# ESCI 386 – Scientific Programming, Visualization and Analysis with Python

## Lesson 11 - 1D Plotting with Matplotlib

# `matplotlib` Overview

- `matplotlib` is a module that contains classes and functions for creating MATLAB-style graphics and plots.

- The primary submodule we will use is `pyplot`, which is commonly aliased as `plt`.

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

# Figure and Axis Objects

- A *figure object* can essentially be thought of as the ‘virtual page’ or ‘virtual piece of paper’ that defines the canvas on which the plot appears.

- The *axes object* is the set of axes (usually x and y) that the data are plotted on.

# Simple Plots with `pyplot.plot()`

- The quickest way to generate a simple, 1-D plot is using the `pyplot.plot()` function.

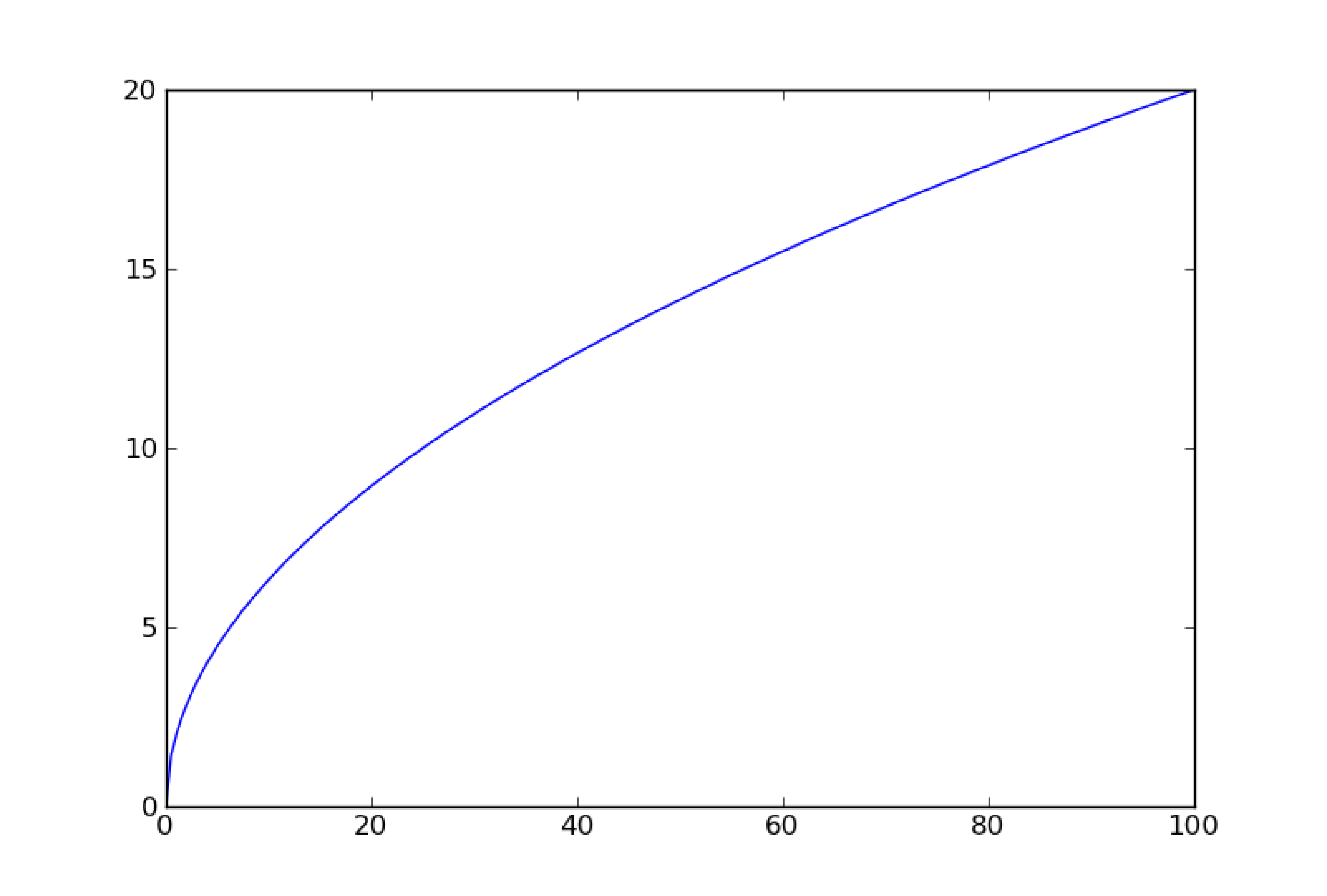
- `pyplot.plot()` automatically creates both the figure and axis objects and plots the data onto them.

# Simple Plot Example

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

x = np.arange(0,100.5,0.5) # Generate x-axis values

y = 2.0*np.sqrt(x)  # Calculate y-values

plt.plot(x,y)   # Create figure and axis objects

plt.show()  # Display plot to screen
```

File:  simple-plot.py

# Simple Plot Result



# Alternate to Using `pyplot.plot()`

- The `pyplot.plot()` function generates the figure and axes objects automatically, and is the simplest way to create a plot.

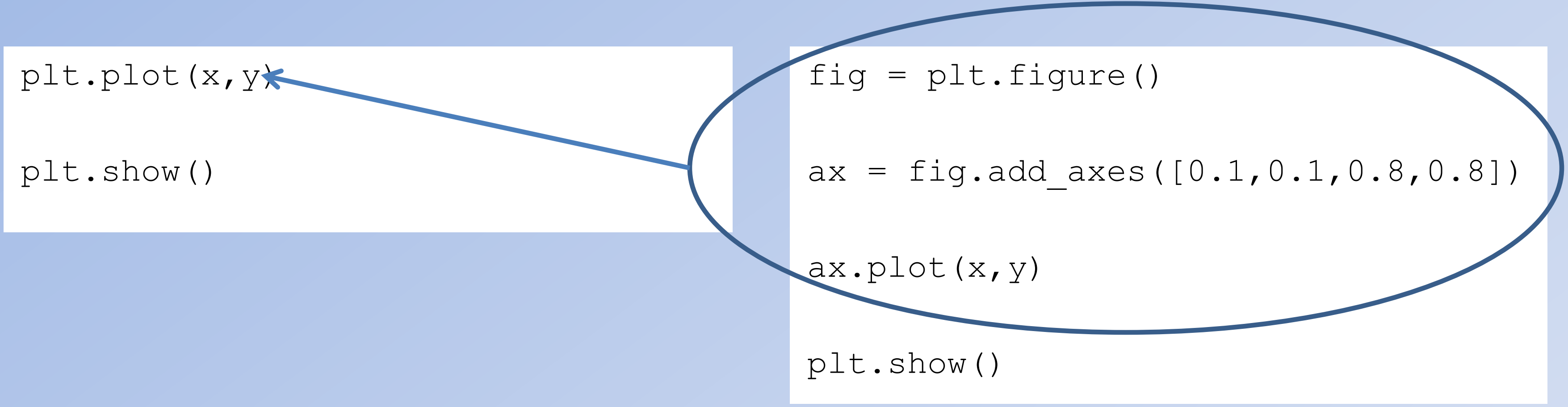
- For more control over axes placement we can use the `pyplot.figure()` function to generate the figure and then use the `add_axes()` method to create the axes.

# Example Using `pyplot.figure()` and `add_axes()` method

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

x = np.arange(0,100.5,0.5) # Generate x-axis values

y = 2.0*np.sqrt(x)  # Calculate y-values

fig = plt.figure()  # Create figure

ax = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # Create axes

ax.plot(x,y)   # Plot data on axes

plt.show()  # Display plot to screen
```

File:  simple-plot-alternate.py

# Comparison of Two Approaches



```
plt.plot(x,y)

plt.show()
```

```
fig = plt.figure()

ax = fig.add_axes([0.1,0.1,0.8,0.8])

ax.plot(x,y)

plt.show()
```

- The `plt.plot()` function performs the three steps shown in the elipse.

# `plot()` is Not the Same in Each Approach

```
plt.plot(x,y)

plt.show()
```

- Here `plot()` is a function within the `pyplot` module.

```
fig = plt.figure()

ax = fig.add_axes([0.1,0.1,0.8,0.8])

ax.plot(x,y)

plt.show()
```

- Here `plot()` is a method belonging to an axes object named `ax`.

# When to use `pyplot.plot()` versus `pyplot.figure()` and `figure.add_axes()`

- Most plotting with only a single set of axes can be accomplished using the `pyplot.plot()` function.

- For plots with multiple axes, or when detailed control of axes placement is required, then the `pyplot.figure()` and `figure.add_axes()` methods, or similar methods, are needed.

# Getting References to the Current Figure and Axes

- A reference to the current figure can always be obtained by using the `pyplot.gcf()` function.
  - Example: `fig = plt.gcf()`

- Likewise, a reference to the current axes is obtained using `pyplot.gca()`:
  - Example: `ax = plt.gca()`

# Plotting Multiple Lines on a Single Set of Axes

- Multiple lines are plotted by repeated use of the `pyplot.plot()` function or `axes.plot()` method.

