

# Introduction to Programming

## Contents

Introduction to Programming .....	1
Lesson 1 – What is a computer program .....	4
Hello World .....	5
How Programs are run (very simplified) .....	5
Interpreters .....	6
REPL.....	6
Lesson 2 - Defining the problem in words and pictures .....	7
Defining the problem in words and pictures .....	7
Pseudo code.....	7
Flow charts.....	9
Lesson 3 – Program Documentation.....	16
Lesson 4 – Python Programming constructs.....	18
Writing and Running Python code .....	18
Python .....	19
Python programming constructs .....	21
Lesson 5 – Coding environments .....	27
Lesson 6 – Debugging.....	29
Types of Errors .....	29
What if I don't have any tools.....	30
How does Debugging differ from Testing? .....	30
Lesson 7 – More Python .....	31
Reading and Writing to a file .....	31
Lists .....	32
String Functions .....	33
Defining Functions .....	33
Using parameters in main program .....	34
Python on-line documentation .....	35
Lesson 8 – Introduction to Testing.....	36
Testing approach.....	36
Constructing a plan .....	36
Testing.....	36
Recording Results.....	38

VVT .....	38
Lesson 9 – Re-using code and packages .....	39
Introduction to pip .....	39
Lesson 10 – Introduction to Small Programs .....	40
Program details .....	40
You should end up producing: .....	40

## Lesson 1 – What is a computer program

Relying as we do on the Internet, it is easy to find a definition of a computer program. Two of which are given below.

**computer program** - a sequence of instructions that a computer can interpret and execute.

(The Free Dictionary)

A **computer program** is a collection of instructions that performs a specific task when executed by a computer.

(Wikipedia)

[**Questions** – What is the difference between sequence and collection? Which of the two definitions do you think is better and why?]

Using the definitions above, if I wanted to define the term ‘human program’ I might come up with something like : “A Human program is a sequence of instructions that a Human can interpret and execute in order to perform a specific task”

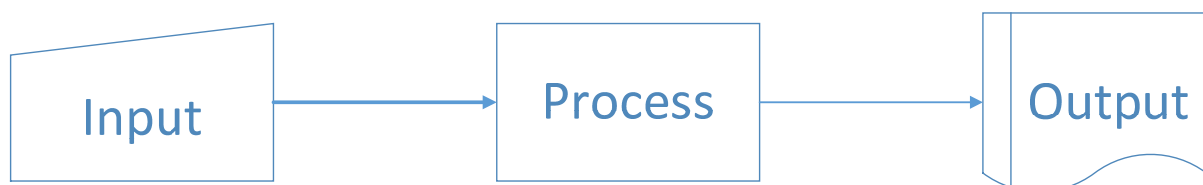
We don’t actually use the term ‘Human program’ we would simply be ‘performing a task’ or even just doing something.

What would be different between a computer program and a Human program, is that different Humans might interpret the instructions differently or the same human might execute the sequence of instructions in a different order on another occasion even though ultimately they would complete the same task.

There will be many different ways of writing a computer program to perform a given task. However, for any given program a computer will always use the same interpretation and the same sequence every time the program is run.

**The one thing computers cannot cope with in ambiguity.**

Another way of looking at a computer program is by considering the inputs and outputs. In its simplest form a computer program, takes input, processes it in some way and produces output.



The process section in the middle is what would be considered the ‘computer program’, although it will also deal with accepting the input by some means and making the output available outside of the program.

The input could be something typed on the keyboard or it could be a mouse click or it could be lines of data read from a file. Similarly, the output could be written to the screen or could be data written to a file

[**Question** – Other examples of input or output]

In computer programming the term ‘Algorithm’ is often used. An algorithm is a step-by-step set of instructions required to complete a given task. This is in fact very like our definition of a program. For small programs, you can probably think of them as being the same, except people don’t generally go around saying that they have ‘written an algorithm’. A larger more complex program may make use of several algorithms to complete a task, each algorithm being used for some sub-task within it.

Computer programs are written in some language or other, an Algorithm is more generic in nature; a given algorithm can be implemented in almost any programming language.

### Hello World

Traditionally the first program a new programmer is shown how to write is a ‘Hello World’ program. The program simply writes the expression ‘Hello World’ on to the screen. An example of such a program, written in the C programming language is shown below.

```
main( ) {  
    printf("hello, world");  
}
```

*"Hello, World" source code. This first known "Hello, world" snippet from the seminal book The C Programming Language originates from Brian Kernighan and Dennis Ritchie in 1974.*

(Wikipedia)

[**Question** - If you hadn’t been told what this program does, how likely would you have been to guess correctly?]

### How Programs are run (very simplified)

When you write a computer program, you write ‘source code’ and it is simple text – straight from the keyboard. Source code is purely for the benefit of the Human reader. It means nothing to a computer.

The source code is given as input to a program called a compiler. The compiler reads your source code and produces output called machine code and writes this to a file – this is the output of the compiler program.

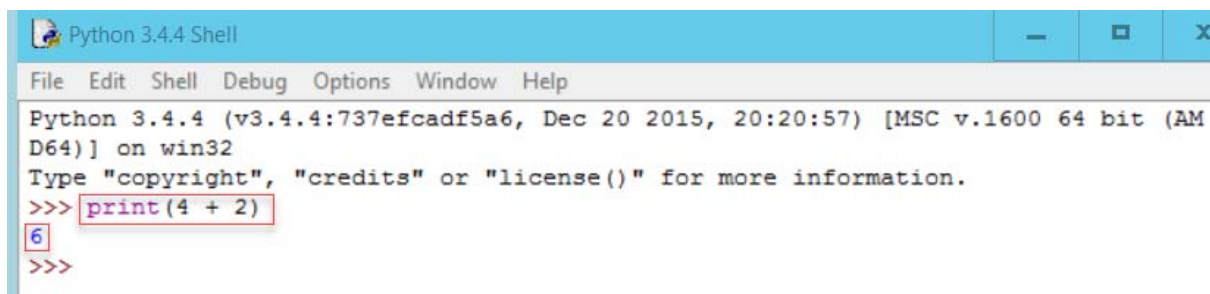
This file of machine code is then given as input to a program called an operating system which executes the instructions in the machine code file and depending on what the instructions say, produces the output for your program.

## Interpreters

The above description is how all programs used to be run, some languages like C still work this way. Many of the more recent languages such as Python use a program called an Interpreter which effectively combine the compile step and the execution step together. You effectively ask the Python Interpreter to read your source code, create the machine code and run it in a single step.

## REPL

REPL which is an acronym for Read, Execute, Print, Loop is programming development environment which is provide by some interpreted languages, like Python. The REPL allows you to write code a line (or more accurately a statement) at a time and have it executed and any output displayed immediately. This can be very useful when developing code. We will be using the Python REPL today when we write some of our code.



```
Python 3.4.4 Shell
File Edit Shell Debug Options Window Help
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 20:20:57) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print(4 + 2)
6
>>>
```

(The IDLE REPL environment for Python)

## Lesson 2 - Defining the problem in words and pictures

### Defining the problem in words and pictures

Before we can start writing code we need to make sure that we have a clear definition of the problem we are trying to solve or of the task we wish to perform. It is very easy to say 'oh yes, I know how to do that'. It is only when you try to put your intended actions into words that potential difficulties, unforeseen circumstances or even questioning your starting assumptions come to mind.

