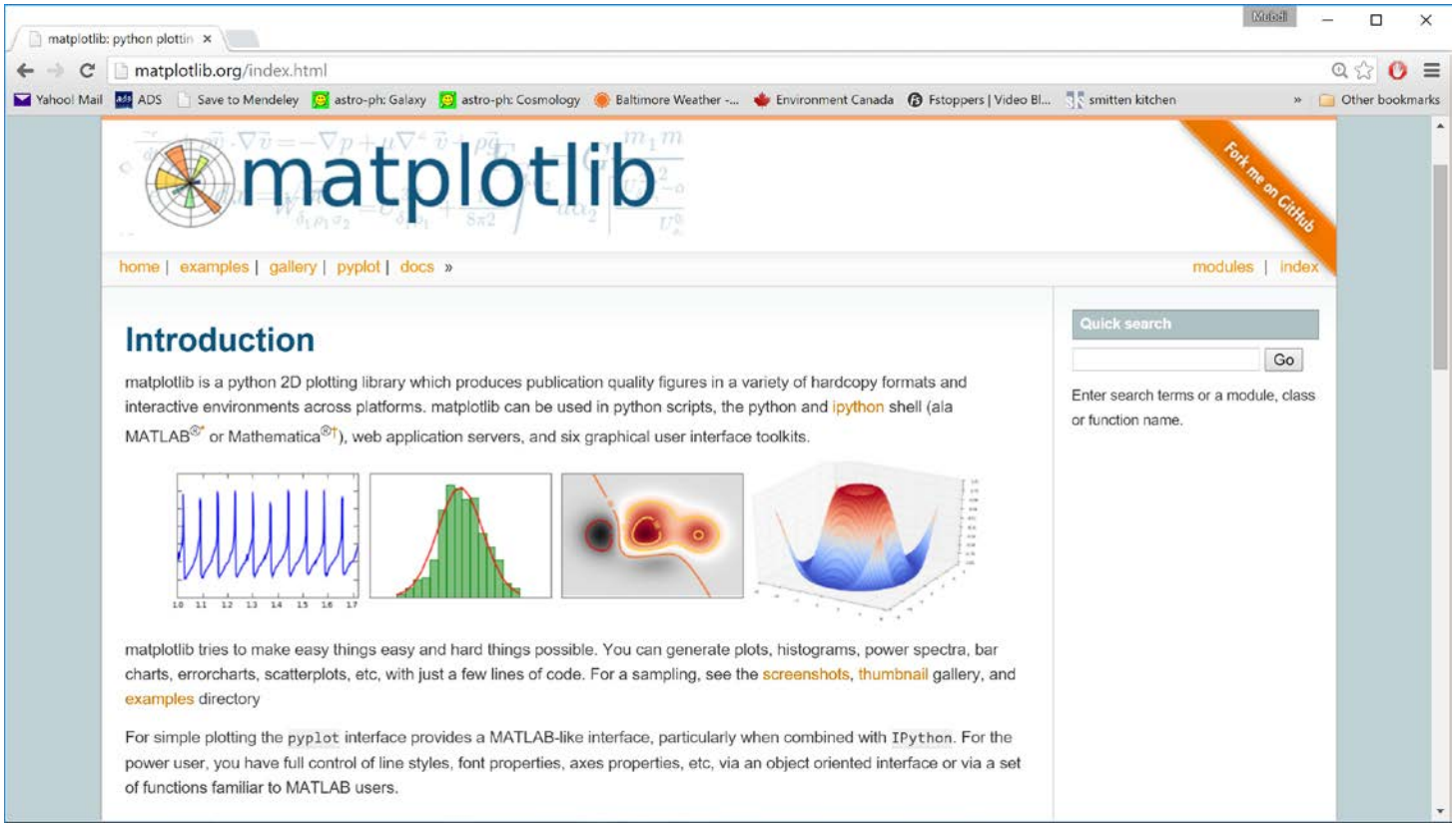
4. BASIC PLOTTING

JHU Physics & Astronomy Python Workshop 2015

Lecturer: Mubdi Rahman

INTRODUCING MATPLOTLIB!



Very powerful plotting package.

The Docs: http://matplotlib.org/api/pyplot_api.html

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** 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
```

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** interface:

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

You can choose to run matplotlib either **interactively** or **non-interactively**. For the interactive m
go along. For non-interactive, the
finished everything. To switch bet

```
plt.ion() # Turn interact
plt.ioff() # Turn interac
plt.show() # Show the plo
```

MUBDI IS A BONEHEAD NOTE:

I started using python back in the "Wild West" days. Some of the defaults of how I code are not the standards suggested today. In particular, I import matplotlib.pyplot as p. Call me on this!

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** interface:

```
import matplotlib.pyplo
```

You can choose to run matplot
interactively. For the interacti
go along. For non-interactive,
finished everything. To switch

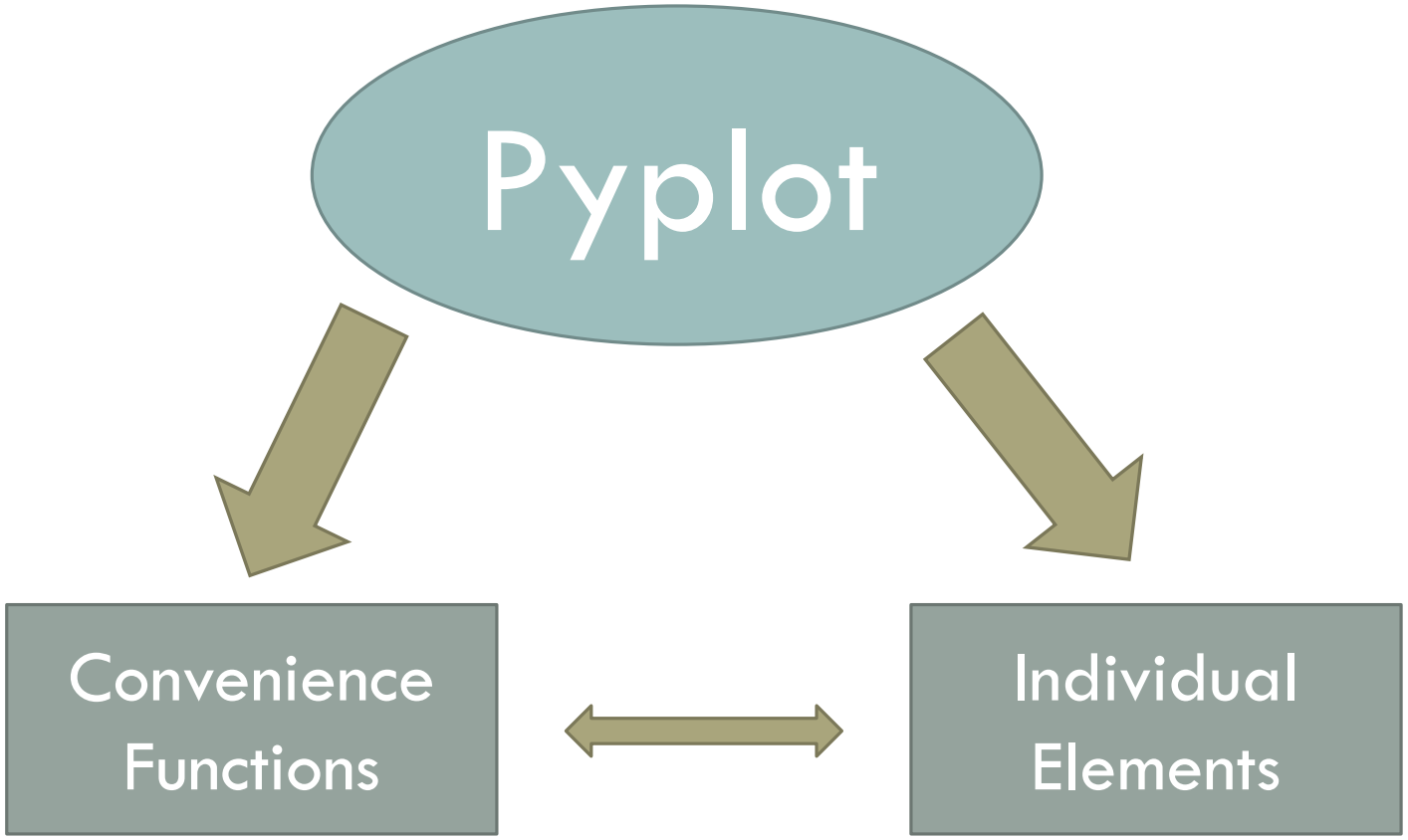
```
plt.ion() # Turn intera
plt.ioff() # Turn inter
plt.show() # Show the p
```

PRO TIP:

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!



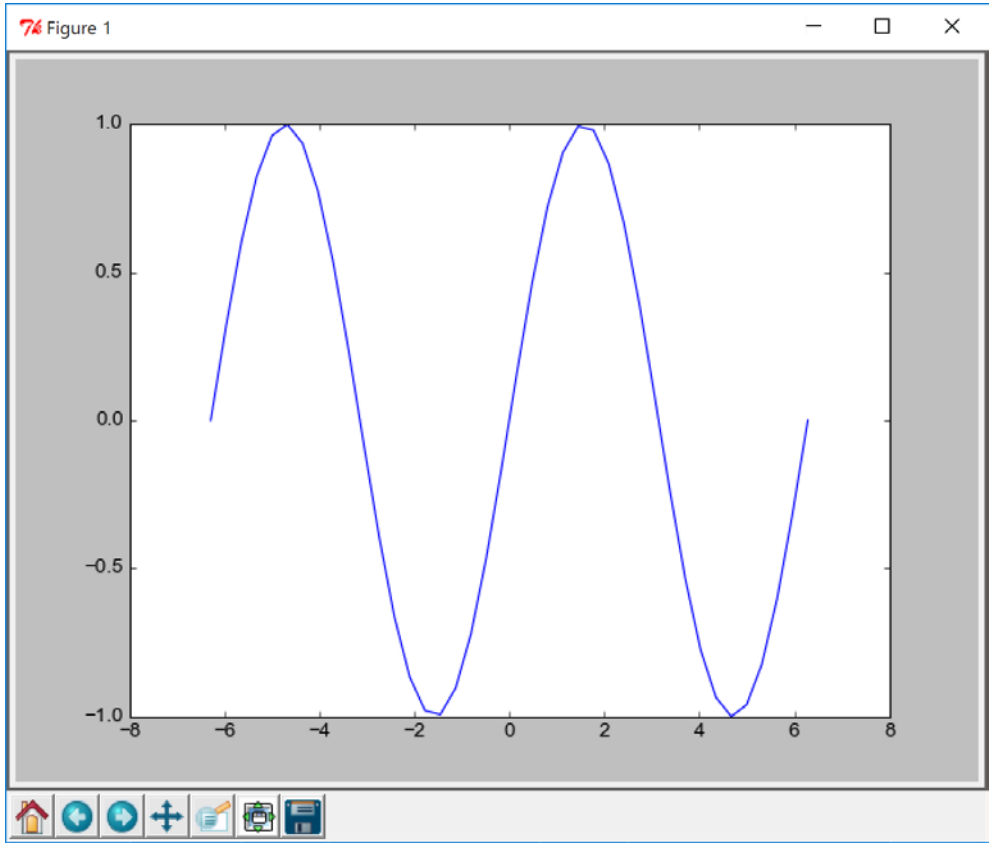
- Really simple to start
- Not as much flexibility

- Requires more coding
- Can plot anything!

SIMPLE PLOTTING BASICS

Much of your power is in the plot command:

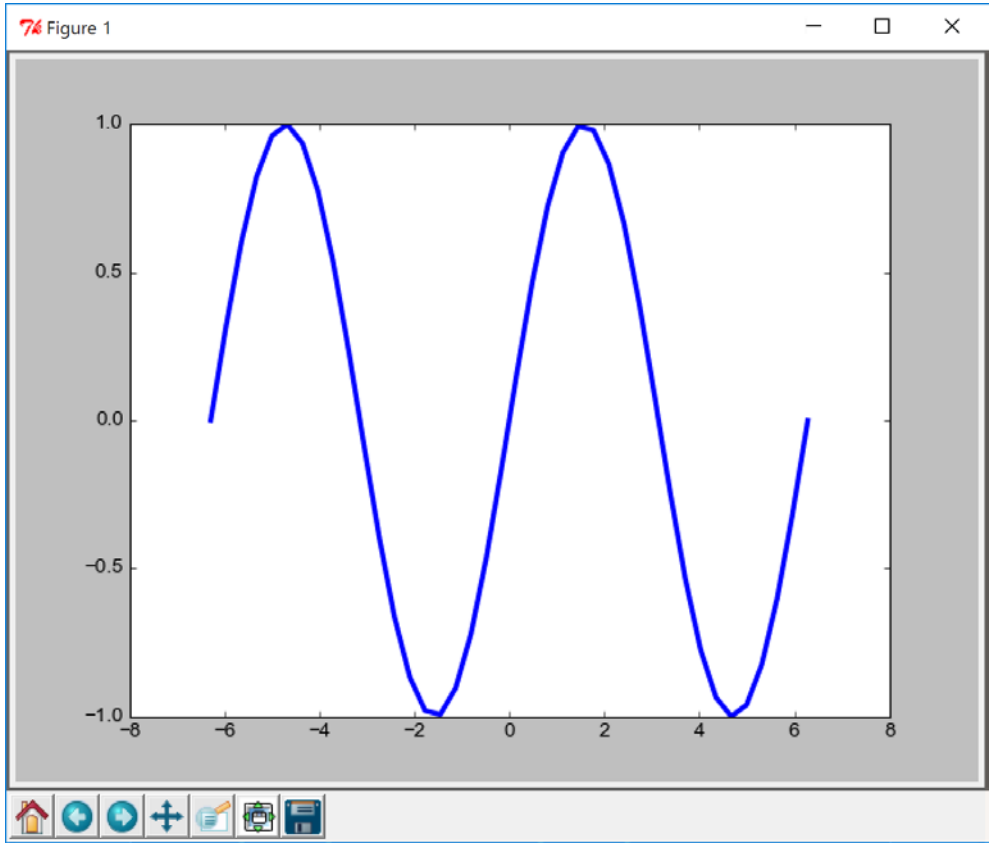
```