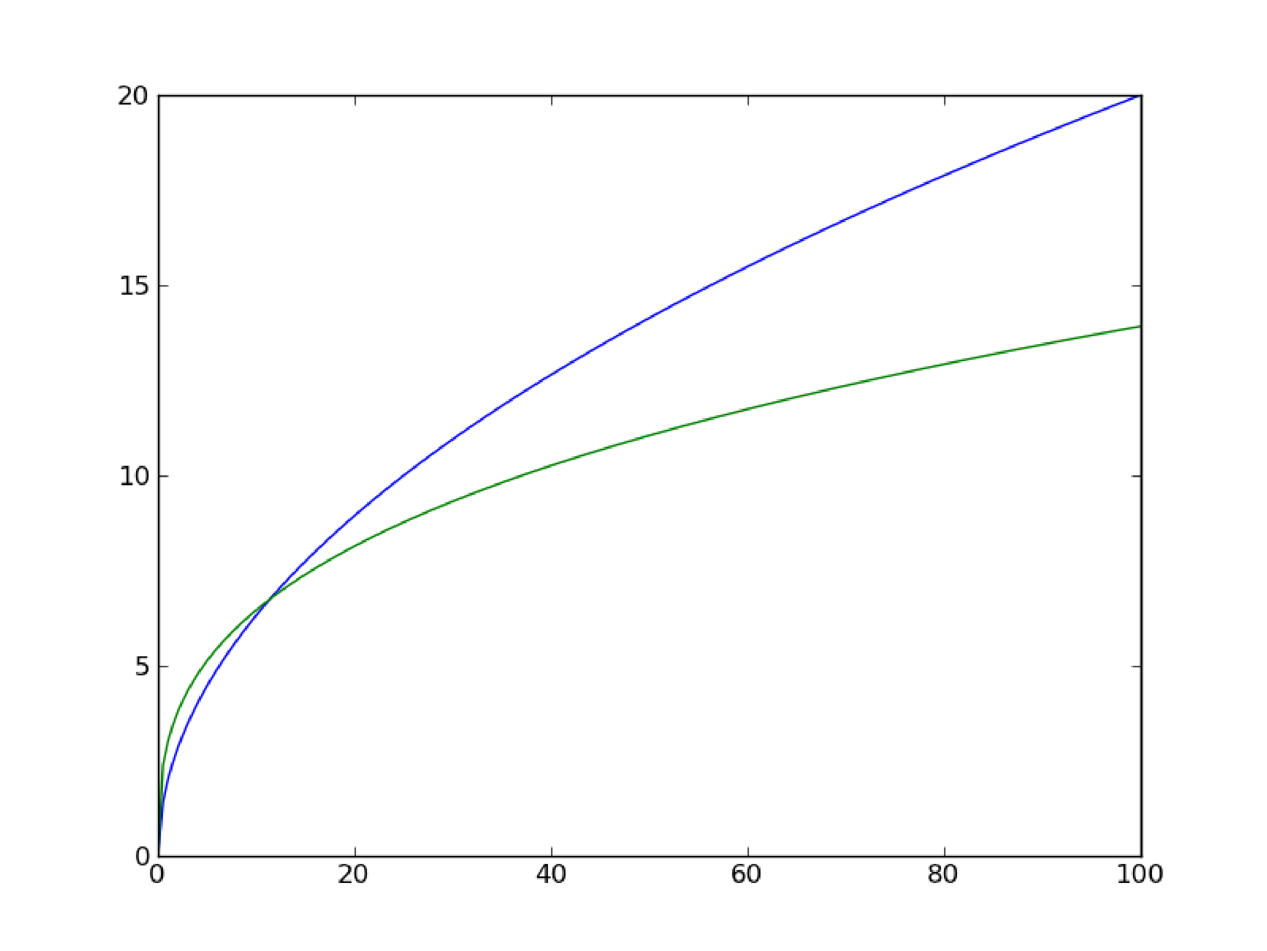
- The color of each set of lines will automatically be different.

# Plotting Multiple Lines Example

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

x = np.arange(0,100.5,0.5) # Generate x-axis values

y1 = 2.0*np.sqrt(x)  # Calculate y1 values
y2 = 3.0*x**(1.0/3.0)  # Calculate y2 values

plt.plot(x,y1)   # Plot y1
plt.plot(x,y2)   # Plot y2

plt.show()  # Display plot to screen
```

File:  two-lines-plot.py

# Plot Multiple Lines Result



# Plotting Multiple Lines-Alternate

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

x = np.arange(0,100.5,0.5) # Generate x-axis values

y1 = 2.0*np.sqrt(x)  # Calculate y1 values
y2 = 3.0*x**(1.0/3.0)  # Calculate y2 values

fig = plt.figure()  #  Create figure

ax = fig.add_axes([0.1, 0.1, 0.8, 0.8])  # Create axes

ax.plot(x,y1)   # Plot y1
ax.plot(x,y2)   # Plot y2

plt.show()  # Display plot to screen
```

File:  two-lines-alternate.py

# Keyword for Line Colors

| Keyword | Purpose | Values |
|---|---|---|
| `color` or `c` | Controls color of plotted line | Any valid `matplotlib` color |

# Colors

- Colors are specified using the names of the basic built-in colors or their single-letter abbreviations:
  - `‘b’, ‘blue’`
  - `‘g’, ‘green’`
  - `‘r’, ‘red’`
  - `‘c’, ‘cyan’`
  - `‘m’, ‘magenta’`
  - `‘y’, ‘yellow’`
  - `‘k’, ‘black’`
  - `‘w’, ‘white’`

# Colors

- Colors can also be specified using HTML color names or their hexadecimal representation (e.g., `‘aquamarine’` or `‘#7FFFD4’`).
  - There are 167 of these, so they are not listed here. They can be found in references on the web.

# Colors

- Gray shades can also be represented using a floating-point number between 0.0 and 1.0 represented as a string.
  - `‘0.0’` is black, `‘1.0’` is white, and any number in between (e.g., `‘0.3’`) will represent different shades of gray.

# Keywords for Line Colors and Styles

| Keyword | Purpose | Values |
|---|---|---|
| `linestyle or ls` | Controls style of plotted line | solid `ls = '-'`<br>dashed `ls = '--'`<br>dash-dot `ls = '-.'`<br>dotted `ls = ':'`<br>no line `ls = 'None'` |
| `linewidth or lw` | Controls width of plotted line | Point value such as `1`, `2`, `3`, etc. |

# Keyword for Marker Styles

| Keyword | Purpose | Values |
|---|---|---|
| marker | Controls marker style | circle : marker = 'o'<br>diamond: marker = 'D'<br>thin diamond: marker = 'd'<br>no marker: marker = 'None'<br>+: marker = '+'<br>x: marker = 'x'<br>point: marker = '.'<br>square: marker = 's'<br>star: marker = '*'<br>triangle down: marker = 'v'<br>triangle up: marker = '^'<br>triangle left: marker = '<'<br>triangle right: marker = '>'<br>pentagon: marker = 'p'<br>hexagon: marker = 'h' or 'H'<br>octagon: marker = '8'<br>down-caret: marker = '7'<br>left-caret: marker = '4'<br>right-caret: marker = '5'<br>up-caret: marker = '6'<br>horizontal line: marker = '_'<br>vertical line marker = '\|' |

# Keywords for Markers Properties

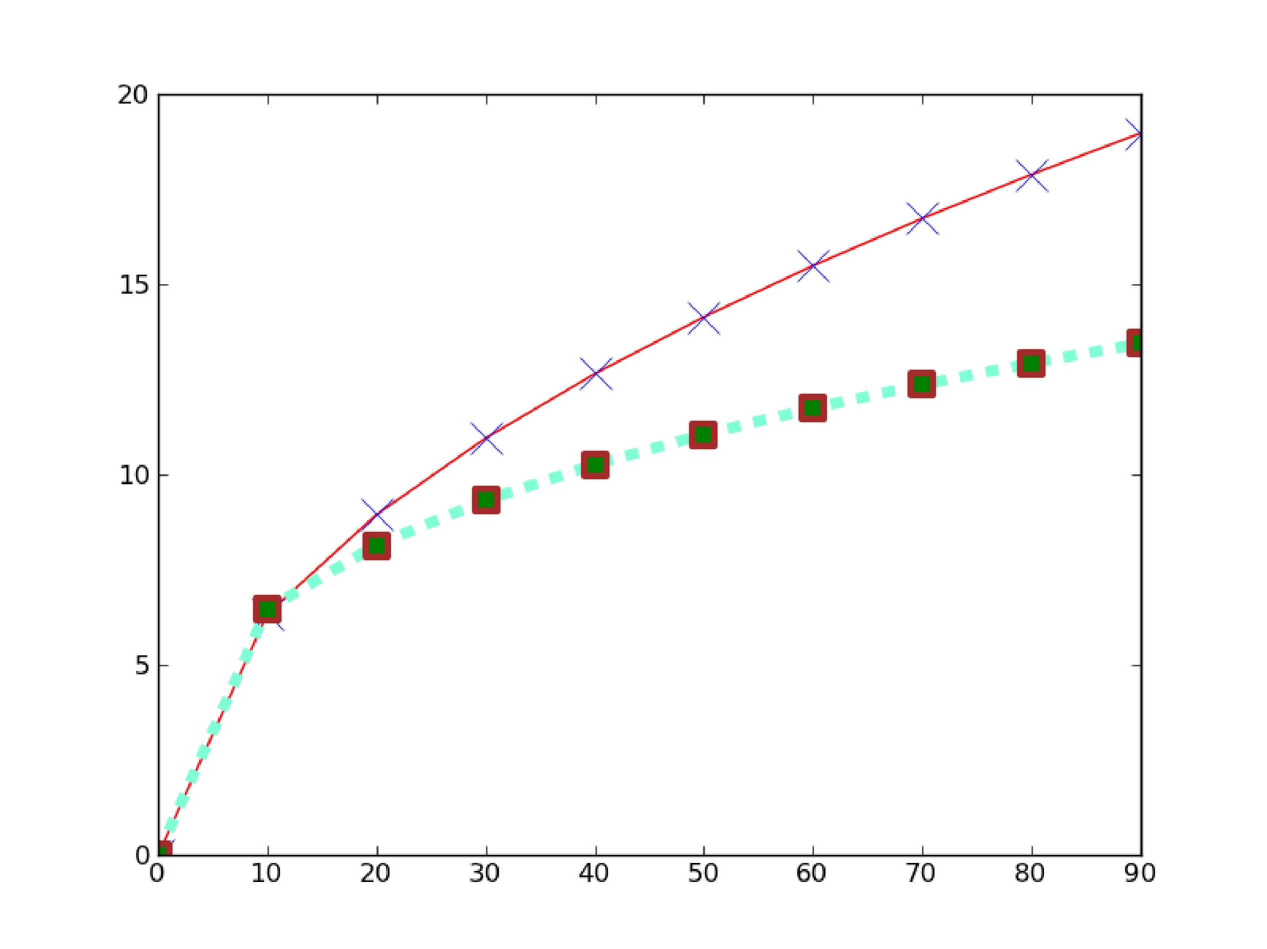
| Keyword | Purpose | Values |
|---|---|---|
| `markersize` or `ms` | Controls size of marker in points. | Point value such as `10`, `14`, `17`, etc. |
| `markeredgecolor` or `mec` | Controls marker outline color | Any valid `matplotlib` color |
| `markeredgewidth` or `mew` | Controls width of marker outline | Point value such as `2`, `3`, `5`, etc. |
| `markerfacecolor` or `mfc` | Controls marker fill color | Any valid `matplotlib` color |
| `label` | A label used to identify the line. This can be used for a legend | Any string. |

