Simple Line Plots

简单的折线图

> Perhaps the simplest of all plots is the visualization of a single function $y = f(x)$. Here we will take a first look at creating a simple plot of this type. As with all the following sections, we'll start by setting up the notebook for plotting and importing the packages we will use:

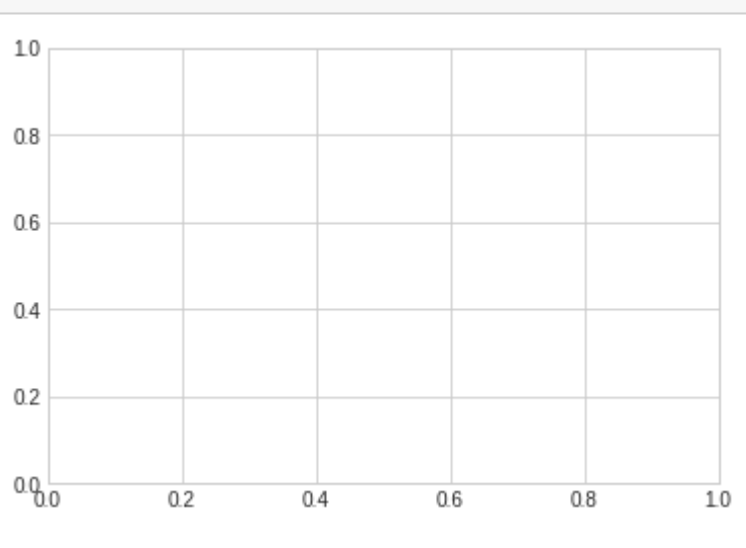
对于图表来说，最简单的莫过于作出一个单一函数$y=f(x)$的图像。本节中我们首先来介绍创建这种类型图表。本节和后续小节中，我们都会使用下面的代码将我们需要的包载入到notebook中：

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

> 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:

对于所有的Matplotlib图表来说，我们都需要从创建图形和维度开始。图形和维度可以使用下面代码进行最简形式的创建：

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



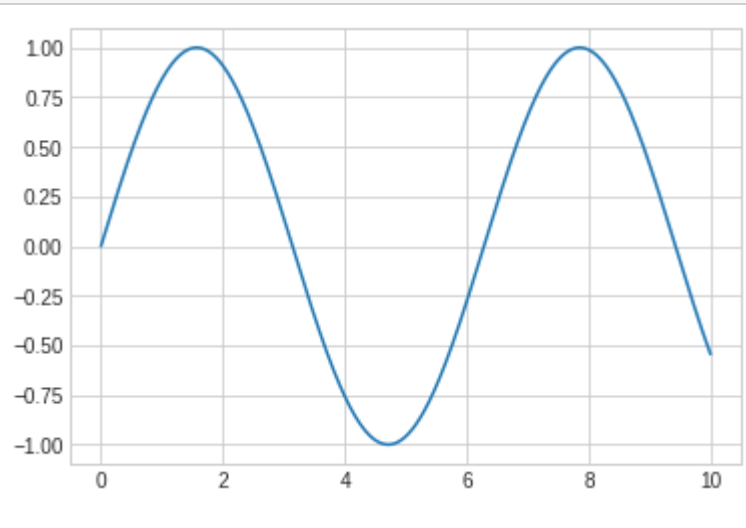
> In Matplotlib, the *figure* (an instance of the class `plt.Figure`) can be thought of as a single container that contains all the objects representing axes, graphics, text, and labels. The *axes* (an instance of the class `plt.Axes`) is what we see above: a bounding box with ticks and labels, which will eventually contain the plot elements that make up our visualization. Throughout this book, we'll commonly use the variable name `fig` to refer to a figure instance, and `ax` to refer to an axes instance or group of axes instances.

