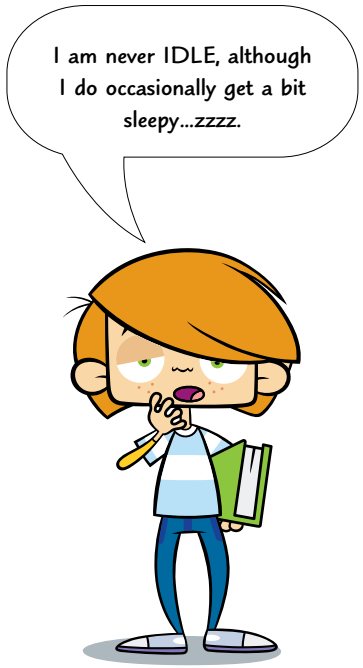
# Chapter 1
# Python, IDLE and your first program

In this chapter you are going to:

- learn about computer programming and the different languages that you can use
- meet the Python programming language
- learn how to use IDLE, which will help organise your programs and allow you to run them easily
- check that your computer has been set up correctly
- write and run your first program.



# Coding

Coding is writing instructions for a computer to perform a task. This code has to be in a form that the computer can understand. This is more formally known as computer programming.

Computers and coding have not been around for a long time but they have sure packed in some interesting history in a short space of time. The first machine that stored instructions in a way that future computers could take advantage of was the Jacquard loom that used holes punched in cards and was invented in 1801. Charles Babbage is often credited with inventing the first computer which he described in 1837 but was not built until 100 years later. In 1989 Guido van Rossum started to create the Python programming language which he named after Monty Python's Flying Circus, a BBC comedy sketch show.

# Programming languages

There are many programming languages currently used by coders around the world. Some are best in one situation, others in another.

- HTML is good for producing web pages.
- SQL is great at making databases do what you want.
- Python is brilliant for writing quick applications, running programming experiments and for building larger applications, including games.

If you have previously programmed in Scratch (produced by MIT) you will find you can pick up Python very quickly. Scratch is great for learning how to think like a programmer and is very good for making games. If you have not tried Scratch before, you might enjoy trying that next because the ability to learn a new programming language is an important skill for coders. You will find it is a lot easier than learning a new human language.

Once you have learned one modern programming language, you can quickly learn others. You simply have to find out how your new language handles variables, loops, etc. (You will know what these are by the end of the book.)

## Python

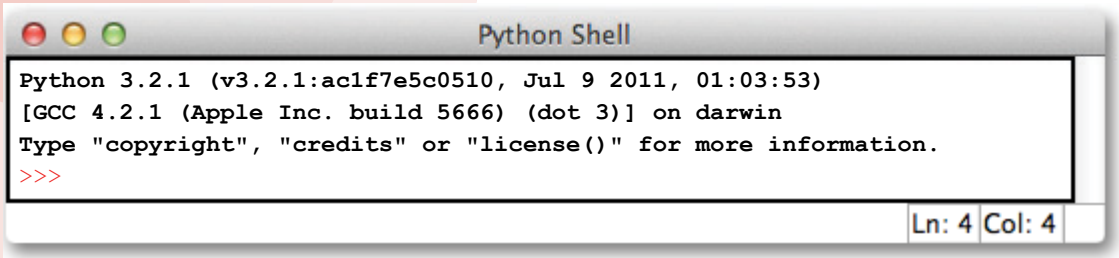
Python is a typed computer language. This makes writing short programs very fast and you can produce almost anything you can imagine.

Python is a powerful, modern programming language used by many famous organisations such as YouTube and NASA. It is one of three programming languages that can be used to write Google Apps. Python is a great language. Enjoy!


## IDLE

You will start programming in IDLE which comes with Python. IDLE is a special text editor like Microsoft Word, except it understands Python and helps you get your code right. IDLE is itself, a Python application.

Let's look at IDLE:



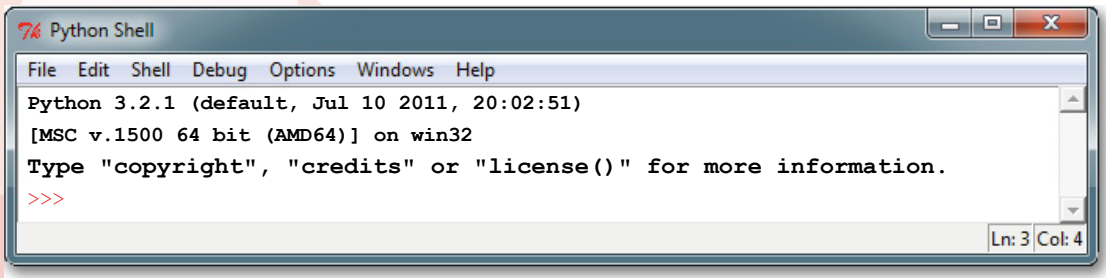

IDLE when started on an Apple Mac.

