Matplotlib Tutorial: 1. Basic Plot Interface

In this notebook, we will explore the basic plot interface using `plt.plot` and `plt.scatter`. We will also discuss the difference between the `pyplot interface`, which offers plotting with the feel of Matlab. In the following sections, we will introduce the `object-oriented interface`, which offers more flexibility and will be used throughout the remainter of the tutorial.

This tutorial is written for Python 3.3; to make it work with Python 2 we'll do some future imports:

In [11]:

```
from __future__ import print_function, division
```

Setting up IPython

IPython has a built-in mode to work cleanly with matplotlib figures. The most common way to invoke it is to use the following magic command:

In [12]:

```
%matplotlib inline
```

A first plot: the PyPlot interface

Now we're ready for a plot. The `%pylab` mode we entered above does a few things, among which is the import of `pylab` into the current namespace. For clarity, we'll do this directly here. We'll also import `numpy` in order to easily manipulate the arrays we'll plot:

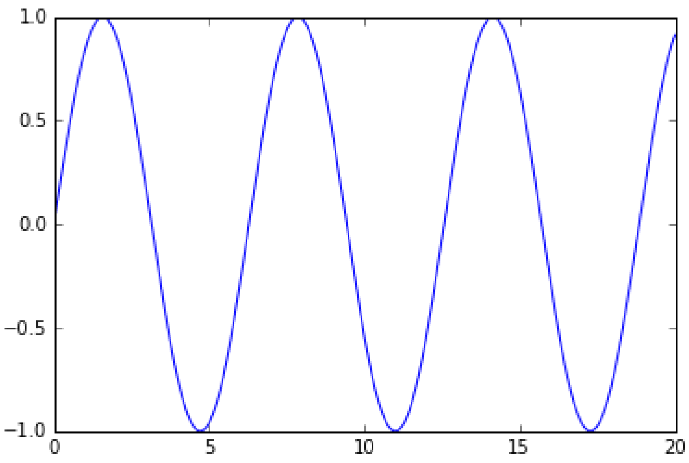
In [13]:

```
import matplotlib.pyplot as plt
import numpy as np
```

Let's make some simple data to plot: a sinusoid

In [14]:

```
x = np.linspace(0, 20, 1000)  # 100 evenly-spaced values from 0 to 50
y = np.sin(x)

plt.plot(x, y);
```

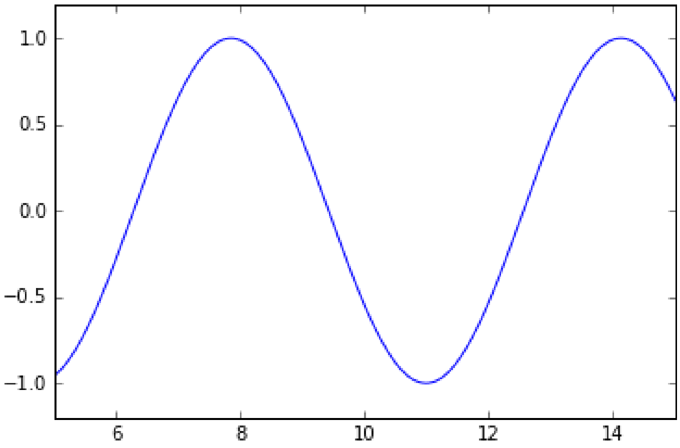


Customizing the plot: Axes Limits

Let's play around with this a bit: first we can change the axis limits using `xlim()` and `ylim()`

In [15]:

```
plt.plot(x, y)
plt.xlim(5, 15)
plt.ylim(-1.2, 1.2);
```

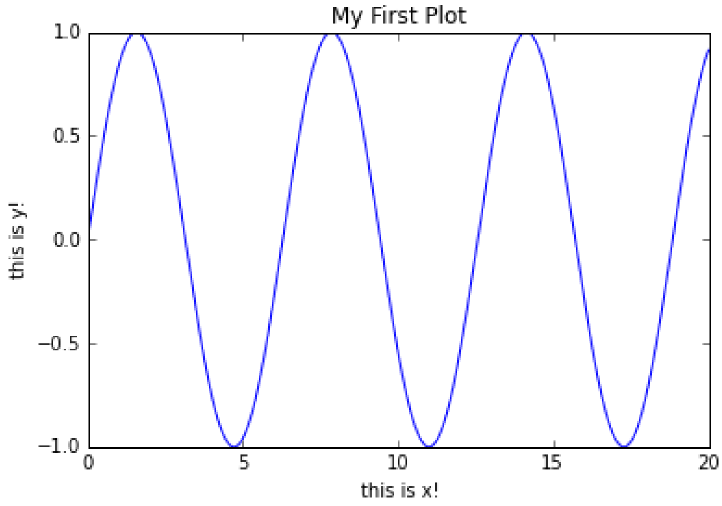


Customizing the plot: Axes Labels and Titles

We can label the axes and add a title:

In [16]:

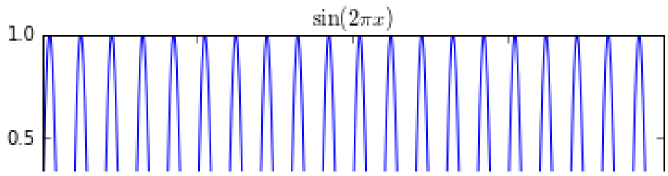
```
plt.plot(x, y)

plt.xlabel('this is x!')
plt.ylabel('this is y!')
plt.title('My First Plot');
```

