Introduction to Python Programming

Packages + Plotting

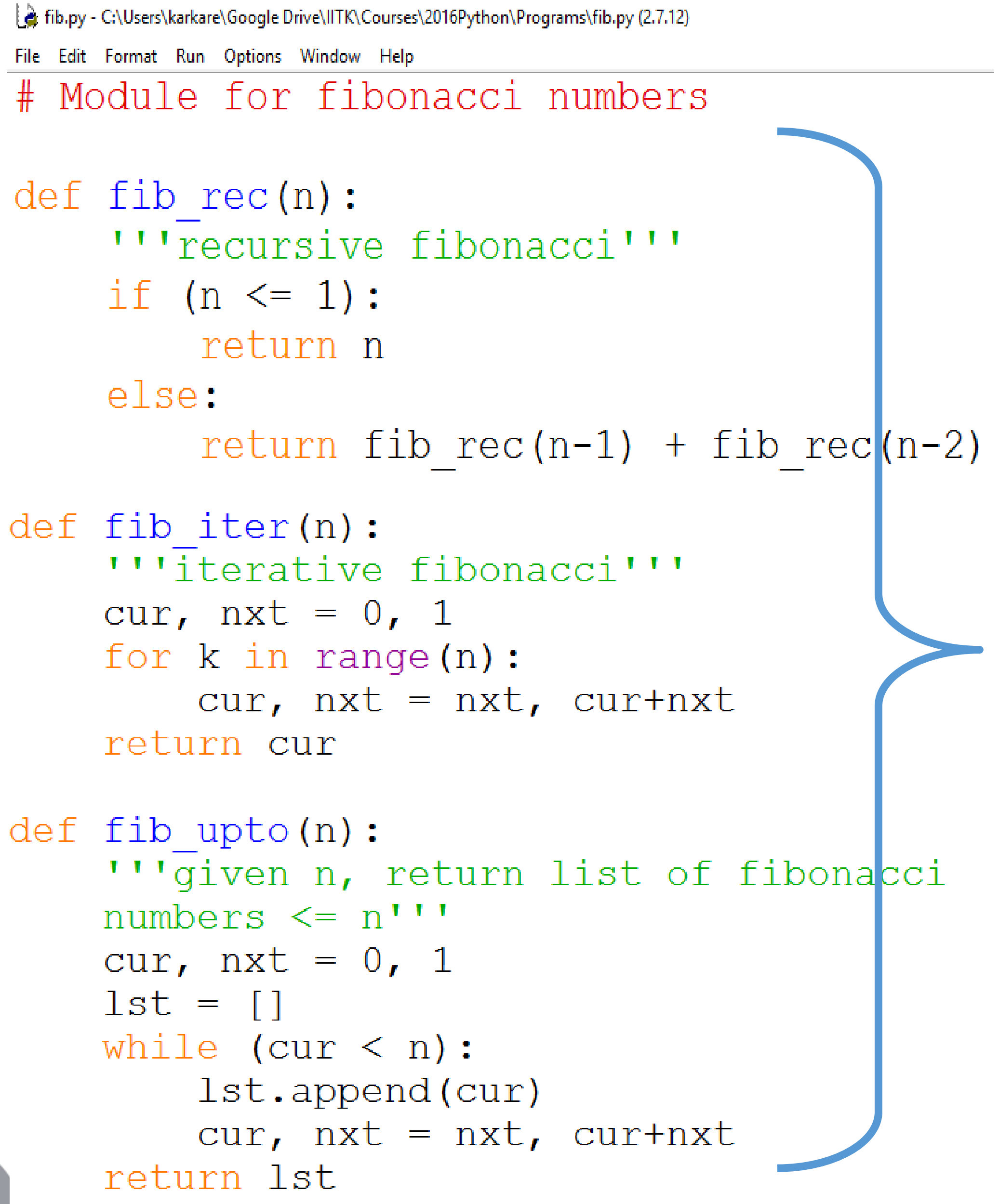
Modules

- As program gets longer, need to organize them for easier access and easier maintenance
- Reuse same functions across programs without copying its definition into each program
- Python allows putting definitions in a file
  - use them in a script or in an interactive instance of the interpreter
- Such a file is called a **module**
  - definitions from a module can be *imported* into other modules or into the *main* module

- A module is a file containing Python definitions and statements
- The file name is the module name with the suffix .py appended

Modules Example

Definition

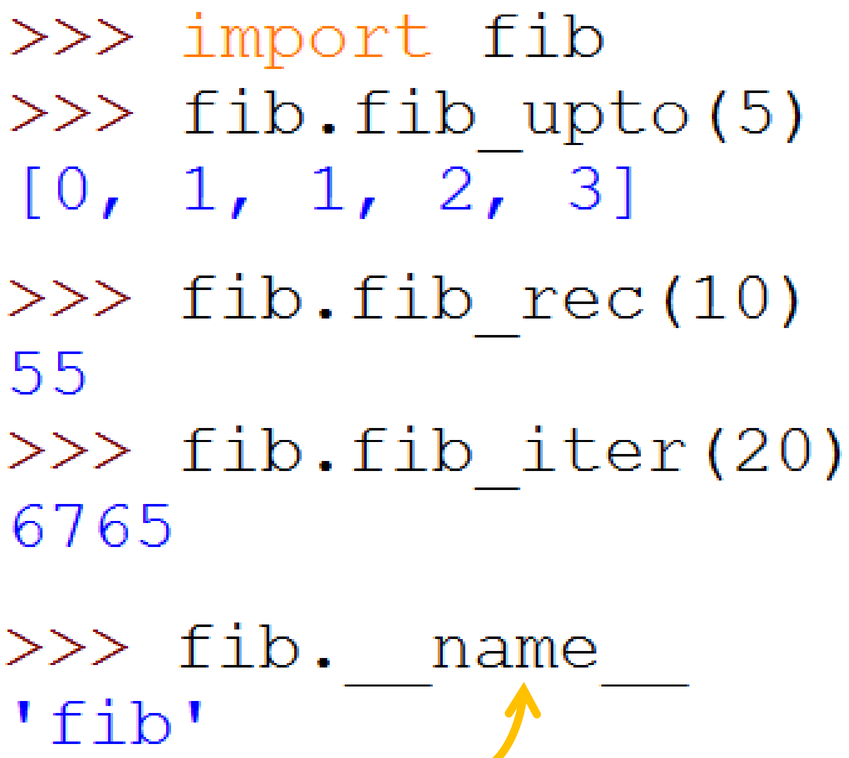

fib.py - C:\Users\karkare\Google Drive\IITK\Courses\2016Python\Programs\fib.py (2.7.12)
File Edit Format Run Options Window Help

```
# Module for fibonacci numbers

def fib_rec(n):
    '''recursive fibonacci'''
    if (n <= 1):
        return n
    else:
        return fib_rec(n-1) + fib_rec(n-2)

def fib_iter(n):
    '''iterative fibonacci'''
    cur, nxt = 0, 1
    for k in range(n):
        cur, nxt = nxt, cur+nxt
    return cur

def fib_upto(n):
    '''given n, return list of fibonacci
    numbers <= n'''
    cur, nxt = 0, 1
    lst = []
    while (cur < n):
        lst.append(cur)
        cur, nxt = nxt, cur+nxt
    return lst
```

fib.py

Modules Example

Usage



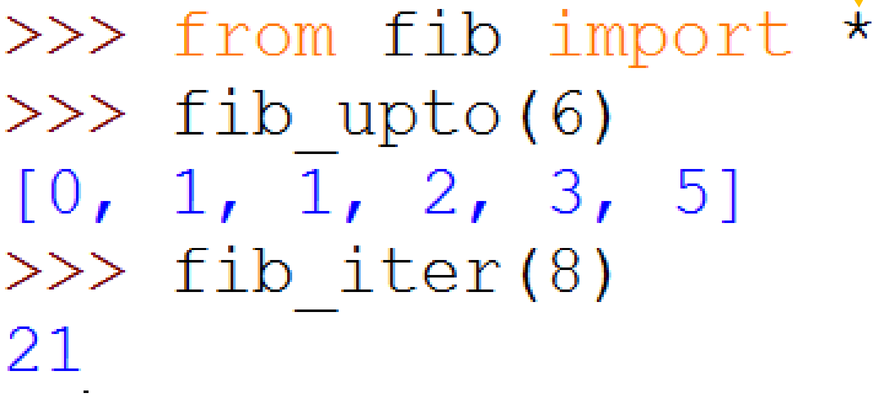
```
>>> import fib
>>> fib.fib_upto(5)
[0, 1, 1, 2, 3]
>>> fib.fib_rec(10)
55
>>> fib.fib_iter(20)
6765
>>> fib.__name__
'fib'
```

Within a module, the module's name is available as the value of the global variable __name__.

