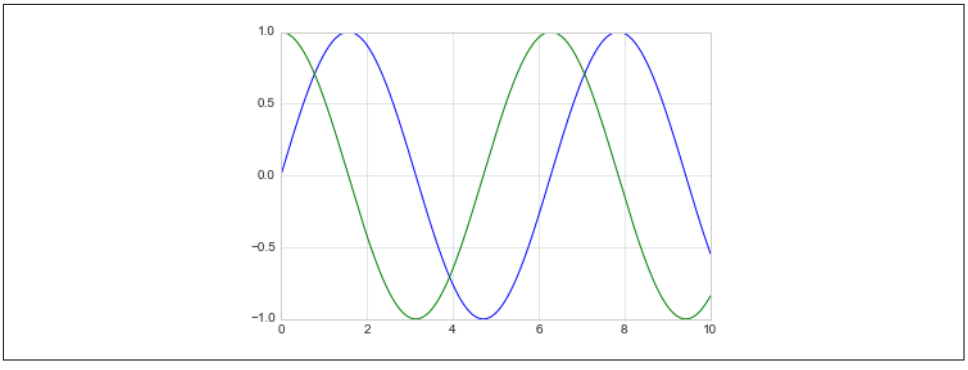

Figure 4-8. Over-plotting multiple lines

That's all there is to plotting simple functions in Matplotlib! We'll now dive into some more details about how to control the appearance of the axes and lines.

Adjusting the Plot: Line Colors and Styles

The first adjustment you might wish to make to a plot is to control the 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 (Figure 4-9):

```
In[6]:
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 and 1
plt.plot(x, np.sin(x - 5), color='chartreuse'); # all HTML color names supported
```

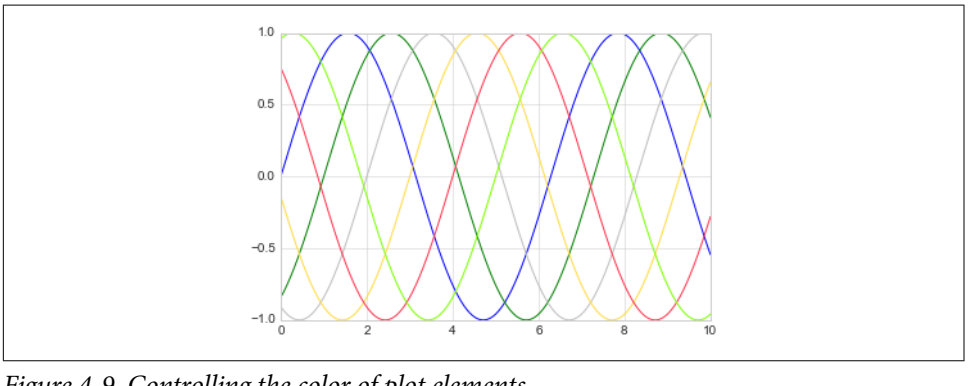

Figure 4-9. Controlling the color of plot elements

If no color is specified, Matplotlib will automatically cycle through a set of default colors for multiple lines.

Similarly, you can adjust the line style using the `linestyle` keyword (Figure 4-10):

```
In[7]: 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
```

