Line Plots

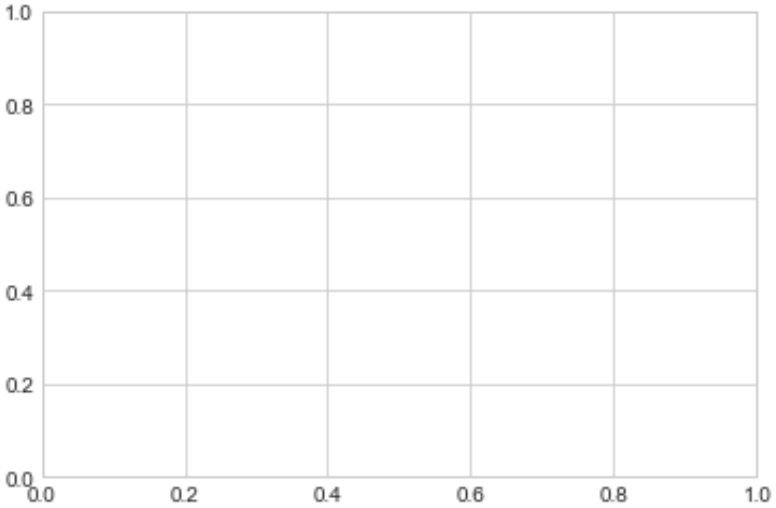
Necessary Settings

In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
```

Simple Form for Creating all Matplotlib Plots

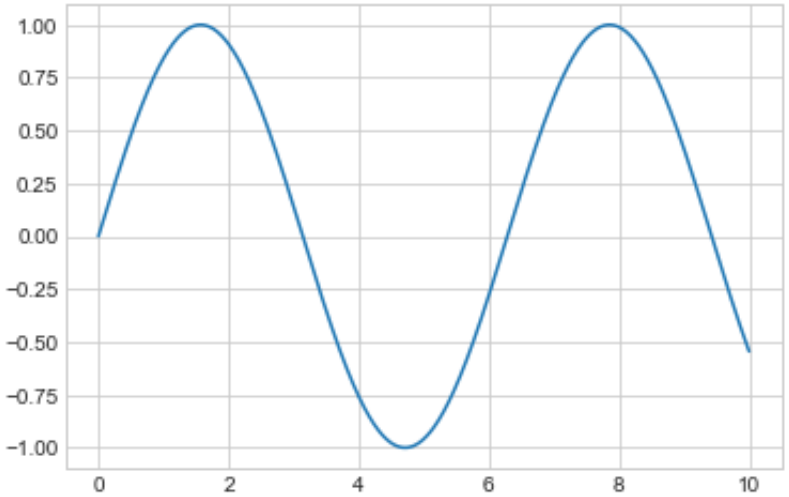
For all Matplotlib plots, we start by creating a figure and an axes. In their simplest form, a figure and axes can be created as follows:

In [2]:
```
fig = plt.figure()
ax = plt.axes()
```



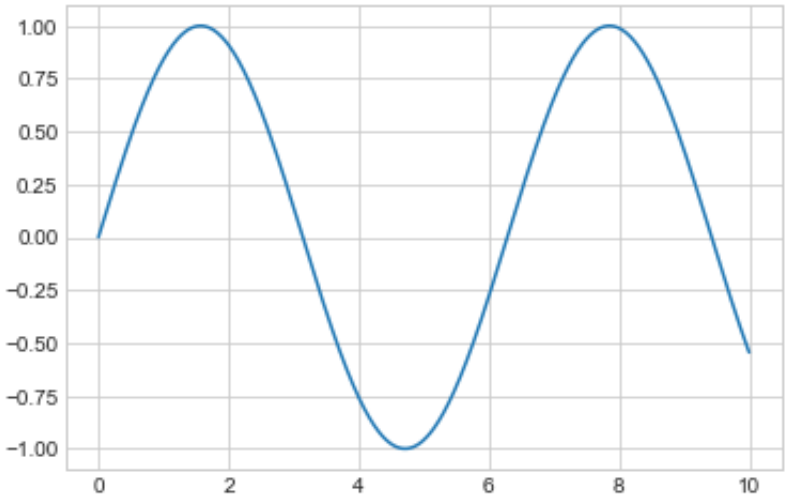
- In Matplotlib, the `fig` (an instance of the class `plt.Figure`) can be thought of as a single container that contains all the objects representing axes, graphics, text, and labels.
- The `ax` (an instance of the class `plt.Axes`) is a bounding box with ticks and labels, which will eventually contain the plot elements that make up our visualization.
- Once an axes is created, the `ax.plot` function can be used to plot some data.

In [3]:
```
fig = plt.figure()
ax = plt.axes()

