Pyplot Tutorial:

matplotlib.pyplot is a collection of command style functions that make matplotlib work like MATLAB.

Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

In [2]:

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

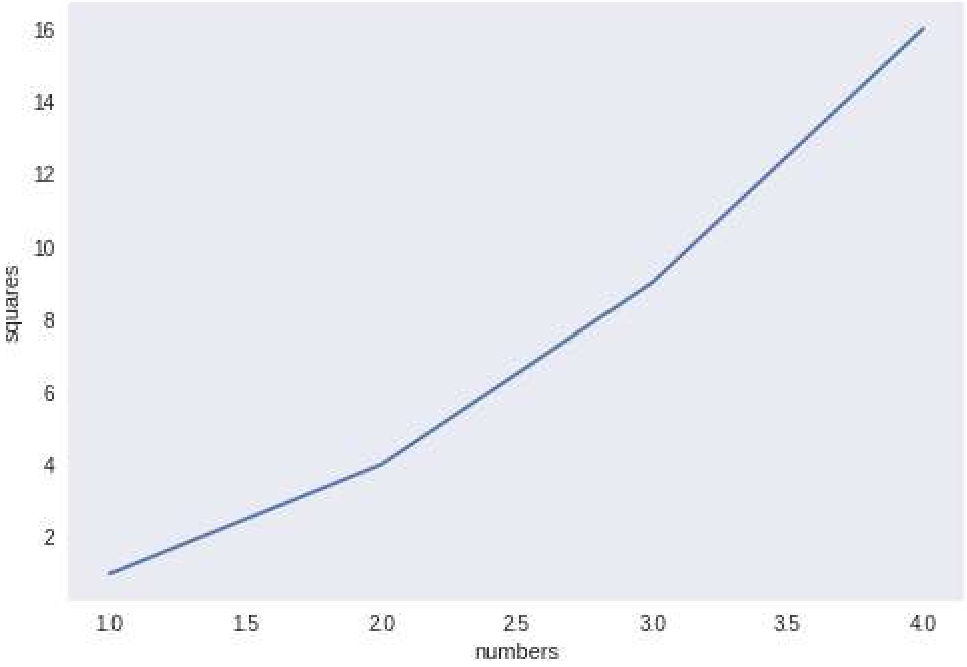
In [ ]:

```
plt.plot([2,4, 6, 4])
plt.ylabel("Numbers")
plt.xlabel('Indices')
plt.title('MyPlot')
plt.show()
```

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].

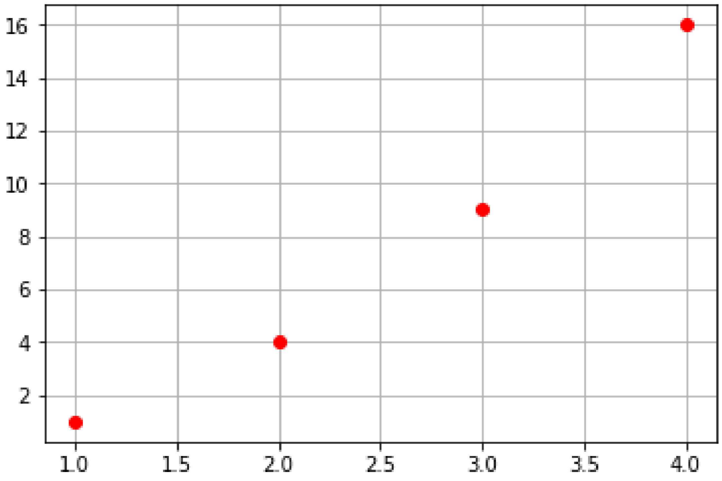
plot x versus y

In [ ]:
```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.ylabel('squares')
plt.xlabel('numbers')
plt.grid() # grid on

plt.show()
```



For every x, y pair of arguments, there is an optional third argument which is the **format string** that indicates the color and line type of the plot.

