Iterating Over a Sequence of Objects

We said earlier that we were going to employ Python's `for` loop here. The `for` loop is *perfect* for controlling looping when you know ahead of time how many iterations you need. (When you don't know, we recommend the `while` loop, but we'll save discussing the details of this alternate looping construct until we actually need it). At this stage, all we need is `for`, so let's see it in action at the >>> prompt.

We present three typical uses of `for`. Let's see which one best fits our needs.

Use "for" when looping a known number of times.

Usage example 1. This `for` loop, below, takes a list of numbers and iterates once for each number in the list, displaying the current number on screen. As it does so, the `for` loop assigns each number in turn to a *loop iteration variable*, which is given the name `i` in this code.

As this code is more than a single line, the shell indents automatically for you when you press Enter after the colon. To signal to the shell that you are done entering code, press Enter *twice* at the end of the loop's suite:

```
>>> for i in [1, 2, 3]:
            print(i)

1
2
3
```

We used "i" as the loop iteration variable in this example, but we could've called it just about anything. Having said that, "i", "j", and "k" are incredibly popular among most programmers in this situation.

As this is a suite, you need to press the Enter key TWICE after typing in this code in order to terminate the statement and see it execute.

Note the *indentation* and *colon*. Like `if` statements, the code associated with a `for` statement needs to be **indented**.

Usage example 2. This `for` loop, below, iterates over a string, with each character in the string being processed during each iteration. This works because a string in Python is a **sequence**. A sequence is an ordered collection of objects (and we'll see lots of examples of sequences in this book), and every sequence in Python can be iterated over by the interpreter.

A sequence is an ordered collection of objects.

```
>>> for ch in "Hi!":
            print(ch)

H
i
!
```

Python is smart enough to work out that this string should be iterated over one-character at a time (and that's why we used "ch" as the loop variable name here).

Nowhere did you have to tell the `for` loop *how big the string is*. Python is smart enough to work out when the string *ends*, and arranges to terminate (i.e., end) the `for` loop on your behalf when it exhausts all the objects in the sequence.