# The simplest of
# plots
plt.plot(x, y)
```



SIMPLE PLOTTING BASICS

Much of your power is in the plot command:

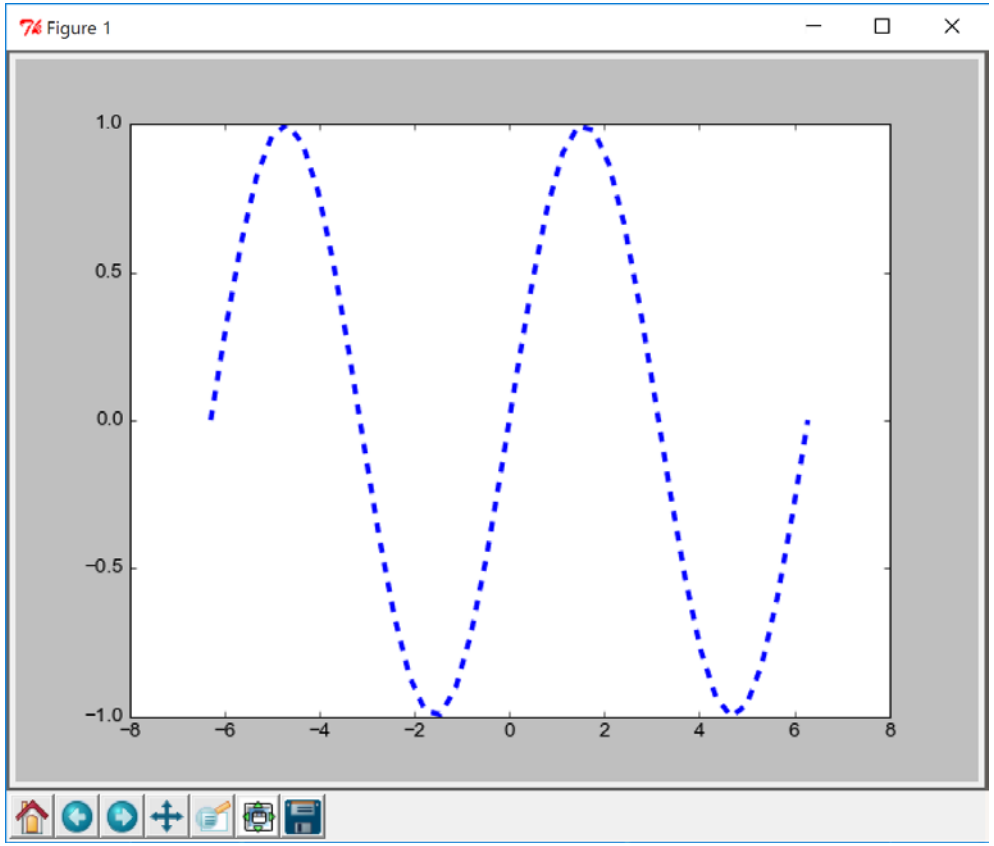
```
plt.plot(x, y, linewidth=3)
```



SIMPLE PLOTTING BASICS

Much of your power is in the plot command:

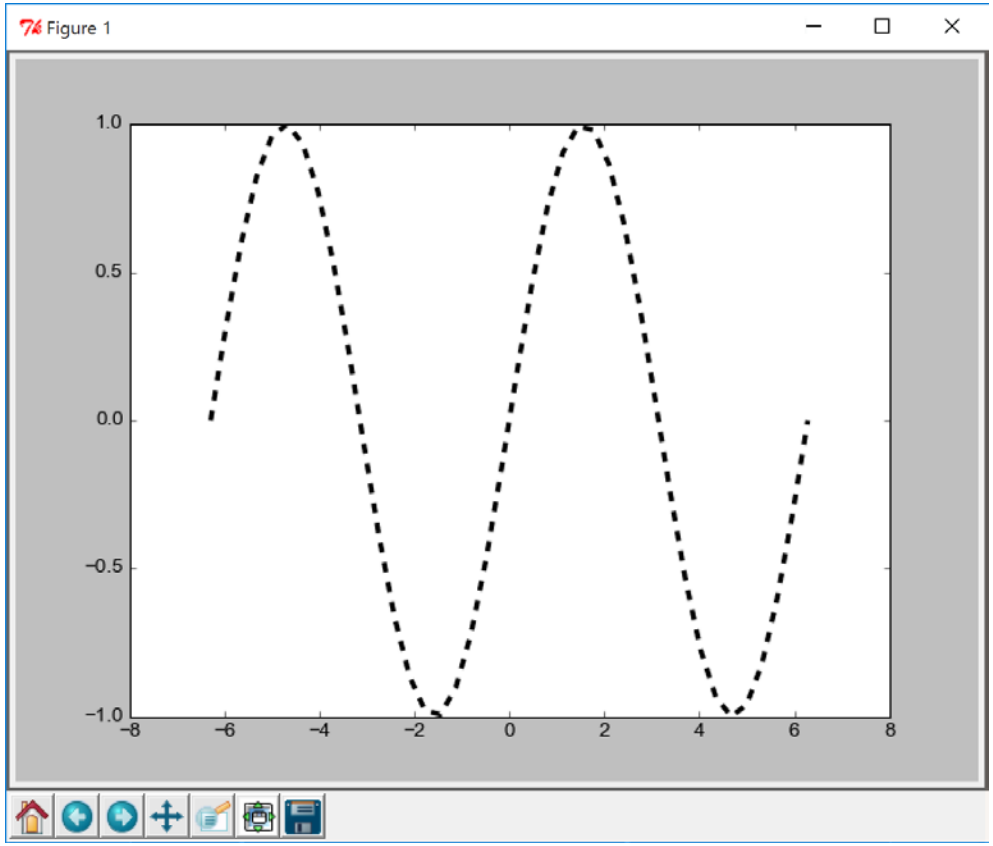
```
plt.plot(x, y,
linewidth=3,
linestyle='dashed')
```



SIMPLE PLOTTING BASICS

Much of your power is in the plot command:

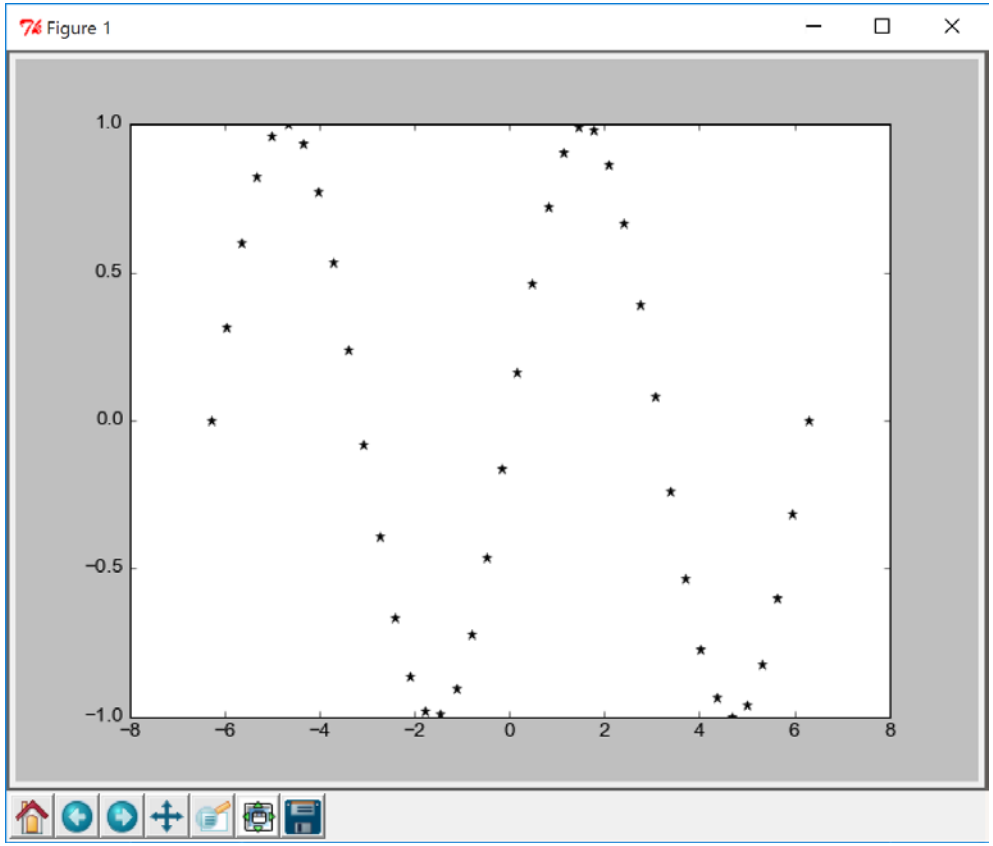
```
plt.plot(x, y,
linewidth=3,
linestyle='dashed',
color='k')
```



SIMPLE PLOTTING BASICS

Much of your power is in the plot command:

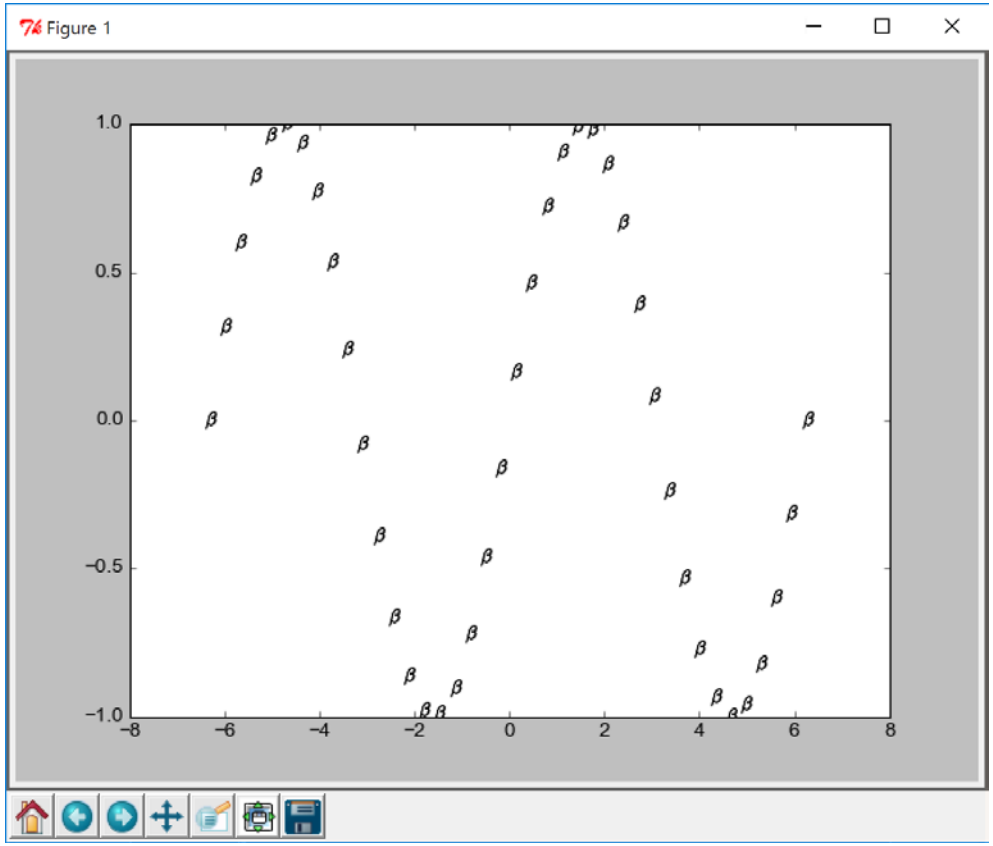
