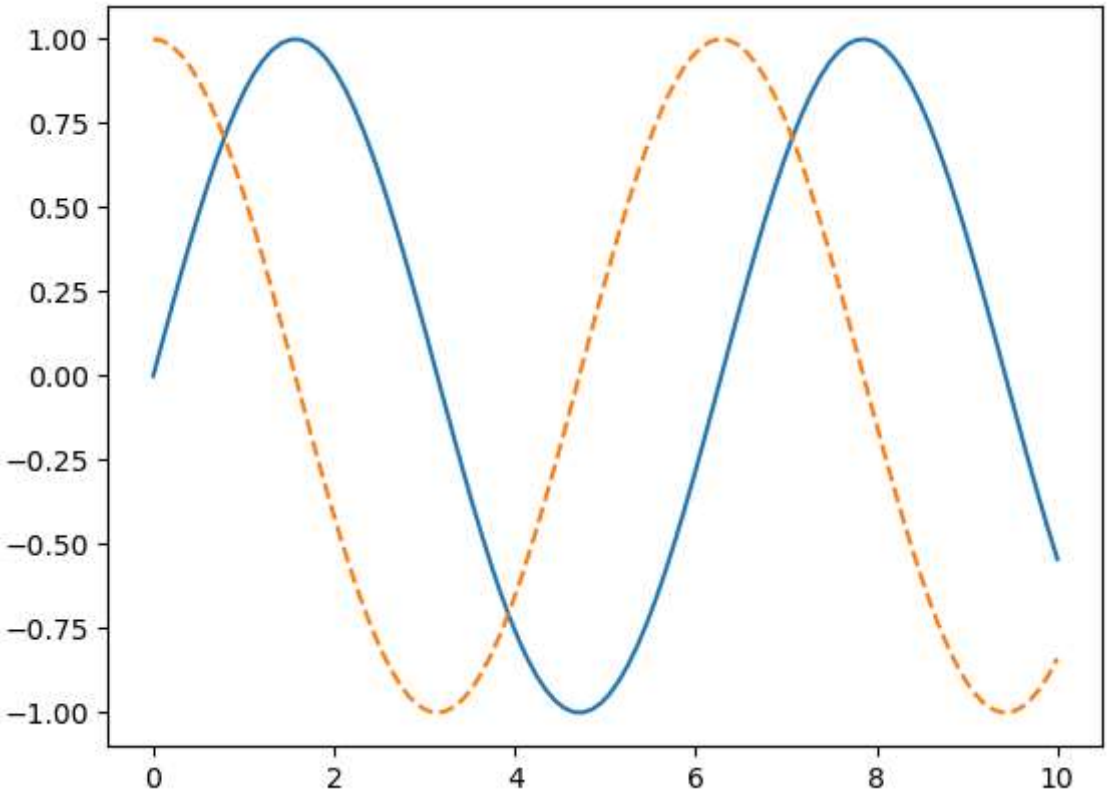
Matplotlib

Matplotlib, the basic data visualization tool of Python programming language.

In [4]:
```
# Import dependencies

import numpy as np
import pandas as pd
```

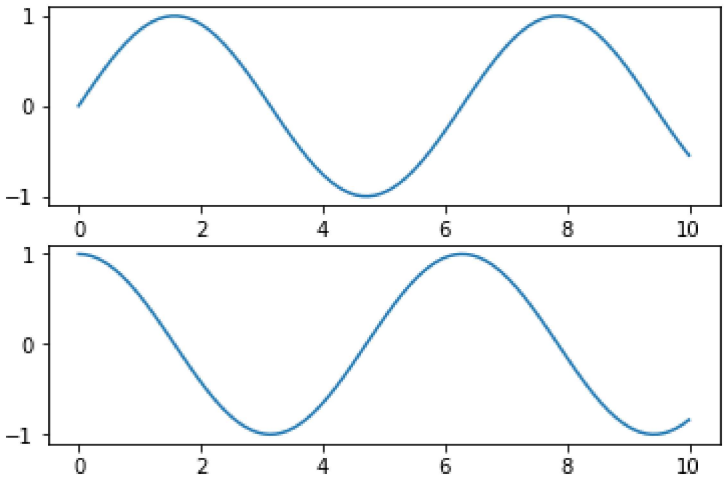
In [5]:
```
# Import Matplotlib
import matplotlib.pyplot as plt
```

In [4]:
```
%matplotlib inline


x1 = np.linspace(0, 10, 100)


# create a plot figure
fig = plt.figure()

plt.plot(x1, np.sin(x1), '-')
plt.plot(x1, np.cos(x1), '--');
```



Produces sine and cosine curves using Pyplot API.

In [4]:
```
# create a plot figure
plt.figure()


# create the first of two panels and set current axis
plt.subplot(2, 1, 1)   # (rows, columns, panel number)
plt.plot(x1, np.sin(x1))

# create the second of two panels and set current axis
plt.subplot(2, 1, 2)   # (rows, columns, panel number)
plt.plot(x1, np.cos(x1));
```

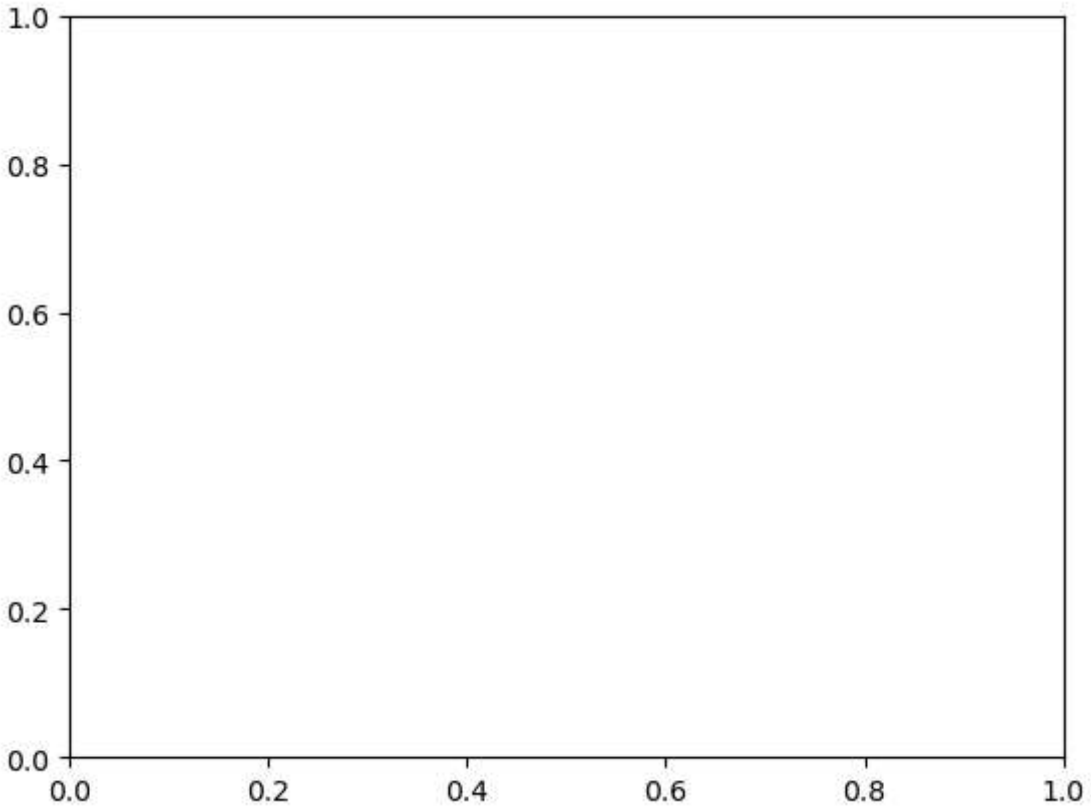


In [4]:
```
# get current figure information

print(plt.gcf())
```

```
Figure(640x480)
<Figure size 640x480 with 0 Axes>
```