Before we start on the computer programs we are going to practice on a few human programs, simple everyday tasks. Things that most of us will have done many times before without thinking too much about them.

#### [Exercise]

Select a task from the list below and write instructions in English as to how the task is to be completed. You can use either prose or as a set of bullet points.

- Make a cup of tea
- Boil an Egg

You may assume that you are in a well-designed kitchen and everything you are going to need is within arm's reach.

Most people when given the choice will choose to write the instructions as a set of bullet points. It is a more natural way of setting out instructions and implying an order of execution.

People will do this even though, if decisions have to be made as part of the process, a linear, ordered sequence of instructions is difficult to construct.

When you are designing code, there are two techniques that you can employ (and you should employ at least one of them) before you start actually writing the code. In fact, you can do this even before you decide on which programming language you intend to use.

The two techniques are pseudo-code and flowcharts and we will discuss both of these in turn.

### Pseudo code

Pseudocode is a made-up informal language that enables coders to easily and quickly write down their thought processes in an algorithmic type way. Pseudocode is a "text-based" detail (algorithmic) design tool.

Although pseudocode is made up, you can have your own version, there are a few rules and constructs which are normally adhered to.

If a line is dependent on the line before, you indent the line. for example;

```
If student's grade is greater than or equal to 60
    Print "passed"
else
    Print "failed"
endIf
```

Although you can use simple English, there are some constructs which are frequently used.

To denote	Use
<b>Loops</b>	Do While...EndDo
	Do Until...EndDo
<b>Selections</b>	If...Endif
	If...Else...Endif
	Case...EndCase
<b>A process or sub-process</b>	Generate
	Compute
	Process
<b>An individual action</b>	set, reset, increment, compute, calculate, add, sum, multiply, print, display, input, output, edit, test ...

A more substantial example:

```
set total, counter and average to 0
Input the first score
Do while the user has not entered the final score
    add this score to the total
    add one to the counter
    input the next score
EndDo

if the counter is not equal to 0
    set the average to the total divided by the counter
    print the average
else
    print 'no grades were entered'
endif
```

Further details on using pseudocode can be found at theses web sites




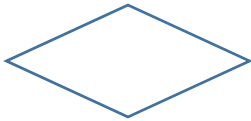

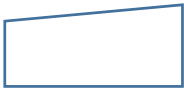



<http://www.unf.edu/~broggio/cop2221/2221pseu.htm> and  
<http://www.bbc.co.uk/education/guides/z3bq7ty/revision/1> .



## Flow charts


A flowchart is a diagrammatic way of representing your algorithm or program. Simple shapes are used to represent the various elements (lines of code) in your program. There is a standard set of shapes which are used to depict the different elements.

### Flowchart Symbols

	Terminator (Start and Finish)
	Process
	Subprocess
	Decision
	Off page connector
	Manual Input
	Internal Storage
	Document
	Stored data

The table above just shows the more common symbols used. A search online will find a lot more. Even the flowcharting shapes in Word have a more comprehensive selection,

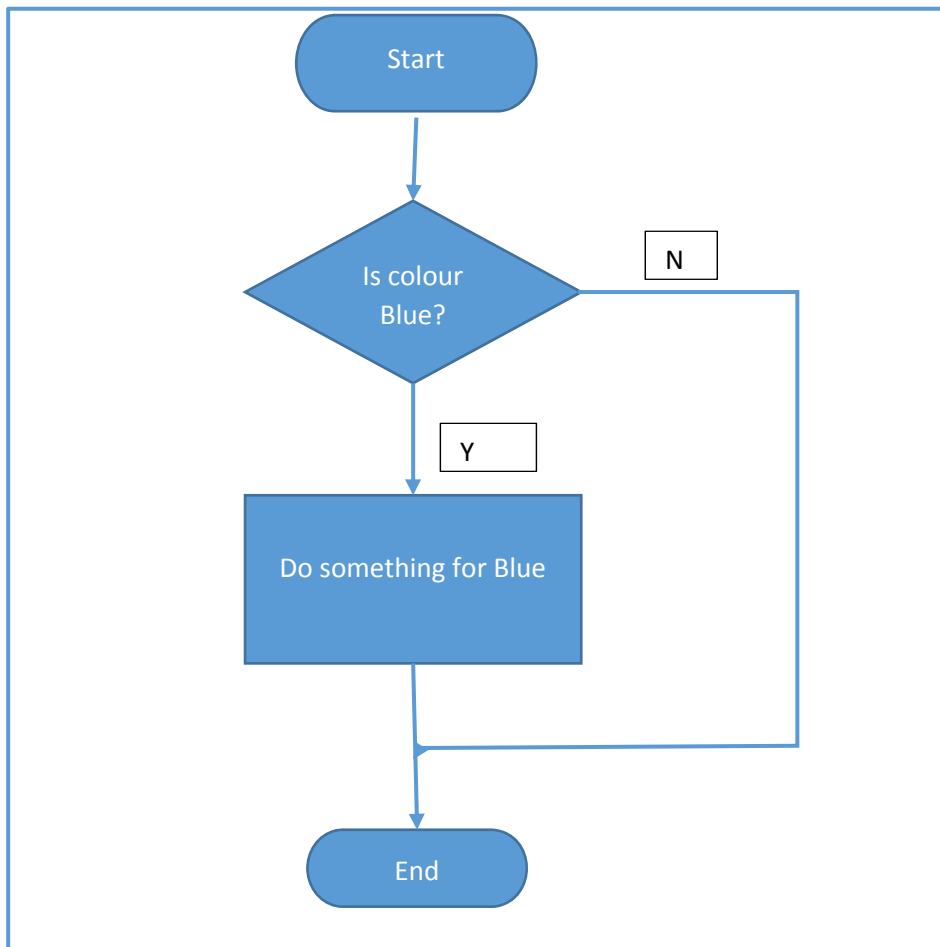


but in reality not many people used punched tape any more. 