The code you want to run is typed after the special entry prompt:

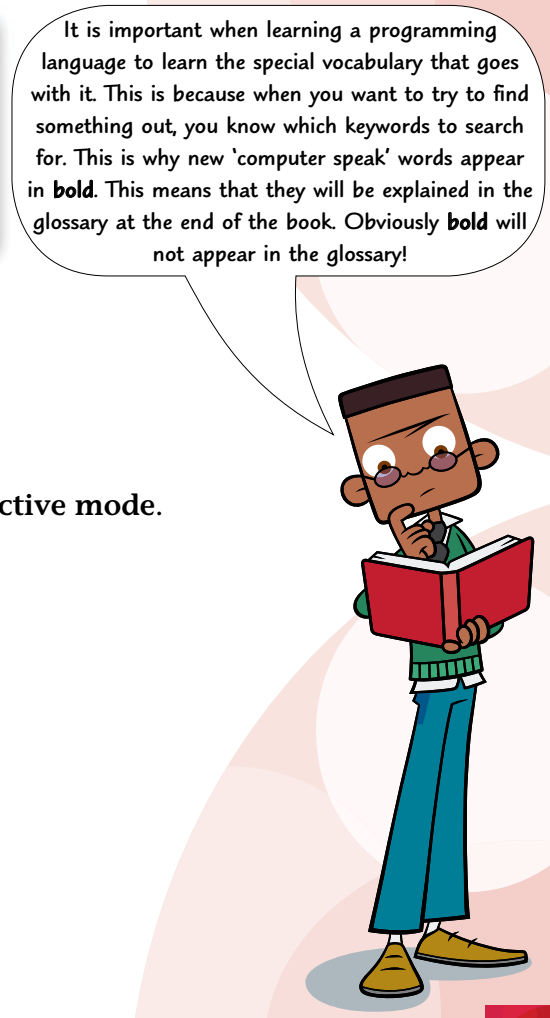
```
>>> my code goes here
```

To run the code we press the return key. This is how Python runs in IDLE's **interactive mode**. Python can run files as well but to start with, this is all we need.

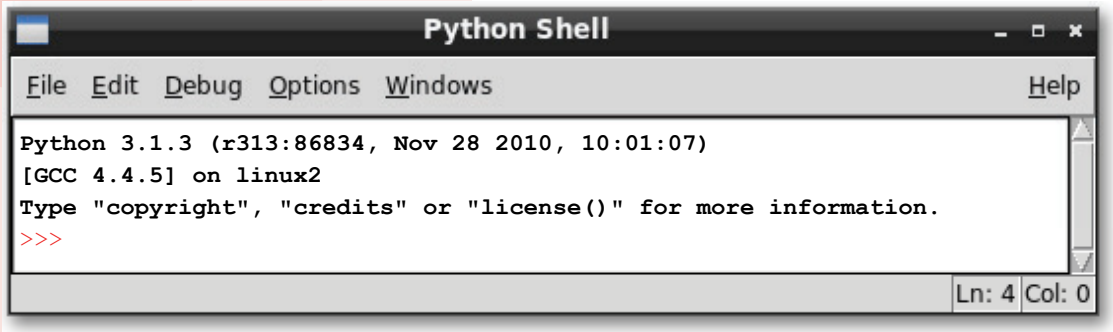
Let's see how IDLE looks on a Windows PC:




IDLE when started on Windows 7.



And finally, how IDLE looks on a Linux computer:




IDLE when started up on the Raspberry Pi computer.

A great reason for learning Python and using IDLE as our **IDE** (Integrated Development Environment) is that it is very similar on all the different types of computers available.

The text before the >>> prompt is unimportant at the moment. However, it is always useful to know what version of Python you are using.

# Hello World!

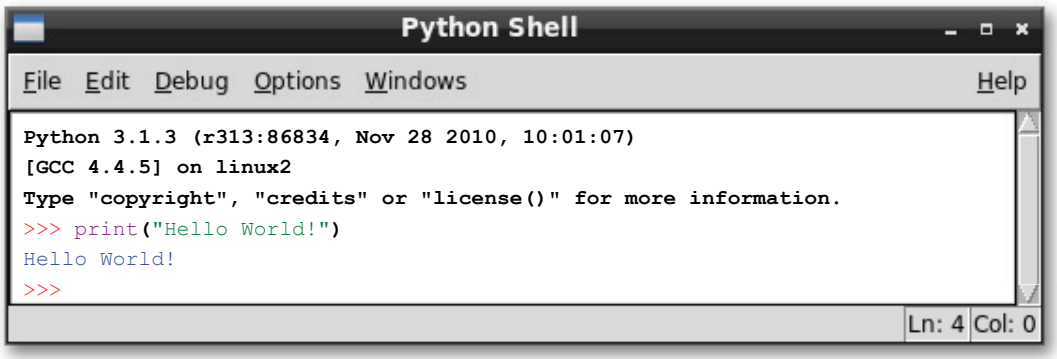
Since the dawn of programming, when the first cave-coders booted up their cave-computers, it has been a tradition that your first program when learning a new language is 'Hello World'. The aim is to try to make the computer say 'hello' to the world. If you can do this you will have tested whether everything that was set up for you is working properly.