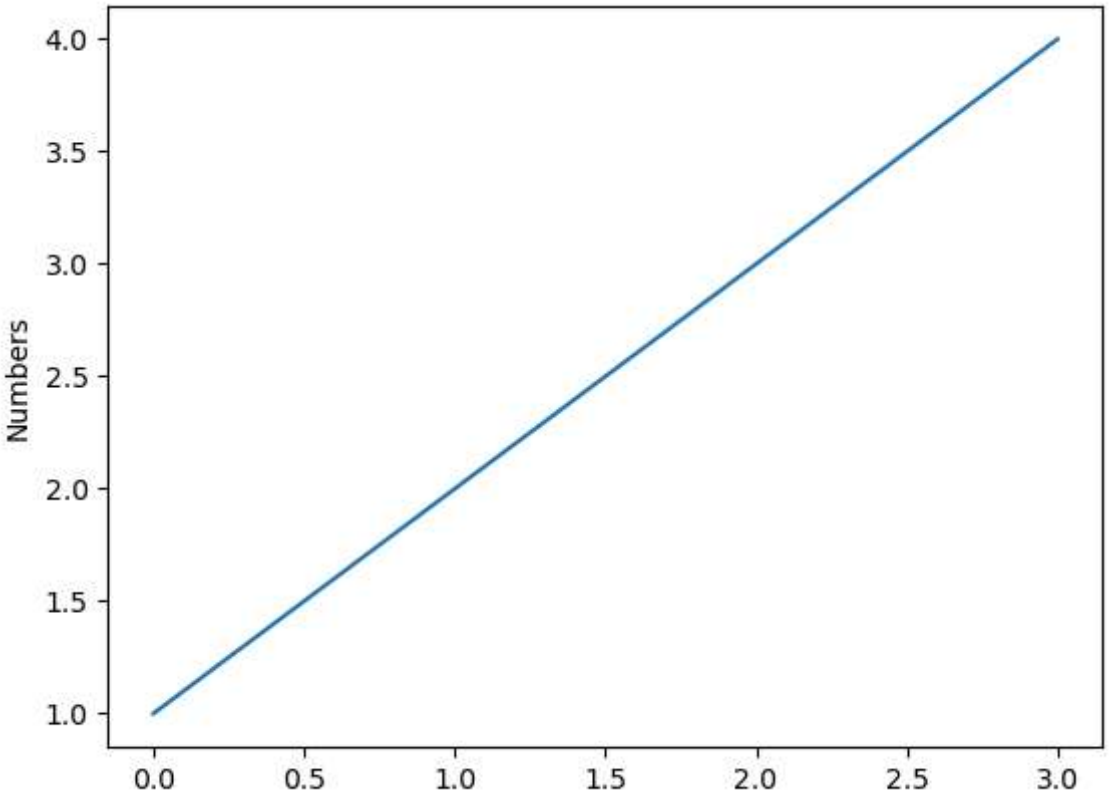
In [5]:
```
# get current axis information

print(plt.gca())
```

```
Axes(0.125,0.11;0.775x0.77)
```

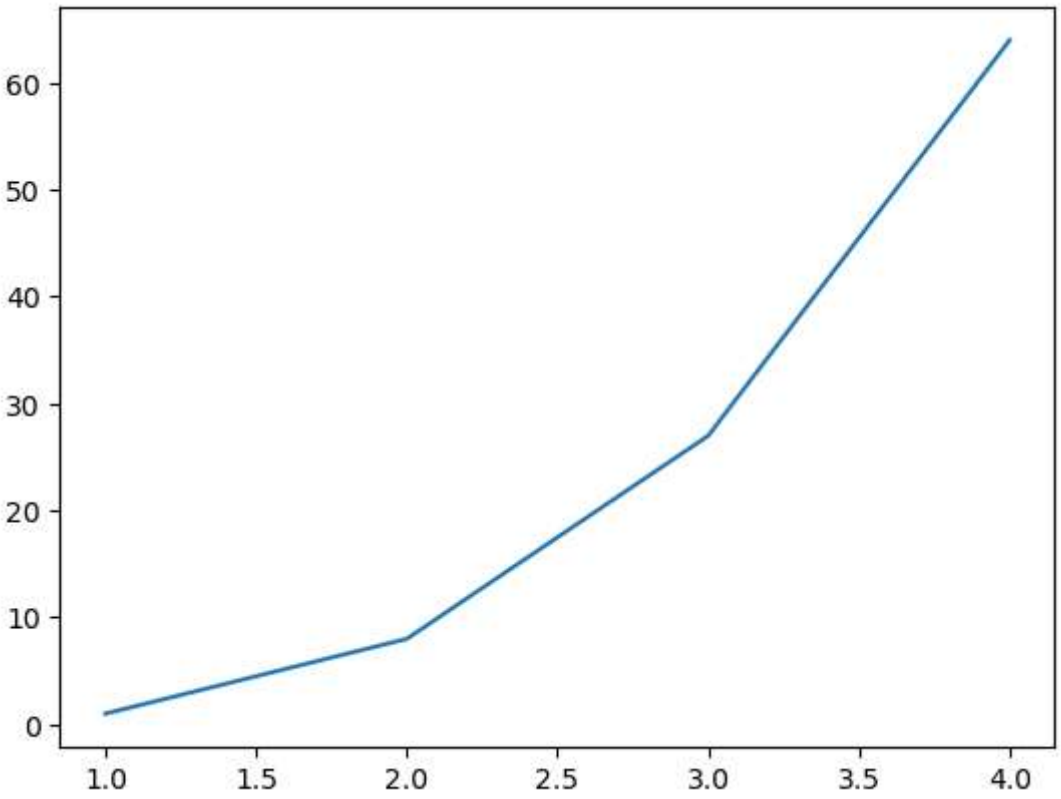


Visualization with Pyplot

In [8]:
```
plt.plot([1, 2, 3, 4])
plt.ylabel('Numbers')
plt.show()
```

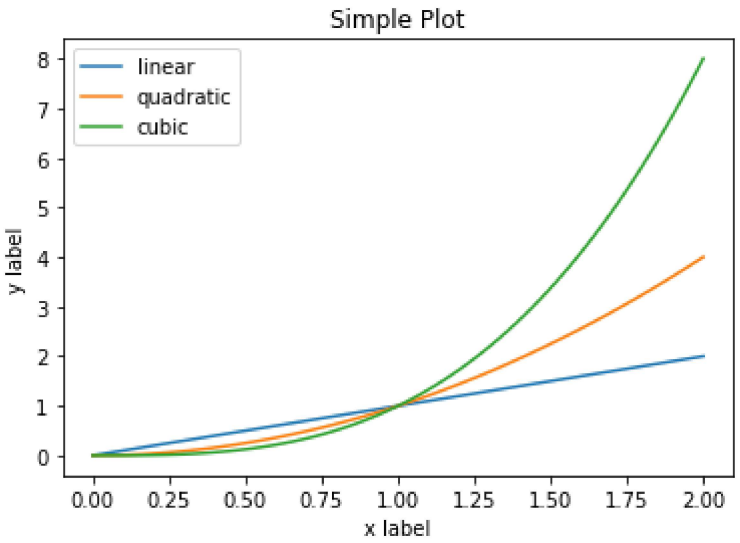


In [3]:
```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4], [1, 8, 27, 64])
plt.show()
```