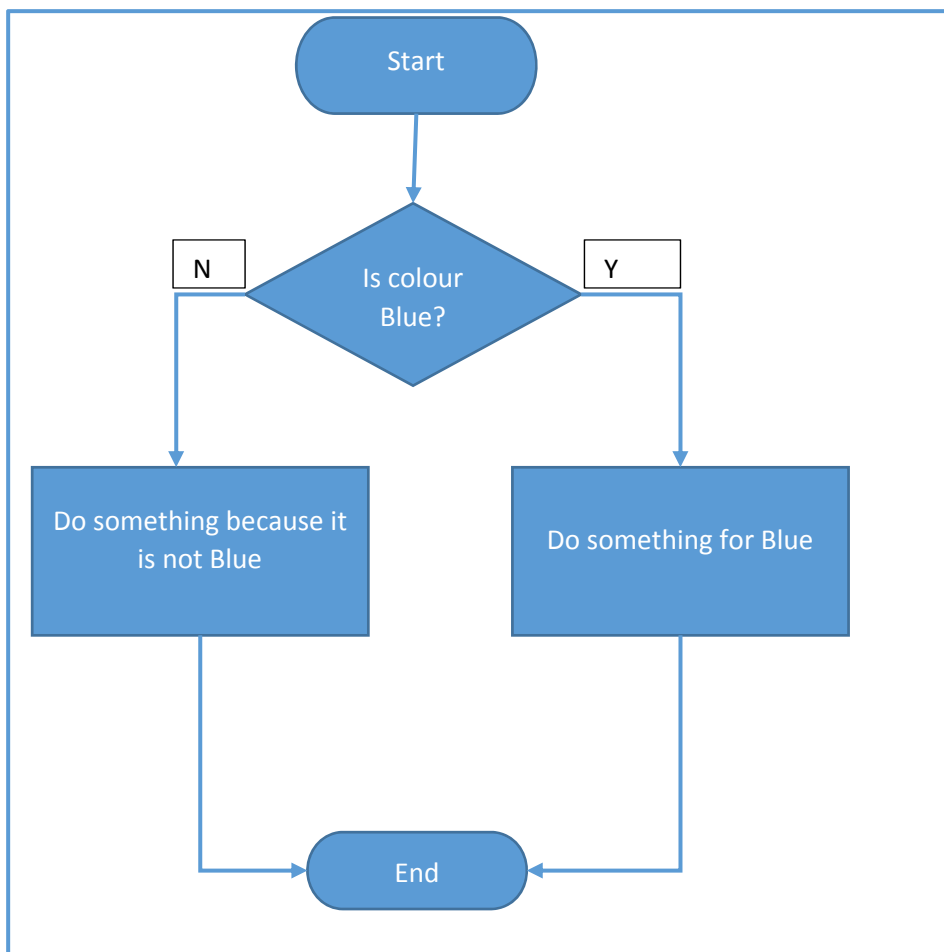
With suitable annotation of the shapes and directed arrows connecting the shapes it is possible to construct a diagrammatic representation of your program.

Rather than trying to construct a whole program, we will start by illustrating some of the pseudo-code structures from the table above.