x = np.linspace(0, 10, 1000)
ax.plot(x, np.sin(x));
```



Alternatively, we can use the pylab interface and let the figure and axes be created for us in the background.

In [8]:
```
plt.plot(x, np.sin(x));
```



If we want to create a single figure with multiple lines, we can simply call the `plot` function multiple times:

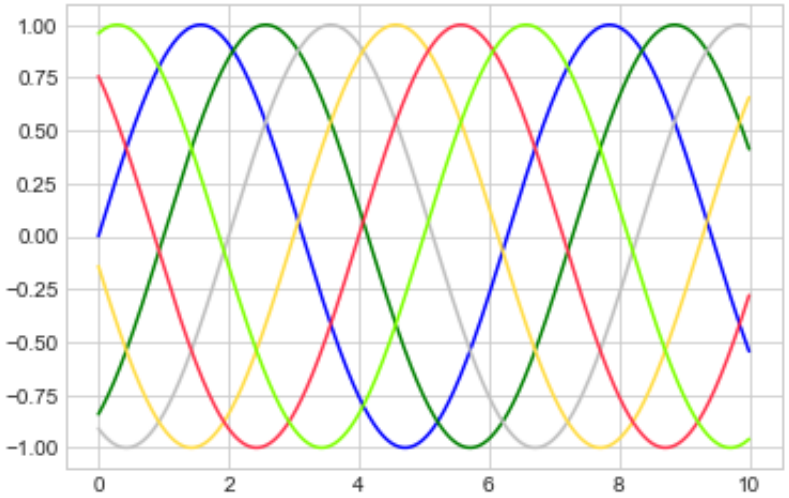
In [7]:
```
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x));
```

Line Colors and Styles

- The `plt.plot()` function takes additional arguments that can be used to specify these.
- To adjust the color, you can use the `color` keyword, which accepts a string argument representing virtually any imaginable color.
- The color can be specified in a variety of ways.
- If no color is specified, Matplotlib will automatically cycle through a set of default colors for multiple lines.

In [23]:

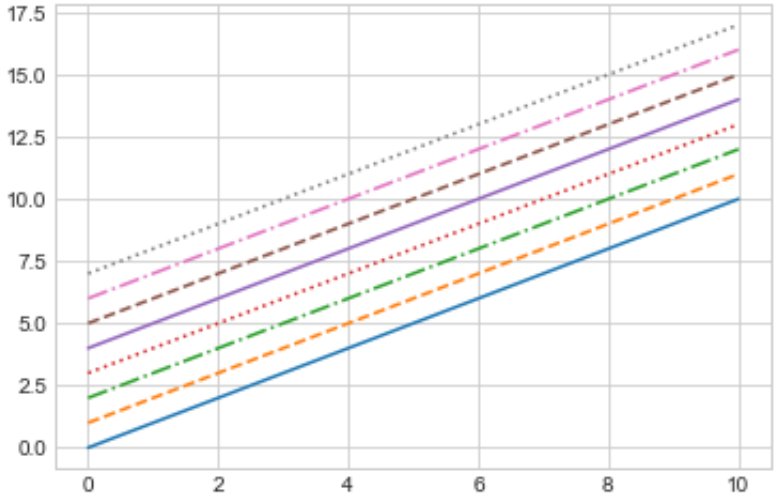
```
plt.plot(x, np.sin(x - 0), color='blue')        # specify color by name
plt.plot(x, np.sin(x - 1), color='g')           # short color code (rgbcmyk)
plt.plot(x, np.sin(x - 2), color='0.75')        # Grayscale between 0 and 1
plt.plot(x, np.sin(x - 3), color='#FFDD44')     # Hex code (RRGGBB from 00 to FF)
plt.plot(x, np.sin(x - 4), color=(1.0,0.2,0.3)) # RGB tuple, values 0 to 1
plt.plot(x, np.sin(x - 5), color='chartreuse'); # all HTML color names supported
```



- Similarly, the line style can be adjusted using the `linestyle` keyword.

In [25]:

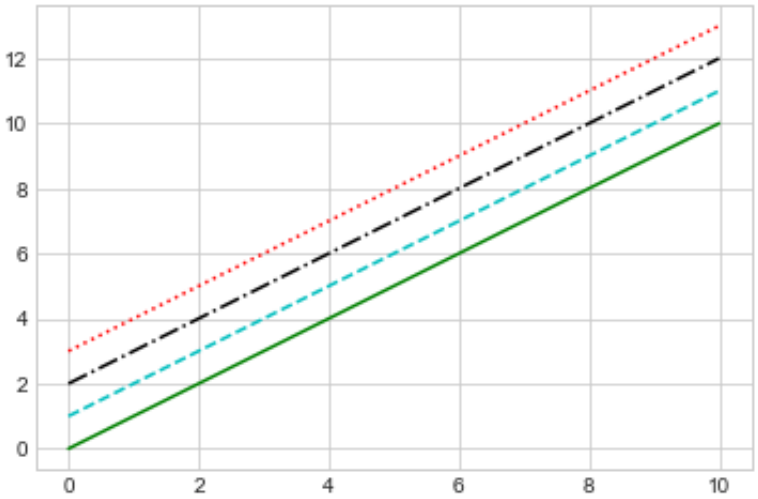
```
plt.plot(x, x + 0, linestyle='solid')
plt.plot(x, x + 1, linestyle='dashed')
plt.plot(x, x + 2, linestyle='dashdot')
plt.plot(x, x + 3, linestyle='dotted');

