8 Flow of control: loops

Sometimes we want to repeat an action, perhaps with variations. One way to do this is with the word `for`. Suppose we want to print out a hundred lines containing the messages:

```
Next number is 1
Next number is 2
   ⋮
Next number is 100
```

Here is a code snippet which does that:

```
for ($i = 1; $i <= 100; ++$i)
  {
  print "Next number is $i\n";
  }
```

The brackets following `for` contain: a variable created for the purpose of this `for` loop and given an initial value; a condition for repeating the loop; and an action to be executed after each pass. The variable `$i` begins with the value 1, `++$i` increments it by one on each pass, and the instruction within the curly brackets is executed for each value of `$i` until `$i` reaches 101, when control moves on to whatever follows the closing curly bracket.

We saw earlier that, within double quotation marks, a symbol like `\n` is translated into what it stands for (newline, in this case), rather than being taken literally as the two characters `\` followed by `n`. Similarly, a variable name such as `$i` is translated into its current value; the lines displayed by the code above read e.g. *`Next number is 3`*, not *`Next number is $i`*. If you really wanted the latter, you would need to "escape" the dollar sign:

```
print "Next number is \$i\n";
```

The little examples in earlier chapters often ended with statements such as

```
print $a;
```

In practice, it would usually be far preferable to write

```
print "$a\n";
```

so that the result appears on a line of its own, rather than jammed together with the next system prompt.

Within the output of the above code snippet, 1 is not a "next" number but the first number. So we might want the message on the first line to read differently. By now, we know various ways to achieve that. Here are two – a straightforward, plodding way, and a more concise way:

(5)

```
1   for ($i = 1; $i <= 100; ++$i)
2     {
3     if ($i == 1)
4       {
5       print "First number is $i\n";
6       }
7     else
8       {
9       print "Next number is $i\n";
10      }
11    }
```

or (quicker to type, though less clear when you come back to it weeks later):

(6)

```
1   for ($i = 1; $i <= 100; ++$i)
2     {
3     $a = ($i == 1 ? "First" : "Next");
4     print "$a number is $i\n";
5     }
```

Another way to set up a repeating loop is the `while` construction. Here is another code snippet which achieves the same as the two we have just looked at:

(7)

```
1   $i = 1;
2   print "First number is $i\n";
3   while ($i < 100)
4     {
5     ++$i;
6     print "Next number is $i\n";
7     }
```

Here, `$i` is incremented within the loop body, and control falls out of the loop after the pass in which `$i` begins with the value 99. The `while` condition reads `$i < 100`, not `$i <= 100`: within the curly brackets, `$i` is incremented before its value is displayed, so if `<=` had been used in the `while` line, the lines displayed would have reached 101.

The `while` construction is often used for reading input lines in from a text file, so the next chapter will show us how that is done.