在Matplotlib中，*图形*（类`plt.Figure`的一个实例）可以被认为是一个包括所有维度、图像、文本和标签对象的容器。*维度*（类`plt.Axes`的一个实例）就是你上面看到的图像，一个有边界的格子包括刻度和标签，最终还有我们画在上面的图表元素。在本书中，我们会使用变量名`fig`来指代图形对象，以及变量名`ax`来指代维度变量。

> Once we have created an axes, we can use the `ax.plot` function to plot some data. Let's start with a simple sinusoid:

一旦我们创建了维度，我们可以使用`ax.plot`方法将数据绘制在图表上。下面是一个简单的正弦函数图形：

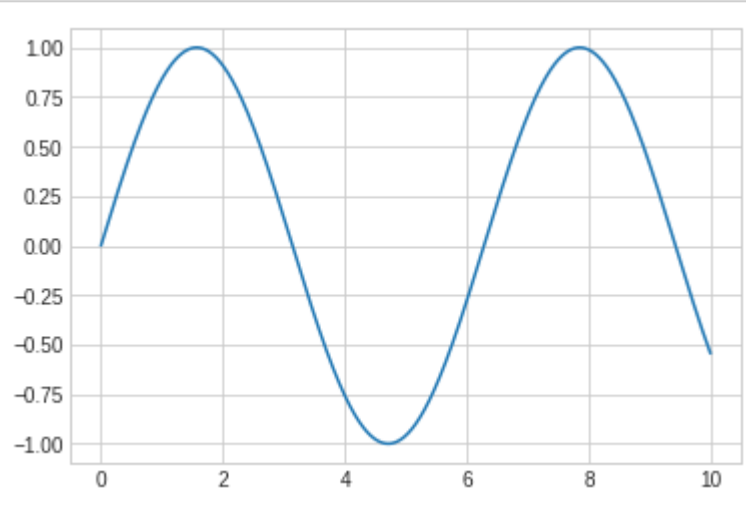
In [3]:
```
fig = plt.figure()
ax = plt.axes()

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 (see Two Interfaces for the Price of One for a discussion of these two interfaces):

同样的，我们可以使用pylab接口（MATLAB风格的接口）帮我们在后台自动创建这两个对象：

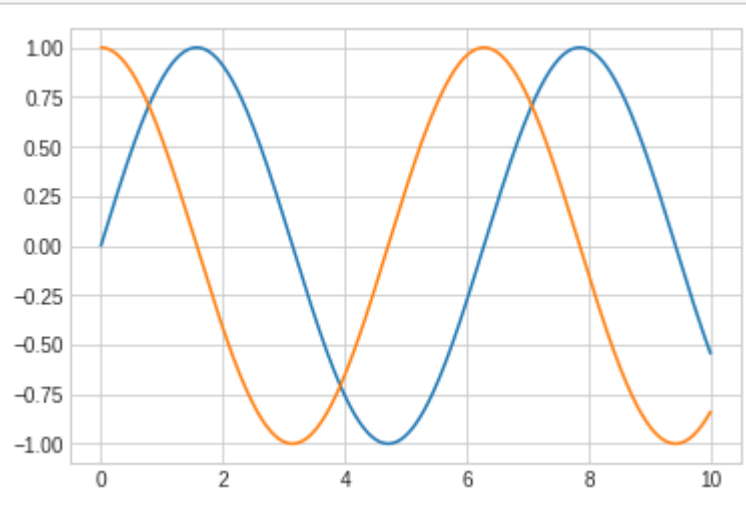
In [4]:
```
plt.plot(x, np.sin(x));
```



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

如果我们需要在同一幅图形中绘制多根线条，只需要多次调用`plot`函数即可：

In [5]:
```
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x));
```



> 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.

这就是在Matplotlib中绘制简单函数图像的所有接口了。下面我们深入了解一下控制坐标轴和线条外观的细节。

Adjusting the Plot: Line Colors and Styles

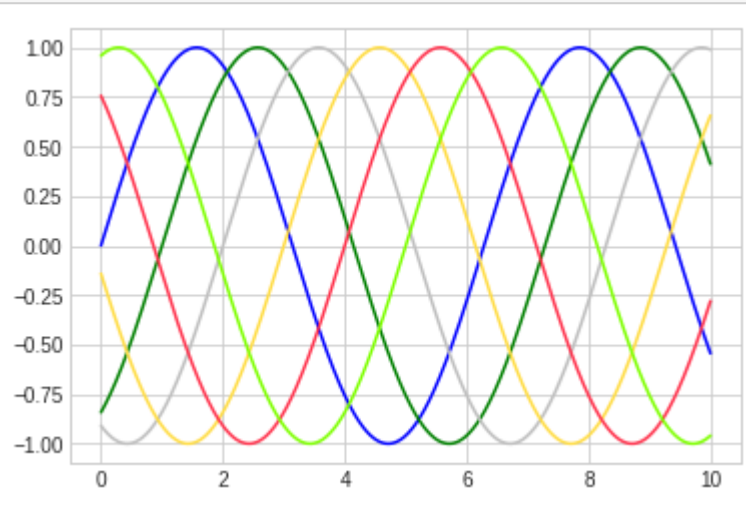
调整折线图：线条颜色和风格