# Line Styles Example

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

x = np.arange(0,100,10) # Generate x-axis values

y1 = 2.0*np.sqrt(x)  # Calculate y1 values
y2 = 3.0*x**(1.0/3.0)  # Calculate y2 values

plt.plot(x,y1,c = 'r', ls = '-', marker = 'x', mec = 'blue',
         ms = 15)
plt.plot(x,y2,c = 'aquamarine', ls = '--', lw = 5, marker = 's',
         mec = 'brown', mfc = 'green', mew = 3, ms = 10)

plt.show()  # Display plot to screen
```

File:  line-styles-example.py

# Line Styles Result



# Shortcuts for Line Styles and Colors

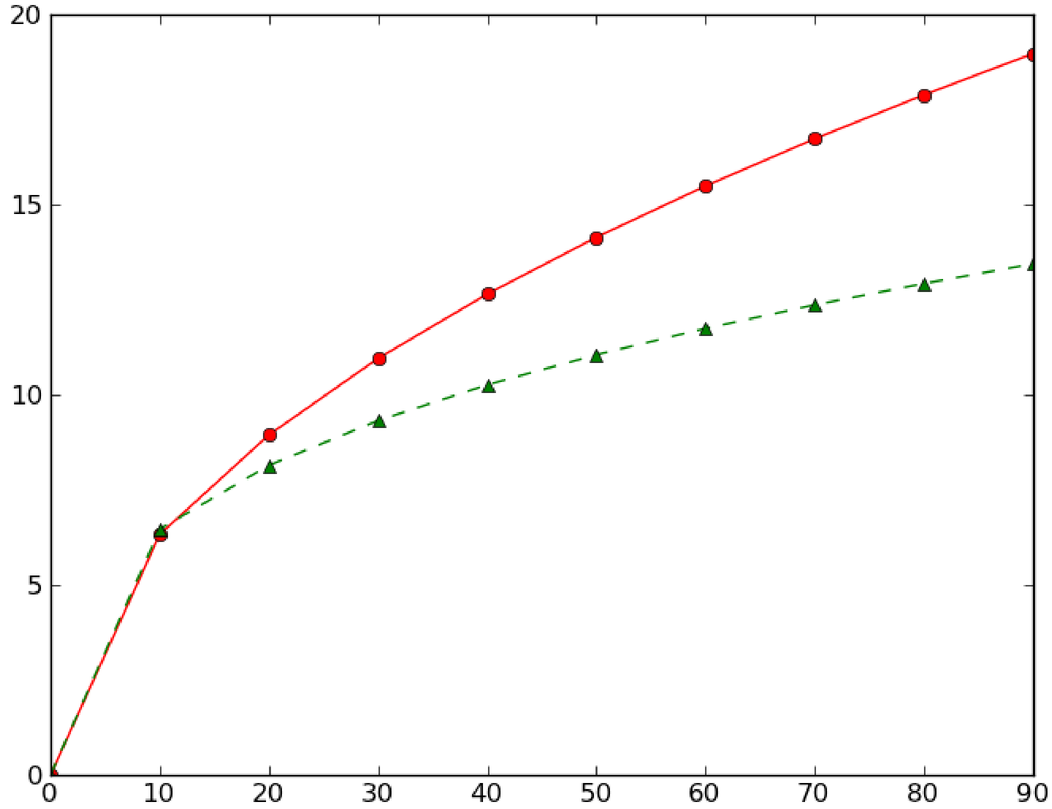
- A shortcut for quickly specifying line colors, styles, and markers is shown in the following example.

# Shortcuts Example

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

x = np.arange(0,100,10) # Generate x-axis values

y1 = 2.0*np.sqrt(x)  # Calculate y1 values
y2 = 3.0*x**(1.0/3.0)  # Calculate y2 values

plt.plot(x,y1,'r-o')
plt.plot(x,y2,'g--^')

plt.show()  # Display plot to screen
```

File: line-styles-shortcut.py

# Shortcuts Result



# Logarithmic Plots

- Logarithmic plots are made by using the following in place of `plot()`:
  - `semilogx()` creates a logarithmic x axis.
  - `semilogy()` creates a logarithmic y axis.
  - `loglog()` creates both x and y logarithmic axe

- Linestyles and colors are controled the same way as with the `plot()` function/method.

# Plot Titles

- Use
  - `pyplot.title(‘title string’)`
  - `axes.title(‘title string’)`

- Control size with the size keyword
  - Options are:

    `‘xx-small’, ‘x-small’, ‘small’, ‘medium’, ‘large’, ‘x-large’, ‘xx-large’,` or a numerical font size in points.

# Axes Labels

- Pyplot functions:
  - `pyplot.xlabel(‘label string’)`
  - `pyplot.ylabel(‘label string’)`

- Axes methods:
  - `axes.set_xlabel(‘label string’)`
  - `axes.set_ylabel(‘label string’)`

- Also accepts `size` keyword.

# Keywords for Axes Labels

- size = *same as for plot titles*
- horizontalalignment = [ ‘center’ | ‘right’ | ‘left’ ]
- verticalalignment = [ ‘center’ | ‘top’ | ‘bottom’ | ‘baseline’ ]
- rotation = [ *angle in degrees* | ‘vertical’ | ‘horizontal’ ]
- color = *any matplotlib color*

# Including Greek Characters and Math Functions in Titles/Labels

- Uses LaTeX-like markup syntax.
- The mathematical text needs to be included within dollar signs ($).
- Use raw strings (`r’string’`) when using mathematical text.
- Items contained within curly braces {} are grouped together.
- Subscripts and superscripts are denoted by the ‘^’ and ‘_’ characters.
- Spaces are inserted by backslash-slash `'\/’`
- Greek letters are rendered by a backslash followed by the name of the letter.

# Including Greek Characters and Math Functions in Titles/Labels

- Some examples:
  - `r'$x^{10}$'` >> $x^{10}$
  - `r'$R_{final}$'` >> $R_{final}$
  - `r'$\alpha^{\eta}$'` >> $\alpha^{\eta}$
  - `r'$\sqrt{x}$'` >> $\sqrt{x}$
  - `r'$\sqrt[3]{x}$'` >> $\sqrt[3]{x}$

- An online tutorial for writing mathematical expressions can be found at [*http://matplotlib.sourceforge.net/users/mathtext.html#mathtext-tutorial*](http://matplotlib.sourceforge.net/users/mathtext.html#mathtext-tutorial)

# Controlling Axes Limits

- pyplot functions
  - `xlim(mn, mx)`
  - `ylim(mn, mx)`

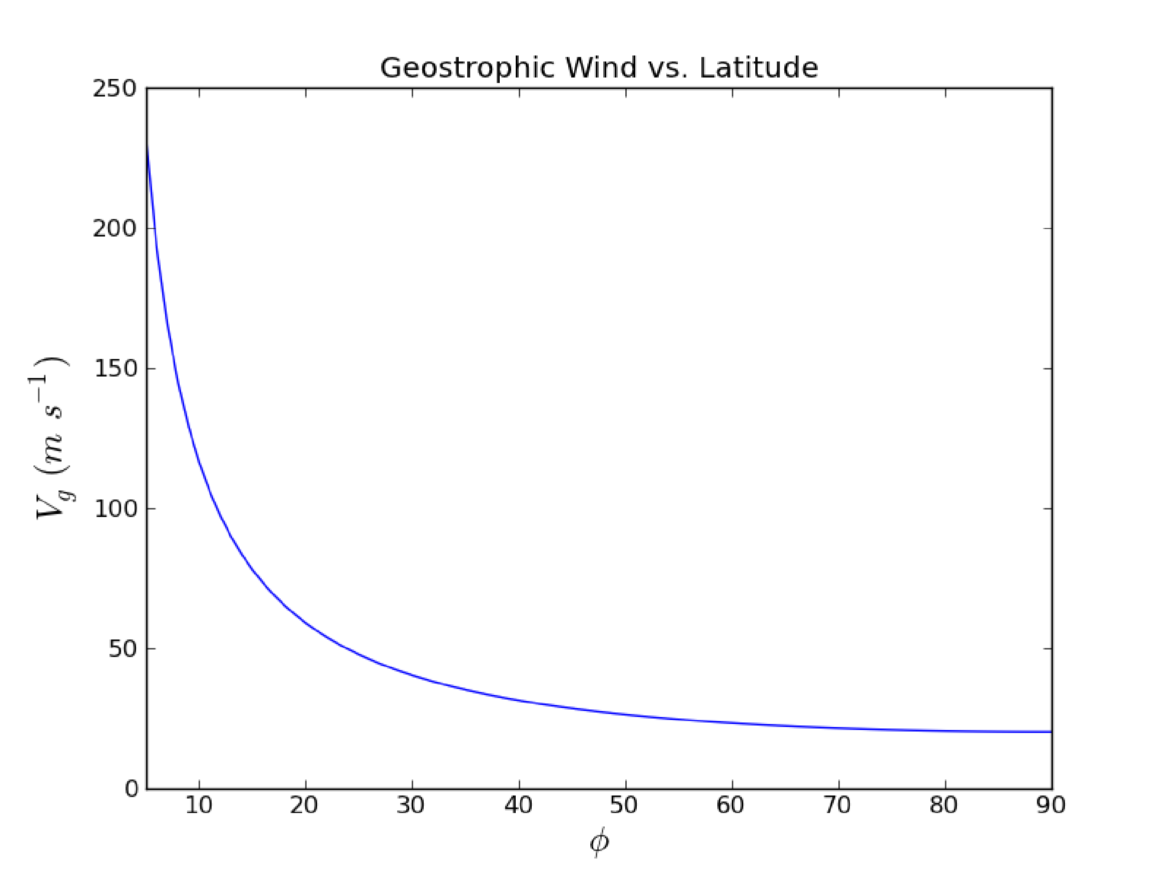
- axes methods
  - `set_xlim(mn, mx)`
  - `set_ylim(mn, mx)`

- *mn* and *mx* are the lower and upper limits of the axis range.

# Plotting Exercise #1

- $g$ = 9.81 m/s$^2$
- $\Omega$ = 7.292e-5 rad/s
- $f = 2\Omega\sin\phi$
- $\Delta Z$ = 60 m
- $\Delta n$ = 2e5 m

$$\boxed{V_g \cong \frac{g_0}{f}\frac{\Delta Z}{\Delta n}}$$



# Controlling Axes Tick Mark Locations and Labels

- pyplot functions
  - `xticks(`*`loc`*`, `*`lab`*`)`
  - `yticks(`*`loc`*`, `*`lab`*`)`

- axes methods
  - `set_xticks(`*`loc`*`)` and `set_xticklabels(`*`lab`*`)`
  - `set_yticks(`*`loc`*`)` and `yticklabels(`*`lab`*`)`