# For short, you can use the following codes:
plt.plot(x, x + 4, linestyle='-')  # solid
plt.plot(x, x + 5, linestyle='--') # dashed
plt.plot(x, x + 6, linestyle='-.') # dashdot
plt.plot(x, x + 7, linestyle=':');  # dotted
```



- (If you would like to be extremely terse) `linestyle` and `color` codes can be combined into a single non-keyword argument to the `plt.plot()` function.
- These single-character color codes reflect the standard abbreviations in the RGB (Red/Green/Blue) and CMYK (Cyan/Magenta/Yellow/blacK) color systems, commonly used for digital color graphics.

In [26]:

```
plt.plot(x, x + 0, '-g')  # solid green
plt.plot(x, x + 1, '--c') # dashed cyan
plt.plot(x, x + 2, '-.k') # dashdot black
plt.plot(x, x + 3, ':r');  # dotted red
```

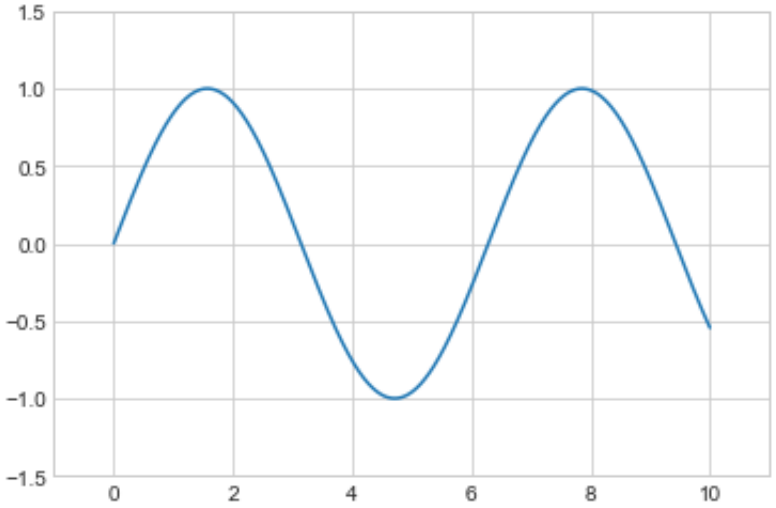


Axes Limits

- The most basic way to adjust axis limits is to use the plt.xlim() and plt.ylim() methods.

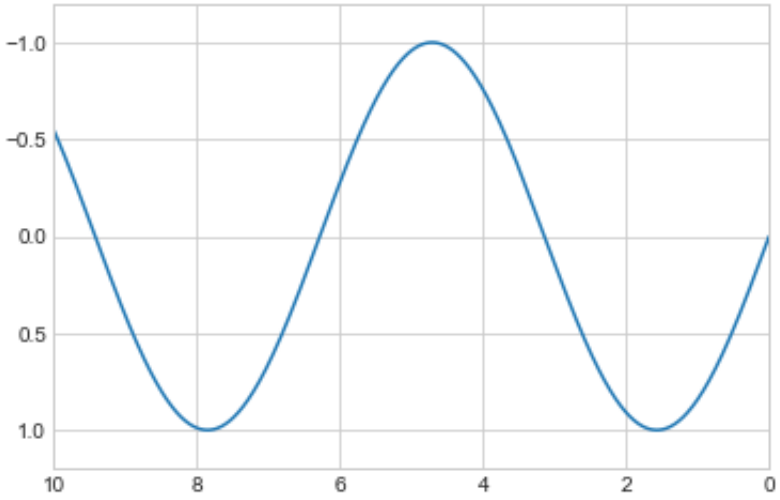
In [27]:

```
plt.plot(x, np.sin(x))

plt.xlim(-1, 11)
plt.ylim(-1.5, 1.5);
```



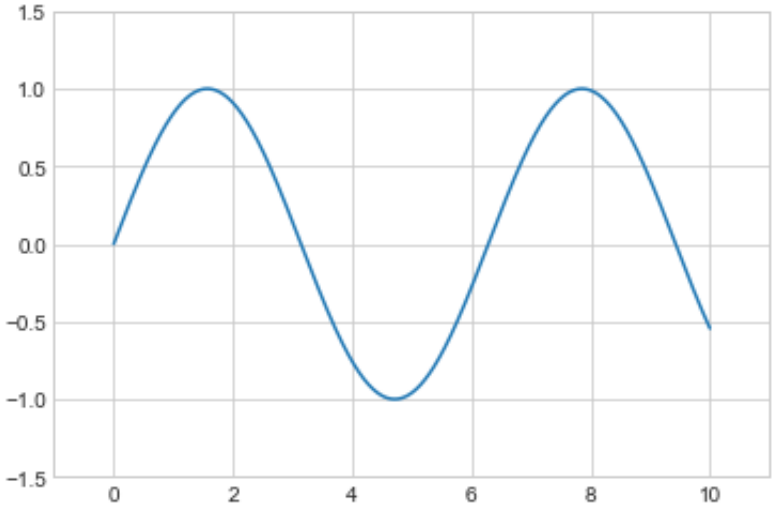
- If for some reason you'd like either axis to be displayed in reverse, you can simply reverse the order of the arguments.

In [28]:
```
plt.plot(x, np.sin(x))

plt.xlim(10, 0)
plt.ylim(1.2, -1.2);
```



- A useful related method is `plt.axis()` (note here the potential confusion between axes with an e, and axis with an i).
- The `plt.axis()` method allows you to set the `x` and `y` limits with a single call, by passing a list which specifies `[xmin, xmax, ymin, ymax]`.

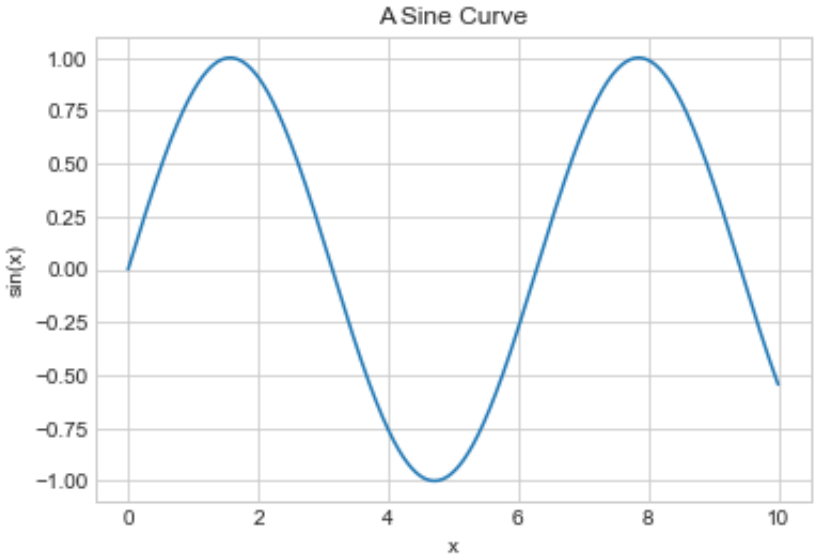
In [29]:
```
plt.plot(x, np.sin(x))
plt.axis([-1, 11, -1.5, 1.5]);
```



Labeling Plots

- `title()`, `xlabel()`, and `ylabel()` methods can be used to quickly set titles and axis labels.

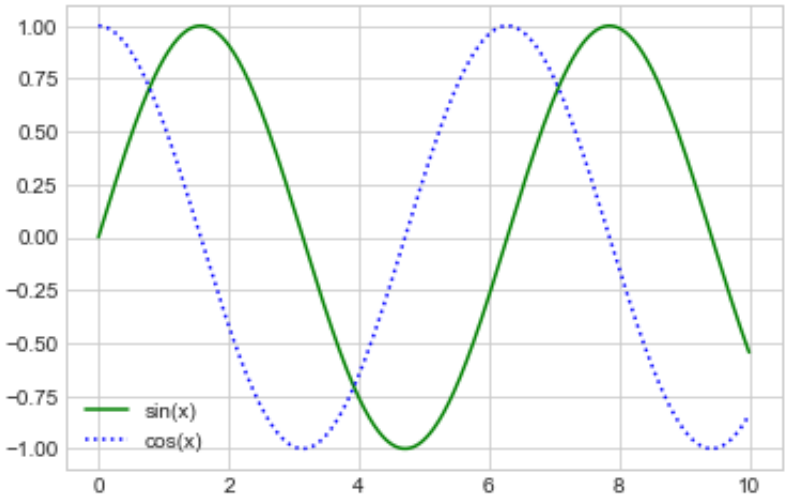
In [30]:
```
plt.plot(x, np.sin(x))
plt.title("A Sine Curve")
plt.xlabel("x")
plt.ylabel("sin(x)");
```



- `plt.legend()` method is used to label when multiple lines shown within a single axes.
- Simply, just specify the label of each line using the `label` keyword of the plot function.
- `plt.legend()` function keeps track of the line style and color, and matches these with the correct label.

In [36]:

```
plt.plot(x, np.sin(x), '-g', label='sin(x)')
plt.plot(x, np.cos(x), ':b', label='cos(x)')

plt.legend();
```

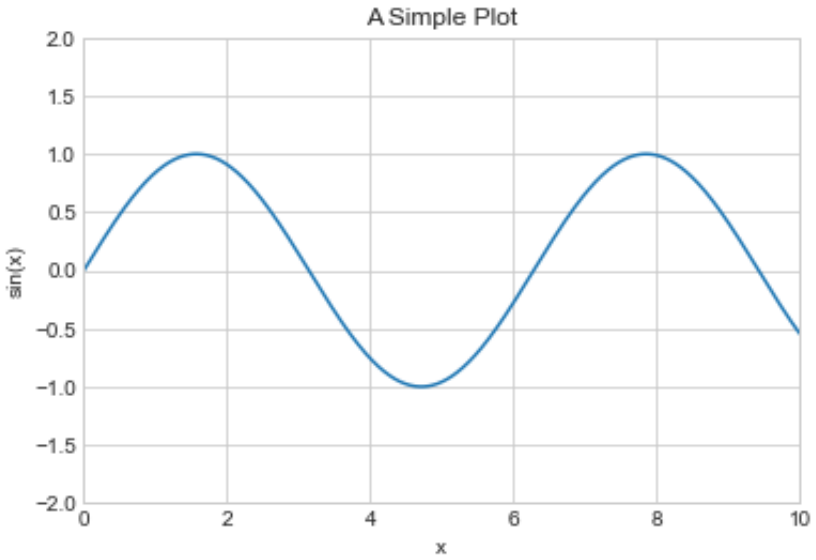


- For transitioning between MATLAB-style functions and object-oriented methods, make the following changes to set tittle, axis labels, and legend:
  - `plt.xlabel()` → `ax.set_xlabel()`
  - `plt.ylabel()` → `ax.set_ylabel()`
  - `plt.xlim()` → `ax.set_xlim()`
  - `plt.ylim()` → `ax.set_ylim()`
  - `plt.title()` → `ax.set_title()`

- In the object-oriented interface to plotting, rather than calling these functions individually, it is often more convenient to use the `ax.set()` method to set all these properties at once.

In [34]:

```
ax = plt.axes()
ax.plot(x, np.sin(x))
ax.set(xlim=(0, 10), ylim=(-2, 2),
       xlabel='x', ylabel='sin(x)',
       title='A Simple Plot');
```


A Simple Plot
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
sin(x)
0
2
4
6
8
10
x