- If it is not already started, start up IDLE.
- After the >>> prompt write in the code from Code Box 1.1 and then press your return key to run the program.

**Code Box 1.1**

```
print("Hello World!")
```

If all is well, you should get something like this:




Hello World!

Python has followed your instruction and **output** 'Hello World!'

You have written your first computer program. Well done!

# Making mistakes

## Did you get a syntax error?

**Syntax errors** are very common when typing in code (as are other errors). If you make one or two it is not your fault. It is because although computers are fast, they can also be a bit stupid. If there are any tiny mistakes in your code, they panic and produce error messages. These messages try to explain to you what the problem is but they are often difficult to understand.

Colons, brackets, speech marks, apostrophes and spelling of Python words have to be just right. Although we can read imperfect sentences, computers cannot.

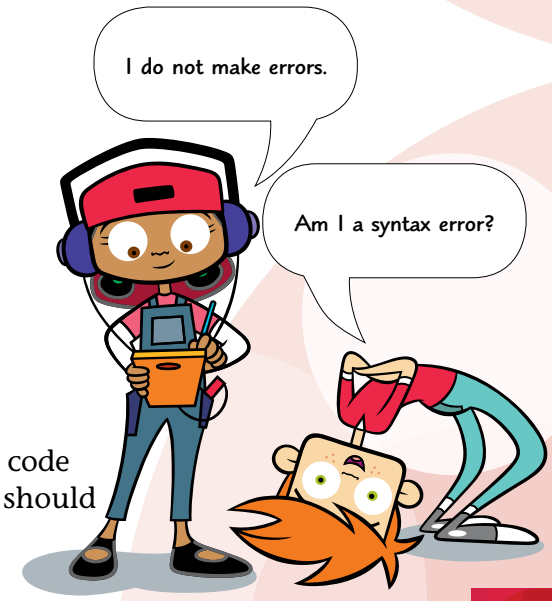
Whether or not you got any errors, try this Quick Quiz.

### Quick Quiz 1

Which of these lines of code are correct?

```
1 Print("Hello world!")
2 print("Hello world!")
3 print(Hello world!)
4 print "Hello world!"
```

Notice how the coloured text helps you spot code that is not going to work. All the code listings in this book use the same coloured text as in IDLE's standard display. This should help you to spot **bugs** in your code.



# Chapter summary

In this chapter you have learned:

- that programming is writing instructions for computers
- that there are many different computer languages
- why Python is a great language to learn
- how to use IDLE in interactive mode
- how to write and run a simple program
- that the `print()` command means 'show on the screen' not 'send to the printer'.

## Idea 1

1 Write some new code so that a short message is displayed that says thank you to whoever got everything ready for you.
2 Run your new code to display the message.
3 Now show them your message. This will make them happy.

## Idea 2

1 Write some code so that the computer will show the text for a joke.

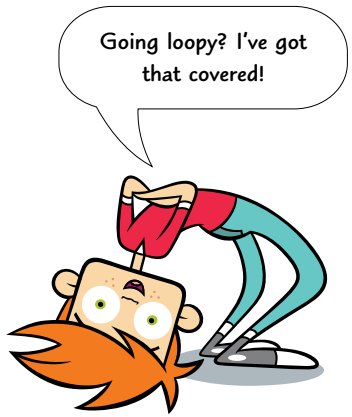
```
>>> print("Question: What goes clip?")
Question: What goes clip?
>>> print("Answer: A one legged horse")
Answer: A one legged horse
>>>
```

# Chapter 2
# 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 `while` loops work
- learn lots of useful operators.

This is a fun chapter as we get to start some real programming!



# Text

## Escape sequences

Try opening IDLE in interactive mode and enter the code in Code Box 2.1.

**Code Box 2.1**

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

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

You should have discovered `\n` has a special purpose. It is an example of an **escape sequence**. Table 2.1 shows some more escape sequences.

| Escape sequence | What it does |
|---|---|
| `\n` | creates a line return in a string of text |
| `\t` | creates a tab style indent in a string of text |
| `\\` | allows a backslash to appear in a string of text |
| `\"` | allows a speech mark to be used in a string of text |

**Table 2.1** Escape sequences.

## Experiment

Try writing a variety of little programs in IDLE using the escape sequences in Table 2.1 until you feel you know what they all do.

