Chapter Two

Some text, some maths and going loopy

In this Chapter you are going to:

◆ Learn how to do some more with text.
◆ Get Python to do some maths for you.
◆ Learn about how loops work.
◆ Learn lots of useful operators.

This is a fun chapter as we get to start real programming!
Try opening IDLE in interactive mode and enter the text in Code Box 2.

```
print("Question: What goes clip?\nAnswer: A one legged horse.")
```

If you have not pressed your enter key yet to see what happens, do so now.

You should have discovered \n has a special purpose. It is called an escape sequence. Table 1 shows some more. Try writing a variety of little programs in IDLE using them and then in your own words fill in the the right-hand column. If you do not like writing in books you could use a pencil!

<table>
<thead>
<tr>
<th>escape sequence</th>
<th>what they do</th>
</tr>
</thead>
<tbody>
<tr>
<td>\n</td>
<td></td>
</tr>
<tr>
<td>\t</td>
<td></td>
</tr>
<tr>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;</td>
<td></td>
</tr>
</tbody>
</table>

[ Table 1 – escape sequences ]

If you are a bit confused about the last two, try running this code:

```
print("Here is a speech mark: \" and here is a slash: \\")
```

The backslash is used to “escape” characters that are used in Python: When we want to print some text to the screen we wrap it in speech marks. This means there is a problem if you want to type some speech marks. Well, now you know what to do about it – put a backslash before it. So what do you do if you want to actually print a backslash to the screen? Put a backslash before it!
Maths

Using *Python* as a calculator is easy if you remember two things. In computer programming, in almost all languages, the multiplication symbol is an asterisk and the division symbol is a forward slash:

```
>>> 10/4
2.5
>>> 3*3
9
>>> 
```

[ figure 6 - some sums using Python ]

There is another way of dividing. If you use two forward slashes instead of one, *Python* will produce an integer as an answer. An integer is a whole number (a decimal such as 2.5 is called a *float*). You can now find the remainder with another mathematical operator called modulus. This is represented by a % sign.

```
>>> 11/4
2.75
>>> 11//4
2
>>> 11%4
3
>>> 
```

[ figure 7 - kinds of division ]

It is also possible to combine text (or *strings*) and numbers like this:

```
>>> print("11 divided by 4 = ", 11/4)
11 divided by 4 = 27.75
>>> 
>>> print("11 divided by 4 = ", 11//4)
11 divided by 4 = 2.75
>>> print("11 divided by 4 also equals: ", 11//4, " remainder: ", 11%4)
11 divided by 4 also equals: 2 remainder: 3
```
print() is called a function (these are covered in chapter 4). What print() will do, is print anything you throw at it inside the brackets. They must be separated by a comma, and strings (bits of text) must be put in speech marks. Everything inside the brackets will be printed out in order. The results from sums can also be output, but without putting the calculations in speech marks. What do you think would happen if you left in the speech marks?

Don’t forget you can also add in escape sequences.

**Coding Time**

It is time to try your own code now. Just experiment in interactive mode. It might be an idea to see what happens if you put a maths sum in speech marks in a print() statement. Here are some more Maths operators:

<table>
<thead>
<tr>
<th>operator</th>
<th>name</th>
<th>example</th>
<th>answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>multiply</td>
<td>2*3</td>
<td>6</td>
</tr>
<tr>
<td>/</td>
<td>divide (normal)</td>
<td>20/8</td>
<td>2.5</td>
</tr>
<tr>
<td>//</td>
<td>divide (integer)</td>
<td>20//8</td>
<td>2</td>
</tr>
<tr>
<td>%</td>
<td>modulus</td>
<td>20%8</td>
<td>4</td>
</tr>
<tr>
<td>+</td>
<td>add</td>
<td>2+3</td>
<td>5</td>
</tr>
<tr>
<td>-</td>
<td>minus</td>
<td>7–3</td>
<td>4</td>
</tr>
</tbody>
</table>

[ Table 2 – maths operators ]
**Going Loopy**

Computers are great at repetitive tasks. So are humans but we get bored easily! Computers are not only good at them, they are fast! Therefore we need to know how to tell them to do repeats. To do this we use a **while loop**. This runs some code while something is true and stops when it becomes false.

Suppose you were trying to write some code in a History lesson at school when you should be doing History. Your teacher might ask you to write fifty lines. Well no matter, Python can do that. Try opening **IDLE** in **interactive mode** and then enter the text in Code Box 3. You will need to press return twice at the end.

```
>>> lines=0
>>> while lines<50:
    print("I will not write code in history lessons.")
    lines=lines+1
```

[ Code Box 3 ]

Another solution to the same problem is this:

```
>>> print("I will not write code in history lessons.\n" *50)
```

[ Code Box 4 ]

TO DO: add a character commenting: “Wow, *Python* can multiply strings as well!”

The code in Box 4 is clever – look carefully to see what is happening. However, although Code Box 4 was shorter, a while loop is far more useful. It can do far more complex tasks. For example, with a while loop you can ask a computer to count to 100. Try entering this code and running it:

```
>>> number=1
>>> while number<101:
    print(number)
    number=number+1
```

[ Code Box 5 ]
How While Loops Work

OK so how does this work? To start with we create a variable and assign a value to it. A variable is a space in the computer’s memory where we can store, for example, a string or an integer. We create a variable by naming it. We called our variable number and with the equals operator we gave it the value 1 to look after.

The next line of code says “while the variable called number is less than 101 do the following”. All of the code that is indented after the colon is what is to be repeatedly performed by the computer. That is, it loops through these two lines of code until number is no longer less than 101. The last line of code

\[
\text{number=number+1}
\]

is in the loop. It keeps adding 1 the variable called number for each passage through the loop. Don’t forget the variable’s value can be changed with the equals operator at any time.

![figure 9 - A While Loop]
There are several operators you can use in a while loop. I have given you some of them in Table 2. Note how we now have another version of equals ==. This form is more like equals in maths. It is an example of a **comparative operator** which are sometimes called **logical operators**.

Therefore

```python
while number==1:
```

means “while the variable called `number` is equal to 1, do the following”.

<table>
<thead>
<tr>
<th>operator</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>equal to</td>
</tr>
<tr>
<td>!=</td>
<td>not equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
</tr>
</tbody>
</table>

[ Table 3 – logic operators ]

We use a double equals sign to compare two values and a single equals sign to assign a value to a variable.

My husband works for a bus operator in Croydon.
Chapter Two Summary

In this chapter you have learnt:
1. How to use the `print()` statement
2. How to write and run simple maths code
3. How to output a mixture of strings, maths or numbers
4. How to write a while loop

Chapter Two – Fun Time

You have learnt a lot in this chapter. It is time you practised. This will help you remember what you have learnt.

Challenge 1

◆ Write some code in IDLE so that the computer counts up to 20 in twos.

Challenge 2

◆ Write some code so that the computer outputs the 5 times table like this:
  
  $1 \times 5 = 5$
  
  $2 \times 5 = 10$
  
  $3 \times 5 = 15$

  Hint: You will need a counter variable which you could call `number`. Figure out how to write one line, then make your loop do it 10 times.

Challenge 3

◆ See if you can re-write the code in Code Box 5 in three different ways. Each program should still produce output which counts to a hundred. In your new code you are not allowed to use the less than operator `<`. Instead you should use one of these in each program: $\leq$, $>$, $!=$