Modules Example

Usage (cont.)

To import **all** functions from a module, in the current symbol table



```
>>> from fib import *
>>> fib_upto(6)
[0, 1, 1, 2, 3, 5]
>>> fib_iter(8)
21
```

Packages (Libraries)

- A Python **package** (*library*) is a collection of Python modules

- Packages are a way of structuring Python's module namespace by using dotted module names
  - The module name A.B designates a submodule named B in a package named A.
  - The use of dotted module names saves the authors of multi-module packages like *NumPy* or *Matplotlib* from having to worry about each other's module names

Importing Modules from Packages

```
import matplotlib.pyplot
```

- Loads the submodule `pyplot` from the package `matplotlib`

```
import numpy as np
```

- Loads all subroutines/functions from the package `numpy`
- Calling a specific function by using the dotted name convention, e.g. `y=np.sqrt(x)` for the square root function $y=x^2$

Popular Packages (libraries)

https://hackr.io/blog/best-python-libraries

• pandas, numpy, scipy ….	Data Handling/Analysis
• matplotlib, seaborn, bokeh ….	Data Visualization
• scikit-learn, tensorflow, pytorch, keras….	Machine Learning
• beautifulsoup, scrapy ….	Web Scraping
• opencv, pillow ….	Computer Vision
• ….	
• ….	

Plotting in Python

- Before creating plots, it is worth spending sometime familiarising ourselves with a famous Python plotting library/package → **matplotlib**
- **`matplotlib`** was originally developed by a neurobiologist in order to emulate aspects of the MATLAB software
- **`matplotlib`** is an open-source Python library often touted as an alternative to the paid solution MATLAB. **`matplotlib`** was made for the purpose **of data visualization** as it's used to create graphs and plots.
- **`matplotlib`** does have a limit — it can only do 2D plotting. Despite this fact, this library remains highly capable of producing publish-ready data visualizations in the form of plots, diagrams, histograms, plots, scatter plots, error charts, and of course, bar charts.

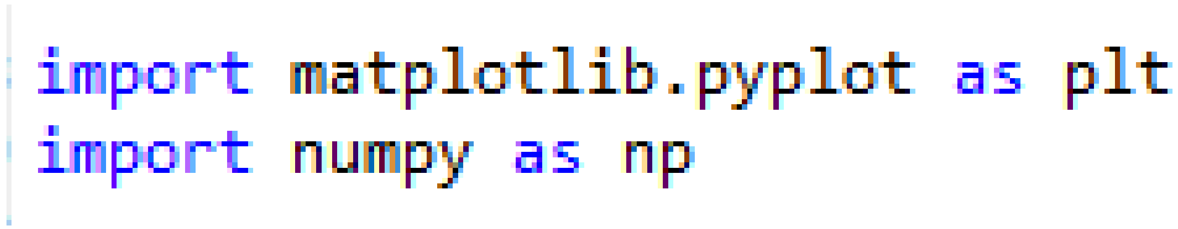
Different Graph Types

- A **simple line graph** can be plotted with `plot()`
- A **histogram** can be created with `hist()`
- A **bar chart** can be created with `bar()`
- A **pie chart** can be created with `pie()`
- A **scatter plot** can be created with `scatter()`

plus many more….

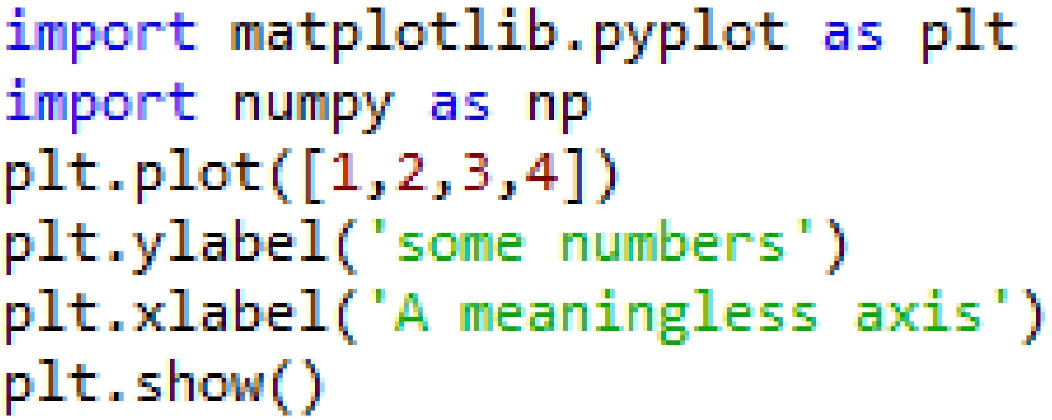
Getting started

- We are also going to import `numpy`, which we are going to use to *generate random data* for our examples

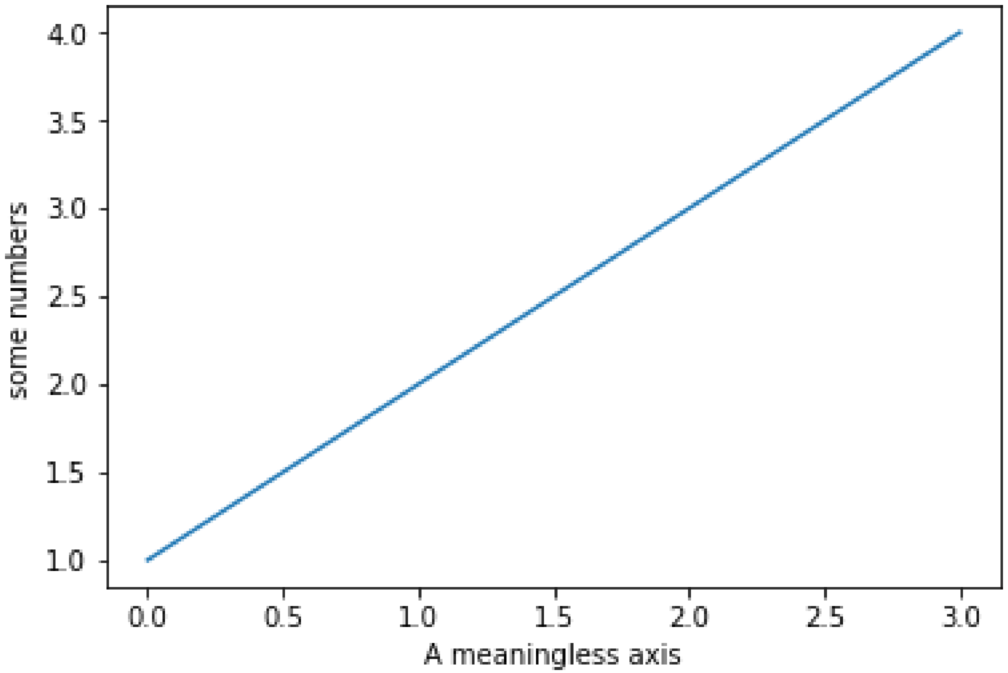


```
import matplotlib.pyplot as plt
import numpy as np
```