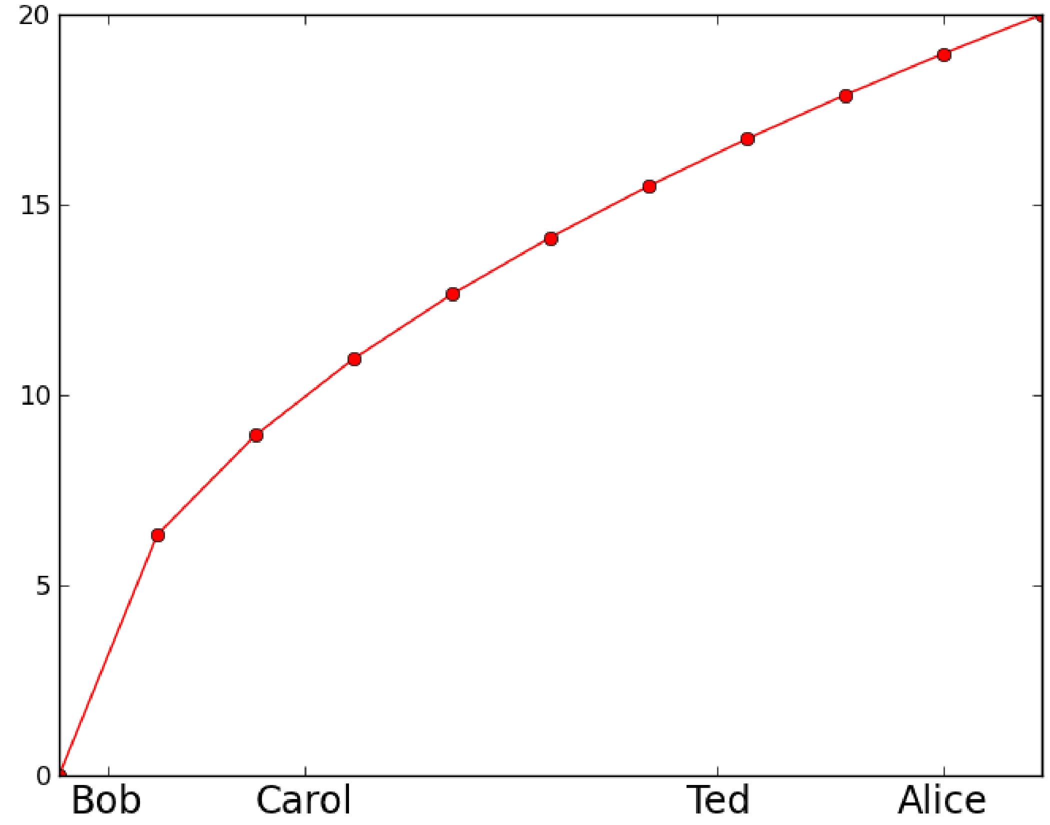
- In these functions/methods the arguments are:
  - *`loc`* is a list or tuple containing the tick locations
  - *`lab`* an optional list or tuple containing the labels for the tick marks. These may be numbers or strings.
  - *`loc`* and *`lab`* must have the same dimensions
  - Can use `size` keyword for font size

# Tick Mark Example

```
…
loc = (5, 25, 67, 90)
lab = ('Bob', 'Carol', 'Ted', 'Alice')
plt.xticks(loc, lab , size = 'x-large')
plt.show()
```

File:  tick-marks.py

# Tick Mark Results



# Axis Grid Lines

- Grid lines can be added to a plot using the `grid()` pyplot function or axes method.

- The two main keywords are

```
 axis = ‘both’|‘x’|‘y’
 which =  ‘major’|‘minor’|‘both’
```

- There are also other keywords for controlling the line type, color, etc.

# Adding Duplicate Axes

- The `twinx()` and `twiny()` pyplot functions or axes methods make duplicate axes with the y or x ticks located on the opposite side of the plot.

# Duplicate Axes Example

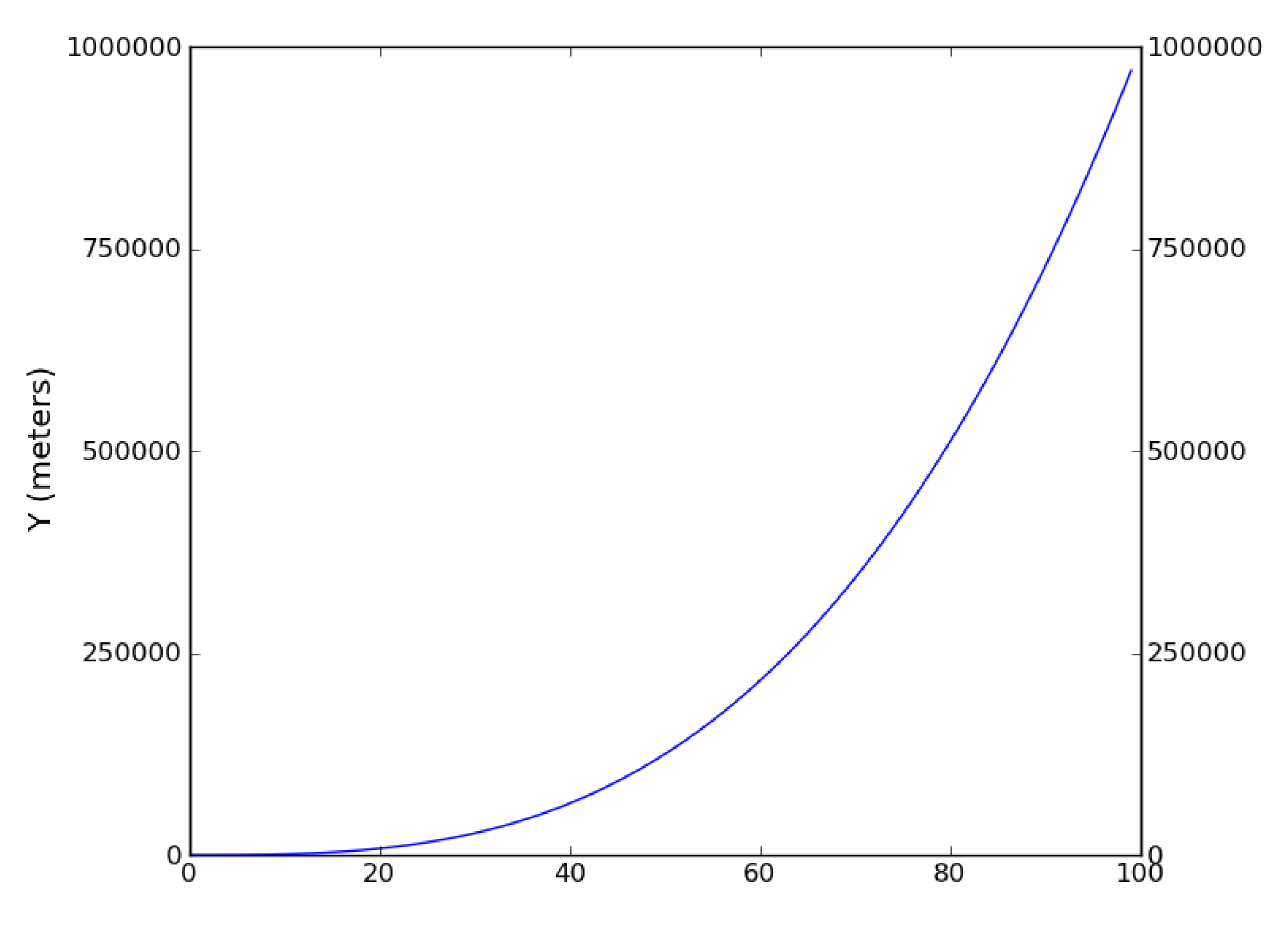
```
f = plt.figure()
a = f.add_axes([0.15, 0.1, 0.75, 0.85])
x = np.arange(0.0,100.0)
y = x**3
a.plot(x,y)
a.set_yticks([0, 250000, 500000, 750000, 1000000])
a.set_ylabel('Y (meters)', size = 'large')

b = plt.twinx(a)

b.set_yticks(a.get_yticks())
plt.show()
```

Creates duplicate axes

Sets new axes ticks to match original

File: duplicate-axes.py

# Duplicate Axes Results



# Creating Legends

# Creating Legends

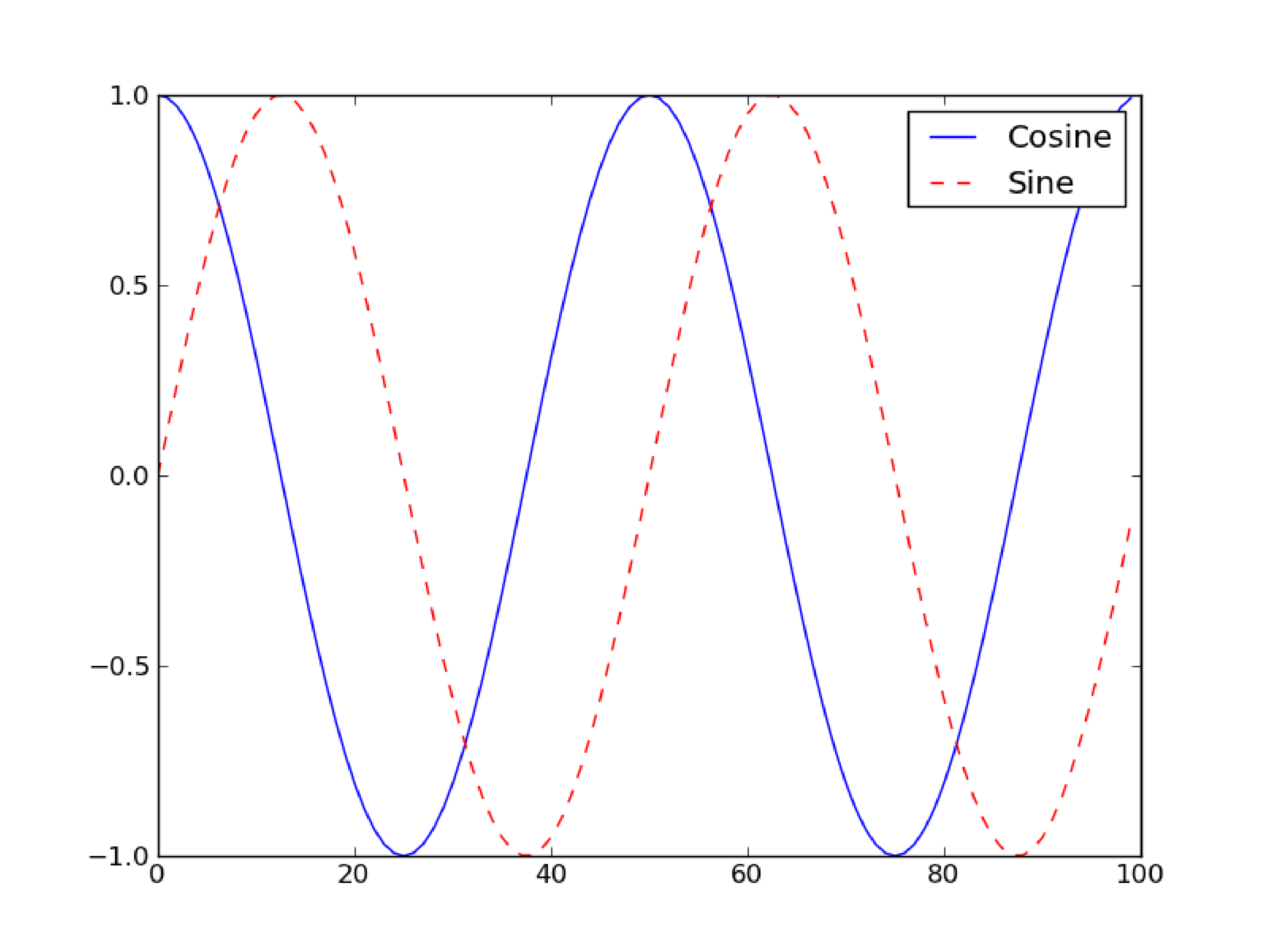
- To create a legend you first need to give each plotted line a label, using the label keyword in the `plot()` function/method.
- The label is merely a string describing the line.
- The legend is then created using the `pyplot.legend()` function or `axes.legend()` method.