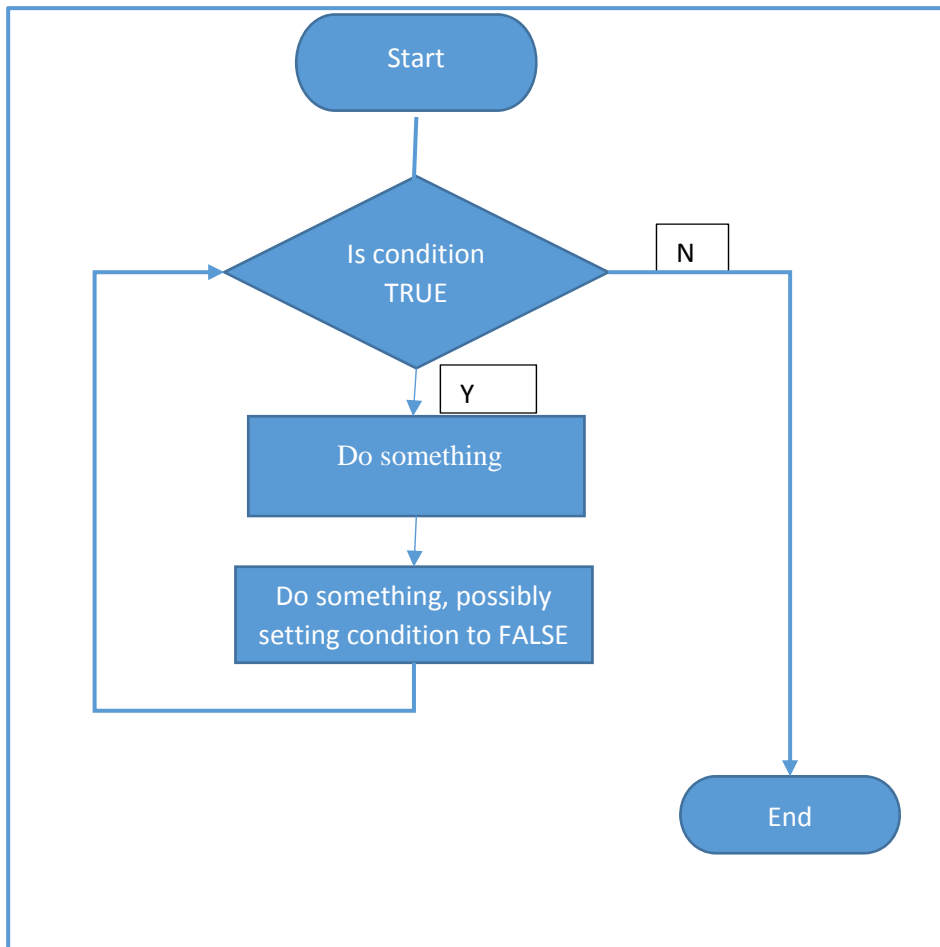
### **If... Endif**



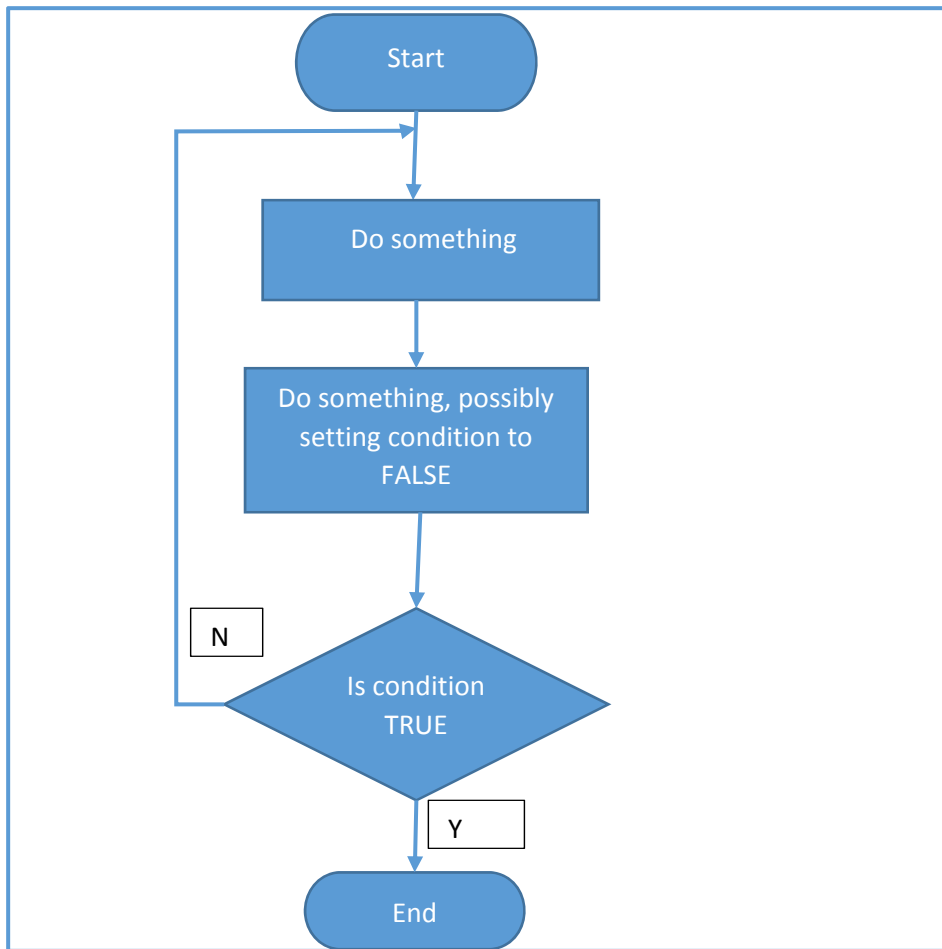
### If...Else...Endif



## Do While...EndDo



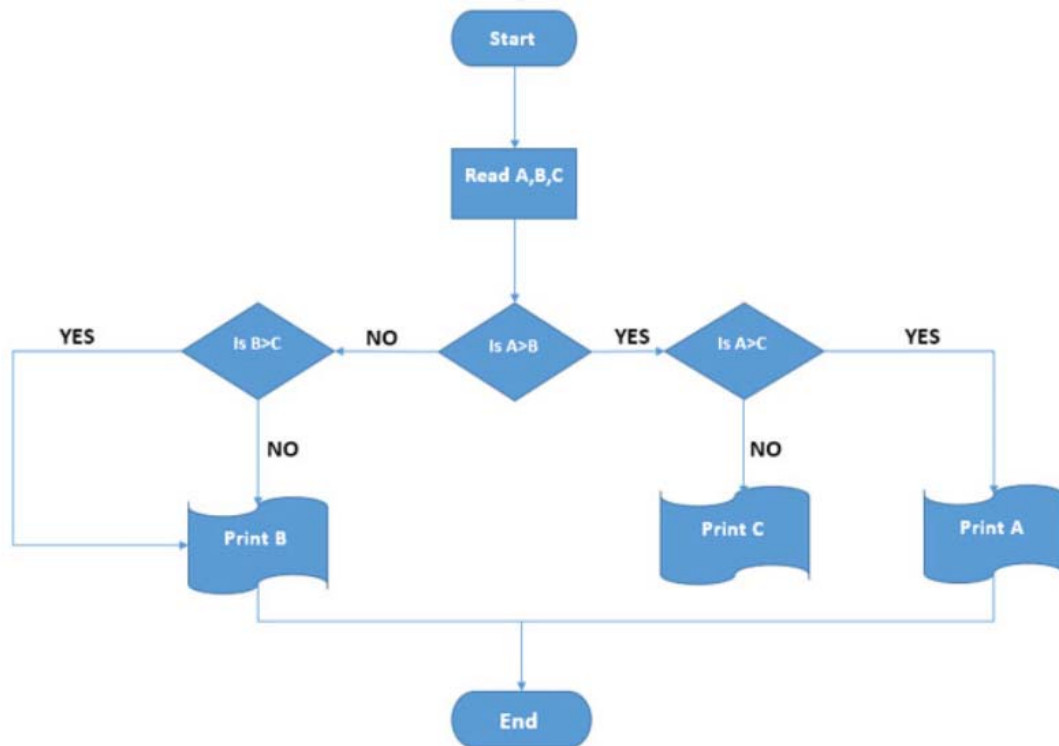
## Do Until...Enddo



The following web site provides instructions on how to create flowcharts using Microsoft Word

<http://www.makeuseof.com/tag/create-stunning-flowcharts-microsoft-word/>

It includes the following example



The text explains that this flowchart can be used to find the largest of three numbers (A, B, C)

[ **Exercise** – Find two things wrong with the flowchart. Can you re-draw it so that it does what it is meant to? ]

[**Exercise** – Convert your English task into pseudo-code and a flowchart.]

In practice, you only need to use one of pseudo-code or a flowchart, whatever you are more comfortable with.

For Large systems where there may be many programs which interact with each other, then an overall system flowchart is quite common.

## Lesson 3 – Program Documentation

All computer programs should be documented in some way. The extent of the documentation will largely depend on the complexity of the program being written. If the program is being written as part of a larger project, then the project might dictate the standards of the documentation required. Hopefully the project may also provide you with a set of documentation templates which can be used.

Documentation can be divided into two areas; documentation external to the program code and documentation used within the program code file.

### External Documentation

We have already seen examples of external documentation. The program title and any pseudo-code or flow charts contribute to the documentation. In most cases you would expect that a program has a meaningful name, which gives some indication of what it does. You would also expect a descriptive paragraph explaining how the program works, mentioning where the input comes from and what output it produces. The pseudo code and flow charts are not a substitute for this descriptive paragraph but can be included to aid understanding.

### Internal Documentation

All programming languages allow the programmer to include comments in the file of program code. These comments will be ignored by the language interpreter and are there purely to assist a human reader of the code to help them understand what is going on.

Different languages have different ways of allowing the programmer to write comments. In Python, there are two different ways.

```
# A comment on a line by itself by starting the line with the '#' sign

print("Hello world") # everything after the '#' is also a comment

"""
You can also have multiline comments
by using the triple quotes at the beginning and end
"""
```

You can use whatever is the more appropriate. However it is common practice to triple quotes comments immediately after the 'def' statement of a function as this has special meaning if the function is part of a class or separate module (Further explained in Lesson 9).

It is good practice to start a code file with a block comment in which you can provide basic information about the program.



```
"""
Program Name :
Author       :
Date written :
Description  :

Inputs      :
Outputs     :

Calls       :
Is called by :
"""
```

You can use your own headings and the complexity of the program may influence how much information is appropriate.

Single line comments or comments after a code statement are used to aid understanding of particular parts of the code. They can be overused and cause distraction. Only use them to introduce a new section of the code or to clarify potentially confusing parts of an algorithm.

While you are developing a program, you can use comments to help your own understanding of what you think should be happening. (We will look at this a bit more in Lesson 6 – Debugging)

## Lesson 4 – Python Programming constructs

So far, we have specified our programs in terms of either pseudo-code or as flow chart diagrams. This can always be done irrespective of the programming language you intend to use.

In almost any programming language, the constructs that we have been using to specify our programs can easily be translated into genuine programming code for the language.

For this workshop we are going to use the Python programming language. This is a freely available language which can be downloaded and installed on any PC.

It is a very popular general purpose language which is used extensively for data manipulation and data analysis.

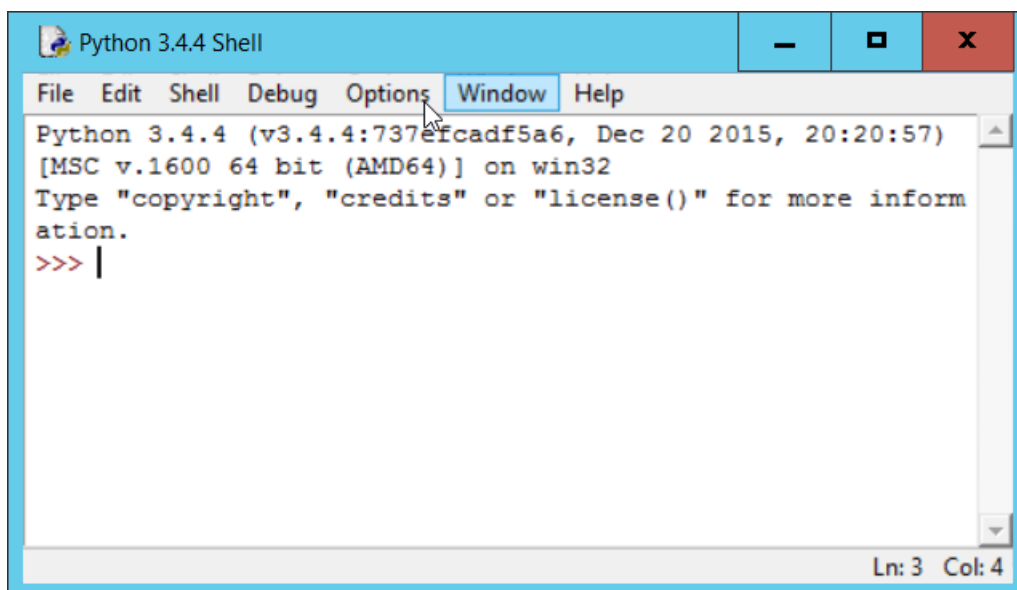
The purpose of this workshop is **not** to teach you the Python language but to provide you with sufficient information about its constructs to allow you to develop your own small programs.