```
plt.plot(x, y,
linestyle='none',
color='k',
marker='*')
```



SIMPLE PLOTTING BASICS

Much of your power is in the plot command:

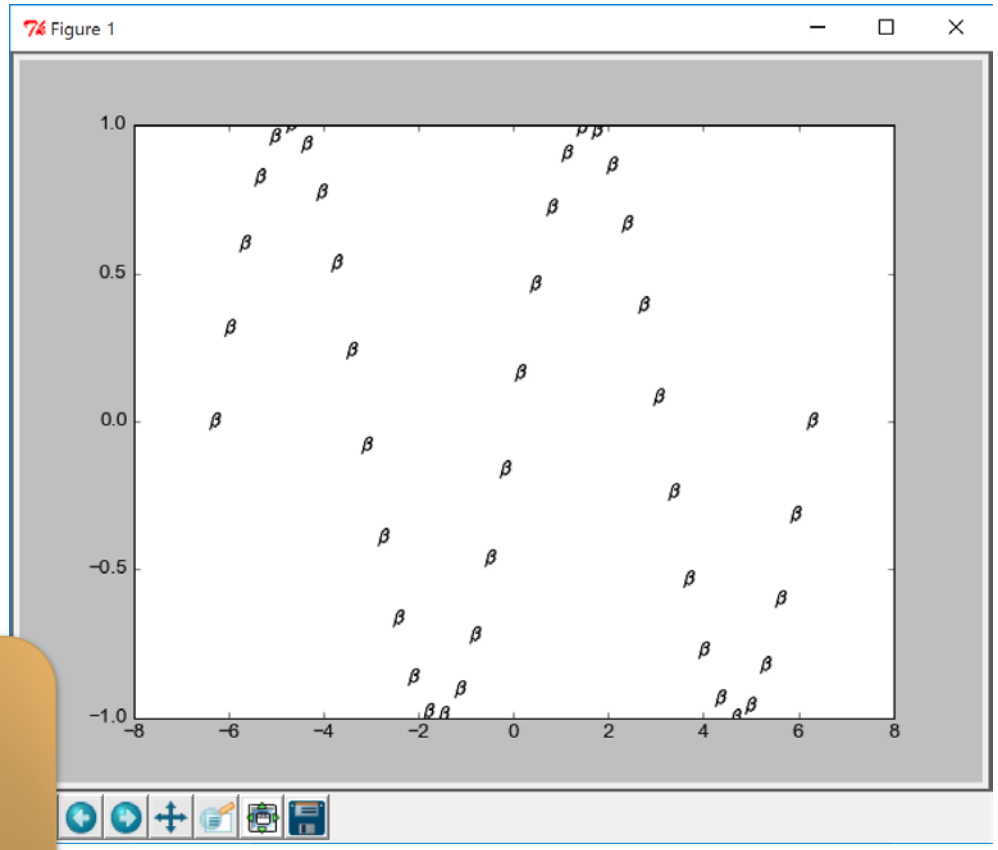
```
plt.plot(x, y,
linestyle='none',
color='k',
marker='$\\beta$',
markersize=10)
```



SIMPLE PLOTTING BASICS

Much of your power is in the plot command:

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



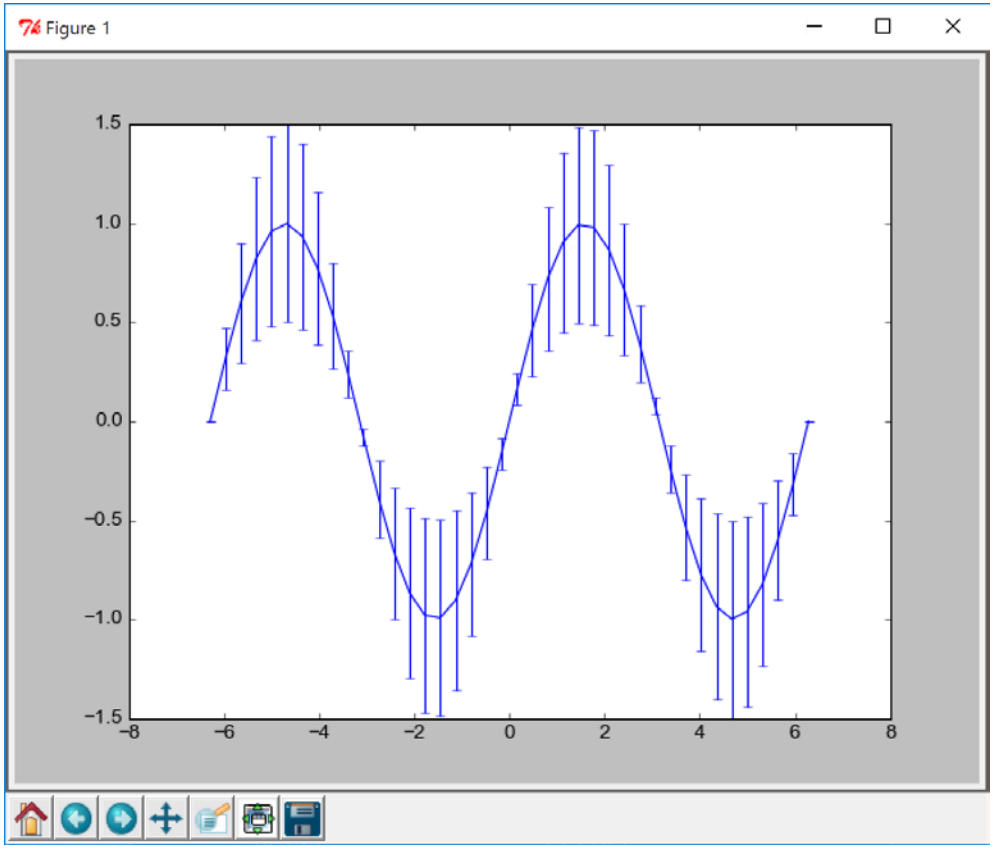
PRO TIP:

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

SIMPLE PLOTTING BASICS

Creating error bars:

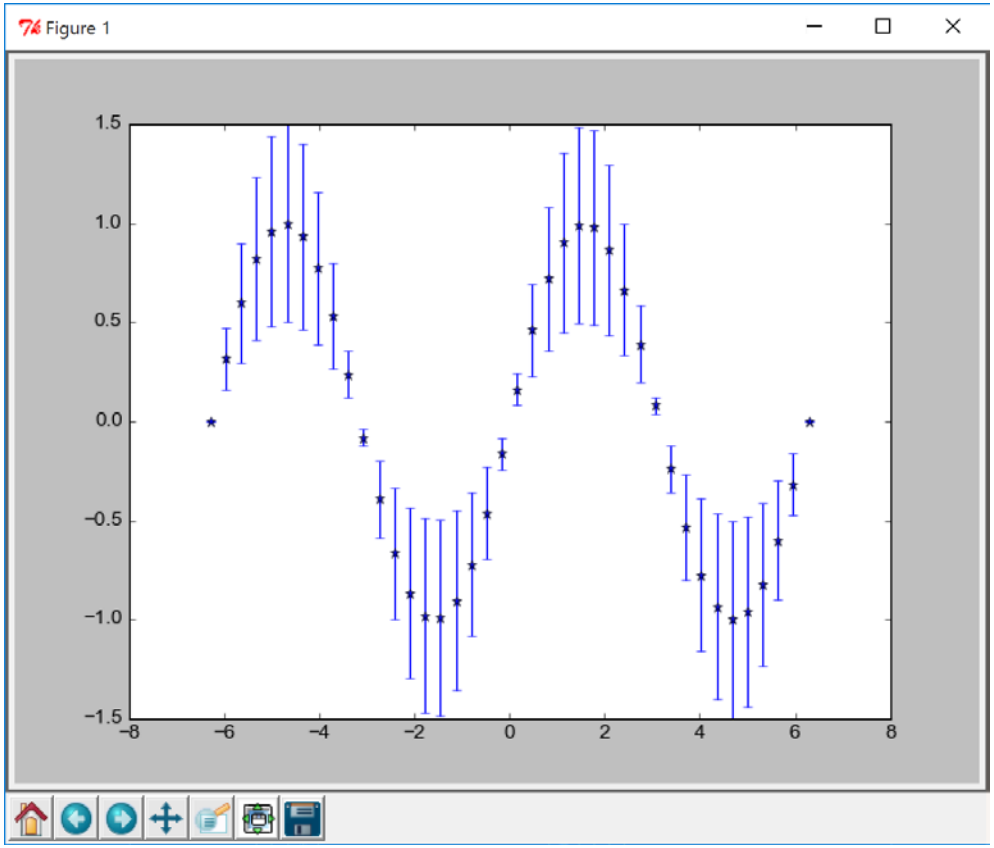
```
plt.errorbar(x, y,
