Chapter 2

For loops

Probably the most powerful thing about computers is that they can repeat things over and over very quickly. There are several ways to repeat things in Python, the most common of which is the for loop.

2.1 Examples

Example 1 The following program will print `Hello` ten times:

```
for i in range(10):
    print('Hello')
```

The structure of a for loop is as follows:

```
for variable name in range( number of times to repeat ):
    statements to be repeated
```

The syntax is important here. The word `for` must be in lowercase, the first line must end with a colon, and the statements to be repeated *must* be indented. Indentation is used to tell Python which statements will be repeated.

Example 2 The program below asks the user for a number and prints its square, then asks for another number and prints its square, etc. It does this three times and then prints that the loop is done.

```
for i in range(3):
    num = eval(input('Enter a number: '))
    print ('The square of your number is', num*num)
print('The loop is now done.')
```

```
Enter a number: 3
The square of your number is 9
Enter a number: 5
The square of your number is 25
Enter a number: 23
The square of your number is 529
The loop is now done.
```

Since the second and third lines are indented, Python knows that these are the statements to be repeated. The fourth line is not indented, so it is not part of the loop and only gets executed once, after the loop has completed.

Looking at the above example, we see where the term *for loop* comes from: we can picture the execution of the code as starting at the `for` statement, proceeding to the second and third lines, then looping back up to the `for` statement.

Example 3 The program below will print A, then B, then it will alternate C's and D's five times and then finish with the letter E once.

```python
print('A')
print('B')
for i in range(5):
    print('C')
    print('D')
print('E')
```

The first two print statements get executed once, printing an A followed by a B. Next, the C's and D's alternate five times. Note that we don't get five C's followed by five D's. The way the loop works is we print a C, then a D, then loop back to the start of the loop and print a C and another D, etc. Once the program is done looping with the C's and D's, it prints one E.

Example 4 If we wanted the above program to print five C's followed by five D's, instead of alternating C's and D's, we could do the following:

```python
print('A')
print('B')
for i in range(5):
    print('C')
for i in range(5):
    print('D')
print('E')
```

2.2 The loop variable

There is one part of a for loop that is a little tricky, and that is the loop variable. In the example below, the loop variable is the variable `i`. The output of this program will be the numbers 0, 1, . . . , 99, each printed on its own line.

```
for i in range(100):
    print(i)
```

When the loop first starts, Python sets the variable `i` to 0. Each time we loop back up, Python increases the value of `i` by 1. The program loops 100 times, each time increasing the value of `i` by 1, until we have looped 100 times. At this point the value of `i` is 99.

You may be wondering why `i` starts with 0 instead of 1. Well, there doesn't seem to be any really good reason why other than that starting at 0 was useful in the early days of computing and it has stuck with us. In fact most things in computer programming start at 0 instead of 1. This does take some getting used to.

Since the loop variable, `i`, gets increased by 1 each time through the loop, it can be used to keep track of where we are in the looping process. Consider the example below:

```
for i in range(3):
    print(i+1, '-- Hello')
```

```
1 -- Hello
2 -- Hello
3 -- Hello
```

Names There's nothing too special about the name `i` for our variable. The programs below will have the exact same result.

```
for i in range(100):              for wacky_name in range(100):
    print(i)                          print(wacky_name)
```

It's a convention in programming to use the letters i, j, and k for loop variables, unless there's a good reason to give the variable a more descriptive name.

2.3 The `range` function

The value we put in the `range` function determines how many times we will loop. The way `range` works is it produces a list of numbers from zero to the value minus one. For instance, `range(5)` produces five values: 0, 1, 2, 3, and 4.

If we want the list of values to start at a value other than 0, we can do that by specifying the starting value. The statement `range(1,5)` will produce the list 1, 2, 3, 4. This brings up one quirk of the `range` function—it stops one short of where we think it should. If we wanted the list to contain the numbers 1 through 5 (including 5), then we would have to do `range(1,6)`.

Another thing we can do is to get the list of values to go up by more than one at a time. To do this, we can specify an optional step as the third argument. The statement `range(1,10,2)` will step through the list by twos, producing 1, 3, 5, 7, 9.

To get the list of values to go backwards, we can use a step of -1. For instance, `range(5,1,-1)` will produce the values 5, 4, 3, 2, in that order. (Note that the `range` function stops one short of the ending value 1). Here are a few more examples:

Statement	Values generated
`range(10)`	`0,1,2,3,4,5,6,7,8,9`
`range(1,10)`	`1,2,3,4,5,6,7,8,9`
`range(3,7)`	`3,4,5,6`
`range(2,15,3)`	`2,5,8,11,14`
`range(9,2,-1)`	`9,8,7,6,5,4,3`

Here is an example program that counts down from 5 and then prints a message.

```
for i in range(5,0,-1):
    print(i, end=' ')
print('Blast off!!')
```

```
5 4 3 2 1 Blast off!!!
```

The `end=' '` just keeps everything on the same line.

2.4 A Trickier Example

Let's look at a problem where we will make use of the loop variable. The program below prints a rectangle of stars that is 4 rows tall and 6 rows wide.

```
for i in range(4):
    print('*'*6)
```

The rectangle produced by this code is shown below on the left. The code `'*'*6` is something we'll cover in Section 6.2; it just repeats the asterisk character six times.

```
******                                  *
******                                  **
******                                  ***
******                                  ****
```