> 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:

你可能第一个想到需要进行调整的部分就是线条的颜色和风格。`plt.plot()`函数接受额外的参数可以用来指定它们。通过指定`color`关键字参数可以调整颜色，这个字符串类型参数基本上能用来代表任何你能想到的颜色。可以通过多种方式指定颜色参数：

译者注：所有HTML颜色名称可以在这里找到。

In [6]:
```
plt.plot(x, np.sin(x - 0), color='blue')        # 通过颜色名称指定
plt.plot(x, np.sin(x - 1), color='g')           # 通过颜色简写名称指定(rgbcmyk)
plt.plot(x, np.sin(x - 2), color='0.75')        # 介于0-1之间的灰阶值
plt.plot(x, np.sin(x - 3), color='#FFDD44')     # 16进制的RRGGBB值
plt.plot(x, np.sin(x - 4), color=(1.0,0.2,0.3)) # RGB元组的颜色值，每个值介于0-1
plt.plot(x, np.sin(x - 5), color='chartreuse'); # 能支持所有HTML颜色名称值
```



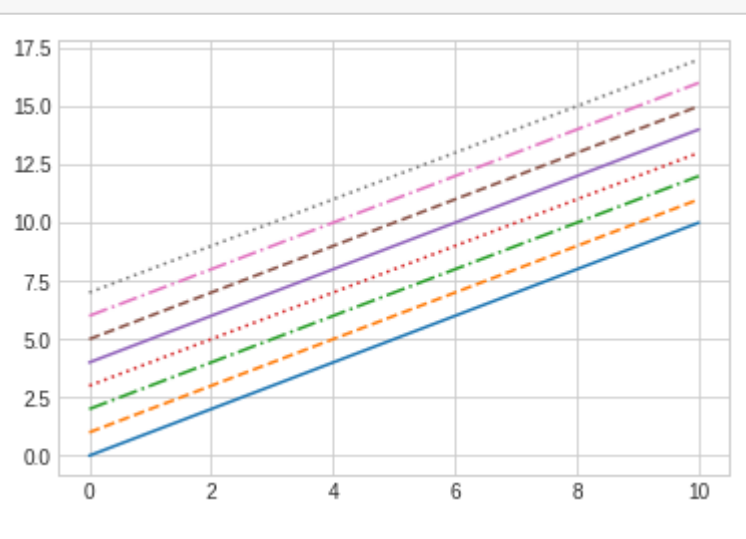
> If no color is specified, Matplotlib will automatically cycle through a set of default colors for multiple lines.

如果没有指定颜色，Matplotlib会在一组默认颜色值中循环使用来绘制每一条线条。

> Similarly, the line style can be adjusted using the `linestyle` keyword:

类似的，通过`linestyle`关键字参数可以指定线条的风格：

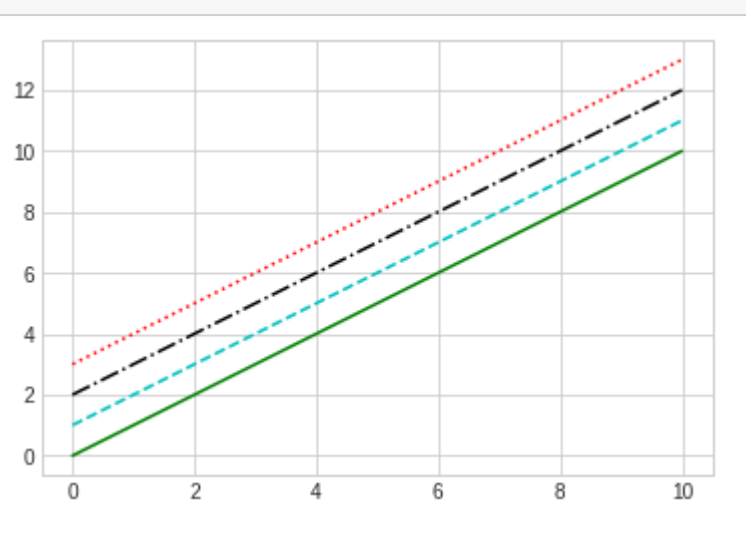
In [7]:
```
plt.plot(x, x + 0, linestyle='solid')
plt.plot(x, x + 1, linestyle='dashed')
plt.plot(x, x + 2, linestyle='dashdot')
plt.plot(x, x + 3, linestyle='dotted');