yerr=yerr)
```



SIMPLE PLOTTING BASICS

Creating error bars:

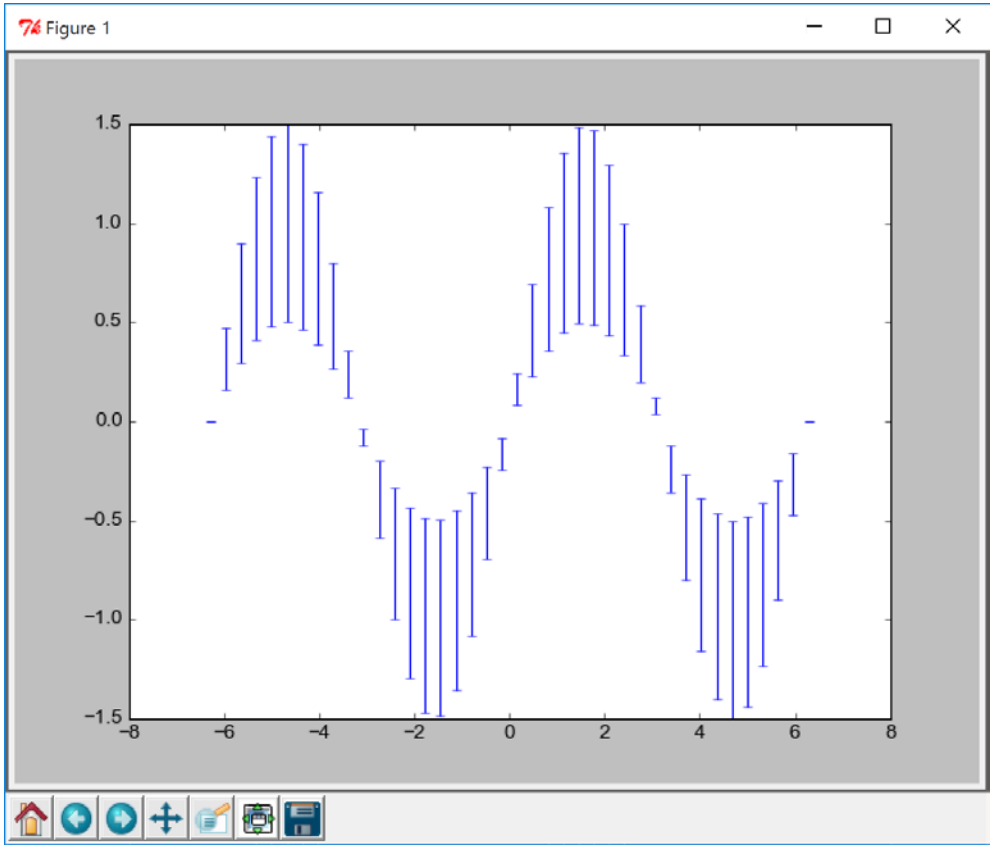
```
plt.errorbar(x, y,
yerr=yerr, fmt='*')
```



SIMPLE PLOTTING BASICS

Creating error bars:

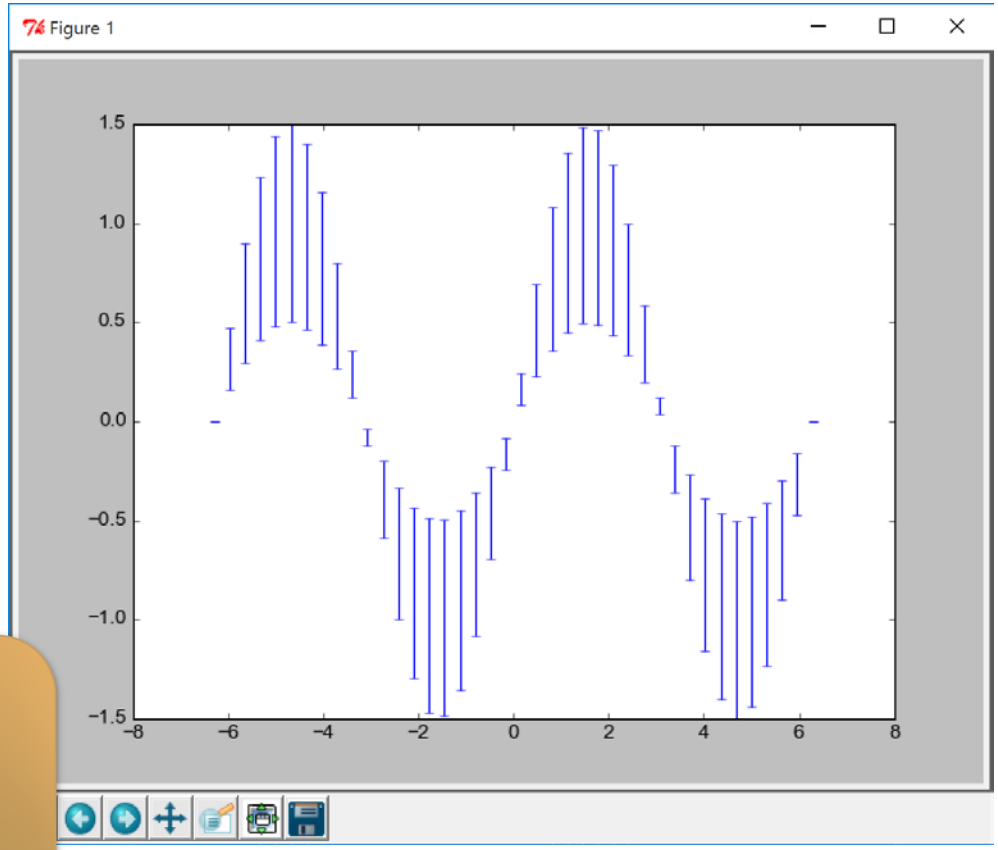
```
plt.errorbar(x, y,
yerr=yerr,
fmt='none')
```



SIMPLE PLOTTING BASICS

Creating error bars:

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



PRO TIP:

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

COLOURS 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:

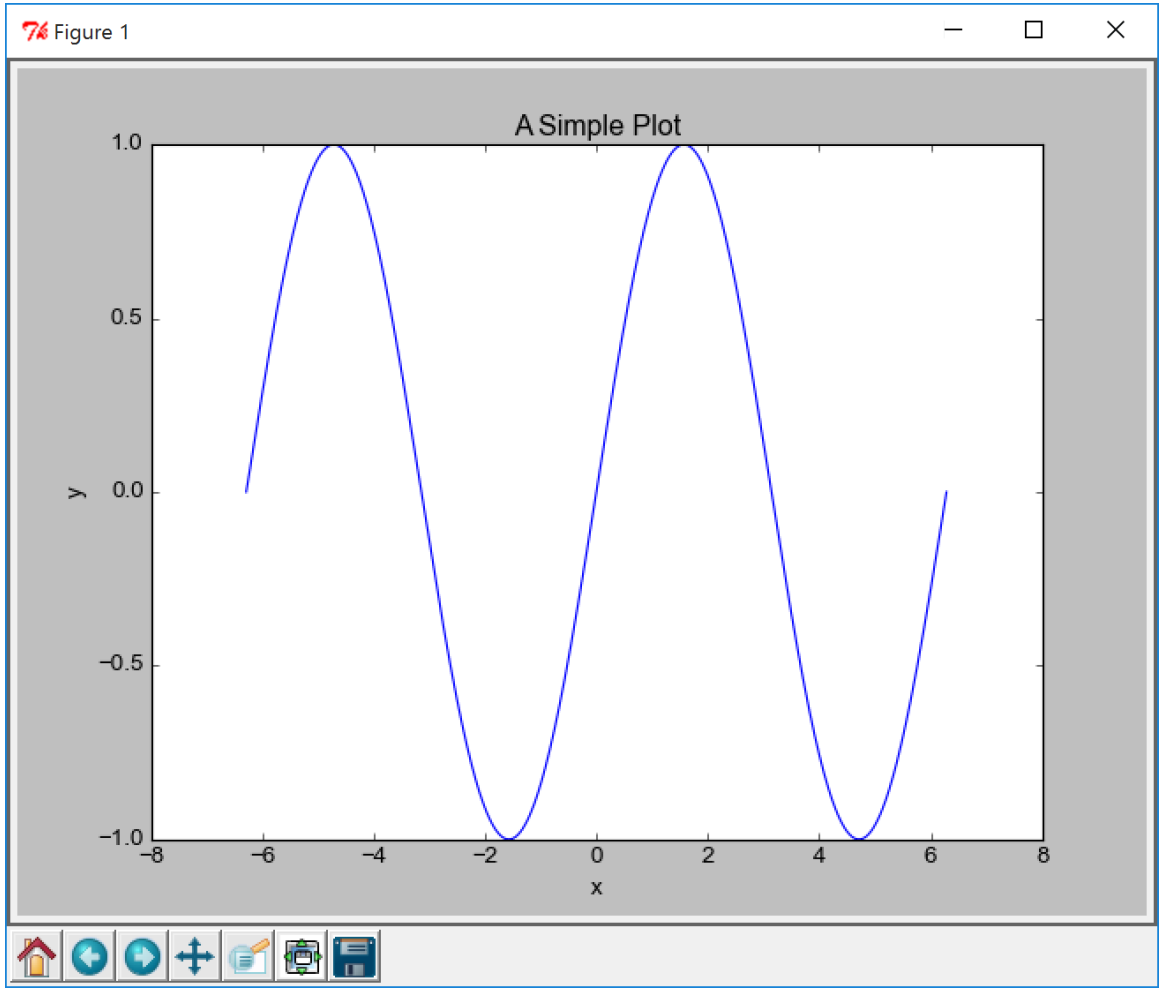
"deeppink"
"slateblue"
"ivory"
"lemonchiffon"

Hex code