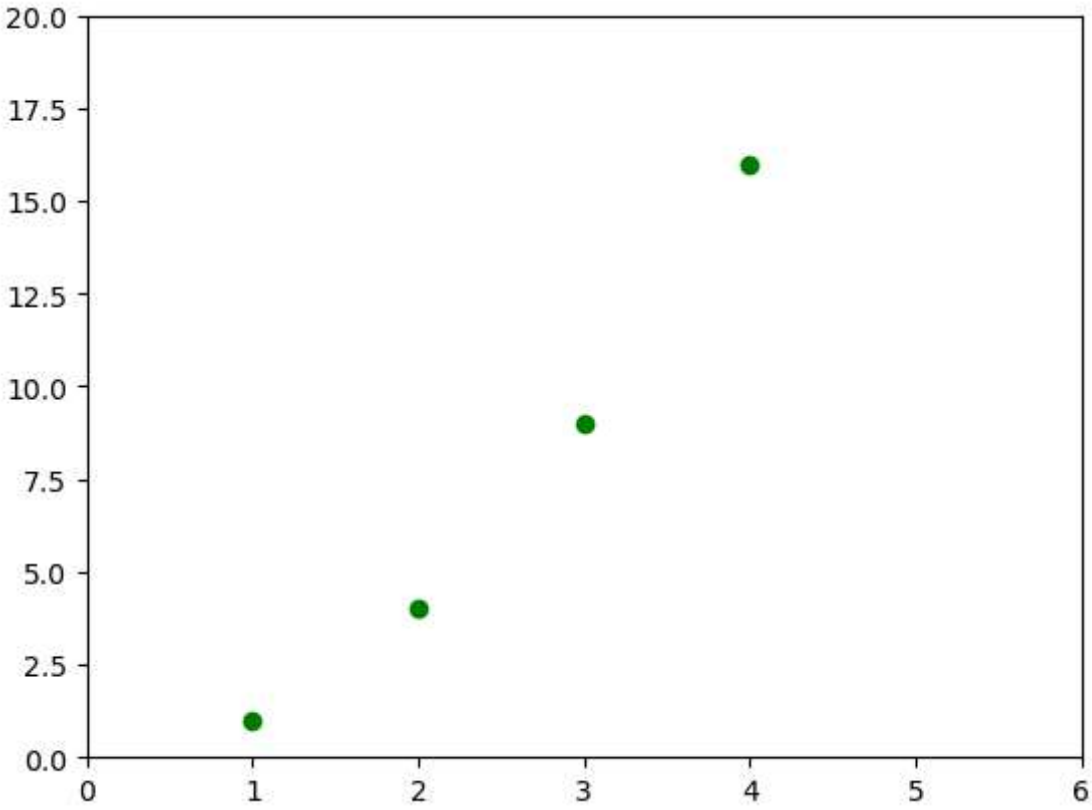
State-machine interface

In [9]:
```
x = np.linspace(0, 2, 100)

plt.plot(x, x, label='linear')
plt.plot(x, x**2, label='quadratic')
plt.plot(x, x**3, label='cubic')

plt.xlabel('x label')
plt.ylabel('y label')

plt.title("Simple Plot")

plt.legend()

plt.show()
```



Formatting the style of plot

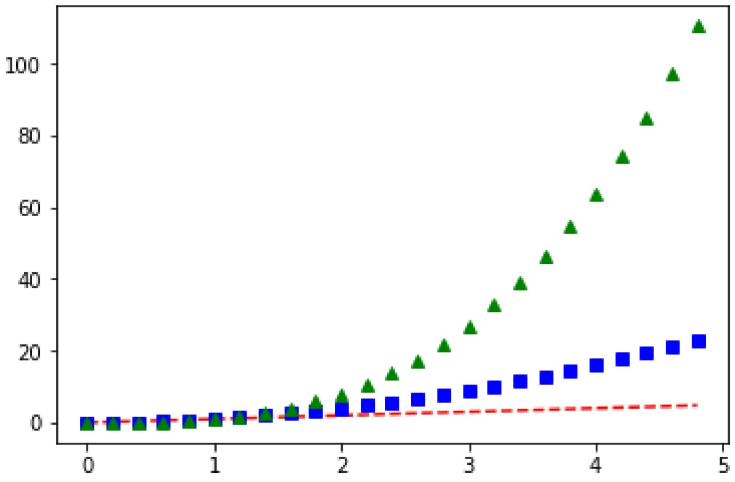
In [7]:
```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'go')
plt.axis([0, 6, 0, 20])
plt.show()
```



The **axis()** command in the example above takes a list of [xmin, xmax, ymin, ymax] and specifies the viewport of the axes.

Working with NumPy arrays

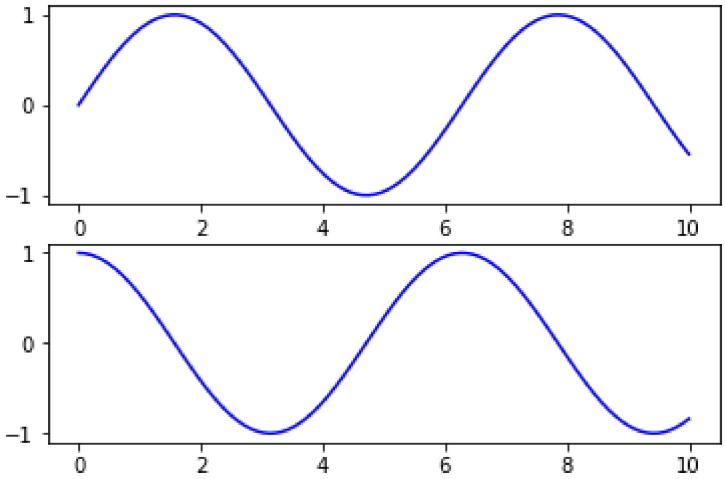
In [11]:
```
# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```



Object-Oriented API

The following code produces sine and cosine curves using Object-Oriented API.

In [12]:
```
# First create a grid of plots
# ax will be an array of two Axes objects
fig, ax = plt.subplots(2)


# Call plot() method on the appropriate object
ax[0].plot(x1, np.sin(x1), 'b-')
ax[1].plot(x1, np.cos(x1), 'b-');
```



Objects and Reference

In [13]:
```
fig = plt.figure()

x2 = np.linspace(0, 5, 10)
y2 = x2 ** 2

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

axes.plot(x2, y2, 'r')

axes.set_xlabel('x2')
axes.set_ylabel('y2')
axes.set_title('title');
```

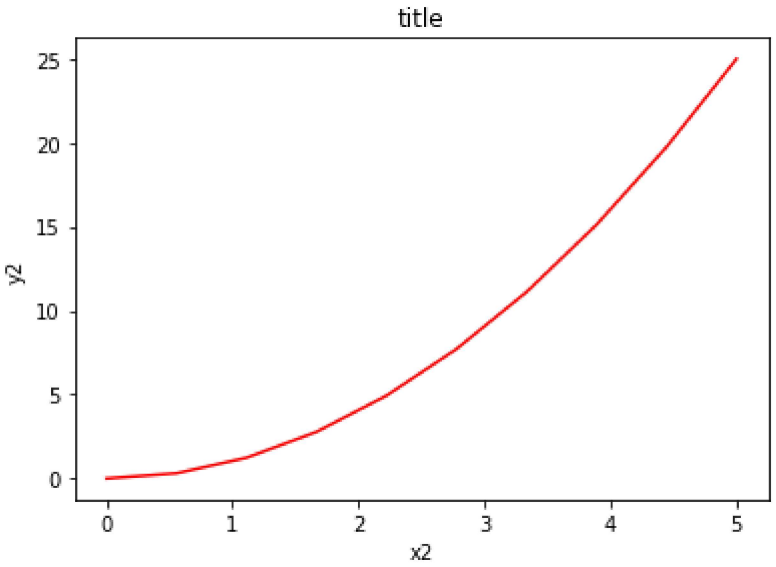
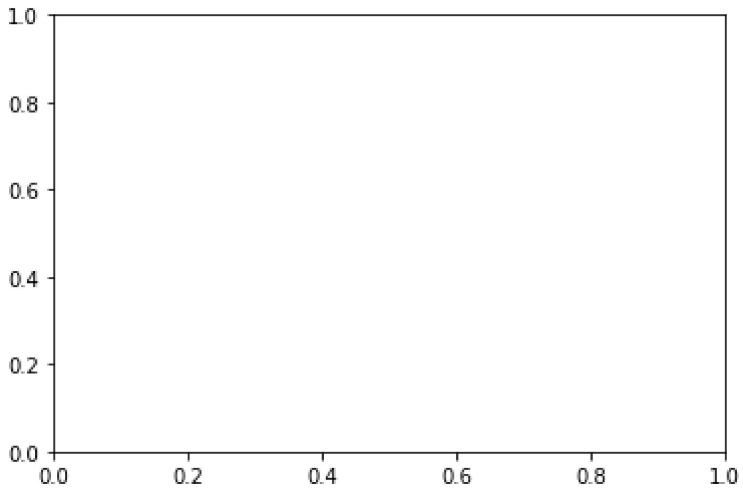


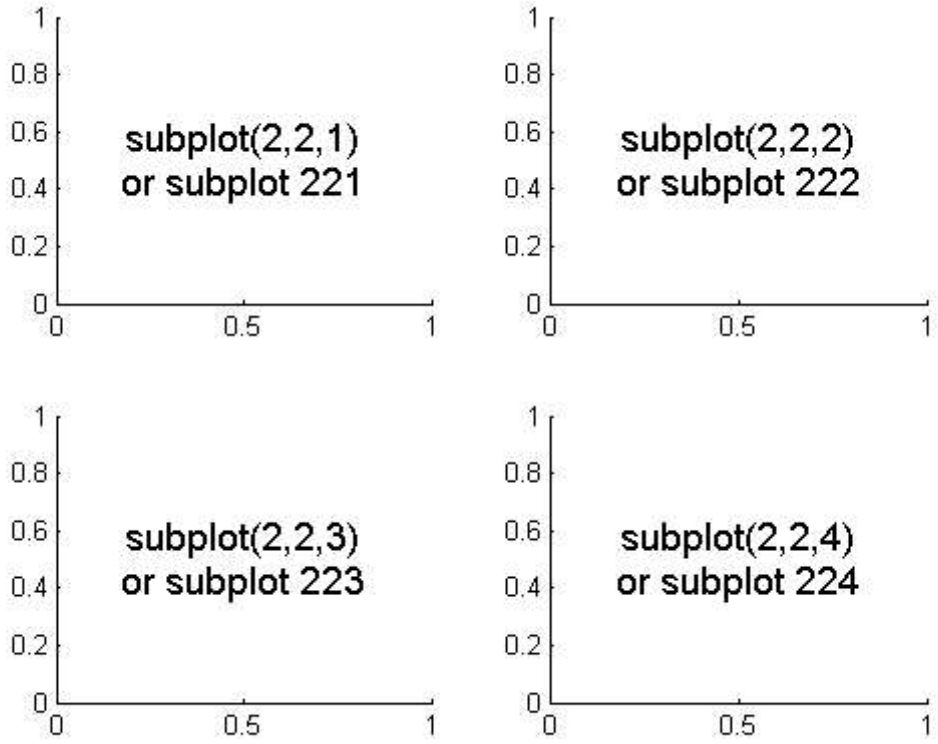
Figure and Axes

In [14]:
```
fig = plt.figure()