# 还可以用形象的符号代表线条风格
plt.plot(x, x + 4, linestyle='-')  # 实线
plt.plot(x, x + 5, linestyle='--') # 虚线
plt.plot(x, x + 6, linestyle='-.') # 长短点虚线
plt.plot(x, x + 7, linestyle=':');  # 点线
```



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

如果你喜欢更简洁的代码，这些`linestyle`和`color`参数能够合并成一个非关键字参数，传递给`plt.plot()`函数：

In [8]:
```
plt.plot(x, x + 0, '-g')  # 绿色实线
plt.plot(x, x + 1, '--c') # 天青色虚线
plt.plot(x, x + 2, '-.k') # 黑色长短点虚线
plt.plot(x, x + 3, ':r');  # 红色点线
```



> 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.

上面的单字母颜色码是RGB颜色系统以及CMYK颜色系统的缩写，被广泛应用在数字化图像的颜色系统中。

> 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).

还有很多其他的关键字参数可以对折线图的外观进行精细调整；可以通过在IPython中使用帮助工具（参见IPython的帮助和文档）查看`plt.plot()`函数的文档来获得更多细节内容。

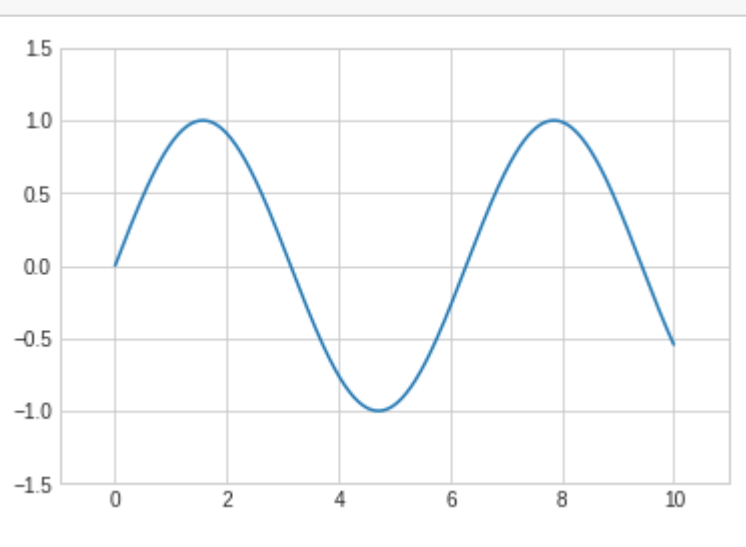
Adjusting the Plot: Axes Limits

调整折线图：坐标轴范围

> 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:

Matplotlib会自动选择非常合适的坐标轴范围来绘制你的图像，但是有些情况下你也需要自己进行相关调整。使用`plt.xlim()`和`plt.ylim()`函数可以调整坐标轴的范围：

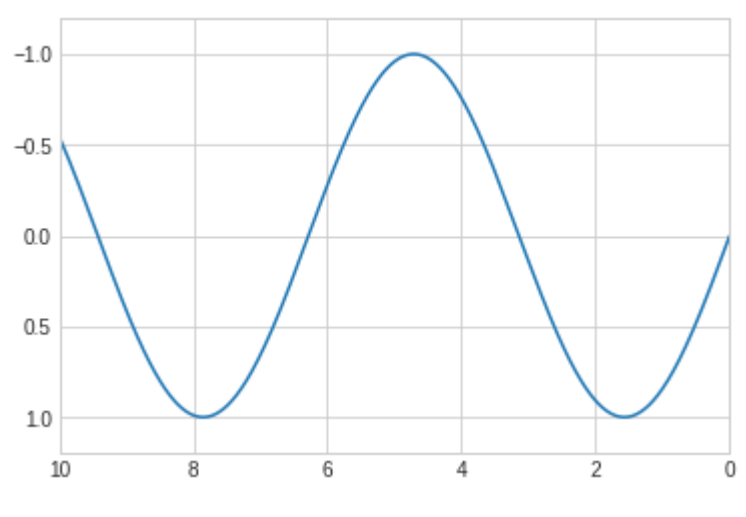
In [9]:
```
plt.plot(x, np.sin(x))