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

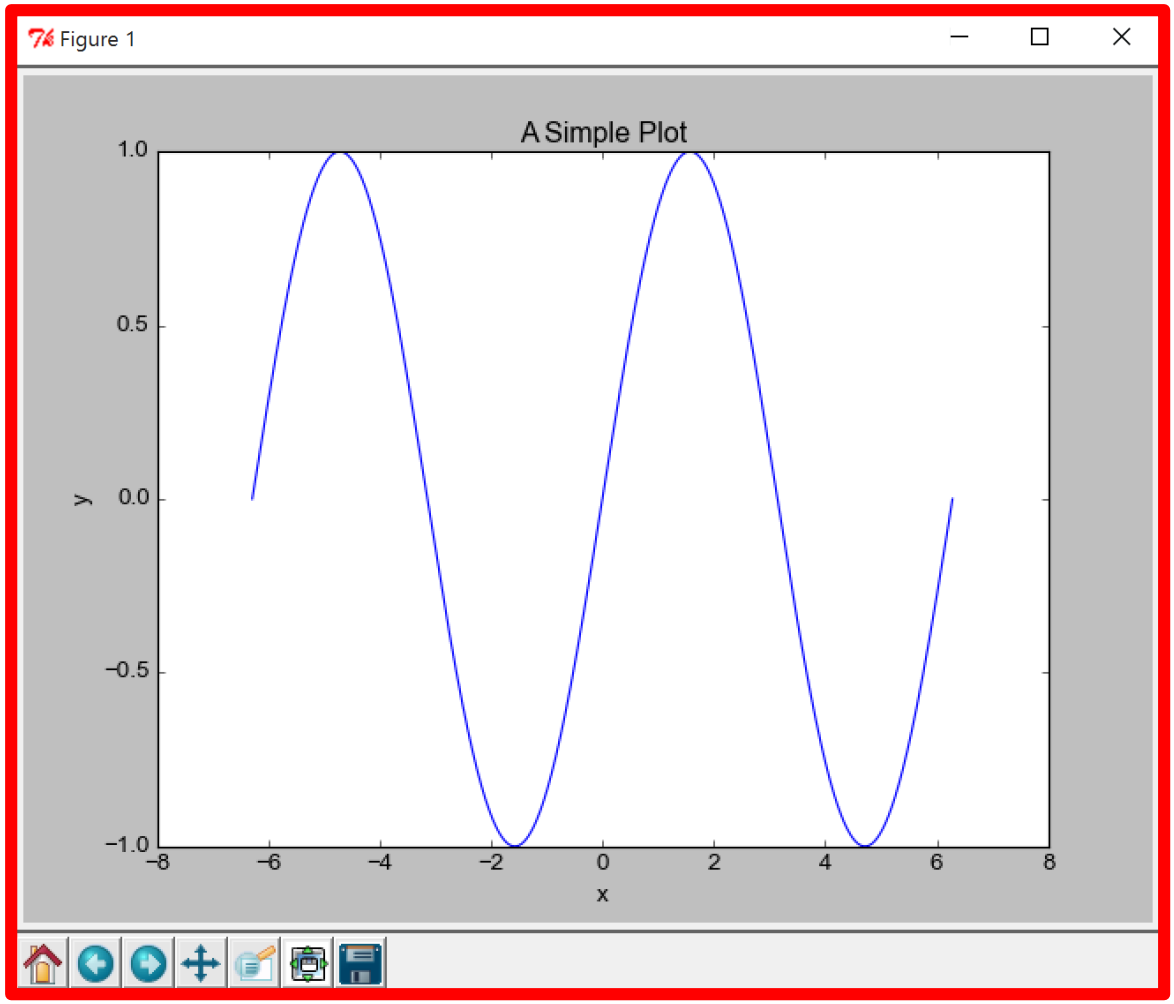
"#ffffff"
"#000000"
"#ff0000"
"#ff00ff"

ANATOMY OF A PLOT WINDOW



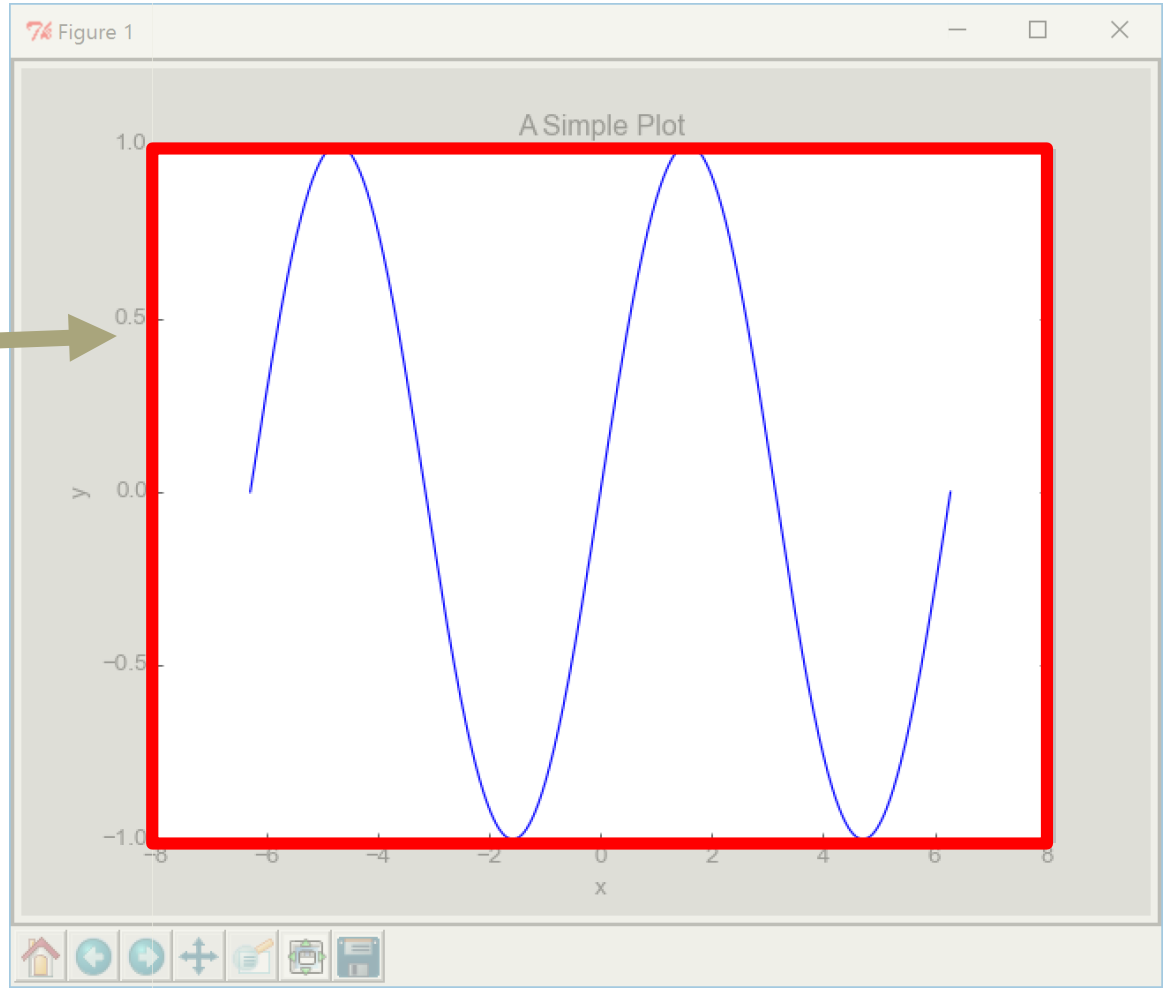
ANATOMY OF A PLOT WINDOW

Figure

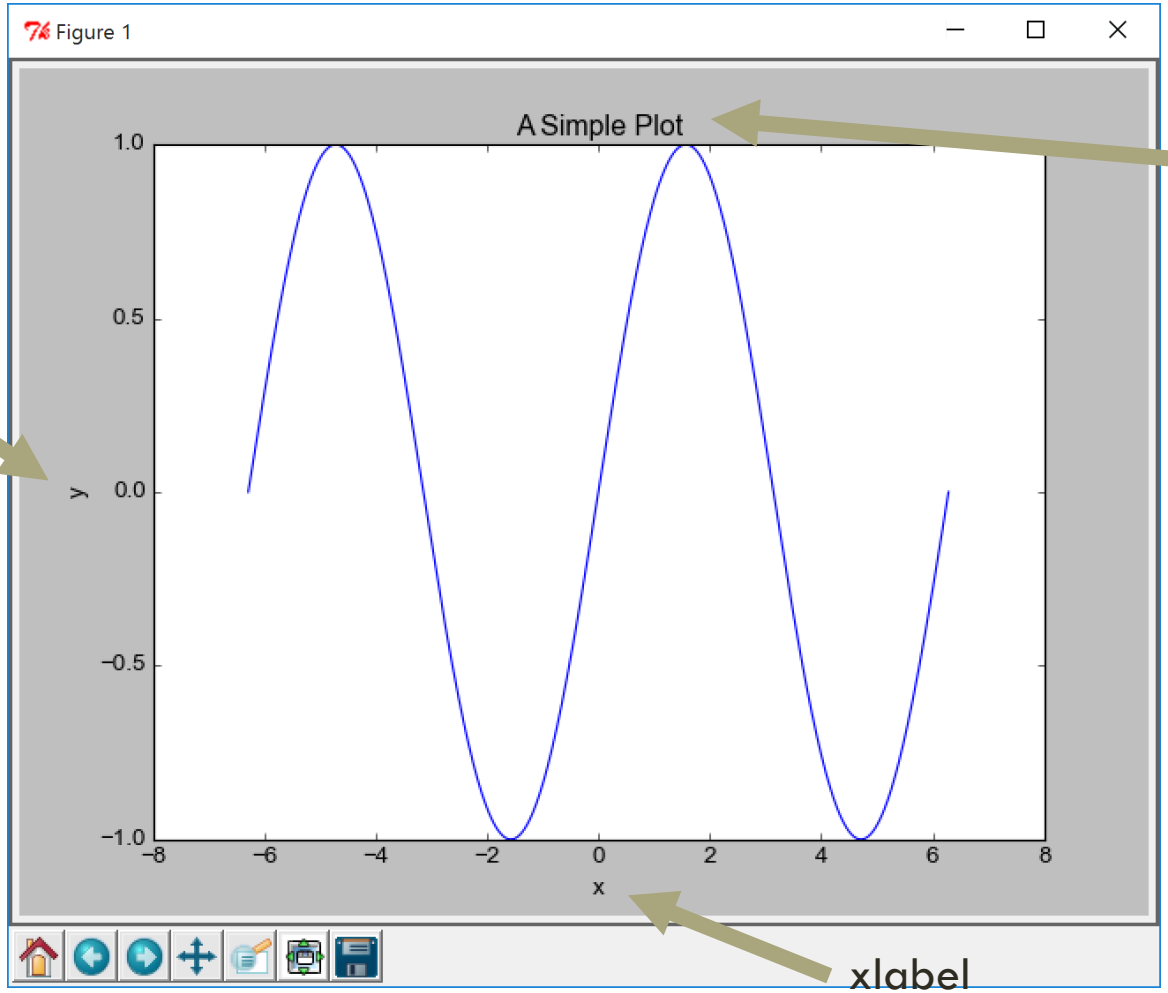


ANATOMY OF A PLOT WINDOW

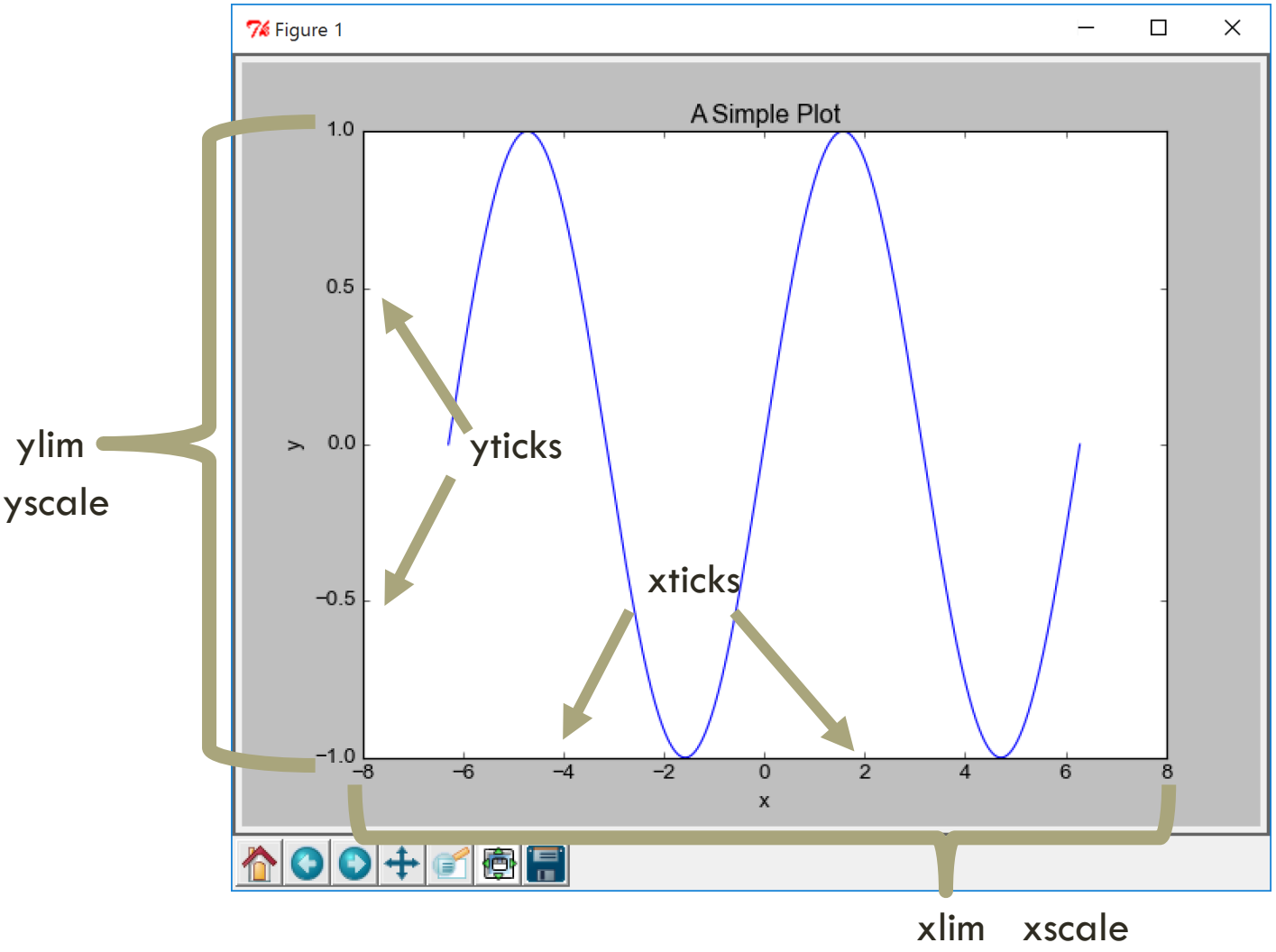
Axis



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

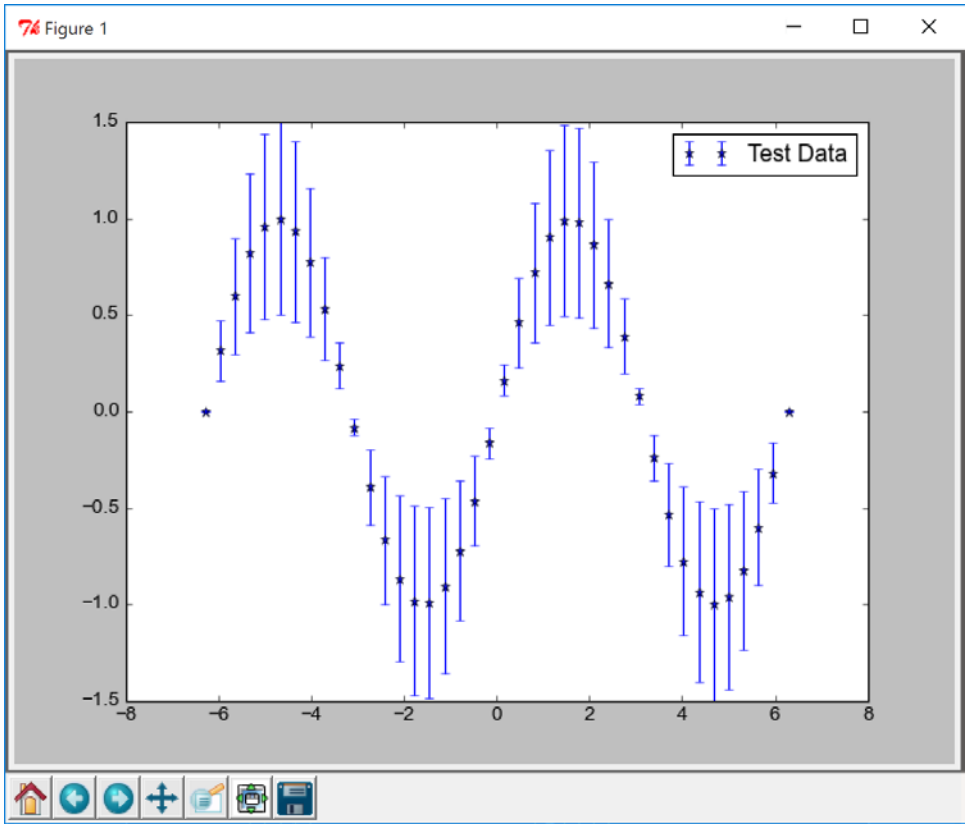
You can (at any time in the plotting) change the range (lim), scale (log or linear), labels or ticks on a plot. Replace x with y (or vice versa) when necessary:

```
# 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, 4, 10, 19]) # Location of x-ticks
```

LABELS AND LEGENDS (OH MY!)

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

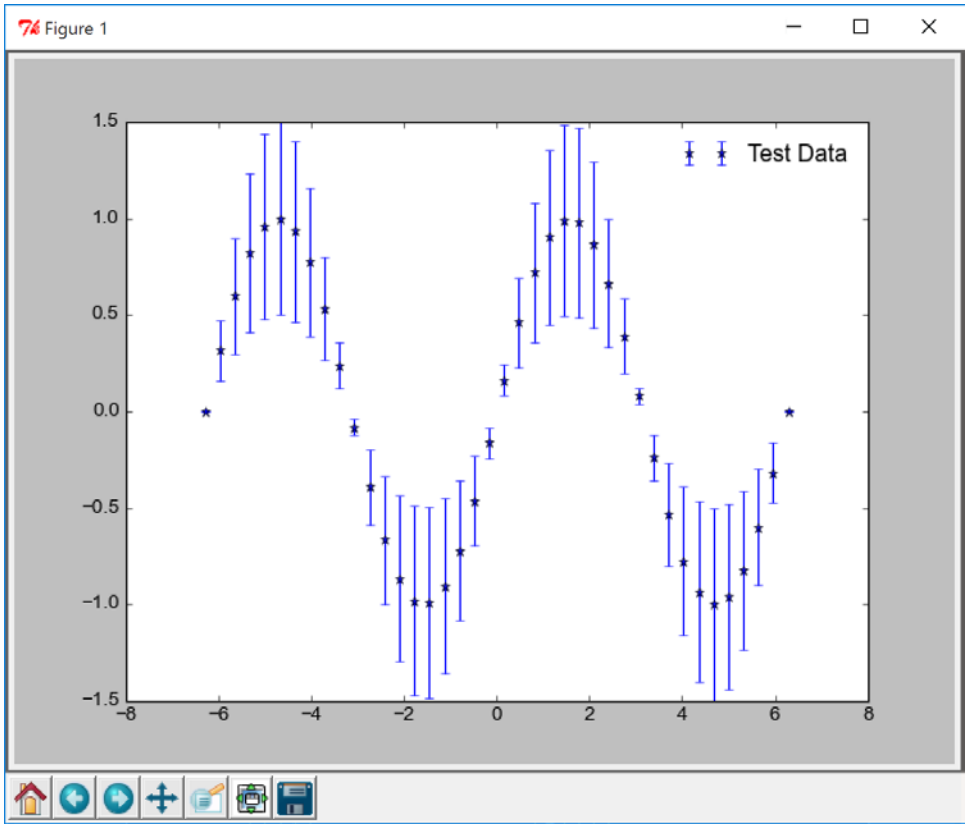
```
plt.errorbar(…,
label="Test Data")

