Underscore Shortcuts and Previous Outputs

The standard Python shell contains just one simple shortcut for accessing previous output; the variable _ (i.e., a single underscore) is kept updated with the previous output; this works in IPython as well:

```
In [9]: print(_)
1.0
```

But IPython takes this a bit further—you can use a double underscore to access the second-to-last output, and a triple underscore to access the third-to-last output (skipping any commands with no output):

```
In [10]: print(__)
-0.4161468365471424

In [11]: print(___)
0.9092974268256817
```

IPython stops there: more than three underscores starts to get a bit hard to count, and at that point it's easier to refer to the output by line number.

There is one more shortcut we should mention, however—a shorthand for `Out[X]` is `_X` (i.e., a single underscore followed by the line number):

```
In [12]: Out[2]
Out[12]: 0.9092974268256817

In [13]: _2
Out[13]: 0.9092974268256817
```

Suppressing Output

Sometimes you might wish to suppress the output of a statement (this is perhaps most common with the plotting commands that we'll explore in Chapter 4). Or maybe the command you're executing produces a result that you'd prefer not to store in your output history, perhaps so that it can be deallocated when other references are removed. The easiest way to suppress the output of a command is to add a semicolon to the end of the line:

```
In [14]: math.sin(2) + math.cos(2);
```

Note that the result is computed silently, and the output is neither displayed on the screen or stored in the `Out` dictionary:

```
In [15]: 14 in Out
Out[15]: False
```