In [ ]:
```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.grid()

plt.show()
```



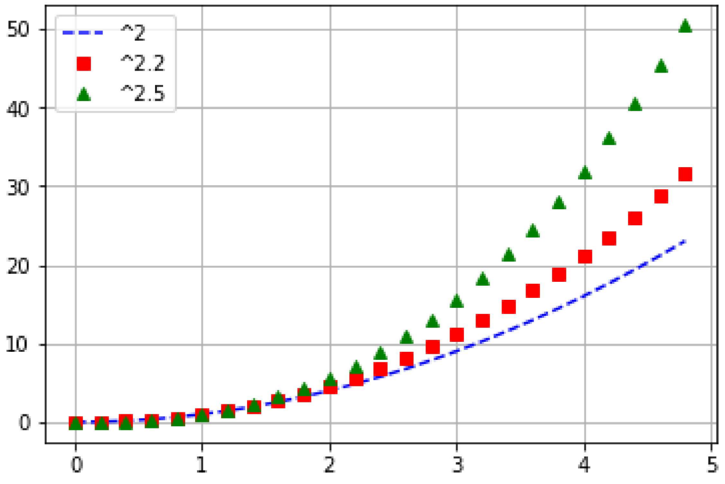
If matplotlib were limited to working with lists, it would be fairly useless for numeric processing. Generally, you will use **numpy arrays**. In fact, all sequences are converted to numpy arrays internally.

In [ ]:

```
import numpy as np
```

In [ ]:

```
import numpy as np
t = np.arange(0., 5., 0.2)

#blue dashes, red squares and green triangles
plt.plot(t, t**2, 'b--', label='^2')#   'rs',   'g^')
plt.plot(t,t**2.2, 'rs', label='^2.2')
plt.plot(t, t**2.5, 'g^', label='^2.5')
plt.grid()
plt.legend() # add legend based on line labels
plt.show()
```

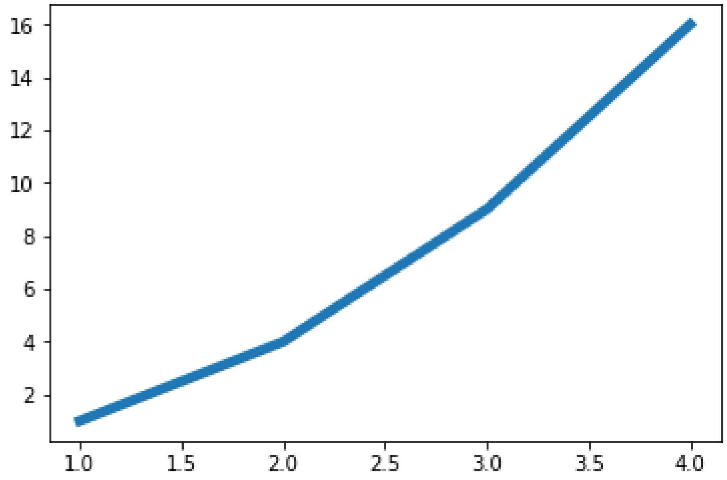


Controlling line properties

use keyword args

In [ ]:

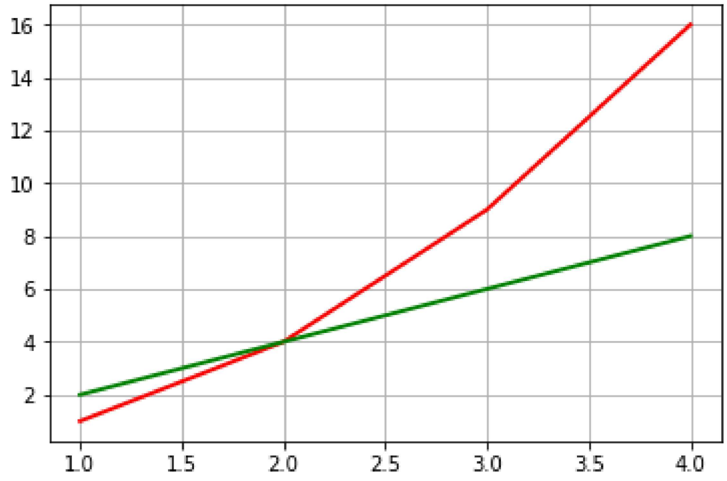
```
x = [1, 2, 3, 4]
y = [1, 4, 9, 16]
plt.plot(x, y, linewidth=5.0)
plt.show()
```



use the setp()

In [ ]:

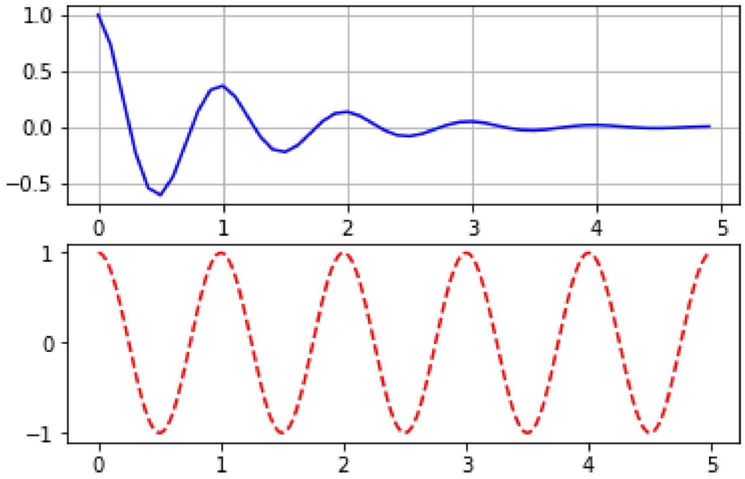
```
x1 = [1, 2, 3, 4]
y1 = [1, 4, 9, 16]
x2 = [1, 2, 3, 4]
y2 = [2, 4, 6, 8]
lines = plt.plot(x1, y1, x2, y2)

# use keyword args
plt.setp(lines[0], color='r', linewidth=2.0)

# or MATLAB style string value pairs
plt.setp(lines[1], 'color', 'g', 'linewidth', 2.0)

plt.grid()
```



working with multiple figures and axes

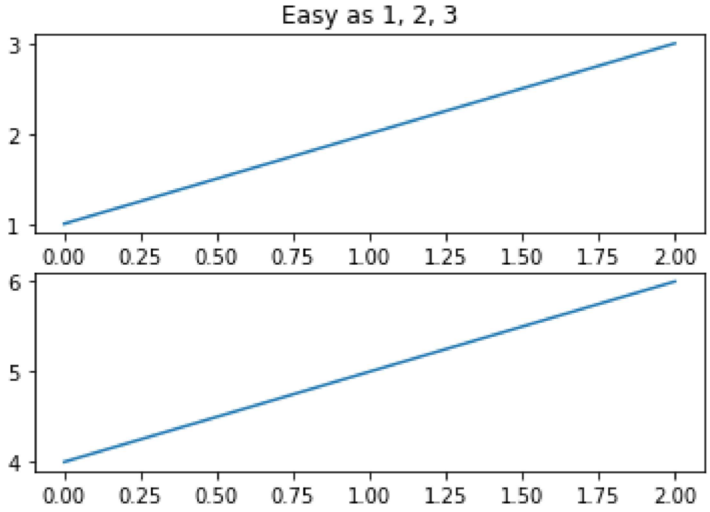
In [ ]:

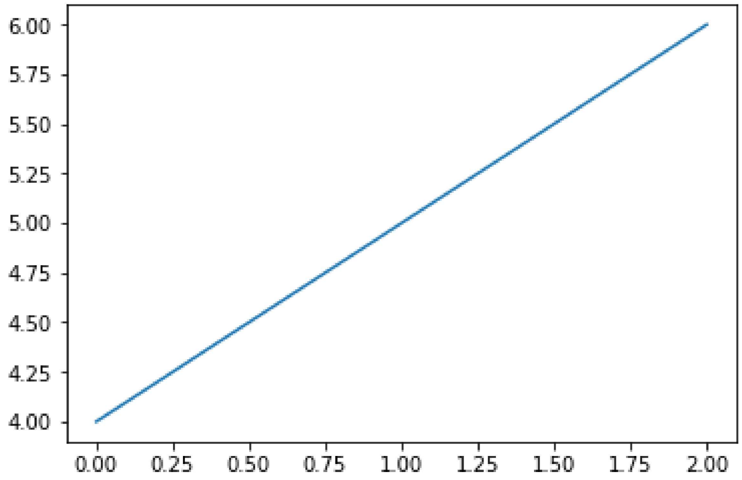
```
def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

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

plt.figure(1)
# The subplot() command specifies numrows, numcols,
# fignum where fignum ranges from 1 to numrows*numcols.
plt.subplot(211)
plt.grid()
plt.plot(t1, f(t1), 'b-')

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



In [ ]:

```
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
plt.show()
```


Easy as 1, 2, 3


6.00
5.75
5.50
5.25
5.00
4.75
4.50
4.25
4.00
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00