ax = plt.axes()
```



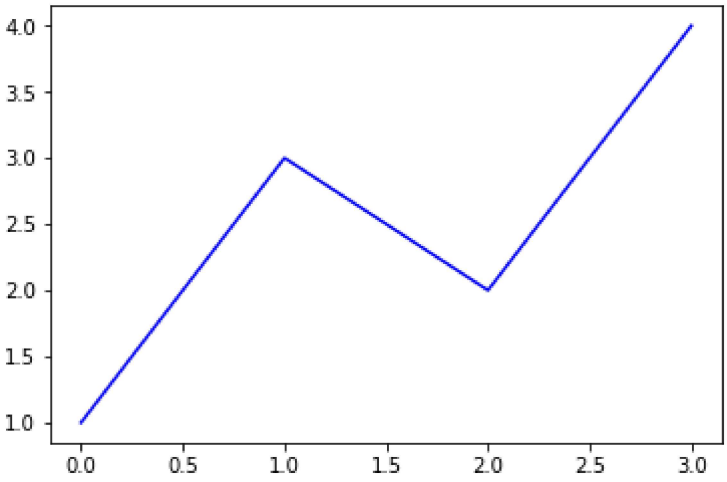
10. Figure and Subplots

The above command result in creation of subplots. The diagrammatic representation of subplots are as follows:-

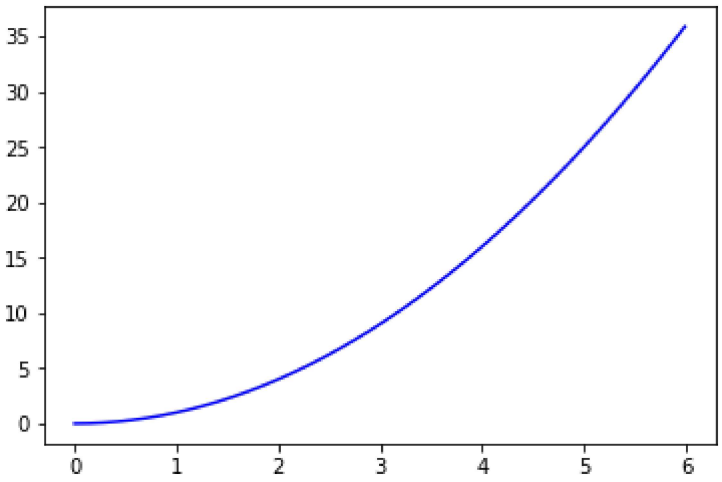


First plot with Matplotlib

In [15]:
```
plt.plot([1, 3, 2, 4], 'b-')

plt.show( )
```

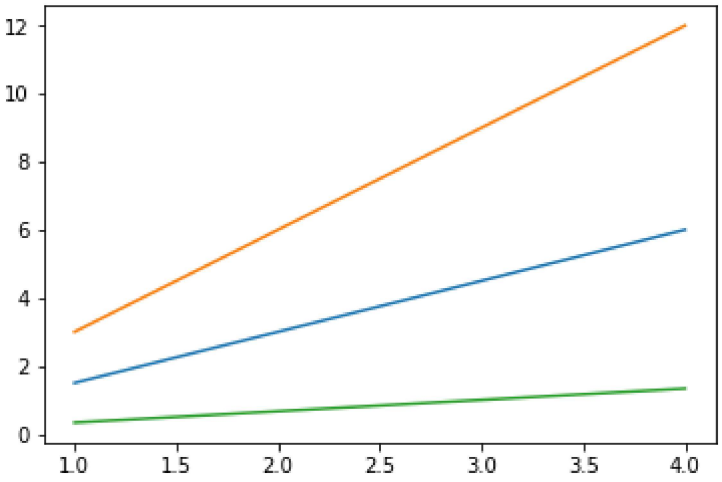


In [16]:
```
x3 = np.arange(0.0, 6.0, 0.01)

plt.plot(x3, [xi**2 for xi in x3], 'b-')

plt.show()
```



Multiline Plots

In [17]:
```
x4 = range(1, 5)

plt.plot(x4, [xi*1.5 for xi in x4])

plt.plot(x4, [xi*3 for xi in x4])

plt.plot(x4, [xi/3.0 for xi in x4])

plt.show()
```



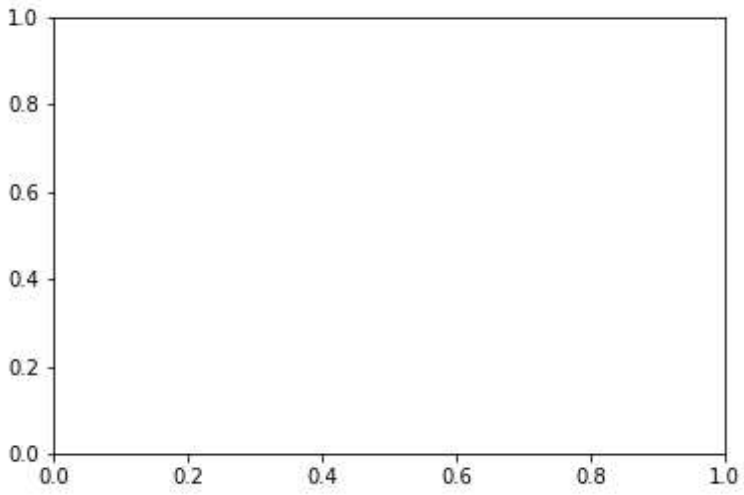
Parts of a Plot

In [18]:
```
# Saving the figure

fig.savefig('plot1.png')
```

In [19]:
```
# Explore the contents of figure

from IPython.display import Image
```

```
Image('plot1.png')
```

Out[19]:

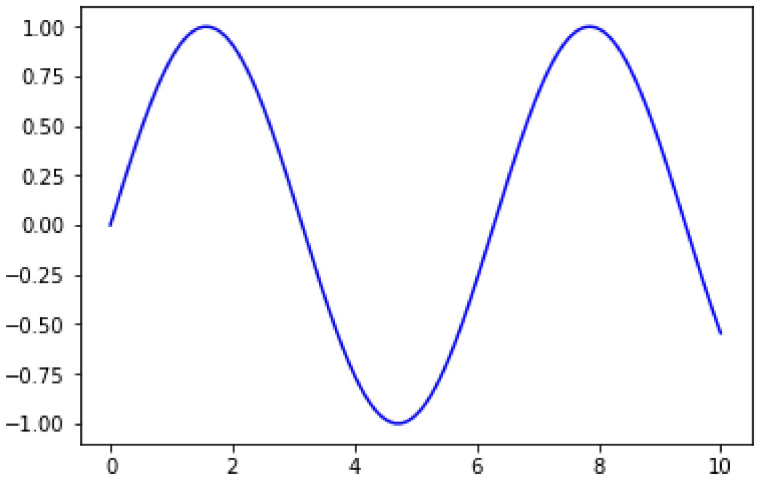


In [20]:
```
# Explore supported file formats