Our first plot



```
import matplotlib.pyplot as plt
import numpy as np
plt.plot([1,2,3,4])
plt.ylabel('some numbers')
plt.xlabel('A meaningless axis')
plt.show()
```



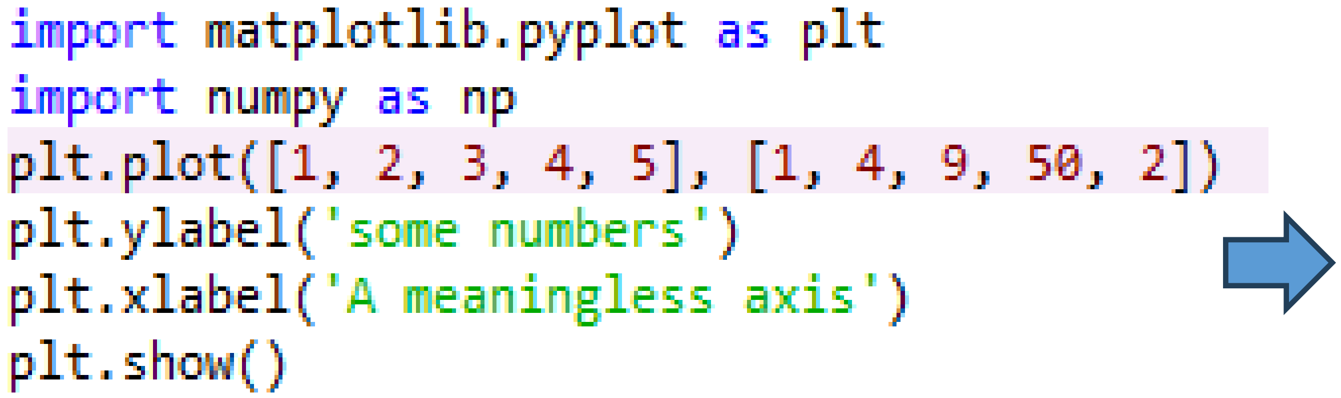
You may be wondering why the x-axis ranges from 0-3 and the y-axis from 1-4.

- If you provide a single list or array to the plot() command, Matplotlib assumes it is a sequence of y values, and automatically generates the x values for you.
- Since python ranges start with 0, the default x vector has the same length as y but starts with 0. Hence the x data are [0,1,2,3].

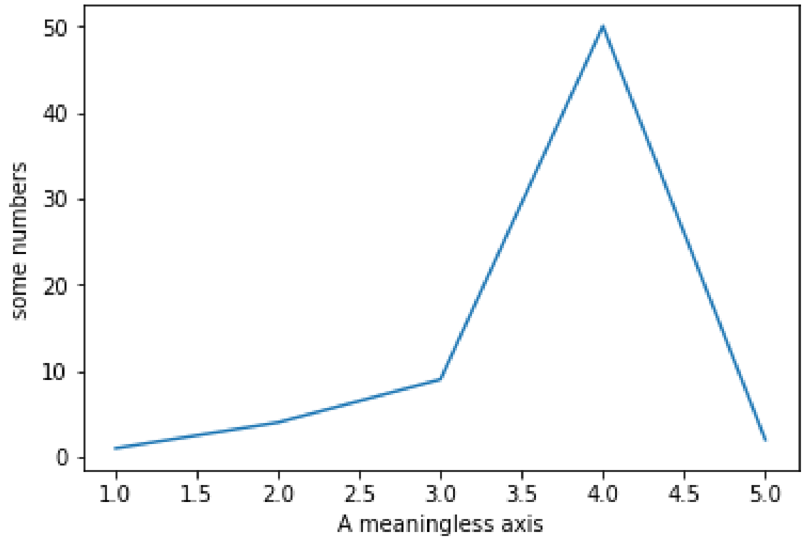
The `plot()` function

- The `plot()` argument is quite versatile, and will take any arbitrary collection of numbers.

For example, if we add an extra entry to the x axis, and replace the last entry in the Y axis and add another entry:



```python
import matplotlib.pyplot as plt
import numpy as np
plt.plot([1, 2, 3, 4, 5], [1, 4, 9, 50, 2])
plt.ylabel('some numbers')
plt.xlabel('A meaningless axis')
plt.show()
```

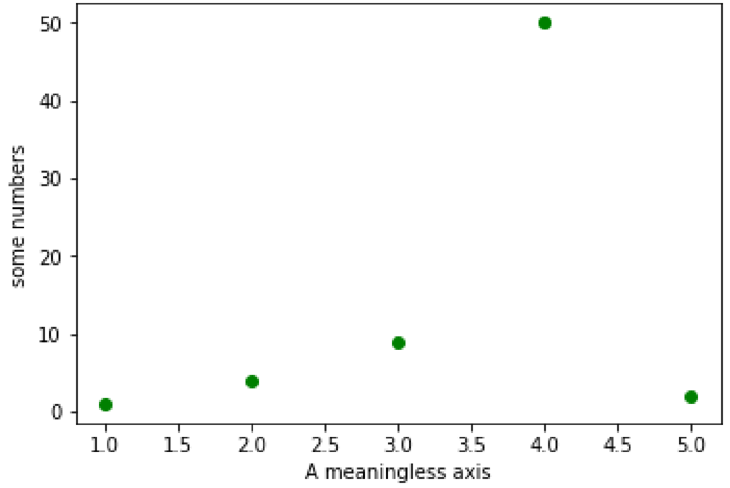


The `plot()` function

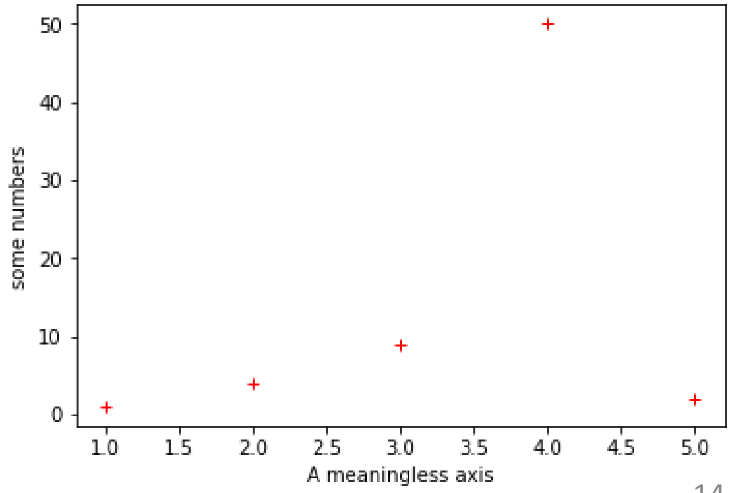
- The `plot()` function has an optional third argument that specifies *the appearance of the data points*.
- The default is `b-`, which is the blue solid line seen in the last two examples.