# Legend Example

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

x = np.arange(0,100.0)
y1 = np.cos(2*np.pi*x/50.0)
y2 = np.sin(2*np.pi*x/50.0)

plt.plot(x, y1, 'b-', label = 'Cosine')
plt.plot(x, y2, 'r--', label = 'Sine')

plt.legend(('Cosine', 'Sine'), loc = 0)

plt.show()  # show plot
```

File:  legend-example.py

# Legend Result



# Values for `loc` Keyword

| Value | Position |
|---|---|
| 0 | best location |
| 1 | upper right |
| 2 | upper left |
| 3 | lower left |
| 4 | lower right |
| 5 | right |
| 6 | center left |
| 7 | center right |
| 8 | lower center |
| 9 | upper center |
| 10 | center |

# Controlling Font Size in Legends

- To control the font sizes in the legends, use the prop keyword as shown below.

```
ax.legend(loc=0,prop=dict(size=12))
```

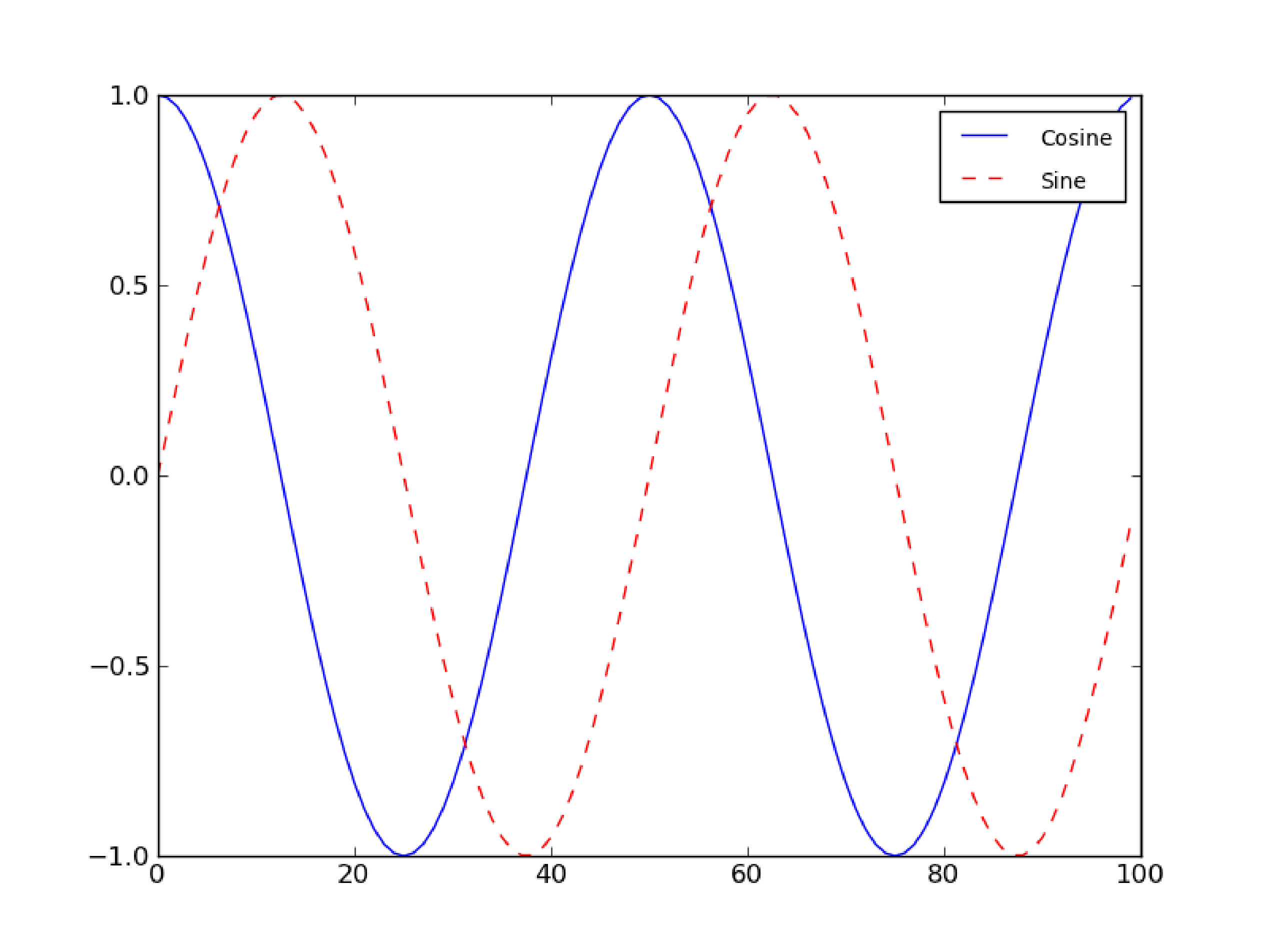
- The size can either be specified as a type point size such as 12, or as ‘large’, ‘small’, etc.

# Legend Fonts Example

```
plt.plot(x, y1, 'b-', label = 'Cosine')
plt.plot(x, y2, 'r--', label = 'Sine')

leg = plt.legend(('Cosine', 'Sine'), loc = 0)
for t in leg.get_texts():
    t.set_fontsize('small')

plt.show()   # show plot
```

File:  legend-fonts.py

# Legend Result



# Other Legend Keywords

| Keyword | Description |
|---|---|
| `numpoints` | How many points are used for the legend line |
| `markerscale` | Ratio of legend marker size to plot marker size |
| `frameon` | `True|False`, controls whether line is drawn for legend frame |
| `fancybox` | `None|True|False`, draws frame with round corners |
| `shadow` | `None|True|False`, draws frame with shadow |
| `ncol` | Number of columns for legend |
| `mode` | `‘expand’|None`, expands legend horizontally to fill plot |
| `title` | A string for the legend title |
| `borderpad` | Amount of whitespace inside legend border |
| `labelspacing` | Vertical spacing of legend entries |
| `handlelength` | length of legend lines |
| `handletextpad` | padding between legend lines and text |
| `borderaxespad` | padding between axes and legend border |
| `columnspacing` | Horizontal spacing between columns |

# Plotting Exercise #2

- For isothermal atmosphere

$$p = p_0 \exp\left(-z/H\right)$$

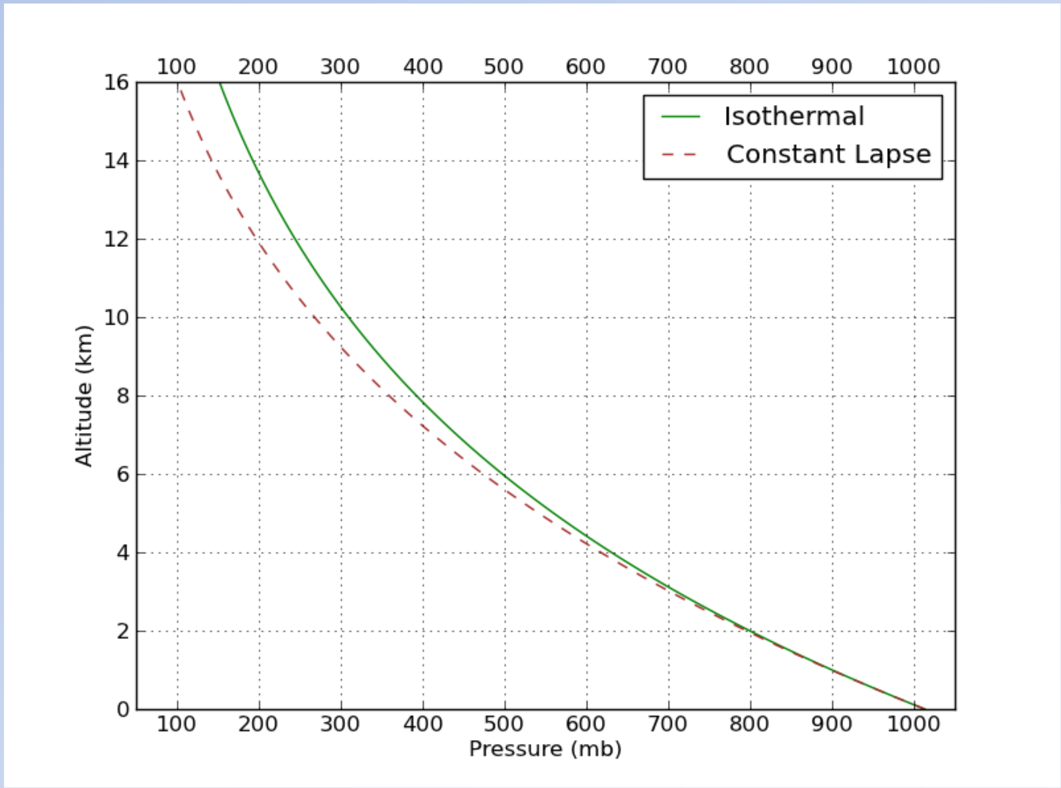
$$H = R_d T/g$$

- For constant lapse rate

$$p = p_0 \left( \frac{T_0 - \gamma z}{T_0} \right)^{g/\gamma R_d}$$

# Plotting Exercise #2

- Create plot shown.
- $p_0$ = 1013 mb
- $R_d$ = 287.1 J kg$^{-1}$ K$^{-1}$
- $g$ = 9.81 m s$^{-2}$
- $T = T_0$ = 288K
- $\gamma$ = 0.006K m$^{-1}$



# Polar Plots

- Polar plots are made with the `pyplot.polar(`*`angle, distance`*`)` function.

- *`angle`* is in radians

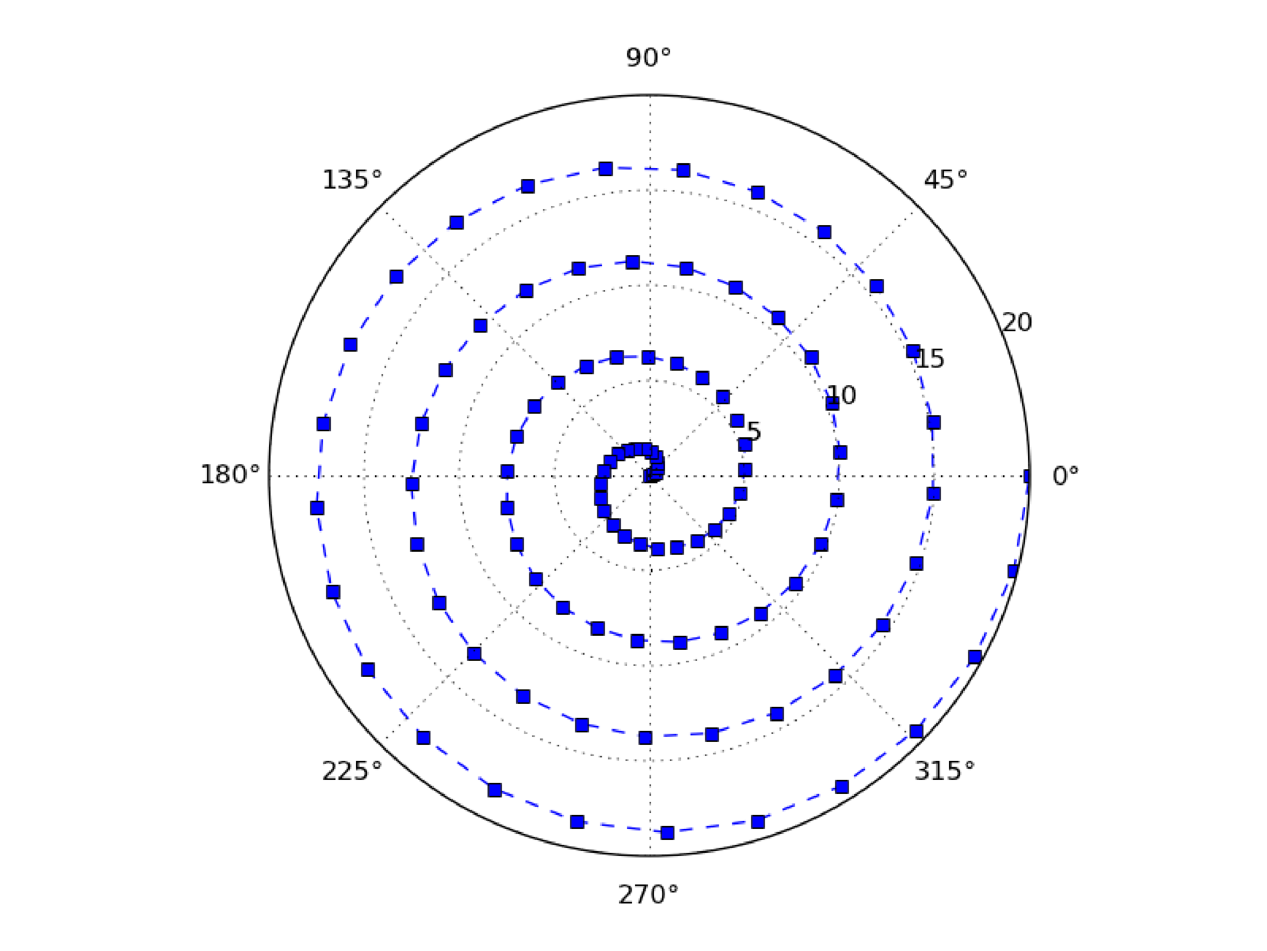
- Many of the keywords for linestyles, colors, symbols, etc. from `plot()` also work with `polar()`

# Polar Plot Example