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:

如果某些情况下你希望将坐标轴反向，你可以通过上面的函数实现，将参数顺序颠倒即可：

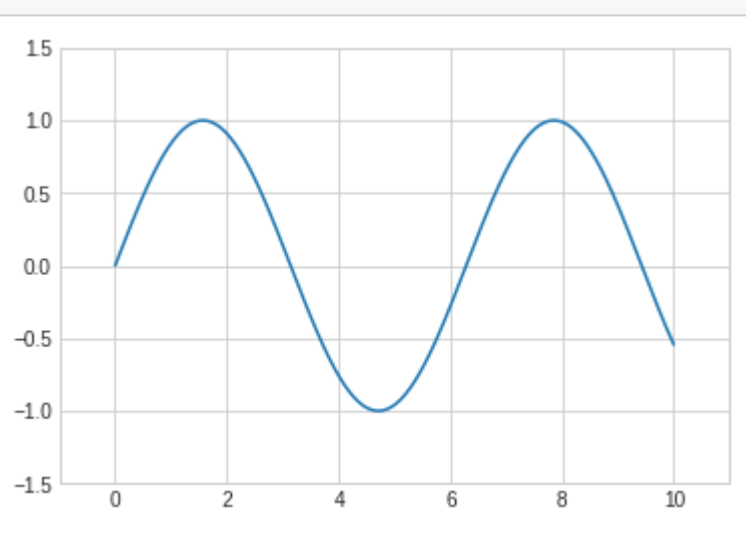
In [10]:
```
plt.plot(x, np.sin(x))

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]`:

相关的函数还有`plt.axis()`（注意：这不是`plt.axes()`函数，函数名称是i而不是e）。这个函数可以在一个函数调用中就完成x轴和y轴范围的设置，传递一个`[xmin, xmax, ymin, ymax]`的列表参数即可：

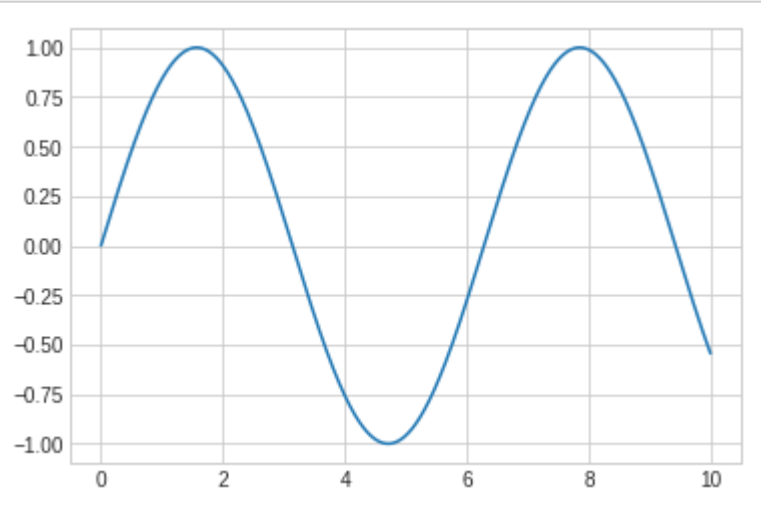
In [11]:
```
plt.plot(x, np.sin(x))
plt.axis([-1, 11, -1.5, 1.5]);
```



> The `plt.axis()` method goes even beyond this, allowing you to do things like automatically tighten the bounds around the current plot:

当然`plt.axis()`函数不仅能设置范围，还能像下面代码一样将坐标轴压缩到刚好足够绘制折线图像的大小：

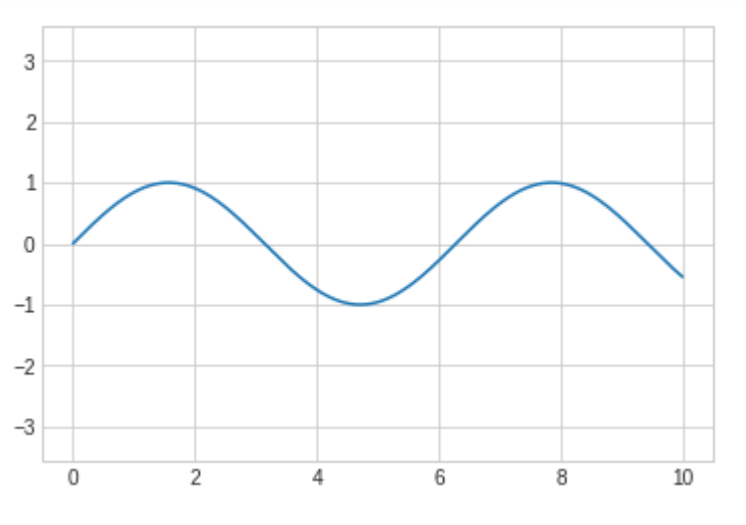
In [12]:
```
plt.plot(x, np.sin(x))
plt.axis('tight');
```



> It allows even higher-level specifications, such as ensuring an equal aspect ratio so that on your screen, one unit in `x` is equal to one unit in `y`:

还可以通过设置`'equal'`参数设置`x`轴与`y`轴使用相同的长度单位：

In [13]:
```
plt.plot(x, np.sin(x))
plt.axis('equal');
```



> For more information on axis limits and the other capabilities of the `plt.axis` method, refer to the `plt.axis` docstring.

更多关于设置axis属性的内容请查阅`plt.axis`函数的文档字符串。

Labeling Plots

折线图标签

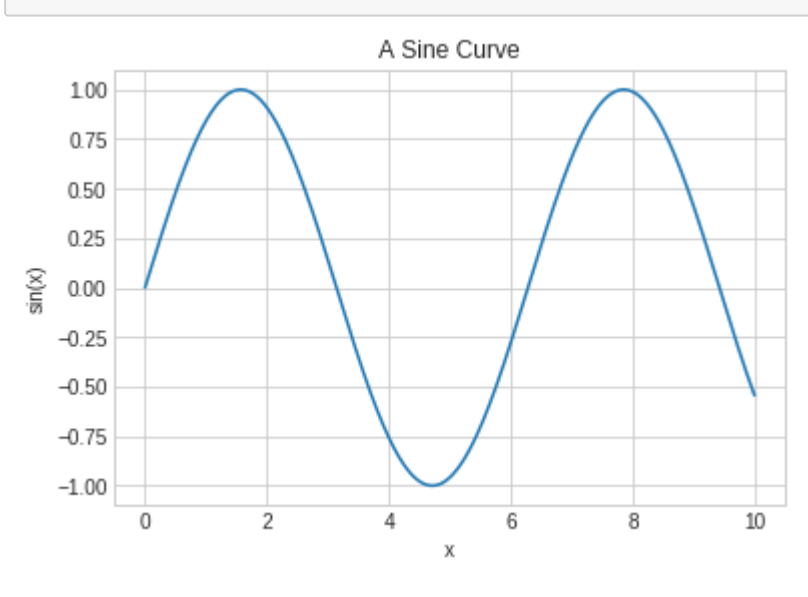
> As the last piece of this section, we'll briefly look at the labeling of plots: titles, axis labels, and simple legends.