The full documentation of the Python Language can be found at the official web site [www.python.org](http://www.python.org) There are two current versions of Python a v2.x and a v3.x. We are going to be using the v2.x version today.

The constructs that we are interested in are those which allow you to translate your flowchart or pseudo-code into Python code.

### Writing and Running Python code

The source code of any computer program is written as text. Consequently, it can be written in any text editor, such as Notepad and the code can be executed (i.e. the program run) from the command line. Although you may wish to run your completed Python program from the command line, in practice no-one really uses Notepad to write Python code. Instead we will use the Python IDE (Interactive Development Environment) called IDLE. IDLE is installed as part of the Python installation. It will allow you to write your code and execute it all from within the same environment.



## Python

The aim of this section is to introduce you to the Python language constructs which will allow you to write in Python code the basic constructs depicted by the flowchart elements we have discussed.

But before we get on to them there are a few other bits of the Python language we need to know about just so that when you write code you can demonstrate that it has some meaning and works correctly.

You will recall that one of our interpretations of a computer program was something which takes input, processes it and produces output. We will see how to provide simple input and output using the Python Input and Print commands.

## Comments

In python comments are anything which follows the '#' symbol up to the end of the line. It can appear at the beginning of the line or after a code statement.

```
# This is a comment at the beginning of the line
print("Hello World")  # This is a comment after the code
```

The code is in the comment.py file

## Simple Data types

In Python, you do not need to say what kind of data is stored in a variable. Python will assume the data type what values you assign and how you use them. Examples are shown below;

```
# simple data types

a = 5
b = 4.3
c = "hello"
d = a + b
print (a,b,c)
print( type(a), type(b), type(c), type(d))

if type(b) is int :
    print(b, "is of type integer")
```

Points to note;

1. We will only be using integers (a), floating point numbers (b) and strings (c).
2. The type() function will tell you what type a variable is
3. If you are checking the type in code you use the 'is' operator not the '==' operator
4. variable d is a float because variable b is. If b had been an integer like a then d would also have been an integer

The code is in the simpletypes.py file

## Print

As the name suggests, the print command will print output, by default to the screen. You can print several items with the same print statement by separating them with a comma. Here are some examples of the print statement.

```

a = 3
print(a)
b = 6.2
print(a,b)
c = "Hello World"
print(c)
print(a, c, b)
print(a+b)

d = "to Peter"
print(c,d)
print(c + d)

```

The code is in the print.py file

### Input

The simplest form of the command is:

```
inputvalue = input()
```

This will allow the user to type some input at the keyboard. Python will assume that they have finished when they hit the Return key.

To make it clearer that the user is to provide input you can include a prompt string when you call input.

```
inputvalue = input("What is your Name? ")
```

The prompt string is output to the screen for the user to see.

```

inputvalue = input("give me a number ")
print(type(inputvalue))

int_inputvalue = int(inputvalue)
print(type(int_inputvalue))

inputvalue = input("give me a number ")
print(type(inputvalue))

int_inputvalue = float(inputvalue)
print(type(int_inputvalue))

```

Whatever the user types is stored in the variable inputvalue. The value is always stored as a string of text. What this means is that if the user types 42, this will be stored as the string with value '42'. So if you wanted to use such a value in arithmetic you would have to convert the value to an integer.

Because this is such a common requirement, Python provides a set of functions to perform conversions between different data types (providing it makes sense to do so). To convert a string to an integer you use the `int()` function. If the string is a number with a decimal point, then you use the `float()` function.

The need to convert input 'strings' to numerical values can cause problems. What if we are expecting an integer and we are given a string. We cannot just check the type, because we know that it will be a string regardless. But if we just try to convert the string to an integer we will get an error if a character string has been given and cannot be converted successfully.

In order to deal with this type of problem many programming language have a 'try...except' construct. It is not something we looked at in the flowcharting session, but it is in fact very similar to the 'if...else...endif' construct. An example which deals with the problem is shown below.

```
# try... except
x = "www"
try:
    int_x = int(x)
    print(x + 1)
except:
    print(x, "is not an integer")

x = 42
try:
    int_x = int(x)
    print(x + 1)
except:
    print(x, "is not an integer")
```

The code is in the tryExcept.py file

## Python programming constructs

### *If...EndIf*

Below are some examples of the 'If' construct in Python

```

# if ... End If

a= 5
b= 4
print("a is", a, "b is",b)
if a > b :
    print(a, "is bigger than ", b)

a= 3
b= 4
print("a is", a, "b is",b)
if a > b :
    print(a , "is bigger than ", b)

a= 4
b= 4
print("a is", a, "b is",b)
if a == b :
    print(a, "is equal to", b)

```

The code is in the if.py file in the code folder.

There are three things to notice;

1. The colon ':' at the end of the 'if' line. Missing this out is a common error.
2. The indentation of the print statement. If you remembered the ':' on the line before, IDLE will automatically do the indentation for you. All of the statements indented at this level are considered to be part of the 'if' statement
3. The equivalent of the 'EndIf' is removing the indent.

In the last example, notice that in Python the operator used to check equality is '=='.

#### *if ... Else ... EndIf*

Below is an example of the 'If ... Else' construct in Python

```

# If ... Else ... EndIf

a = 4
b = 5
print(a,b)

if a > b :
    print(a, " is greater than ", b)
else :
    print(a, " is NOT greater than ", b)

```

The code is in the ifelse.py file in the code folder.

The same structure and formatting rules apply as to the 'if' statement. The block of statements associated with the 'if' part is ended by the 'Else' clause not being indented. The 'Else' clause also need a ':' at the end of it.

### *If ... Elif ... Else ... EndIf*

Below is an example of the 'If ... Elif ... Else ... EndIf' construct in Python

```
# If ... Elif ... Else ... EndIf

a = 5
b = 4
print(a,b)

if a > b :
    print(a, " is greater than ", b)
elif a == b :
    print(a, " equals ", b)
else :
    print(a, " is less than ", b)
```

The code is in the ifelif.py file in the code folder.

The overall structure is similar to the 'if ... Else' statement. There are three additional things to notice;

1. Each 'Elif' clause has its own test expression.
2. You can have as many 'Elif' clause as you need
3. Execution of the whole statement stops after an 'Elif' expression is found to be True. Therefore, the ordering of the 'Elif' clause can be significant, as they are in the example above.
4. Notice that in Python the operator used to check equality is '=='

### *for*

The for construct is the basic looping construct where the number of iterations through the loop is known in advance. It is essentially a special case of the more general while construct. It has its own syntax because of its general usefulness.