```
import numpy as np
import matplotlib.pyplot as plt
theta = np.linspace(0,8*np.pi,100)
r = np.linspace(0,20,100)
plt.polar(theta, r, 'bs--')
plt.show()
```

File:  polar-plot.py

# Polar Plot Result



# Bar Charts

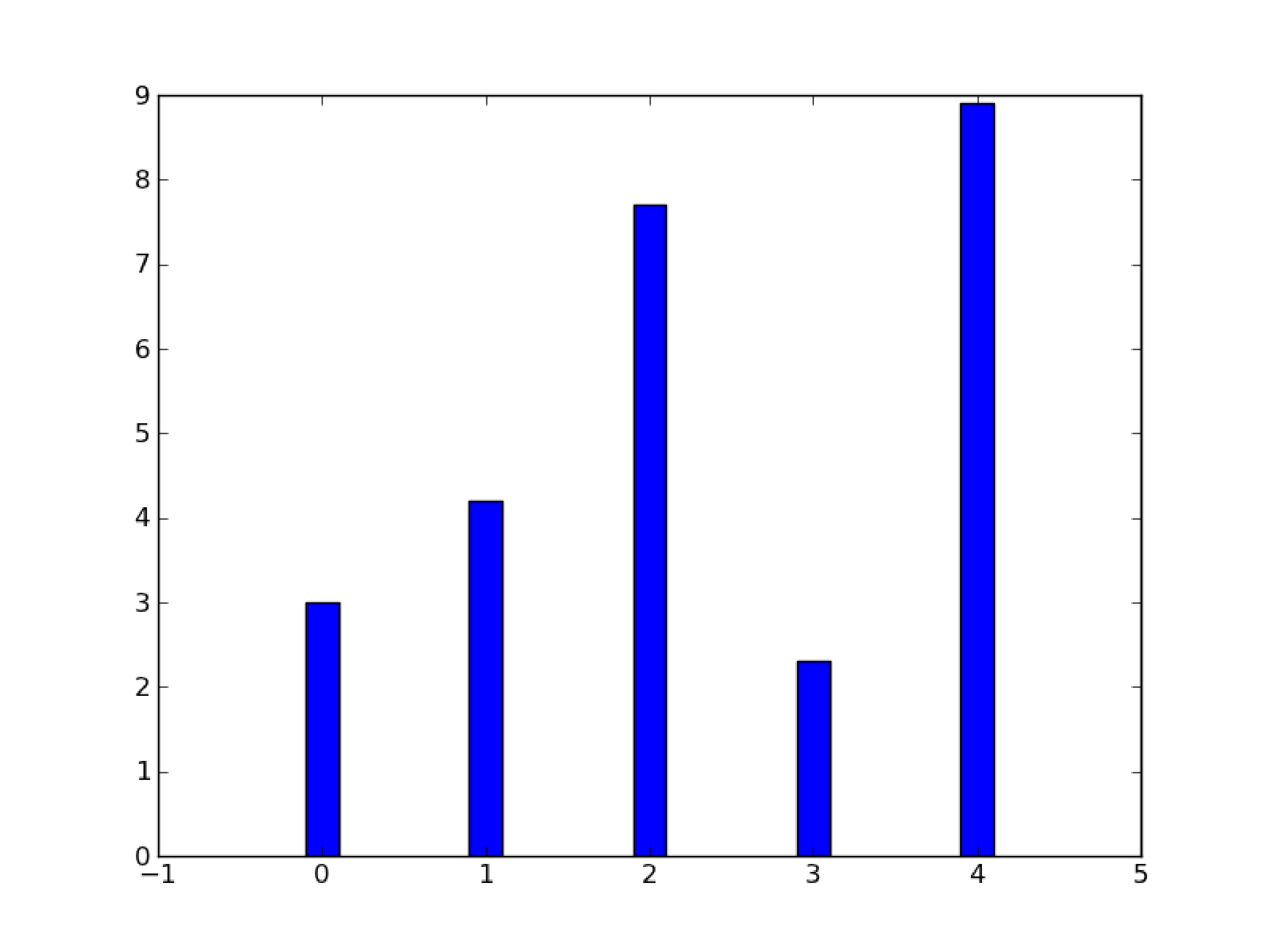
- The `bar()` axes method or pyplot function creates bar charts.

# Example Bar Chart

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(0,5)
y = [3.0, 4.2, 7.7, 2.3, 8.9]
plt.bar(x,y, width = 0.2, align = 'center')
plt.show()
```

File:  bar-chart.py

# Bar Chart Results



# Keywords for `bar()`

| Keyword | Purpose | Values |
|---|---|---|
| `color` | Controls color of  the bars | Any valid `matplotlib` color such as `'red','black','green'` etc. |
| `edgecolor` | Controls color of bar edges | Any valid `matplotlib` color such as `'red','black','green'` etc. |
| `bottom` | y coordinate for bottom of bars | List of floating point values.  Useful for creating stacked bar graphs. |
| `width` | Controls width of bars | Floating point value |
| `linewidth` or `lw` | Controls width of bar outline | Floating point value.  `None` for default linewidth, `0` for no edges. |
| `xerr` or `yerr` | Generates error bars for chart. | List of floating point numbers |
| `capsize` | Controls size of error bar caps | Floating point.  Default is 3. |
| `align` | Controls alignment of bars with axis labels | `‘edge’` or `‘center’` |
| `orientation` | Controls orientation of bars | `‘vertical’` or `‘horizontal’` |
| `log` | Sets log axis | `False` or `True` |

# Pie Charts

- Pie charts are created using the `pie()` axes method or pyplot function.

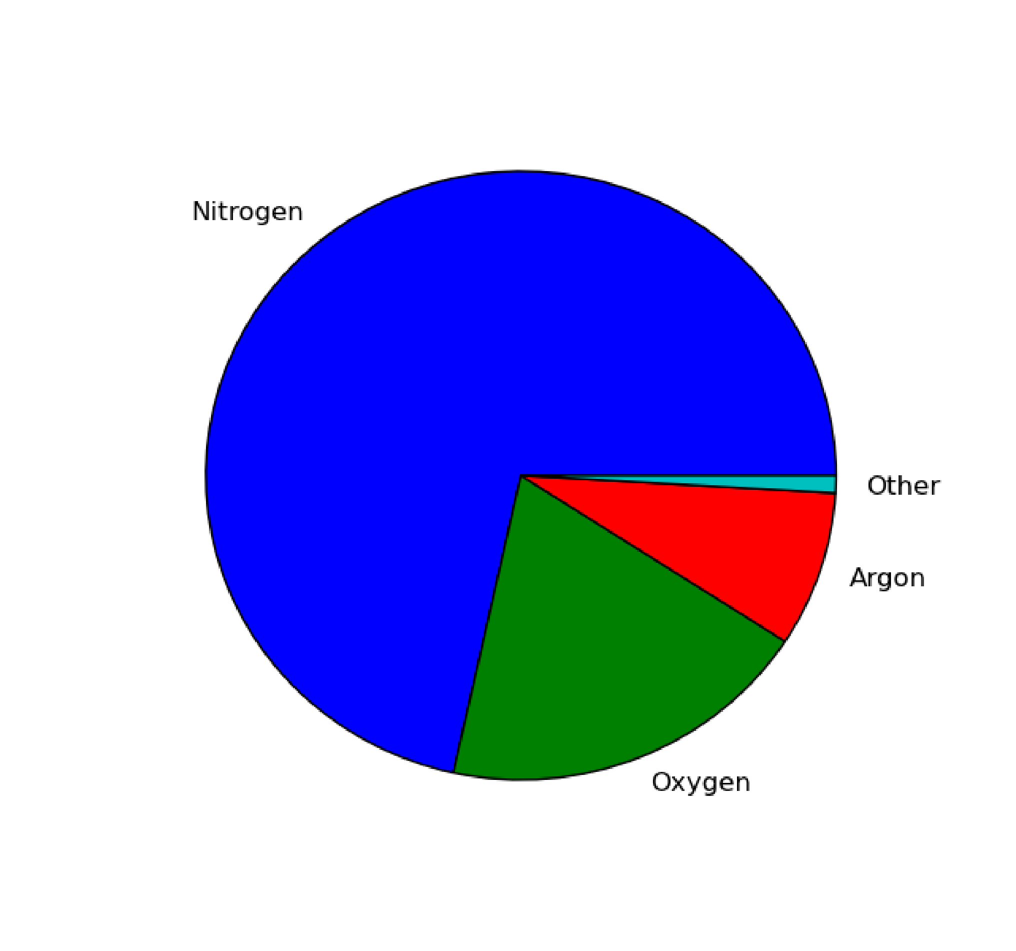
- There are also keywords for controlling the colors of the wedges, shadow effects, and labeling the wedges with numerical values (see online documentation for details.)