  The full list of styles can be found in the documentation for the plot() on the Matplotlib page

```
plt.plot([1, 2, 3, 4, 5], [1, 4, 9, 50, 2], 'go')
```

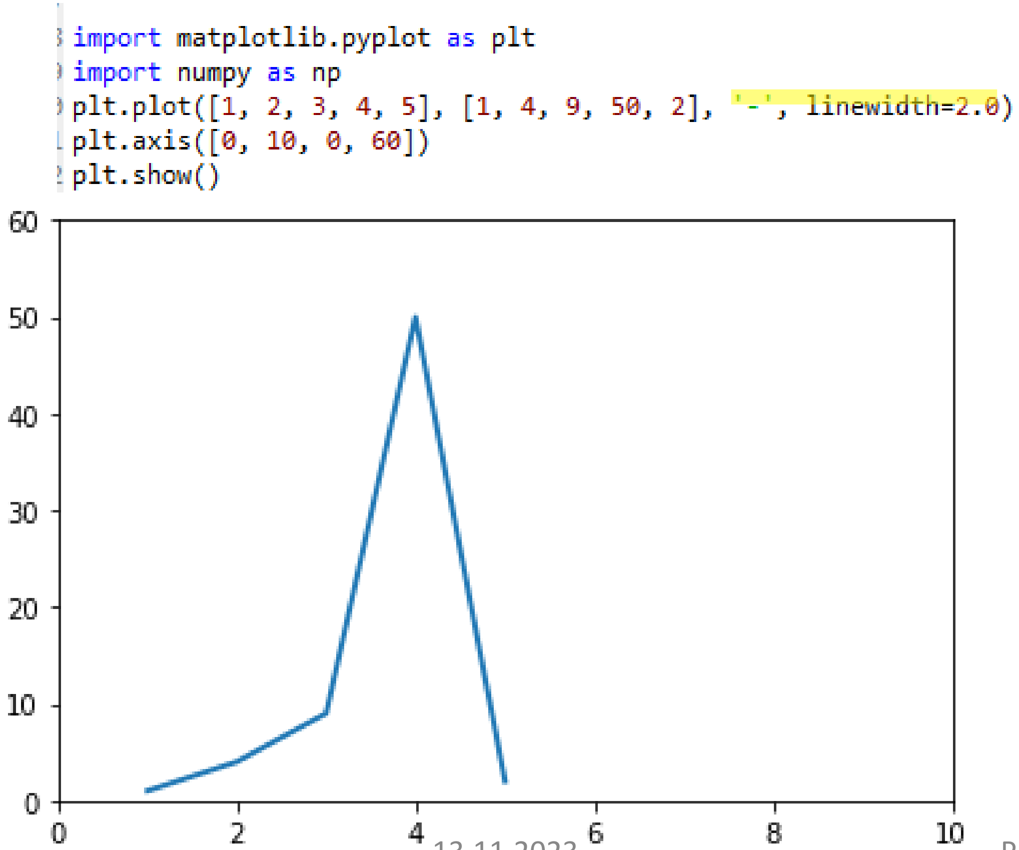


```
plt.plot([1, 2, 3, 4, 5], [1, 4, 9, 50, 2], 'r+')
```

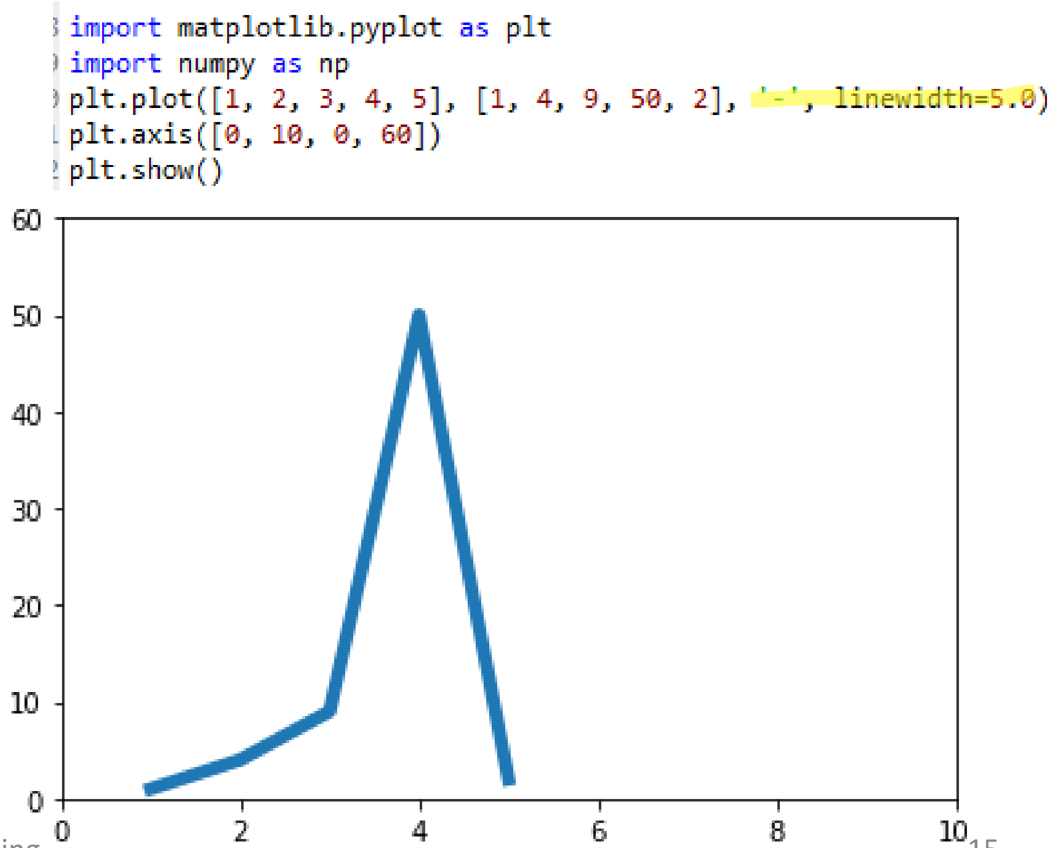


The `plot()` function

You can quite easily alter the *properties of the line* with the `plot()` function.



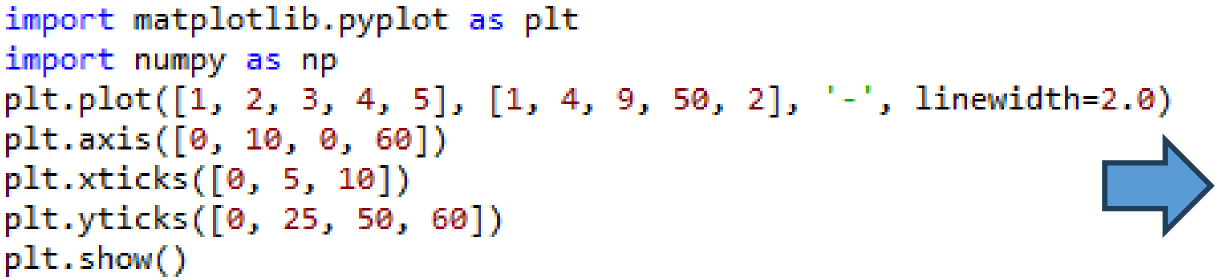
```
import matplotlib.pyplot as plt
import numpy as np
plt.plot([1, 2, 3, 4, 5], [1, 4, 9, 50, 2], '-', linewidth=2.0)
plt.axis([0, 10, 0, 60])
plt.show()
```



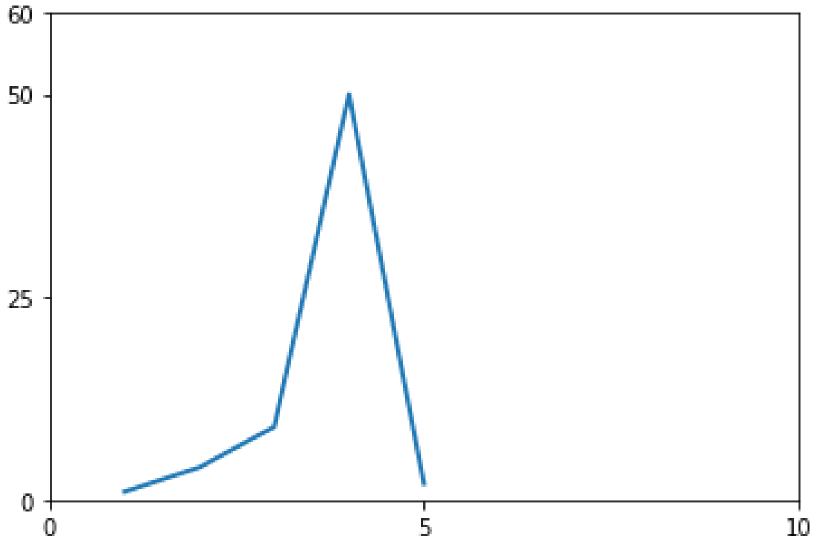
```
import matplotlib.pyplot as plt
import numpy as np
plt.plot([1, 2, 3, 4, 5], [1, 4, 9, 50, 2], '-', linewidth=5.0)
plt.axis([0, 10, 0, 60])
plt.show()
```

Altering tick labels

- The `plt.xticks()` and `plt.yticks()` allows you to manually alter the ticks on the x-axis and y-axis respectively.
  Note that the tick values have to be contained within a list object.