fig.canvas.get_supported_filetypes()
```

Out[20]:
```
{'ps': 'Postscript',
 'eps': 'Encapsulated Postscript',
 'pdf': 'Portable Document Format',
 'pgf': 'PGF code for LaTeX',
 'png': 'Portable Network Graphics',
 'raw': 'Raw RGBA bitmap',
 'rgba': 'Raw RGBA bitmap',
 'svg': 'Scalable Vector Graphics',
 'svgz': 'Scalable Vector Graphics',
 'jpg': 'Joint Photographic Experts Group',
 'jpeg': 'Joint Photographic Experts Group',
 'tif': 'Tagged Image File Format',
 'tiff': 'Tagged Image File Format'}
```

Line Plot

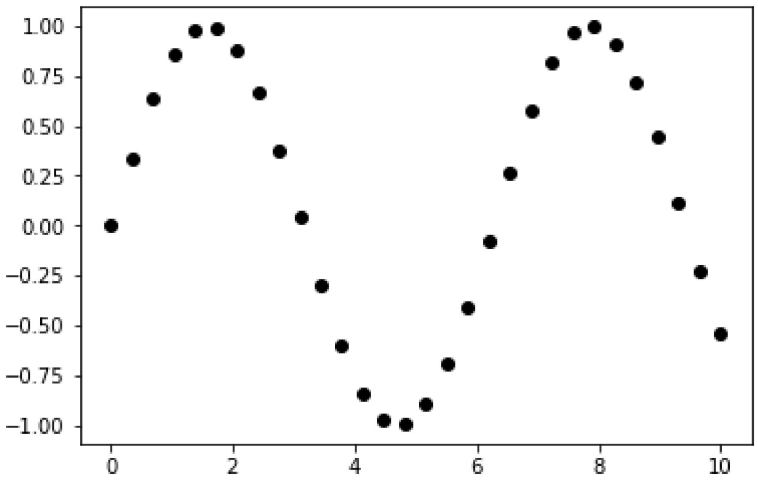
In [21]:
```
# Create figure and axes first
fig = plt.figure()

ax = plt.axes()

# Declare a variable x5
x5 = np.linspace(0, 10, 1000)


# Plot the sinusoid function
ax.plot(x5, np.sin(x5), 'b-');
```



Scatter Plot

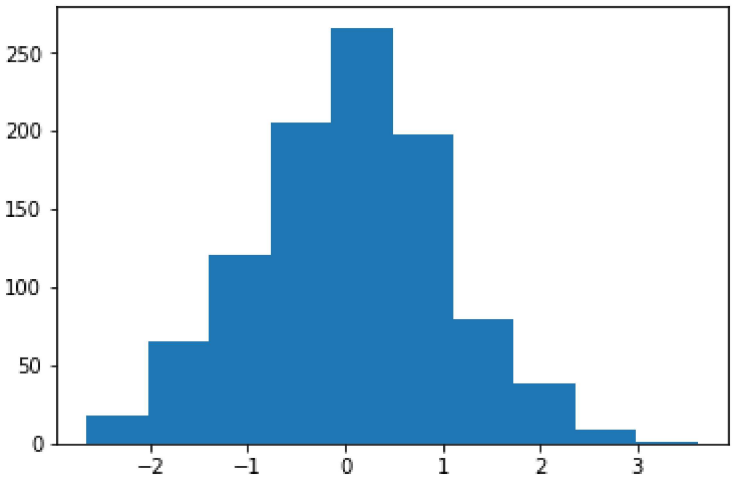
Scatter Plot with plt.plot()

In [22]:
```
x7 = np.linspace(0, 10, 30)

y7 = np.sin(x7)

plt.plot(x7, y7, 'o', color = 'black');
```



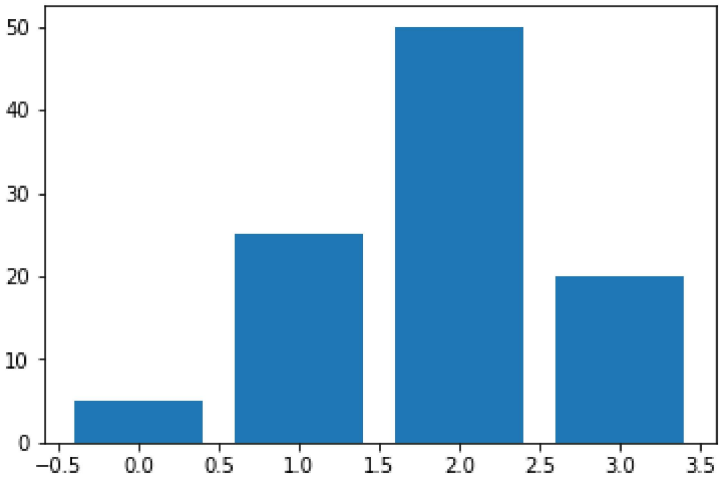
Histogram

In [23]:
```
data1 = np.random.randn(1000)

plt.hist(data1);
```



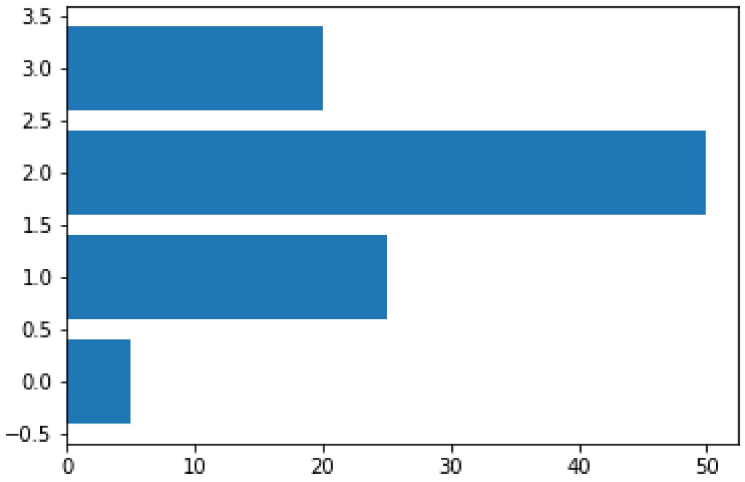
Bar Chart

In [24]:
```
data2 = [5. , 25. , 50. , 20.]

plt.bar(range(len(data2)), data2)

plt.show()
```



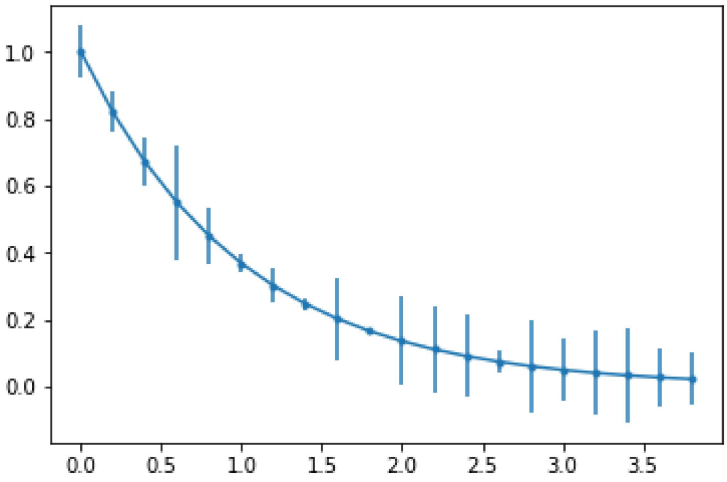
Horizontal Bar Chart

In [25]:
```
data2 = [5. , 25. , 50. , 20.]