## Backslashes

Are you a bit confused about the last two escape sequences? If so, type in and run the code from Code Box 2.2.

**Code Box 2.2**

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

Try typing in the code from Code Box 2.3 to see how to avoid having to escape speech marks. This takes advantage of the fact that you can choose whether to surround strings in double speech marks or single ones. Watch out though, you will get a lot of syntax errors if you do not do this carefully.

**Code Box 2.3**

```
print('I say "High", you say "Low". You say "Why?" and I say "I don\'t know". Oh no.')
```

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 now means that there is a problem if you want to type some speech marks. Well, 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!


## Functions

`print()` is called a **function** (these are covered in chapter 4, page 53). 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 you must not put 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.

# Maths

Using Python as a calculator is easy, if you remember two things:

1. In Python, as in almost all programming languages, the multiplication symbol is an asterisk.
2. The division symbol is a forward slash.

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

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 the **modulus**. This is represented by a `%` sign.

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

Table 2.2 lists some more mathematical operators.

| Operator | Name | Example | Answer |
|---|---|---|---|
| * | multiply | 2*3 | 6 |
| / | divide (normal) | 20/8 | 2.5 |
| // | divide (integer) | 20//8 | 2 |
| % | modulus | 20%8 | 4 |
| + | add | 2+3 | 5 |
| - | subtract | 7-3 | 4 |

**Table 2.2** Maths operators.

## Experiment

In interactive mode, check that the examples in Table 2.2 do give the correct answers and then try out some of your own favourite sums. You might like to see what happens if you wrap a maths sum inside speech marks in the `print()` function.


# Combining text and maths

It is also possible to combine text (or strings) and numbers in the `print()` function. The comma is used here as a separator between the text and the maths.

```
>>> print("111 divided by 4 = ", 111/4)
111 divided by 4 = 27.75
>>>
>>> print("11 divided by 4 = ", 11/4)
11 divided by 4 = 2.75
>>>
```

## Quick Quiz 2

Can you work out what the output from this code will be?

```
>>> print("11 divided by 4 also equals: ", 11//4, " remainder: ", 11%4)
```

# 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.

Imagine 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 code in Code Box 2.4. You will need to press return twice at the end.

**Code Box 2.4**

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

Here is another solution to the same problem:

**Code Box 2.5**

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

The code in Code Box 2.5 is clever – look carefully to see what is happening. Run it if you are not sure. Although the code in Code Box 2.4 is longer, a while loop is often more useful as it can do far more complex tasks. For example, with a while loop you can ask a computer to count to 100. Try entering the code from Code Box 2.6 and running it.

**Code Box 2.6**

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

## How do while loops work?

### Variables

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. In Code Box 2.6 we called our variable 'number' and with the **equals operator** we give it the value '1'.

The next line of code `>>> while number<101:` says 'while the variable called `number` is less than `101` do the following'. All of the code that is indented after the colon 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`.

create a variable and assign it the value 1

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

the colon says to run all of the indented code

there are no speech marks here as we want the value of `number` not the word "number"

the variable `number` is incremented by 1

IDLE automatically indents for you!

The last line of code `number = number+1` is in the loop. It keeps adding `1` to `number` for each passage through the loop. Don't forget the variable's value can be changed with the equals operator at any time.


## Delving Deeper

The equals sign is used differently to the way it is used in maths. In computing, the equals sign means 'point this variable name at this piece of data' (an integer for example). So `number=1` means 'create a variable called `number` and point it at the integer `1`'. Another way of saying this is 'assign the value `1` to the variable `number`'. Later we may assign another value to `number`.

## Operators

There are several operators you can use in a while loop. Some examples are given in Table 2.3. Note how we now have another version of equals ==. This form is more like the equals in maths. It is an example of a **comparative operator**. Therefore, `while number==1:` means 'while the variable called `number` is equal to `1`, do the following'.

| Operator | Meaning |
|---|---|
| == | equal to |
| != | not equal to |
| > | greater than |
| < | less than |
| >= | greater than or equal to |
| <= | less than or equal to |

**Table 2.3** Comparative operators.

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

# Chapter summary

In this chapter you have learned:

- how the `print()` function is very flexible
- how to write and run simple maths code
- how to output a mixture of strings, maths or numbers
- how to write a while loop and use comparative operators

## Puzzle 1

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

## Puzzle 2

Write some code so that the computer outputs the 5 times table like this.

```
1x5=5
2x5=10
3x5=15
```

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

## Puzzle 3

See if you can re-write the following code in three different ways so that each program still produces output which counts to a hundred.

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

In your new code, you are not allowed to use the less than operator <.
Instead you should use one of these comparative operators in each program:
<= > !=

Answers to all of these puzzles can be found on the companion website www.codingclub.co.uk.