# Example Pie Chart

```
import matplotlib.pyplot as plt
import numpy as np
c = [0.78, 0.21,0.09,0.01]
l = ['Nitrogen', 'Oxygen', 'Argon', 'Other']
plt.pie(c,labels = l)
plt.show()
```

File:  pie-chart.py

# Pie Chart Result



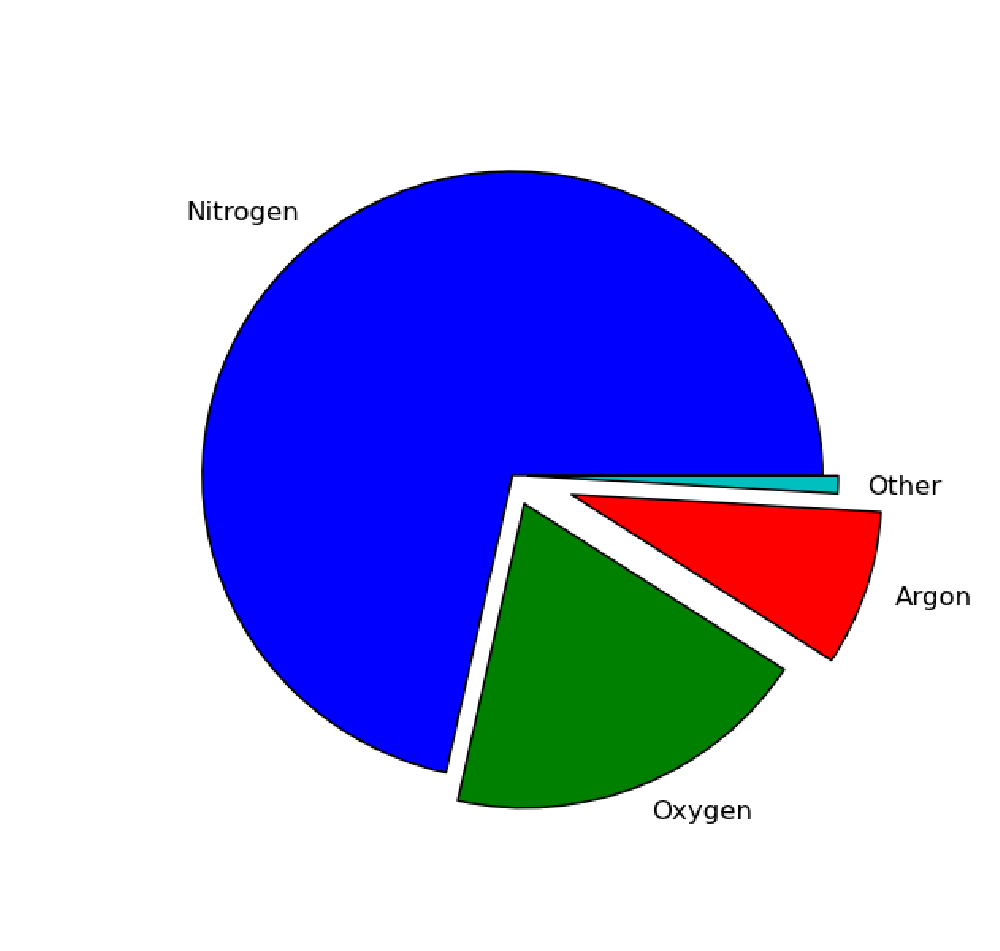
# Example Exploded Pie Chart

```
import matplotlib.pyplot as plt
import numpy as np
c = [0.78, 0.21,0.09,0.01]
l = ['Nitrogen', 'Oxygen', 'Argon', 'Other']
plt.pie(c, explode = [0,0.1,0.2,0.05], labels = l)
plt.show()
```

Offsets for wedges

File:  pie-chart-explode.py

# Exploded Pie Chart Result



# Placing Text on Plots

- Text can be placed on plots using the `text(`*`x`*`, `*`y`*`, `*`s`*`)` pyplot function or axes method.
- The arguments are:
  - *x* is the x-coordinate for the text
  - *y* is the y-coordinate for the text
  - *s* is the string to be written

# Keywords for `text()`

- Many of the same keywords that were used for axis labels and plot titles also work for the `text()` function/method.
- Common ones `size`, `color`, `rotation`, `backgroundcolor`, `linespacing`, `horizontalalignment`, and `verticalalignment`.

# Data Coordinates versus Axes Coordinates

- The x, y coordinates for text() can be specified in *data coordinates* or *axes coordinates*.

- Data coordinates correspond to the data values for the plotted lines.

- Axes coordinates are relative to the axes, with (0,0) being the lower-left corner, (1,0) being the lower right corner, (0,1) being the upper-left, and (1,1) being the upper-right corner.

# Data Coordinates versus Axes Coordinates (cont.)

- Data coordinates are the default.

- To use axes coordinates you must use the `transform` keyword as follows:
  - `transform = ax.transAxes`

- The `transform` keyword requires an axes instance, so you may have to use `ax=plt.gca()` first.

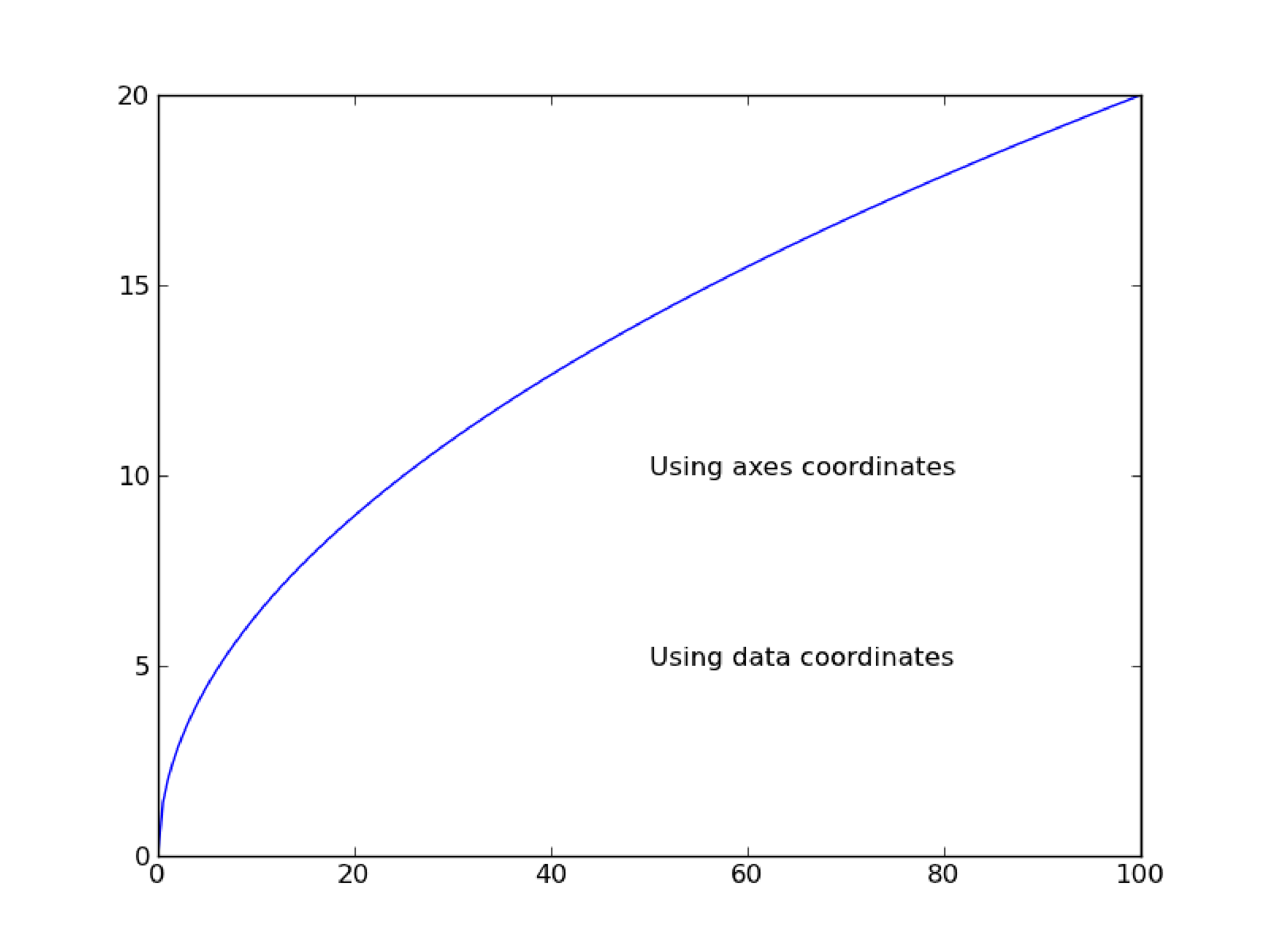
# `Text()` Example

```
…
x = np.arange(0,100.5,0.5)
y = 2.0*np.sqrt(x)

plt.plot(x,y)
plt.text(50,5,'Using data coordinates')
ax = plt.gca()
plt.text(0.5, 0.5,'Using axes coordinates',
        transform = ax.transAxes)
…
```

Need reference to current axes.

File: text-example.py

# `text()` Example Results



# Drawing Horizontal and Vertical Lines on Plots

- Vertical and horizontal lines are drawn using `hlines()` and `vlines()`, either as pyplot functions or axes methods.
  - `hlines(`*`y`*`, `*`xmn`*`, `*`xmx`*`)` draws horizontal lines at the specified *y* coordinates.
  - `vlines(`*`x`*`, `*`ymn`*`, `*`ymx`*`)` draws vertical lines at the specified *x* coordinates.