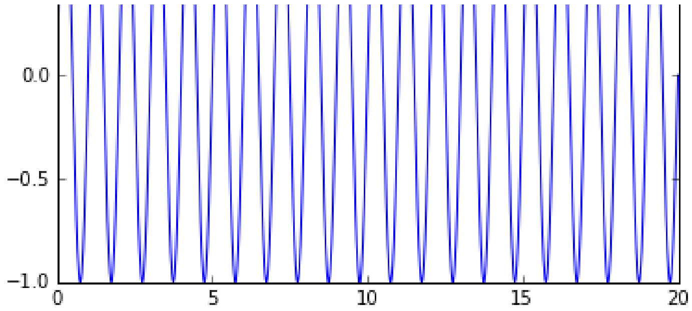


Labels can also be rendered using LaTeX symbols:

In [17]:

```
y = np.sin(2 * np.pi * x)
plt.plot(x, y)
plt.title(r'$\sin(2 \pi x)$')  # the `r` before the string indicates a "raw string";
```



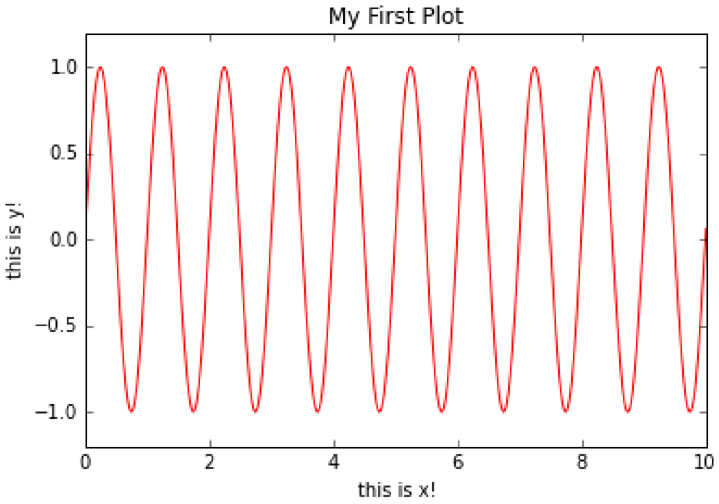


Customizing the plot: Line Styles

We can vary the line color or the line symbol:

In [18]:

```
plt.plot(x, y, '-r')  # solid red line ('r' comes from RGB color scheme)
plt.xlim(0, 10)
plt.ylim(-1.2, 1.2)

plt.xlabel('this is x!')
plt.ylabel('this is y!')
plt.title('My First Plot');
```



Other options for the color characters are:

```
'r' = red
'g' = green
'b' = blue
'c' = cyan
'm' = magenta
'y' = yellow
'k' = black
'w' = white
```

Options for line styles are

```
'-' = solid
'--' = dashed
':' = dotted
'-.' = dot-dashed
'.' = points
'o' = filled circles
```

```
    '^' = filled triangles
```

and many, many more.

For more information, view the documentation of the plot function. In IPython, this can be accomplished using the ? functionality:

In [19]:

```
plt.plot?
```

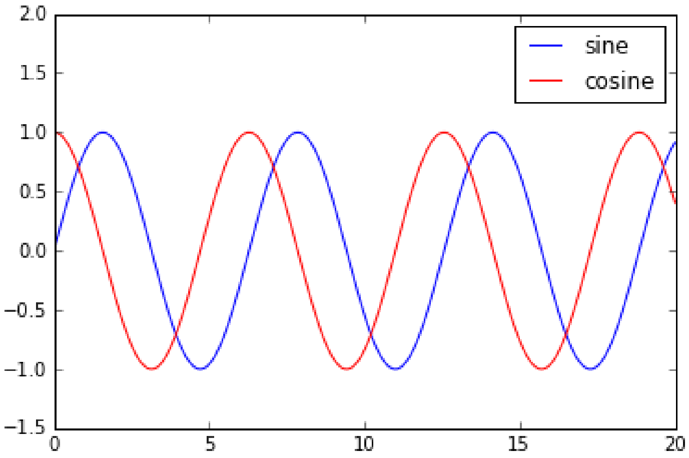
Also see the online version of this help: http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot

Cusomizing the Plot: Legends

Multiple lines can be shown on the same plot. In this case, you can use a legend to label the two lines:

In [20]:

```
x = np.linspace(0, 20, 1000)
y1 = np.sin(x)
y2 = np.cos(x)

plt.plot(x, y1, '-b', label='sine')
plt.plot(x, y2, '-r', label='cosine')
plt.legend(loc='upper right')
plt.ylim(-1.5, 2.0);
```



Exercise: Linestyles & Plot Customization

Below are two sets of arrays `x1, y1`, and `x2, y2`. Create a plot where `x1` and `y1` are represented by blue circles, and `x2` and `y2` are represented by a dotted black line. Label the symbols "sampled" and "continuous", and add a legend. Adjust the y limits to suit your taste.

In [21]:

```
x1 = np.linspace(0, 10, 20)
y1 = np.sin(x1)

x2 = np.linspace(0, 10, 1000)
y2 = np.sin(x2)
```