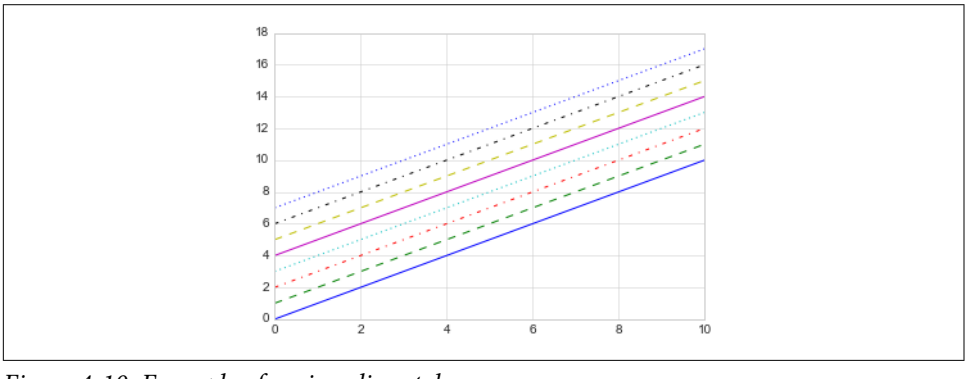

Figure 4-10. Example of various line styles

If you would like to be extremely terse, these `linestyle` and `color` codes can be combined into a single nonkeyword argument to the `plt.plot()` function (Figure 4-11):

```
In[8]: 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
```

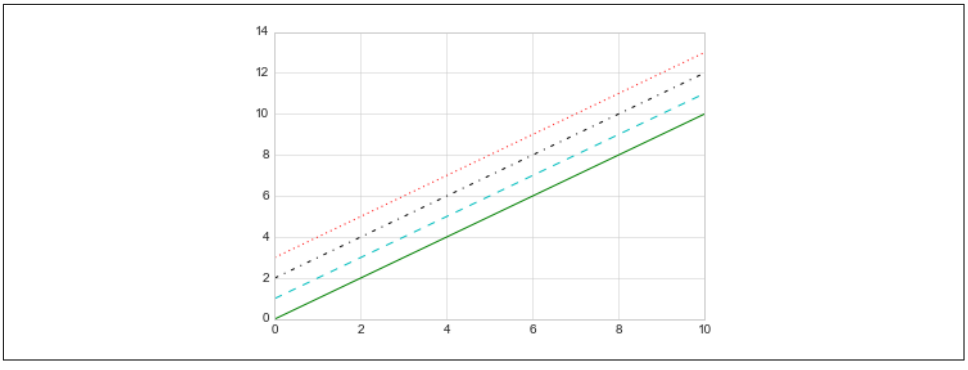

Figure 4-11. Controlling colors and styles with the shorthand syntax

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.

There are many other keyword arguments that can be used to fine-tune the appearance of the plot; for more details, I'd suggest viewing the docstring of the `plt.plot()` function using IPython's help tools (see "Help and Documentation in IPython" on page 3).

Adjusting the Plot: Axes Limits

Matplotlib does a decent job of choosing default axes limits for your plot, but sometimes it's nice to have finer control. The most basic way to adjust axis limits is to use the `plt.xlim()` and `plt.ylim()` methods (Figure 4-12):

```
In[9]: plt.plot(x, np.sin(x))

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

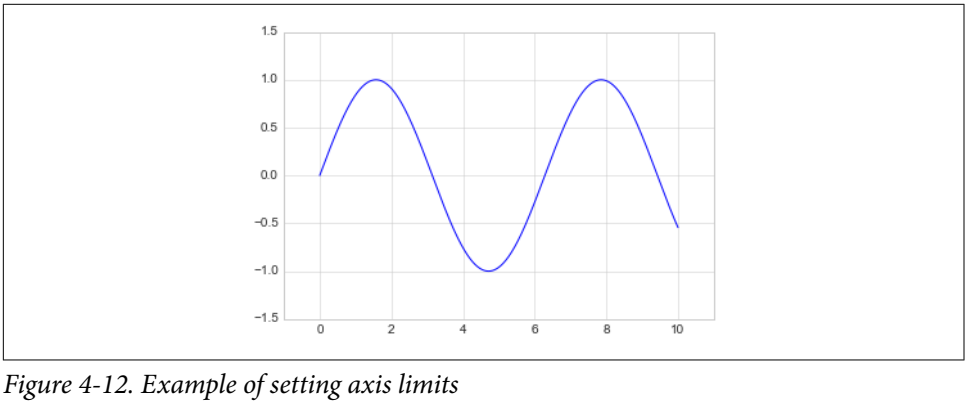

Figure 4-12. Example of setting axis limits