Below are some examples of the 'for' loop

```

# for loop
x = 23
for i in [1,2,3] :
    print(i)

for name in ["Tom", "Dick", "Harry"] :
    print(name)

for name in ["Tom", 42, 3.142] :
    print(name)

for i in range(3) :
    print(i)

for i in range(1,4) :
    print(i)

for i in range(2, 11, 2) :
    print(i)

```

The general format of the 'for' loop is:

```

for <variable> in <sequence>:
    <statements>

```

'variable' can be any variable, typically is named and used just for that particular 'for' loop or is a general 'counting' type variable like 'i'.

Things to note about the for loop include;

1. The ':' at the end of the 'for' statement. This is required and if you are using IDLE will automatically indent the next line.
2. '<sequence>' is anything that you can count through. In the first example a simple list of integers is used and in the second a list of strings.
3. The list in the 3<sup>rd</sup> example has a mixture of data types. This is more a function of lists than the 'for' loop.
4. The last 3 examples use the range() built-in function to generate the sequence. You might think that range(3) is equivalent to the list [1, 2, 3] but in fact it is the list [0, 1, 2].
5. Similarly the range(1, 4) does not equate to the list [1, 2, 3, 4] but to [1, 2, 3]. The first parameter represents the start position in the sequence and the second parameter is one beyond the last value.
6. In the last example, the 3<sup>rd</sup> parameter is a step value, so in this case only every second value in the sequence will be used.

The code is in the for.py file in the code folder.

*while*



```
# while loop
n = 10
sum = 0
# sum of n numbers
i = 1
while i <= 10 :
    sum = sum + i
    i = i + 1
print("The sum of the numbers from 1 to", n, "is ", sum)
```

Points to note;

1. The condition clause ( $i \leq n$ ) in the while statement can be anything which when evaluated would return a Boolean value of either **True** or **False**.
2. The clause can be made more complex by use of parentheses and and and or operators
3. The statements after the while clause are only executed if the condition evaluates as **True**.
4. Within the statements after the while clause there should be something which potentially will make the condition evaluate as **False** next time around.

The code is in the for.py file in the code folder.

*do ... until*

There is no direct equivalent of the 'do ... until' construct in Python. Instead a specially designed while loop is used in conjunction with the Python Break statement

Below is an example of the 'do ... until' loop

```
# do ... until

while True :
    num = input("Give me a number >")
    print(num)
    if num == 0 :
        break
```

[Exercise – what is the error in the code above?]

Points to note about this structure;

1. The condition in the while clause is simply the Boolean value True. This means that in theory this program will loop forever (a very common programming error). Instead of using the Boolean value here some people will write an expression like ' $1 == 1$ '.
2. The if clause is used to check some other condition and if (when) it becomes True, the Break statement is executed.
3. The break statement has the effect of exiting the complete while loop.
4. You need to make sure within the statements following the while clause there is something which will allow the condition in the if clause to become True.

5. There is another Python statement 'continue' which is similar to Break but only takes you to the end of the current iteration of the loop. This can be useful in the for loop construct.

## Lesson 5 – Coding environments

As we have already said, computer program code is plain text. You can type it into any simple text editor such as Notepad (but not Word!)

### Notepad

For our 'Hello World' python example all I have to do is to give the text file with the code ( `print("Hello World")` ) in it a .py extension when I save it and then I can run the program from the commandline using;

```
python helloworld.py
```

In fact, using the .py extension here is simply convention for the naming of files containing Python program code. I could have left the extension as the default of .txt and run;

```
python helloworld.txt
```

Although simple text editors like notepad are readably available in all operating systems, and they are very easy to use they are not particularly well suited for developing code.

There are a variety of more sophisticated text editors (freely available for download) such as Notepad++ (or Atom) which are more understanding when it comes to code development

### Notepad++

In Notepad++, if you indicate that the file you are creating contains Python code (or any of the other many languages Notepad++ knows about) then Notepad++ will offer assistance as you write the code. It cannot tell you if your program is correct or indeed perform detailed checks on the syntax. It can however colour code the reserved words (that is words the programming language doesn't allow you to use as variable names), offer to complete function names and automatically indent the code as required.

### IDLE

When Python is installed an editor called IDLE is installed with it. IDLE is an example of a simple IDE (Interactive Development Environment). This allows python code to be developed and run directly from the Idle environment. In addition to the colour coding of the syntax available with Notepad++ IDLE will also provide you with argument lists when you use builtin functions or even your own functions.

IDLE also provides you with a REPL environment as well as the ability to write a file of code and have it executed.

### Jupyter Notebooks

Jupyter Notebooks, formerly known as IPYTHON Notebooks is a web server / browser system which allows you to develop python code in a way which assists reproducibility and documentation. In some ways it is not as rich in functionality as the IDLE interpreter, in that it will does not provide code completion although there is colour coding of keywords. The strength of the Jupyter notebook approach is that it enables you to integrate code, documentation (with some degree of formatting) and results (including graphics) into a single notebook

## **Pycharm**

Pycharm is a more substantial IDE than IDLE. It does all that everything before does, including running Jupyter. But it also allows your Python environment to be managed. It allows you to see what packages are installed (Packages are covered in Lesson 9) and if a package isn't installed then it will allow you to install it. PyCharm also has more extensive debugging (Lesson 6) tools than IDLE

## Lesson 6 – Debugging

The term debugging is attributed to Grace Hopper who was a Rear Admiral in the US Navy and one of the developers of the COBOL programming language. The word 'bug' to represent a problem with a program was already in use. 'debugging' came about when she removed a dead moth from within a mechanical relay which formed part of the computer system she was using. (Back in the 50's!)

More details here - <http://www.anomalies-unlimited.com/Science/Grace%20Hopper.html>

### Types of Errors

There are four basic types of errors

- Syntax
- 'Compile' time
- Run time
- Wrong answers

Each type may have different causes;

Error Type	Caused By
Syntax	Written code doesn't follow the language syntax
'Compile' time	Ambiguous or impossible instructions
Run Time	Incorrect Algorithm operation
Wrong answers	Incorrect Algorithm operation

If you are using an IDE syntax errors are typically highlighted to you in much the same way as a word processor indicates spelling errors.

For an Interpreted language like Python, to the developer there is no real difference between a 'compile' time error and a run time error in the way in which they are reported

One of the most common errors, which can be a compile or run time error is the misspelling of variable names. This is particularly the case in languages like Python which are case sensitive. i.e. myVar and Myvar are different variables.

Debugging is really needed to deal with Run time errors or wrong answers. You may not realise the need to debug for wrong answer errors until they occur during testing (Lesson 8).

Run time errors you need to fix as you go as they can effectively stop development until resolved.

### [Demo Debugger in IDLE]

A tutorial on using the IDLE debugger can be found [here](#)

### [Demo Debugger in Pycharm – showing use of breakpoints]

### What if I don't have any tools

If you do not have access to an IDE, you can create a simple debugging environment for yourself by adding additional code to your program. Typically, you might add;

- Dummy input to stop the program
- print statements for key variables

When you have resolved the problem in one part of the program you could either remove these statements or simply comment them out, in case they are needed again. When you are happy that the program is working OK, you may want to remove the comments (or better, create a copy without the comments) as for interpreted languages such as Python, they will introduce a slight processing overhead.