plt.barh(range(len(data2)), data2)

plt.show()
```



Error Bar Chart

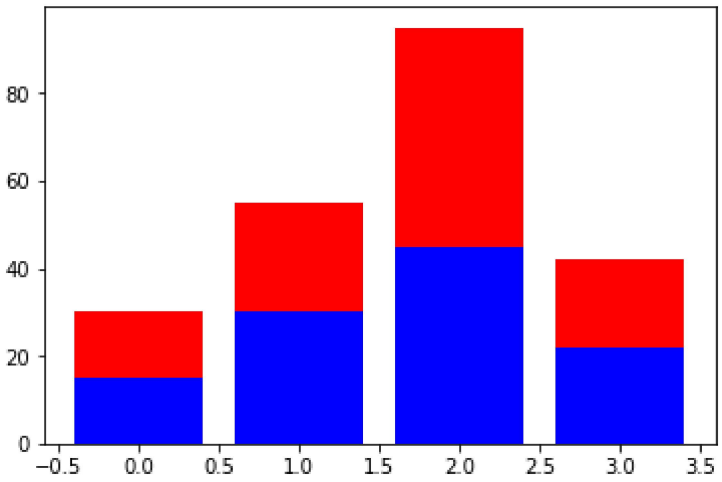
In [26]:
```
x9 = np.arange(0, 4, 0.2)

y9 = np.exp(-x9)

e1 = 0.1 * np.abs(np.random.randn(len(y9)))

plt.errorbar(x9, y9, yerr = e1, fmt = '.-')

plt.show();
```



Stacked Bar Chart

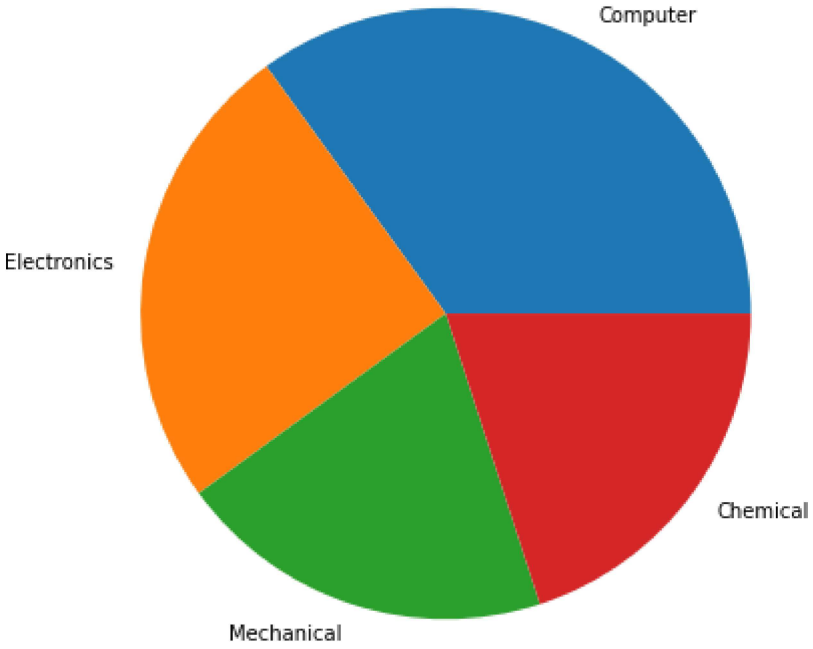
In [27]:
```
A = [15., 30., 45., 22.]

B = [15., 25., 50., 20.]

z2 = range(4)

plt.bar(z2, A, color = 'b')
```

```
plt.bar(z2, B, color = 'r', bottom = A)

plt.show()
```



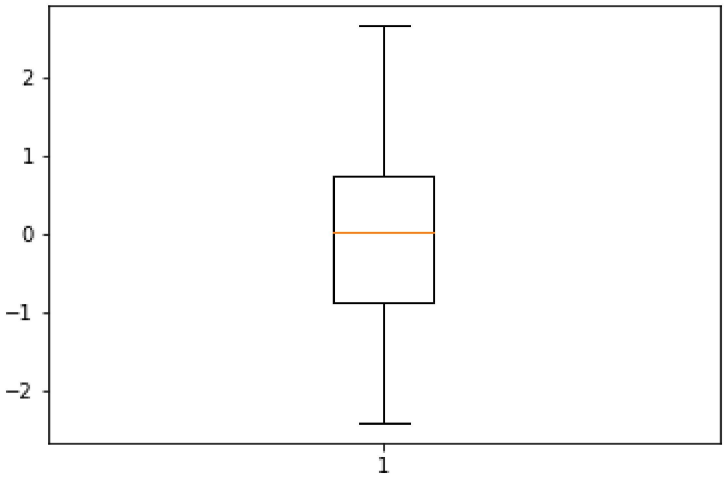
Pie Chart

In [28]:
```
plt.figure(figsize=(7,7))

x10 = [35, 25, 20, 20]

labels = ['Computer', 'Electronics', 'Mechanical', 'Chemical']

plt.pie(x10, labels=labels);

plt.show()
```



Boxplot

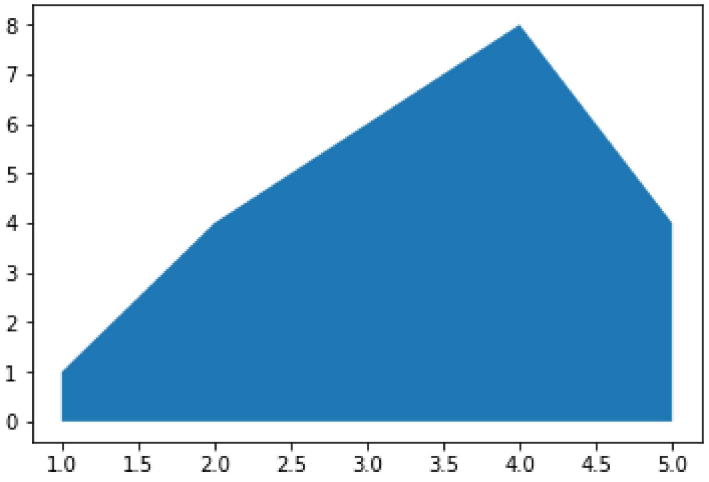
In [29]:
```
data3 = np.random.randn(100)

plt.boxplot(data3)

plt.show();
```



Area Chart

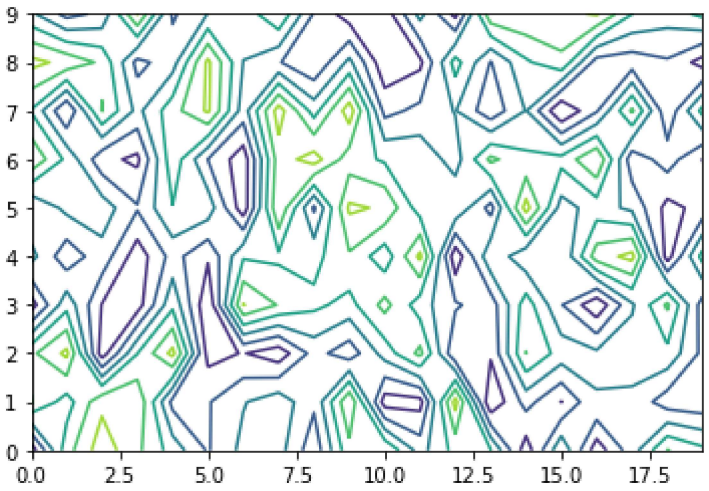
In [30]:
```
# Create some data
x12 = range(1, 6)
y12 = [1, 4, 6, 8, 4]
```