plt.legend()
```



LABELS AND LEGENDS (OH MY!)

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

```
plt.errorbar(…,
label="Test Data")

plt.legend(
frameon=False
)
```



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

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

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("filename.pdf") # Saving as a PDF
plt.savefig("filename.png") # Saving as a PNG
plt.savefig("filename.eps") # Saving as an EPS

# Can also determine what output DPI:
plt.savefig("filename.jpg", dpi
```

PRO TIP:

EPS files do not support transparency natively

BUILDING FROM THE GROUND UP

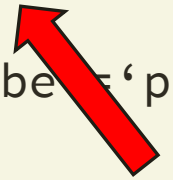
This method gives you a lot more flexibility. Instead of using convenience functions, you use methods on each of the objects:

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

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

ax1.set_xlim([1, 10])
ax1.set_ylim([0, 5])

ax1.set_xscale('log')
ax1.set_xtitle('X Label')
ax1.set_ytitle('Y Label')
fig1.savefig(filename)
```

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])

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

ax1.set_xlim([1, 10])
ax1.set_ylim([0, 5])

ax1.set_xscale('log')
ax1.set_xtitle('X Label')
ax1.set_ytitle('Y Label')
fig1.savefig(filename)
```



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

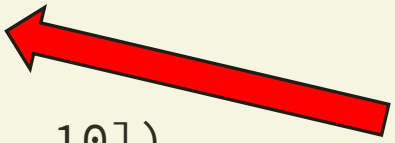
[left, bottom, width, height]

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])

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

ax1.set_xlim([1, 10])
ax1.set_ylim([0, 5])

ax1.set_xscale('log')
ax1.set_xtitle('X Label')
ax1.set_ytitle('Y Label')
fig1.savefig(filename)
```



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])

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

ax1.set_xlim([1, 10])
ax1.set_ylim([0, 5])

ax1.set_xscale('log')
ax1.set_xtitle('X Label')
ax1.set_ytitle('Y Label')
fig1.savefig(filename)
```

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])

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

ax1.set_xlim([1, 10])
ax1.set_ylim([0, 5])

ax1.set_xscale('log')
ax1.set_xtitle('X Label')
ax1.set_ytitle('Y Label')
fig1.savefig(filename)
```

Saving the figure is a method of the figure itself

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])

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

ax1.set_xlim([1, 10])
ax1.set_ylim([0, 5])

ax1.set_xscale('log')
ax1.set_xtitle('X Label')
ax1.set_ytitle('Y Label')
fig1.savefig(filename)
```

PRO TIP:

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)

CUSTOMIZING DEFAULTS

There's a lot of different parameters that matplotlib chooses by default, but you can set your own using a **mat**
will not exist by default, but you can downlo

http://matplotlib.org/_static/matplotlibrc

The place to put this file depends on your pl

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

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

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

PRO TIP:

The default matplotlib font is a crime against typography. Change it as soon as you can.

If you want to replace it with an open source font, may I suggest either **Open Sans** or **Source Sans Pro**?

EXERCISE TIME!

It is I, Captain Vegetable.
With my carrot, and my celery.