### How does Debugging differ from Testing?

Debugging is the process of fixing things that are wrong with the program code.

Testing is a planned process which aims to discover if anything is wrong with the program code. There can be many problems with your code which do not show up in the development process. They only become apparent because of methodical testing. Testing will only tell you that something is wrong, it won't necessarily point you to the point in the code where the error is.

Although testing can reveal run time errors, the testing process mainly highlights errors in the logic of the algorithm used in the program.

Testing is covered in more detail in Lesson 8.

## Lesson 7 – More Python

The purpose of this lesson is to simply introduce you to a few more elements of the Python language which will be useful when you write your own complete programs. This is not an attempt to teach you the Python language, but simply to provide a set of snippets of Python code which you can adapt as needed.

### Reading and Writing to a file

In the examples so far, any input required was either ‘hard wired’ into the program code by assigning values to variables in the code or has used the simple Python ‘input()’ function to get input from the user.

In general, this is impractical and in most cases input and output comes from files and is sent to files.

The following examples show files being read from and written to.

```
# read a file and print the length of each line and its contents
infile = 'D:\python_data\Rio_medals_table.csv'
fr = open(infile, 'r')
for line in fr:
    print(str(len(line)) + "<>" + line)

fr.close()

# read a file and write each line to a different file
infile = 'D:\python_data\Rio_medals_table.csv'
outfile = 'D:\python_data\Rio_medals_table_out.csv'
fr = open(infile, 'r')
fw = open(outfile, 'w')

for line in fr:
    # this is where your main processing goes
    # that changes the input to produce the output
    fw.write(line)

fr.close()
fw.close()
```

I

Points to note;

1. The basic function to work with a file is open(). It takes two parameters the first is the name of the file you wish to open and the second indication of how you intend to use the file. ('r' = read, 'w' = write, 'a' = append)
2. If you say you are going to write to a file and it doesn't exist, then it will be created for you.
3. If you say you are going to write to a file and it does exist, then you will overwrite any existing content
4. Append will also create the file if needed, but if the file does exist then anything written is written at the end of the existing contents in the file
5. The open() function return a value of type file. This is often referred to as a file handle.
6. A type of file is iterable. You can think of the file as being a list of items where each item is a single record in the file.

7. Because it is iterable you can use the file as the sequence in a for loop in order to process each line of the file in turn. the loop variable (line in the examples) will automatically contain the next record read from the file.
8. You write records to a file using the write() function of the file handle associated with the file you are writing to. In the example fw.write(line). line in this case is the whole record read but it could be any string value.
9. If you just wanted to read a single record from a file you could use a statement like;  
'line = fr.readline()'
10. When you have finished processing a file you should close the file using the close() procedure of the file handle (e.g fr.close() )

The code is in the file files.py in the code folder.

## Lists

We have already seen a list in use when we looked at the 'for' construct in lesson 4.

The examples below illustrate the use of lists.

```
# Lists

#create an empty list
myList = []

#create list with values
myList2 = [ 5,8,9]

# Items in the list don't all have to be the same type
myList3 = ['Hello', 42, 3.142 ]

# individual elements in the list can be referenced
print myList3[0]

# to iterate over all of the elements in a list
for x in myList3 :
    print x

#to add an element
myList3.append("new item")

# how many items in the list
print len(myList3)

# print last element remembering that the indexing starts at 0
print myList3[len(myList3) - 1]

# or more simply (counting backwards from the end)
print myList3[-1]

#to delete an item
del myList3[2]

for x in myList3 :
    print x
```

Points to note;



1. Lists are enclosed with [] brackets.
2. The entries in the list don't have to be the same type.
3. Individual items can be accessed by using the index number in {} brackets. Remember that indexing starts at 0 not 1
4. To access all of the items, you can iterate over them using a 'for' loop.
5. Items can be added using the append() function and removed using the del function.

The code is in the file lists.py in the code folder

A more complete list of Python list functions can be found here

[http://www.tutorialspoint.com/python/python\\_lists.htm](http://www.tutorialspoint.com/python/python_lists.htm)

## String Functions

There is a whole variety of string functions available in Python. A full list is provided in the official documentation for v2.x here <https://docs.python.org/2/library/string.html#string-functions>.

For our programming tasks today we really only need one and that is called split. The following examples demonstrate the use of the string split function.

```
# string.split()

mystring = "5,GER,17,10,15,42,View medals by sport for Germany"

mylistofitems = mystring.split(",")

print(mylistofitems)

print(mylistofitems[1])

for item in mylistofitems :
    print(item)

for item in mylistofitems :
    print(type(item))
```

Points to note;

1. split is a function which returns a list of strings.
2. The single parameter to split is the character that you want to split the string on. A ',' being quite typical but it could be any single character.
3. You can print all the items as a list, as individual items from the list or
4. A specific entry in the list using an index (remember indexing starts from 0)

The code is in the file split.py in the code folder

## Defining Functions

All programming language allow developers to create their own functions or procedures. These are sections of code which are written separately from the main program code, but possibly in the same file and used by the main program. They are generally a way of avoiding repetition and making the code more tidy and easier to read.

The only real difference between a function and a procedure is that a function returns a value and a procedure does not.

The following examples show the definition and use of some local functions (i.e. in the same code file as the main program)

```
# functions

# this is a procedure because nothing is returned
def myprocedure(printme) :
    print printme

myprocedure("Hello World")

|
def xtimesprint(printme, howoften) :
    returnstring = ""
    for x in range(howoften) :
        returnstring = returnstring + printme
    return returnstring

printme = "Hello World "
printme = xtimesprint(printme, 3)
print printme
```

Points to note;

1. The definition of a function (or procedure) starts with the def keyword and is followed by the name of the function with any parameters used by the function in brackets.
2. The definition clause is terminated with a ':' which causes indentation on the next and subsequent lines. All of these lines form the statements which make up the function. The function ends after the indentation is removed.
3. Within the function, the parameters behave just like variables whose initial values will be those that they were given when the function was called.
4. Functions have a return statement which specifies the value to be returned. This is the value assigned to the variable on the left-hand side of the call to the function. (printme in the example above)
5. You call (run the code) a function simply by providing its name and values for its parameters just as you would for any builtin function.

The code is in the file functions.py in the code folder

### Using parameters in main program

If you want to run your program directly from the command line as we did in lesson 5 and don't want to be prompting the user for input, you can provide values for variables directly on the commandline when you run the program.

You provide the parameters immediately after the program name separating the parameters with a space.

The code below gives an example of how the arguments can be retrieved from the command line call;

```
#command line arguments
import sys

print "first parameter is ", sys.argv[1]
print "second parameter is ", sys.argv[2]
print "the program name is ", sys.argv[0]

print len(sys.argv)
for arg in sys.argv:
    print arg
```

It can be called from the command line like this;

```
>commandlineargs.py "abc" "def"
```

Points to note;

1. In the call the parameters are separated from the program name and each other by spaces.
2. The arguments are returned a list of strings.
3. The program name is counted as an argument (sys.argv[0])
4. In order to access the arguments from within your program, you need to import the 'sys' package.