```
# Area plot
plt.fill_between(x12, y12)
plt.show()
```



Contour Plot

In [31]:
```
# Create a matrix
matrix1 = np.random.rand(10, 20)

cp = plt.contour(matrix1)

plt.show()
```



Styles with Matplotlib Plots

In [32]:
```
# View list of all available styles

print(plt.style.available)
```

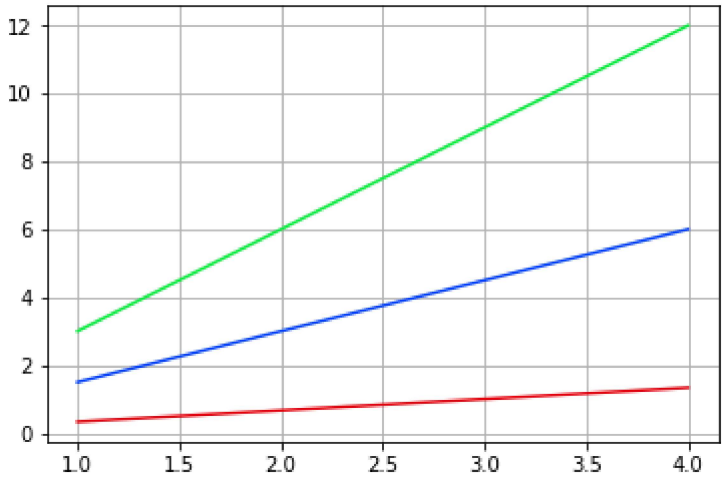
```
['classic', 'seaborn-poster', 'dark_background', 'seaborn-ticks', 'seaborn-muted',
'seaborn-deep', 'fivethirtyeight', 'grayscale', 'seaborn-notebook', 'ggplot', 'bmh',
'seaborn-paper', 'fast', 'tableau-colorblind10', 'seaborn-bright', 'seaborn-colorbli
nd', 'seaborn-pastel', 'seaborn', 'seaborn-talk', 'seaborn-white', 'seaborn-dark',
'seaborn-whitegrid', 'seaborn-dark-palette', 'Solarize_Light2', '_classic_test', 'se
aborn-darkgrid']
```

In [33]:
```
# Set styles for plots

plt.style.use('seaborn-bright')
```

I have set the **seaborn-bright** style for plots. So, the plot uses the **seaborn-bright** Matplotlib style for plots.

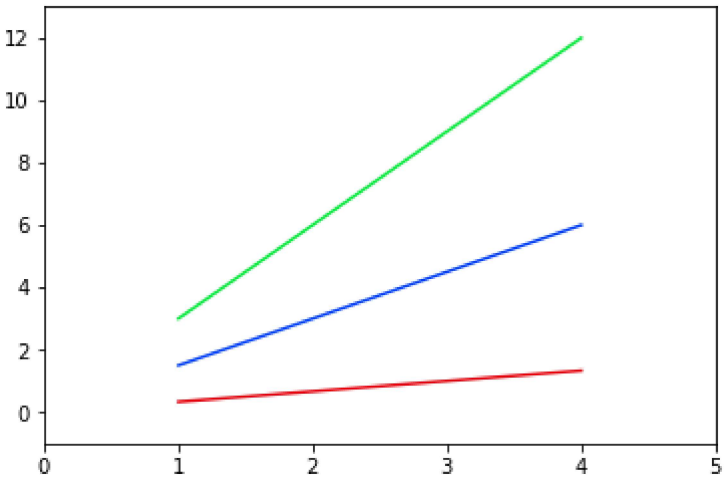
Adding a grid

In [34]:
```
x15 = np.arange(1, 5)

plt.plot(x15, x15*1.5, x15, x15*3.0, x15, x15/3.0)

plt.grid(True)

plt.show()
```



Handling axes

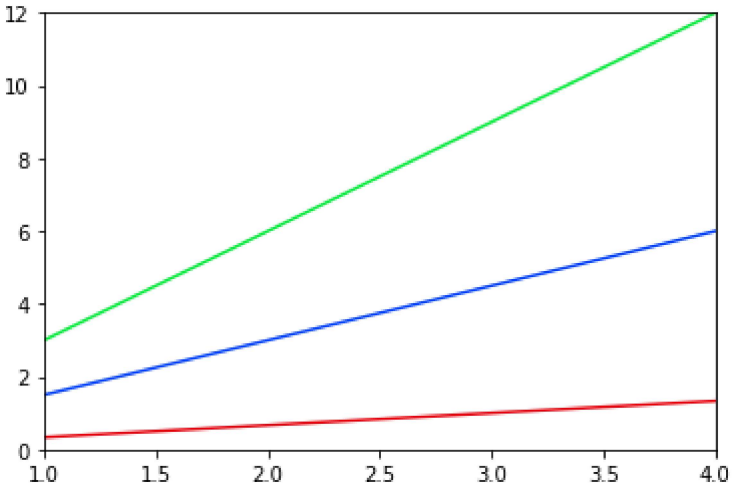
In [35]:
```
x15 = np.arange(1, 5)

plt.plot(x15, x15*1.5, x15, x15*3.0, x15, x15/3.0)

plt.axis()   # shows the current axis limits values

plt.axis([0, 5, -1, 13])

plt.show()
```



We can see that we now have more space around the lines.

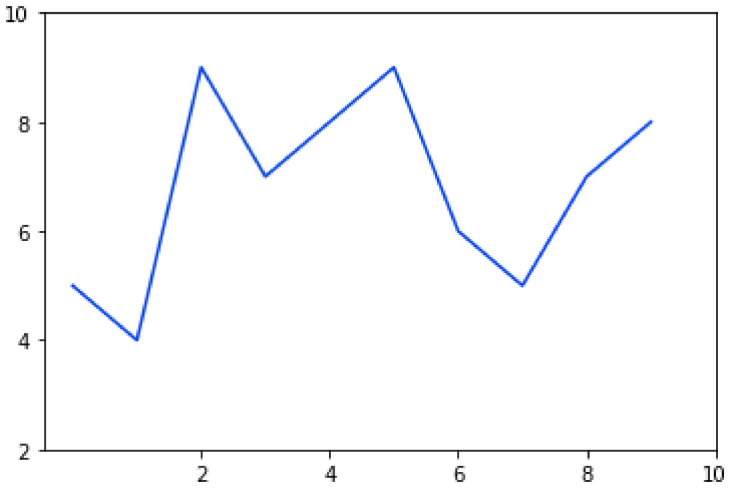
In [36]:
```
x15 = np.arange(1, 5)

plt.plot(x15, x15*1.5, x15, x15*3.0, x15, x15/3.0)

plt.xlim([1.0, 4.0])

plt.ylim([0.0, 12.0])
```

Out[36]: (0.0, 12.0)



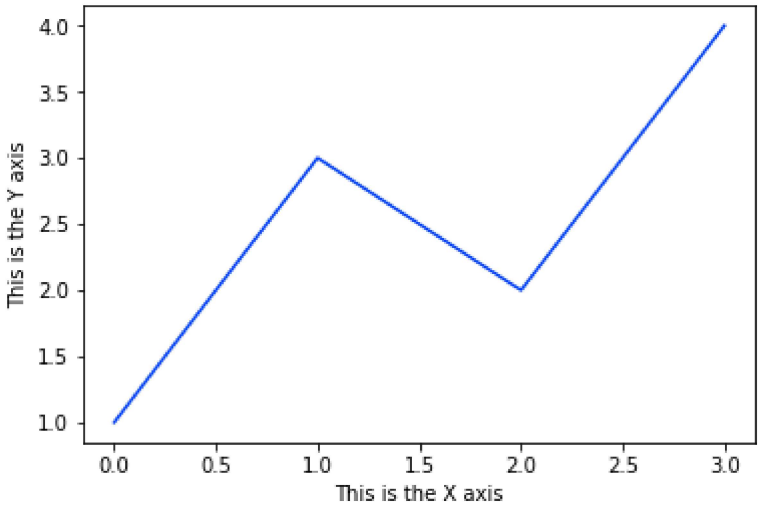
Handling X and Y ticks

In [37]:
```
u = [5, 4, 9, 7, 8, 9, 6, 5, 7, 8]

plt.plot(u)

plt.xticks([2, 4, 6, 8, 10])
plt.yticks([2, 4, 6, 8, 10])