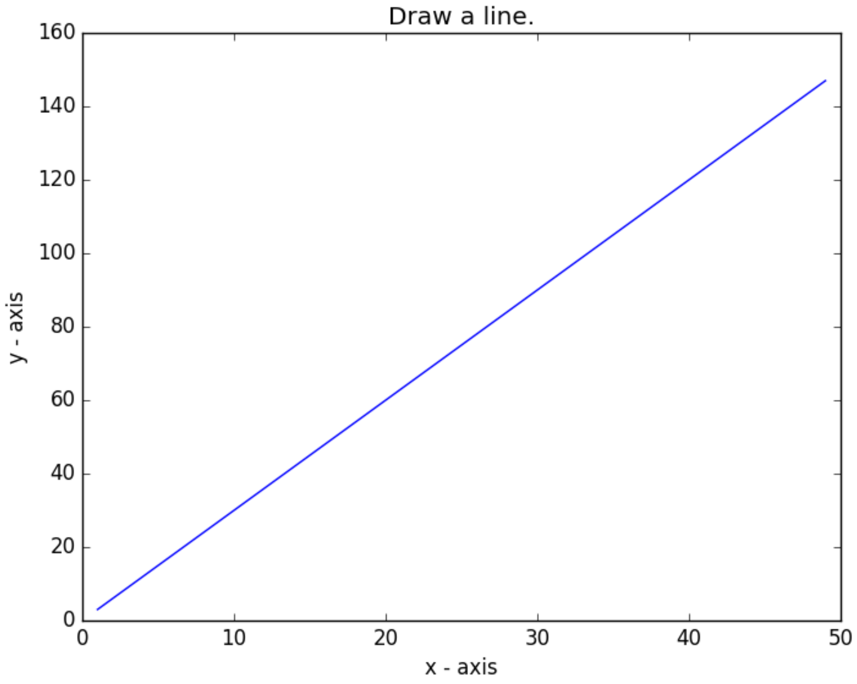


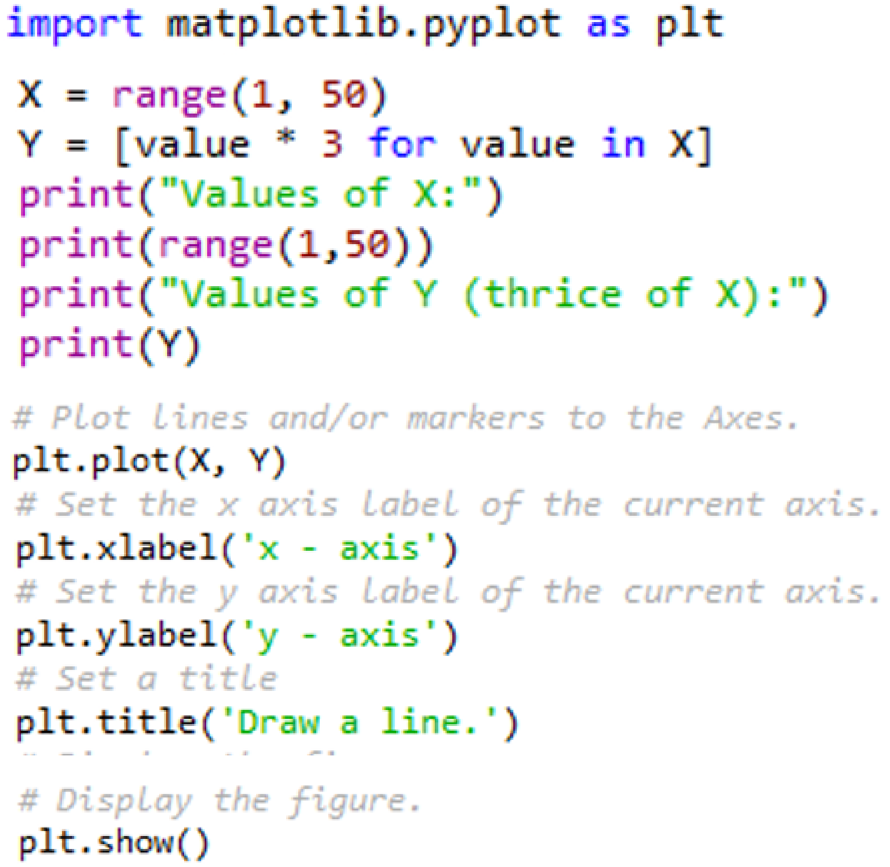
```
import matplotlib.pyplot as plt
import numpy as np
plt.plot([1, 2, 3, 4, 5], [1, 4, 9, 50, 2], '-', linewidth=2.0)
plt.axis([0, 10, 0, 60])
plt.xticks([0, 5, 10])
plt.yticks([0, 25, 50, 60])
plt.show()
```



Task - Basic Line Graph

Let’s write a Python program to draw a line graph with suitable labels for the x-axis and y-axis. Include a title.

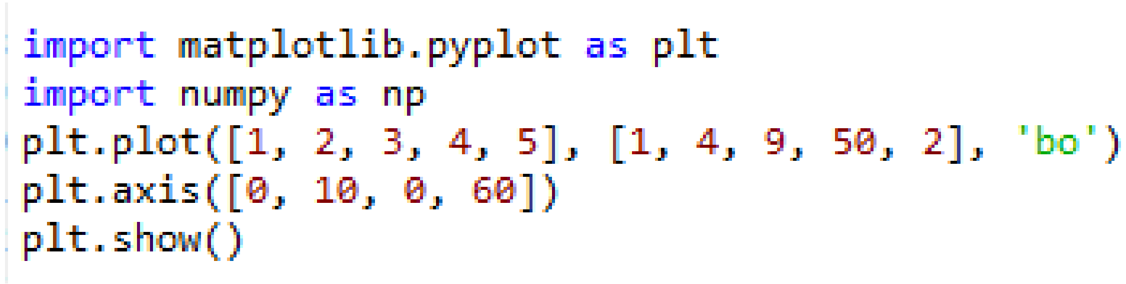




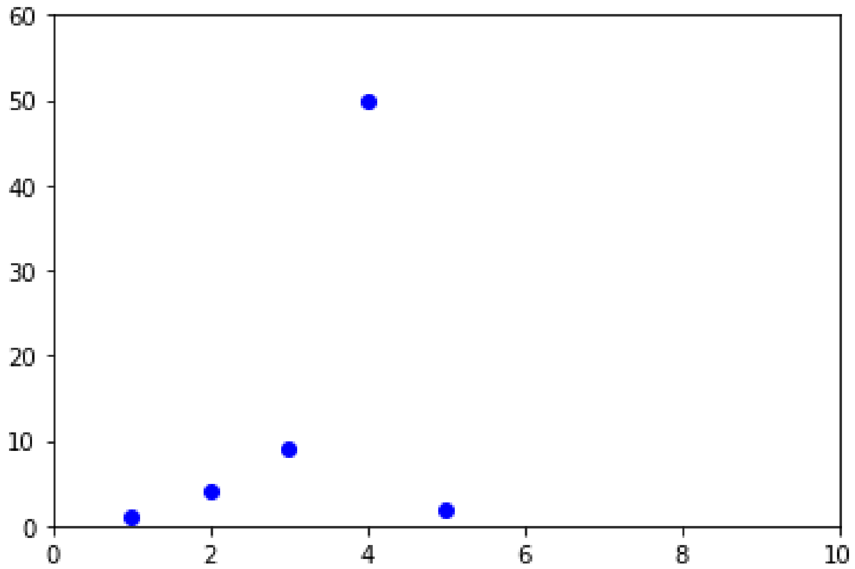
```
import matplotlib.pyplot as plt

X = range(1, 50)
Y = [value * 3 for value in X]
print("Values of X:")
print(range(1,50))
print("Values of Y (thrice of X):")
print(Y)

# Plot lines and/or markers to the Axes.
plt.plot(X, Y)
# Set the x axis label of the current axis.
plt.xlabel('x - axis')
# Set the y axis label of the current axis.
plt.ylabel('y - axis')
# Set a title
plt.title('Draw a line.')

# Display the figure.
plt.show()
```