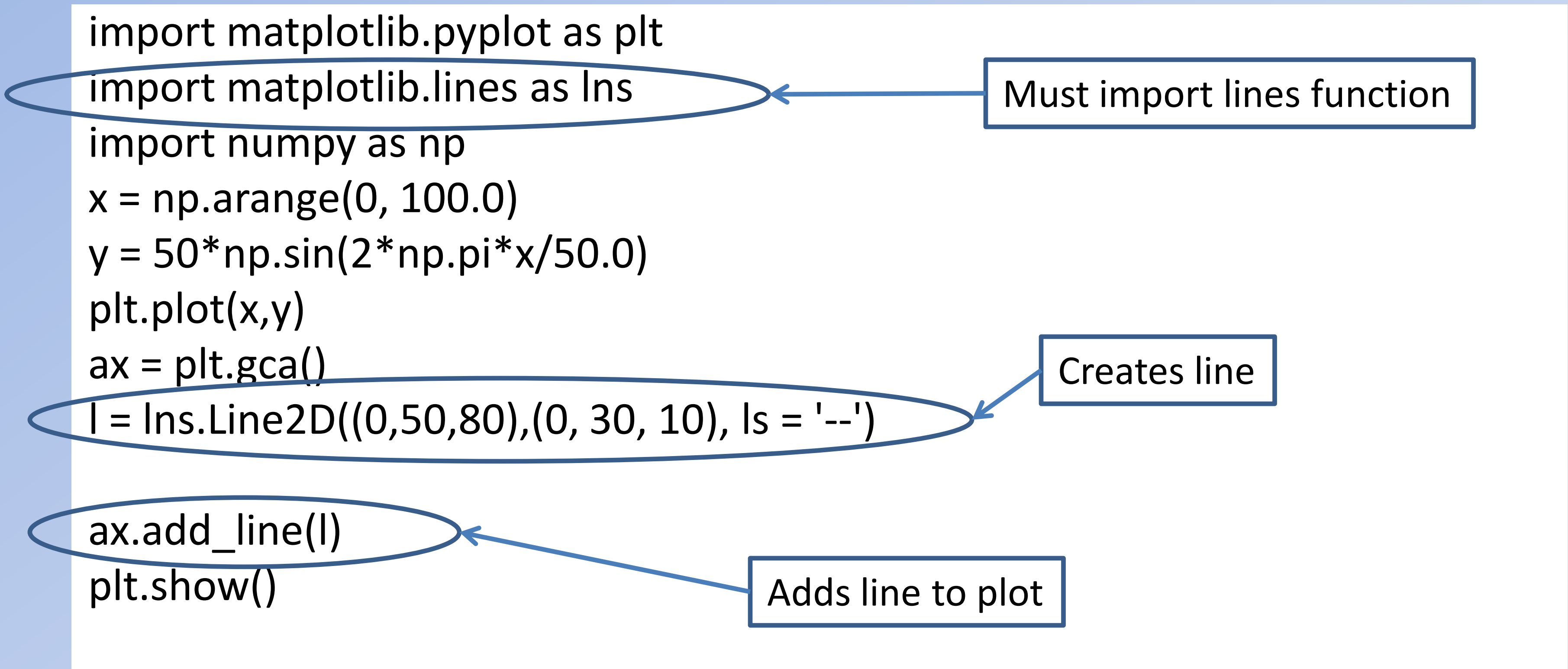
- *xmn*, *xmx*, *ymn*, and *ymx* are optional, and control the min and max coordinates of the lines

# Drawing Arbitrary Lines

- Arbitrary lines can be drawn using the `Line2D()` method from the `matplotlib.lines` module.

- Note that for this method you have to import the matplotlib.lines module.  You also have to use the `add_line()` method for the current axes for each line you want to add to the plot.

# Lines Example



```
import matplotlib.pyplot as plt
import matplotlib.lines as lns
import numpy as np
x = np.arange(0, 100.0)
y = 50*np.sin(2*np.pi*x/50.0)
plt.plot(x,y)
ax = plt.gca()
l = lns.Line2D((0,50,80),(0, 30, 10), ls = '--')

ax.add_line(l)
plt.show()
```

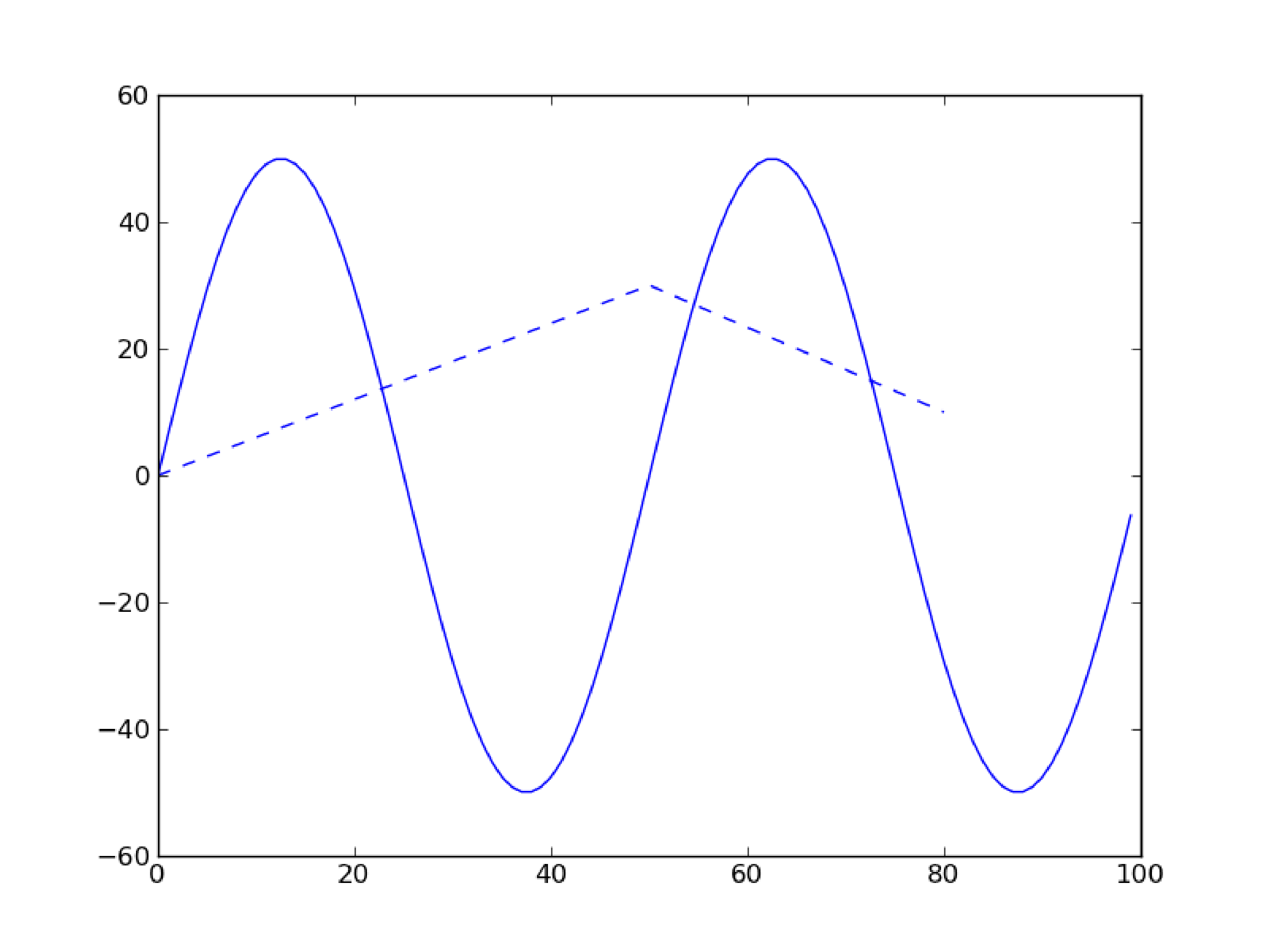
Must import lines function

Creates line

Adds line to plot

File:  lines-example.py

# Lines Result



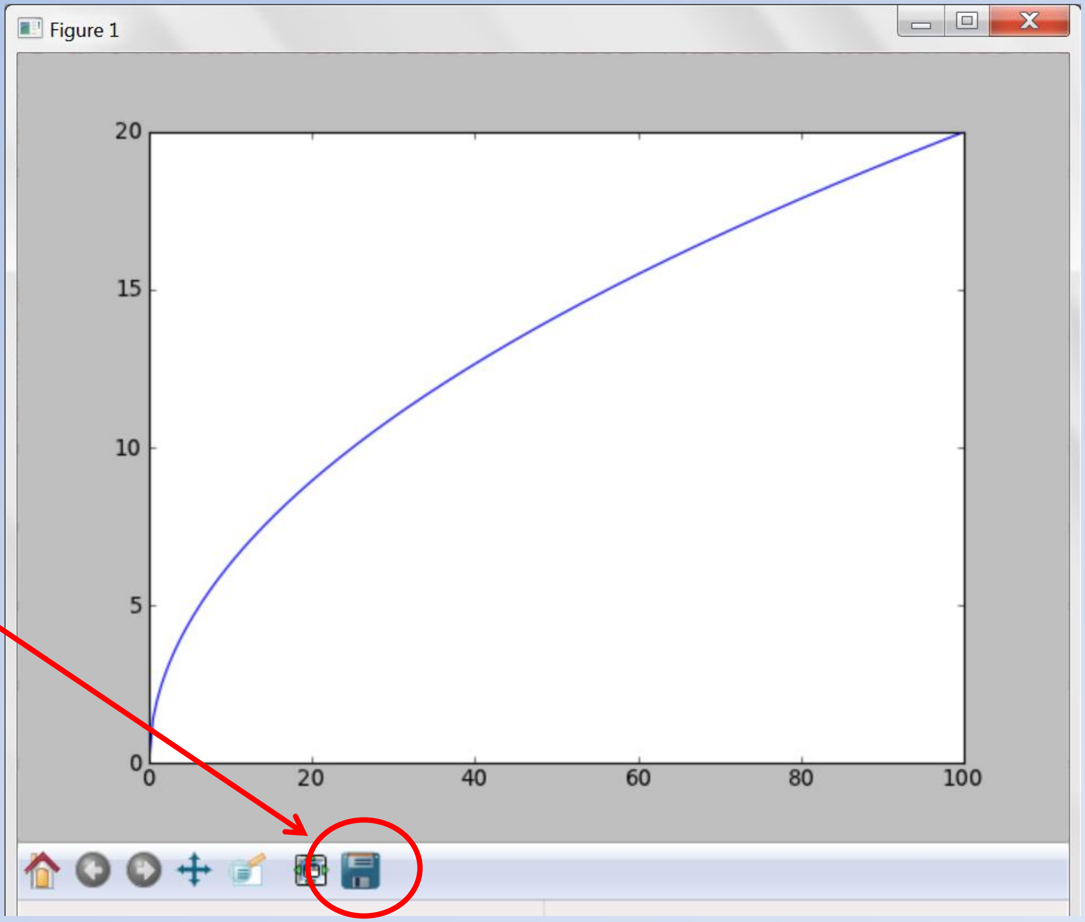
# Annotations

- There is also a method/function called `annotate()` which will place arrows and text onto plots.

- We won’t cover this here, but you can find it in the online documentation.

# Saving Images of Plots

- A plot can be saved directly by clicking here



# Saving Images of Plots (cont.)

- Images can also be saved within a program by using the `savefig(`*`filename`*`)` method of a figure object.

- filename is a string giving the name of the file in which to save the image.

- The file type is dicated by the extension given.

- Commonly supported file extensions are: emf, eps, jpeg, jpg, pdf, png, ps, raw, rgba, svg, svgz, tif, and tiff.

# Saving Plots Example

```
…
x = np.arange(0,100.5,0.5)
y = 2.0*np.sqrt(x)

plt.plot(x,y)   # Create figure and axis objects
fig = plt.gcf()  #  Get reference to current figure
fig.savefig('saved-plot.png')
```

File:  save-plot.py

# Specifying Figure Size

- The size of a figure can be specified using the `set_size_inches(`*w*`, `*h*`)` method of the figure object.
  - *w* and *h* are the width and height of the figure in inches.

- This needs to be done before saving the figure to a file