The code is in the file commandlineargs.py in the code folder.

## Python on-line documentation

The python.org website has a comprehensive set of documents describing all aspects of the Python language, both the current and previous releases. Note that there are two 'current' versions at any one time; one is the v3.x stream and the other is the v2.7.x stream. The languages are slightly different, although in the code that we are writing the only noticeable difference will be in the print statement as it is in v2.7.x and the print function as it is v3.x stream. There is also a difference in how v2.7 handles input strings. The function raw\_input() should be used in preference to input().

The formal documentation can be a bit heavy going to anyone new to coding. If you have a simple problem like 'how does the str function work in python', you might be better off putting it in to Google and checking out the responses. Organisations like StackOverflow are particularly good at providing answers with examples. You will also find many other forums this way which will allow you to put questions to more experienced Python users.

## Lesson 8 – Introduction to Testing

What is the difference between debugging and Testing?

In Lesson 6 we said that; Testing is a planned process which aims to discover if anything is wrong with the program code. We also said that the most likely problem was that the programming logic for the algorithm was incorrect. Debugging on the other hand is the process of detecting and correction the errors in the programming logic.

Perhaps an indication of how important software testing is considered to be, is the fact that it is covered by a set of ISO (International Standards Organisation) Standards (ISO 20119-1 to 5).

All computer programs, whether they appear to be working or not should be comprehensively tested. The actual extent of the testing will depend on the size complexity and criticality of the program.

The answer to the question; When have we done enough testing? is often quoted as; *‘When you have either run out of money or run out of time’*. This may seem like a rather flippant response, but what is significant about it is that it makes no attempt to suggest that it is when you have found (and dealt with) all of the Bugs. There is simply no way of knowing if this has been done or not.

### Testing approach

There are three key points;

- Have a plan! – start it early!
- Know what you are testing for
- Know what results you expect from any given set of test conditions

### Constructing a plan

You can start constructing a test plan as soon as you have the program specification. In our case we will assume the description of the program will serve as a specification, although in reality a program specification is a more rigorous document than a simple description.

There are two aspects of your program that you are looking to test as thoroughly as possible.

The first is to ensure that you program will behave as expected with all possible inputs and values for the variables used in the program. The testing of inputs is particularly important, as you may not be in complete control of this when the program is run. It is very difficult to test all combinations of internal variables so we will focus on the inputs.

The second is to ensure that you have sufficient test cases to exercise every path or part of a path through the program code. In terms of a flowchart this means that every branch caused by a decision needs to be tested. If you cannot construct a test that will follow a path from the start to the end, then you should have a series of tests to cover the different sections of the branches.

### Testing

We have said that we will limit our testing to the input variables, so you need;

- a list of all of the variables in the program so that you can identify those which represent input.
- an understanding of what they are being used for
- to know the range of values that could take. There may be clues in the program specification, there may not.

Variable type	Expected Range	Possible test values
<b>Integer</b>	1 to 100	-1,0,1,50,100,101 any invalid type like a string or a floating point number or an empty value
<b>Integer</b>	Unknown	big -ve, -1,0,1, large +ve any invalid type like a string or a floating point number or an empty value
<b>Float</b>	32.0 to 212.0	-10.0, 31.9999,32, 32.0001, 100.0000000, 211.5, 213.5 any invalid type like a string or an empty value
<b>Float</b>	Unknown	big -ve, 0, 0.000, range of +ve and -ve values with differing degrees of precision any invalid type like a string or an empty value
<b>String</b>	From known list	1 <sup>st</sup> and last in list, random selection from list, random selection not on list with differing lengths, empty string, Upper and lower case
<b>String</b>	For given alphabet or character set	Random selection of different lengths only from the character set. Selection containing characters not in the set, empty string, upper and lower case (where applicable)
	Y/N	Y, y ,Yes, yes, N, n, No, no, YesNo, yesno, empty string

Even for a program with only a single integer, floating point number and a string input. you can see that there could be a good number of test cases to produce.

In the case of the Boolean values, anything starting with Y or y might be acceptable and similarly N or n. Be aware of programs that are coded to expect a specific length of an input string.

For each of the relevant cases above you should draw up a test sheet indicating what you are testing. I.e. what variable or set of variables are being tested, what values you are testing with (in this particular test) and **most importantly what you expect the results to be**. From your understanding of what the program does. you should know in advance what results are expected

## Recording Results

Your test plan and test results should always be recorded. The plan can be used to demonstrate that the program has been thoroughly tested.

If as the result of finding and fixing bugs your program code has been changed, then you will need to re-test. So having the tests and results of the test runs to hand can be invaluable.

## VVT

VVT stands for Verification, Validation and Testing. Testing we have already described as checking if anything wrong with the program code.

There are however another couple of things we need to consider:

Have we written the right program? Does it match the specification we were given? – This is the process of Validation

Have we written the program right? Does the programming logic we have used result in getting the right answers - This is the process of Verification.

A program could pass any of the tests relating specifically to the code but still fail either Verification or Validation.

## Lesson 9 – Re-using code and packages

### Introduction to pip

Python is an extensible language. In addition to the basic functionality provided in the standard installation there is a wealth of additional ‘packages’ which can be added to the installation.

A small command line program called pip is installed as part of the Python installation process. This is the program which can be used to find out what packages have already been installed, find out what packages are available and to install the packages of your choice.

From a command line prompt you should be able to call pip directly. The table below shows the common usages of pip.

Command	What it does
>pip list	Lists all of the packages currently in your Python installation
>pip search *	Provides a list of all available packages (requires Internet connection)
>pip install <package name>	Installs the given package

Once a package has been installed, you can use it in your program by using the import statement. An example of doing this is given below

### import

This example shows the use of using the import command. This makes all of the functions defined within the package available to your program.

```
import math  
  
print(math.sqrt(4))
```

If you need to know what functions are available, you can use the dir() command. If you want to know what a specific function does you can use the help() command.

```
import math  
  
dir(math)  
help(math.sin)
```

## Lesson 10 – Introduction to Small Programs

This lesson is all about putting into practice what you have learnt.

These notes are being given to you now, so that as the day progresses and we cover the various bits of output you are being asked for, you will have the opportunity to start thinking about and planning your program.

### Program details

You will be provided with a file which contains a table representing the final medals table from the 2016 Rio Olympic games. The table is in final position order and includes the number of Gold, Silver and Bronze medals won by each team.

Pos	Country	Gold	Silver	Bronze	Total	Total2
1	US	46	37	38	121	View medals by sport for United States
2	GB	27	23	17	67	View medals by sport for Great Britain & N. Ireland
3	CHN	26	18	26	70	View medals by sport for China

The task is to write a program in Python which will accept as input two Country codes (as specified in the Country column) and should output the number of additional medals the second country would have needed to beat the first country. All combinations of additional medals should be found.

### You should end up producing:

- A program name
- A program specification / description
- Pseudo-code and / or flowcharts for your algorithm
- A code file containing the code of the program
  - Including a suitable block comment at the beginning
  - Any appropriate in-line comments
- A test plan
  - an example of expected output
  - with test data
  - the test results
- An example run of the program
  - Indicating the input used
  - and the output produced.