The **axis()** function

- The **axis()** function allows us to specify the range of the axis.
- It requires a list that contains the following:
  `[The min x-axis value, the max x-axis value, the min y-axis, the max y-axis value]`

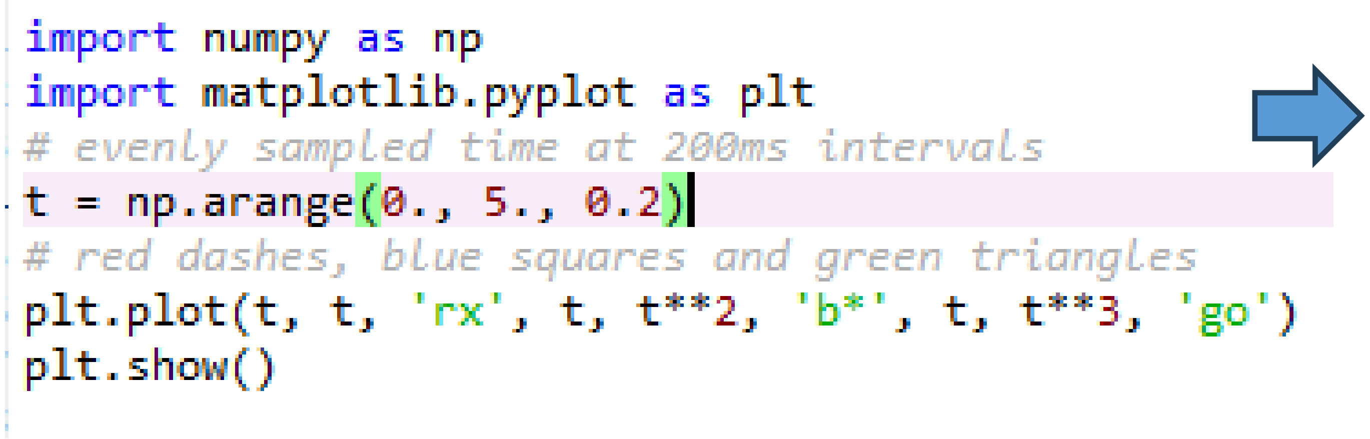


```python
import matplotlib.pyplot as plt
import numpy as np
plt.plot([1, 2, 3, 4, 5], [1, 4, 9, 50, 2], 'bo')
plt.axis([0, 10, 0, 60])
plt.show()
```

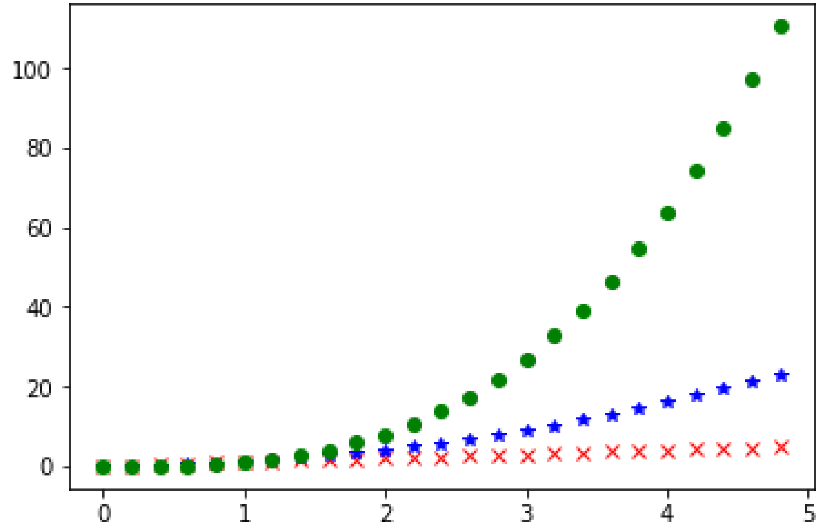


Matplotlib and NumPy Arrays

- Normally when working with numerical data, you’ll be using **NumPy** arrays.
- This is still straight forward to do in **Matplotlib**; in fact all sequences are converted into NumPy arrays internally anyway.



```
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, 'rx', t, t**2, 'b*', t, t**3, 'go')
plt.show()
```



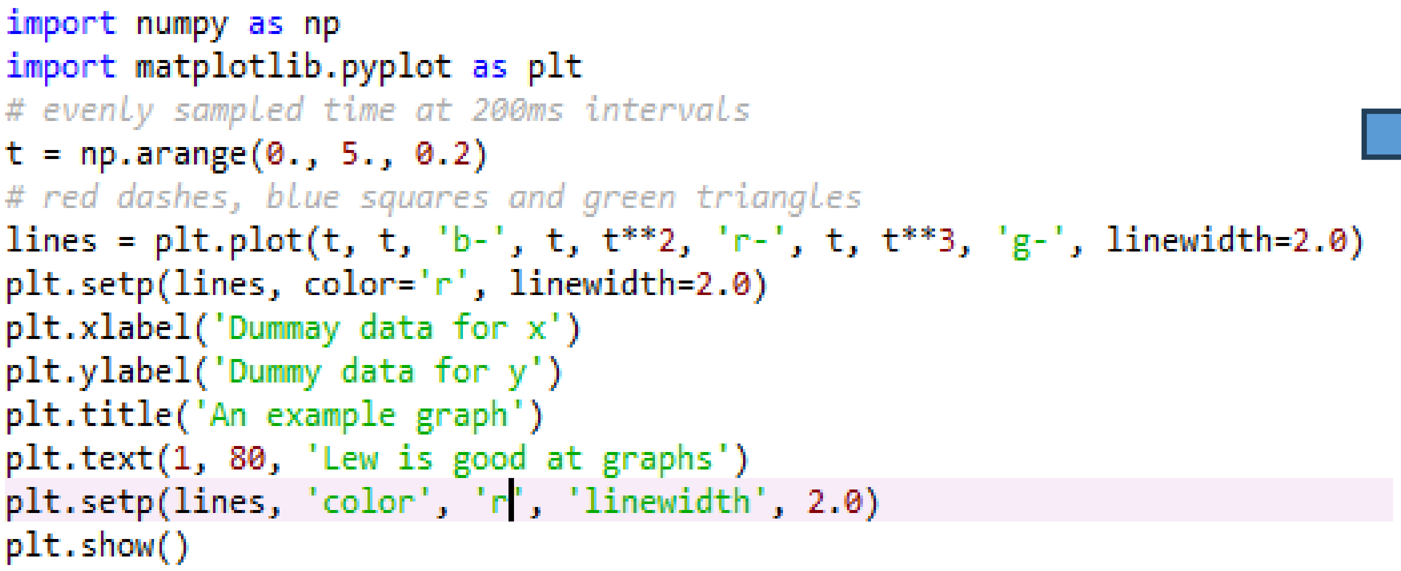
Working with Text

- There are a number of different ways in which to **add text to your graph**:
  - `title()` = Adds a title to your graph, takes a string as an argument
  - `xlabel()` = Add a title to the x-axis, also takes a string as an argument
  - `ylabel()` = same as `xlabel()`
  - `text()` = Can be used to add text to an arbitrary location on your graph. Requires the following arguments:

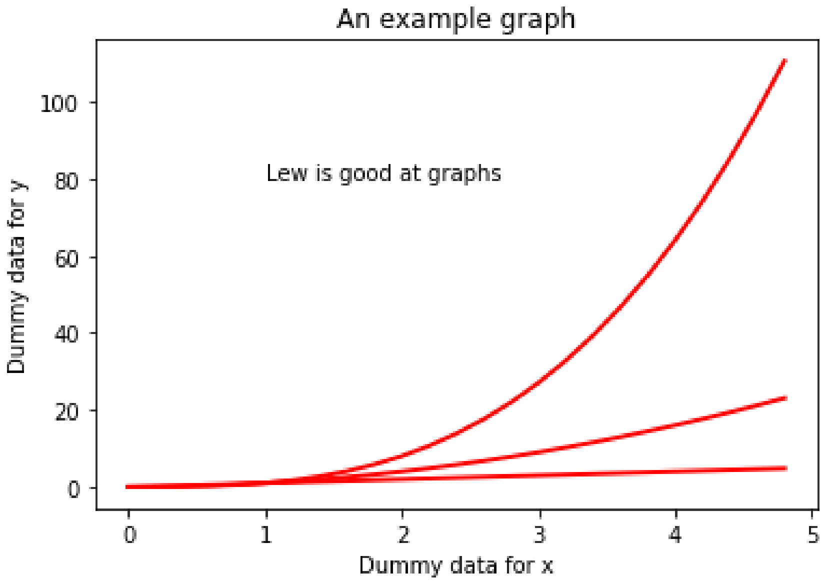
    `text(x-axis location, y-axis location, the string of text to be added)`

- Note: Matplotlib uses TeX equation expressions. So, as an example, if you wanted to put $\sigma_i = 15$ in one of the text blocks, you would write `plt.title(r'$\sigma_i=15$')`

Example - Working with Text



```python
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
lines = plt.plot(t, t, 'b-', t, t**2, 'r-', t, t**3, 'g-', linewidth=2.0)
plt.setp(lines, color='r', linewidth=2.0)
plt.xlabel('Dummay data for x')
plt.ylabel('Dummy data for y')
plt.title('An example graph')
plt.text(1, 80, 'Lew is good at graphs')
plt.setp(lines, 'color', 'r', 'linewidth', 2.0)
plt.show()
```



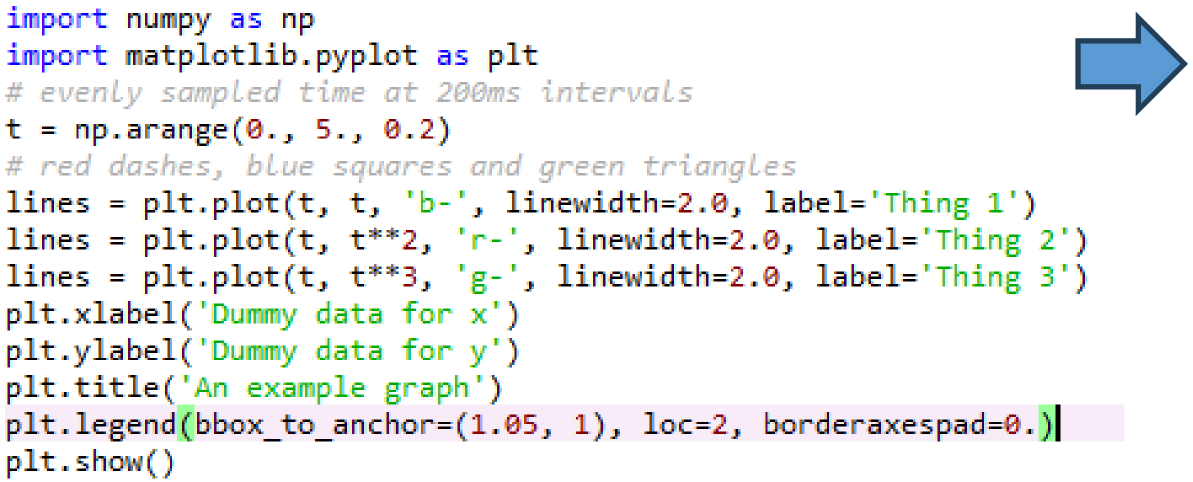
Legends

- The location of a legend is specified by the `loc` command.

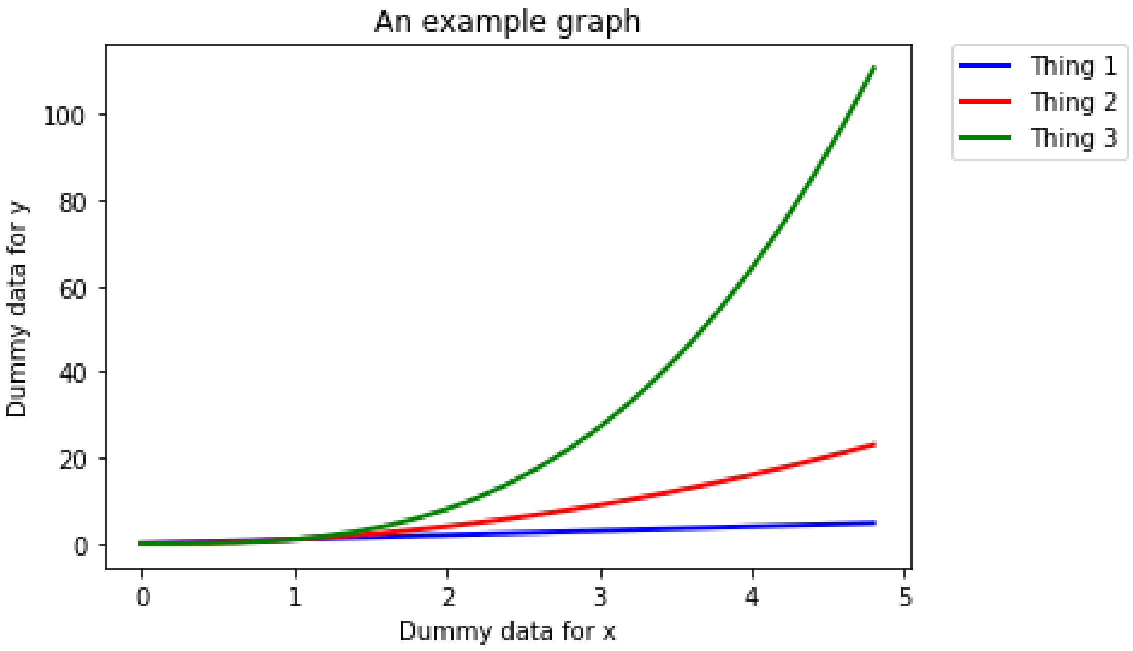
  There are a number of in-built locations that can be altered by replacing the number. The Matplotlib website has a list of all locations in the documentation page for `location().`

- You can then use the `bbox_to_anchor()` function to manually place the legend, or when used with `loc`, to make slight alterations to the placement.

