Cython sources on which Pandas is built. Details on this installation can be found in the Pandas documentation. If you followed the advice outlined in the preface and used the Anaconda stack, you already have Pandas installed.

Once Pandas is installed, you can import it and check the version:

```
In[1]: import pandas
       pandas.__version__

Out[1]: '0.18.1'
```

Just as we generally import NumPy under the alias `np`, we will import Pandas under the alias `pd`:

```
In[2]: import pandas as pd
```

This import convention will be used throughout the remainder of this book.

Reminder About Built-In Documentation

As you read through this chapter, don't forget that IPython gives you the ability to quickly explore the contents of a package (by using the tab-completion feature) as well as the documentation of various functions (using the `?` character). (Refer back to "Help and Documentation in IPython" on page 3 if you need a refresher on this.)

For example, to display all the contents of the `pandas` namespace, you can type this:

```
In [3]: pd.<TAB>
```

And to display the built-in Pandas documentation, you can use this:

```
In [4]: pd?
```

More detailed documentation, along with tutorials and other resources, can be found at *http://pandas.pydata.org/*.

Introducing Pandas Objects

At the very basic level, Pandas objects can be thought of as enhanced versions of NumPy structured arrays in which the rows and columns are identified with labels rather than simple integer indices. As we will see during the course of this chapter, Pandas provides a host of useful tools, methods, and functionality on top of the basic data structures, but nearly everything that follows will require an understanding of what these structures are. Thus, before we go any further, let's introduce these three fundamental Pandas data structures: the `Series`, `DataFrame`, and `Index`.

We will start our code sessions with the standard NumPy and Pandas imports:

```
In[1]: import numpy as np
       import pandas as pd
```

The Pandas Series Object

A Pandas `Series` is a one-dimensional array of indexed data. It can be created from a list or array as follows:

```
In[2]: data = pd.Series([0.25, 0.5, 0.75, 1.0])
       data

Out[2]: 0    0.25
        1    0.50
        2    0.75
        3    1.00
        dtype: float64
```

As we see in the preceding output, the `Series` wraps both a sequence of values and a sequence of indices, which we can access with the `values` and `index` attributes. The `values` are simply a familiar NumPy array:

```
In[3]: data.values

Out[3]: array([ 0.25,  0.5 ,  0.75,  1.  ])
```

The `index` is an array-like object of type `pd.Index`, which we'll discuss in more detail momentarily:

```
In[4]: data.index

Out[4]: RangeIndex(start=0, stop=4, step=1)
```

Like with a NumPy array, data can be accessed by the associated index via the familiar Python square-bracket notation:

```
In[5]: data[1]

Out[5]: 0.5

In[6]: data[1:3]

Out[6]: 1    0.50
        2    0.75
        dtype: float64
```

As we will see, though, the Pandas `Series` is much more general and flexible than the one-dimensional NumPy array that it emulates.

Series as generalized NumPy array

From what we've seen so far, it may look like the `Series` object is basically interchangeable with a one-dimensional NumPy array. The essential difference is the presence of the index: while the NumPy array has an *implicitly defined* integer index used to access the values, the Pandas `Series` has an *explicitly defined* index associated with the values.