本节最后介绍一下在折线图上绘制标签：标题、坐标轴标签和简单的图例。

> Titles and axis labels are the simplest such labels—there are methods that can be used to quickly set them:

标题和坐标轴标签是最简单的这类标签，Matplotlib提供了函数用来方便的设置它们：

In [14]:
```
plt.plot(x, np.sin(x))
plt.title("A Sine Curve")
plt.xlabel("x")
plt.ylabel("sin(x)");
```



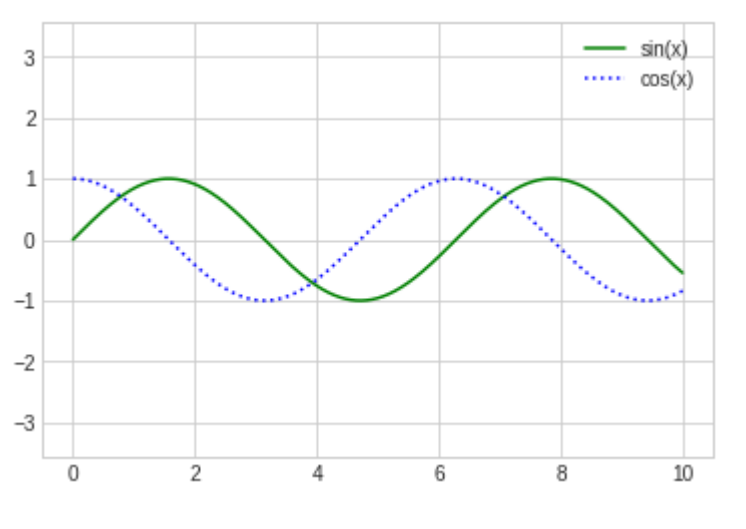
> The position, size, and style of these labels can be adjusted using optional arguments to the function. For more information, see the Matplotlib documentation and the docstrings of each of these functions.

这些标签的位置、大小和风格可以通过上面函数的可选参数进行设置。参阅Matplotlib在线文档和这些函数的文档字符串可以获得更多的信息。

> When multiple lines are being shown within a single axes, it can be useful to create a plot legend that labels each line type. Again, Matplotlib has a built-in way of quickly creating such a legend. It is done via the (you guessed it) `plt.legend()` method. Though there are several valid ways of using this, I find it easiest to specify the label of each line using the `label` keyword of the plot function:

当一幅图中绘制了多条折线时，如果能够绘制一个线条对应的图例能让图表更加清晰。Matplotlib也内建了函数来快速创建图例。估计你也猜到了，通过`plt.legend()`函数可以实现这个需求。虽然有很多种正确的方法来指定图例，作者认为最简单的方法是通过在绘制每条线条时指定对应的`label`关键字参数来使用这个函数：

In [15]:
```
plt.plot(x, np.sin(x), '-g', label='sin(x)')
plt.plot(x, np.cos(x), ':b', label='cos(x)')
plt.axis('equal')