Legends

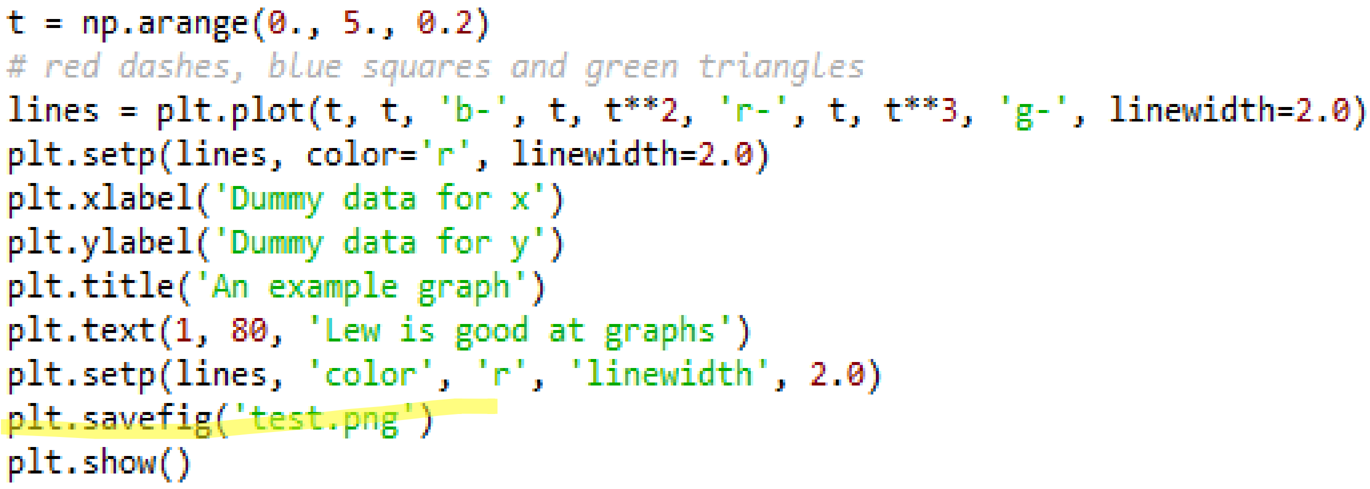


```
import numpy as np
import matplotlib.pyplot as plt
# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)
# red dashes, blue squares and green triangles
lines = plt.plot(t, t, 'b-', linewidth=2.0, label='Thing 1')
lines = plt.plot(t, t**2, 'r-', linewidth=2.0, label='Thing 2')
lines = plt.plot(t, t**3, 'g-', linewidth=2.0, label='Thing 3')
plt.xlabel('Dummy data for x')
plt.ylabel('Dummy data for y')
plt.title('An example graph')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```



Saving a Figure as a File

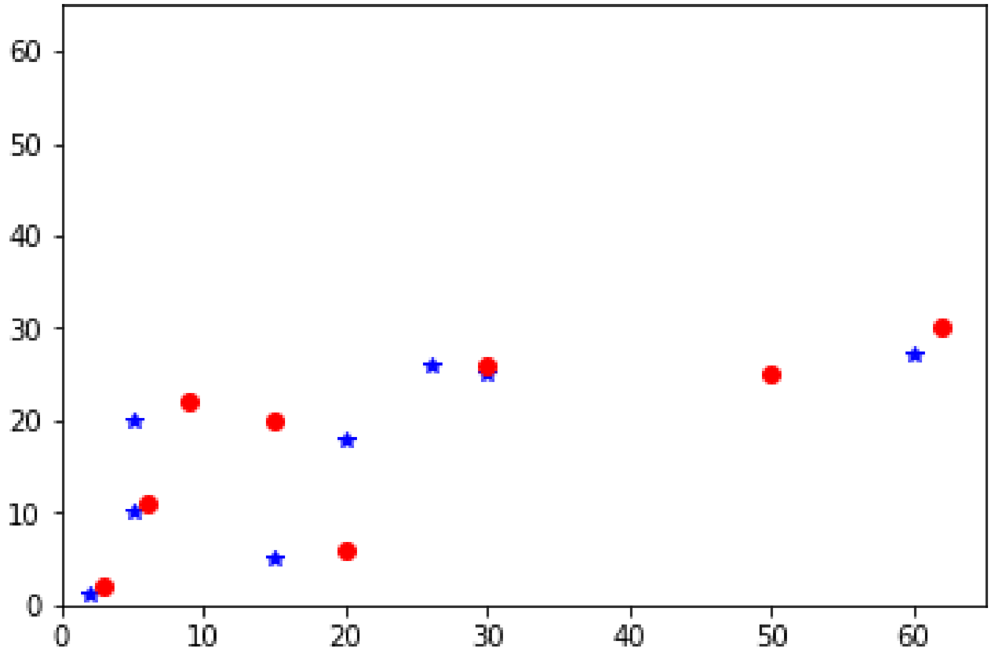
- The plt.savefig() allows you to save your plot as a file.
- It takes a string as an argument, which will be the name of the file. You must remember to state which file type you want the figure saved as; i.e. png or jpeg.
- Make sure you put the plt.savefig() before the plt.show() function. Otherwise, the file will be a blank file.

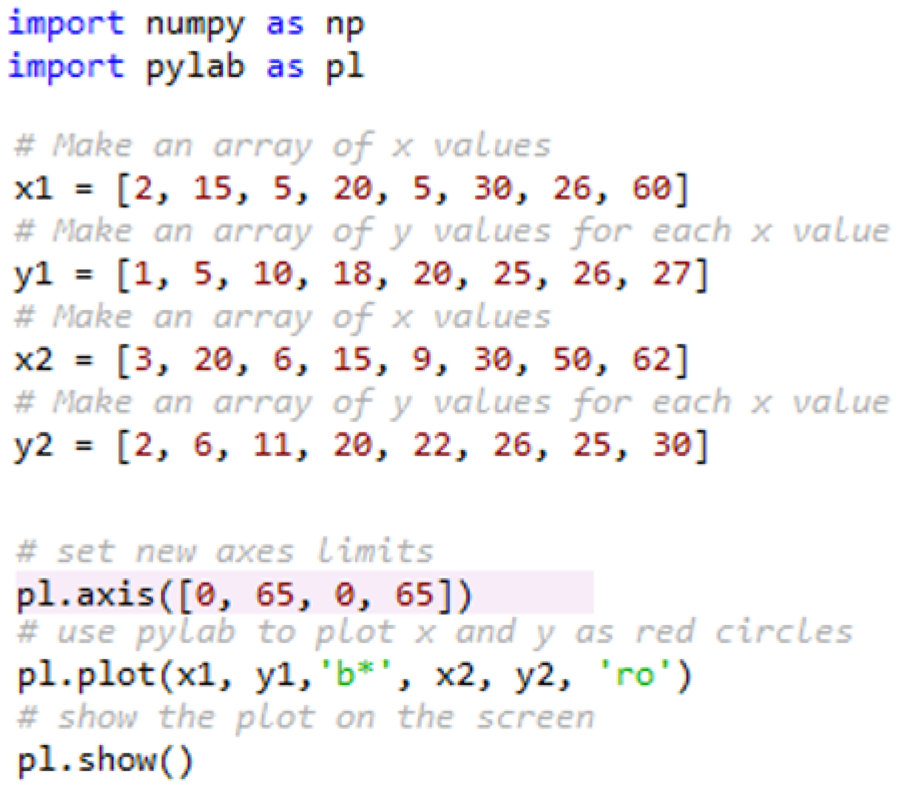


```
t = np.arange(0., 5., 0.2)
# red dashes, blue squares and green triangles
lines = plt.plot(t, t, 'b-', t, t**2, 'r-', t, t**3, 'g-', linewidth=2.0)
plt.setp(lines, color='r', linewidth=2.0)
plt.xlabel('Dummy data for x')
plt.ylabel('Dummy data for y')
plt.title('An example graph')
plt.text(1, 80, 'Lew is good at graphs')
plt.setp(lines, 'color', 'r', 'linewidth', 2.0)
plt.savefig('test.png')
plt.show()
```

Task – Scatter Plot

Let’s write a Python program to plot quantities which have an x and y position; a scatter graph.





```
import numpy as np
import pylab as pl

# Make an array of x values
x1 = [2, 15, 5, 20, 5, 30, 26, 60]
# Make an array of y values for each x value
y1 = [1, 5, 10, 18, 20, 25, 26, 27]
# Make an array of x values
x2 = [3, 20, 6, 15, 9, 30, 50, 62]
# Make an array of y values for each x value
y2 = [2, 6, 11, 20, 22, 26, 25, 30]


# set new axes limits
pl.axis([0, 65, 0, 65])
# use pylab to plot x and y as red circles
pl.plot(x1, y1,'b*', x2, y2, 'ro')
# show the plot on the screen
pl.show()
```