plt.show()
```



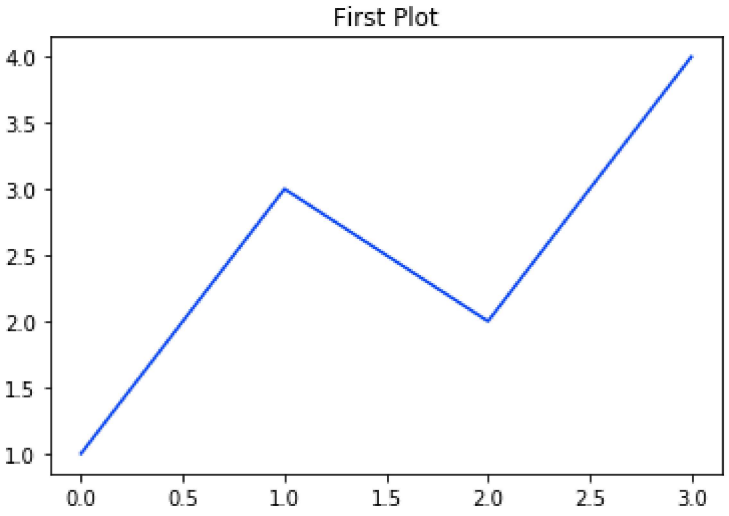
Adding labels

In [38]:
```
plt.plot([1, 3, 2, 4])

plt.xlabel('This is the X axis')

plt.ylabel('This is the Y axis')

plt.show()
```



Adding a title

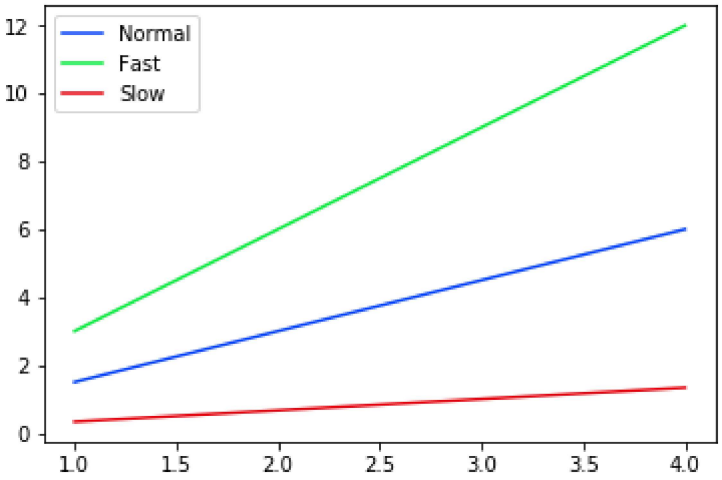
In [39]:
```
plt.plot([1, 3, 2, 4])

plt.title('First Plot')

plt.show()
```



The above plot displays the output of the previous code. The title `First Plot` is displayed on top of the plot.

Adding a legend

In [40]:
```
x15 = np.arange(1, 5)

fig, ax = plt.subplots()

ax.plot(x15, x15*1.5)
ax.plot(x15, x15*3.0)
ax.plot(x15, x15/3.0)

ax.legend(['Normal','Fast','Slow']);
```

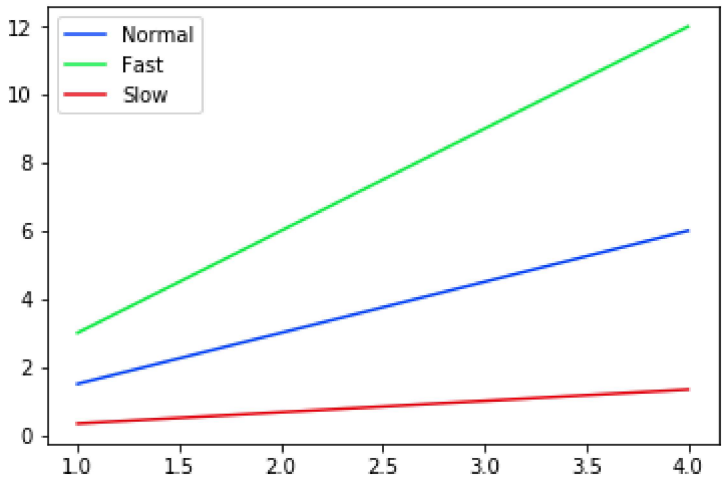


The above method follows the MATLAB API. It is prone to errors and unflexible if curves are added to or removed from the plot. It resulted in a wrongly labelled curve.

A better method is to use the **label** keyword argument when plots are added to the figure. Then we use the **legend** method without arguments to add the legend to the figure.

The advantage of this method is that if curves are added or removed from the figure, the legend is automatically updated accordingly. It can be achieved by executing the code below:-

In [41]:
```
x15 = np.arange(1, 5)

fig, ax = plt.subplots()

ax.plot(x15, x15*1.5, label='Normal')
ax.plot(x15, x15*3.0, label='Fast')
ax.plot(x15, x15/3.0, label='Slow')

ax.legend();
```



The **legend** function takes an optional keyword argument **loc**. It specifies the location of the legend to be drawn. The **loc** takes numerical codes for the various places the legend can be drawn. The most common **loc** values are as follows:-

ax.legend(loc=0) # let Matplotlib decide the optimal location

ax.legend(loc=1) # upper right corner

ax.legend(loc=2) # upper left corner

ax.legend(loc=3) # lower left corner

ax.legend(loc=4) # lower right corner

ax.legend(loc=5) # right

ax.legend(loc=6) # center left

ax.legend(loc=7) # center right

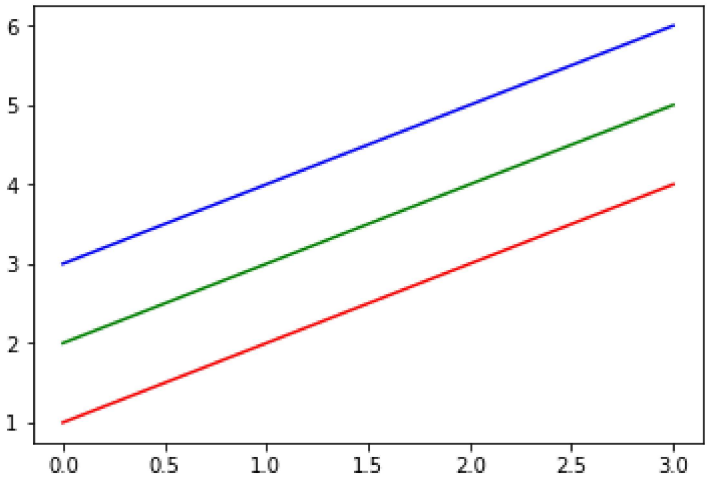
ax.legend(loc=8) # lower center

ax.legend(loc=9) # upper center

ax.legend(loc=10) # center

Control colours

In [42]:
```
x16 = np.arange(1, 5)

plt.plot(x16, 'r')
plt.plot(x16+1, 'g')
plt.plot(x16+2, 'b')

plt.show()
```



There are several ways to specify colours, other than by colour abbreviations:

• The full colour name, such as yellow

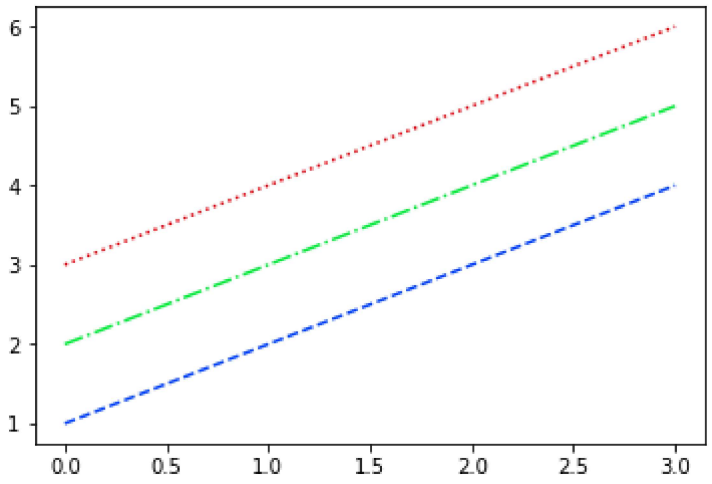
• Hexadecimal string such as ##FF00FF

• RGB tuples, for example (1, 0, 1)

• Grayscale intensity, in string format such as '0.7'.

Control line styles

In [43]:
```
x16 = np.arange(1, 5)

plt.plot(x16, '--', x16+1, '-.', x16+2, ':')

plt.show()
```



The above code snippet generates a blue dashed line, a green dash-dotted line and a red dotted line.

All the available line styles are available in the following table:

Style abbreviation Style

- `solid line`

-- dashed line

-. dash-dot line

: dotted line

Now, we can see the default format string for a single line plot is 'b-'.