plt.legend();
```



> As you can see, the `plt.legend()` function keeps track of the line style and color, and matches these with the correct label. More information on specifying and formatting plot legends can be found in the `plt.legend` docstring; additionally, we will cover some more advanced legend options in Customizing Plot Legends.

上图可见，`plt.legend()`函数绘制的图例线条与图中的折线无论风格和颜色都保持一致。查阅`plt.legend`文档字符串可以获得更多相关信息；我们在自定义图表图例一节中也会讨论更高级的图例应用。

Aside: Matplotlib Gotchas

额外内容：Matplotlib的坑

> While most `plt` functions translate directly to `ax` methods (such as `plt.plot()` → `ax.plot()`, `plt.legend()` → `ax.legend()`, etc.), this is not the case for all commands. In particular, functions to set limits, labels, and titles are slightly modified. For transitioning between MATLAB-style functions and object-oriented methods, make the following changes:

虽然大多数的`plt`函数都可以直接转换为`ax`的方法进行调用（例如`plt.plot()` → `ax.plot()`，`plt.legend()` → `ax.legend()`等），但是并不是所有的命令都能应用这种情况。特别是用于设置极值、标签和标题的函数都有一定的改变。下表列出了将MATLAB风格的函数转换为面向对象的方法的区别：

- `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:

在面向对象接口中，与其逐个调用上面的方法来设置属性，更常见的使用`ax.set()`方法来一次性设置所有的属性：

In [16]:
```
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');
```