
Introduction to Programming

Workshop

15 December 2016

Peter Smyth
UK Data Service

UK Data Service



Accessing the course materials

The code snippets used, the file needed for the final exercise, the additional exercises and a copy of the notes are all available on Github at;

<https://github.com/pfsmyth/Cambridge>

Download the zip file and extract it to your laptop, somewhere where you can access it easily



Overview of this workshop

This is what we will cover:

1. What is a computer program?
2. Defining the problem in words and pictures
3. Documentation
4. Python programming constructs
5. Coding environments
6. Debugging
7. More Python
8. Introduction to testing
9. Re-using code and packages
10. Introduction to small programs



Computer programming

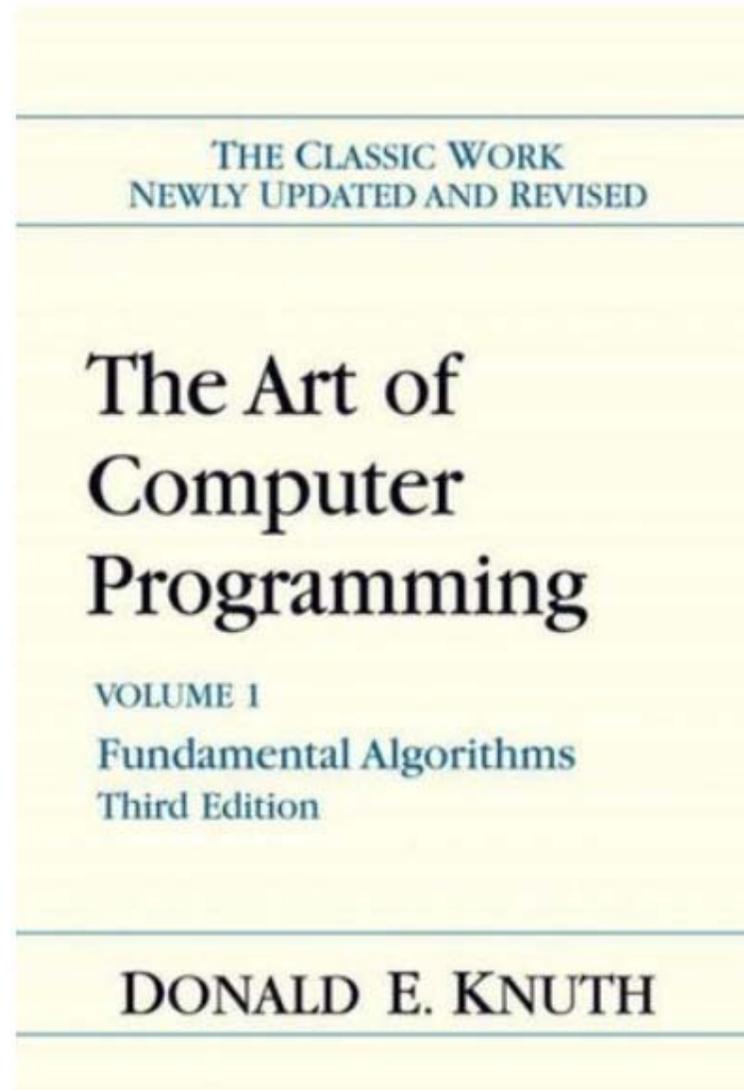
Art ?

or

Science?



Computer programming



Lesson 1

What is a computer Program?



Lesson 1

Definitions

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)



Lesson 1

In pictures



- **Question** – Give examples of input and output

Lesson 1

A simple program

```
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.



Lesson 1

- It is generally easier to read programs than to write them
- Regardless of the language
- It is usually necessary to add symbols (or tokens) to the written code in order to make it understandable by the computer.
- This can help to ensure that the code is not ambiguous



Lesson 1 – stupid computers

Computers can;

- Add
- Subtract
- Compare

And that's about it!



Lesson 1 – How Programs are run

In very simple terms –

- You write your program code to a file as simple text
- A compiler program is run which takes your file of code as input and produces a file of machine code as output
- The machine code is written to a file. It just looks like a load of numbers to you and me.
- Compilers are language specific, in two ways
 - They only understand code written in their language
 - They produce machine code that the computer it is running on understands
- The Operating System takes the machine code file as input executes the instructions and produces output depending on what your program code said (another program)



Lesson 1 – How Programs are run

Not all language use a compiler –

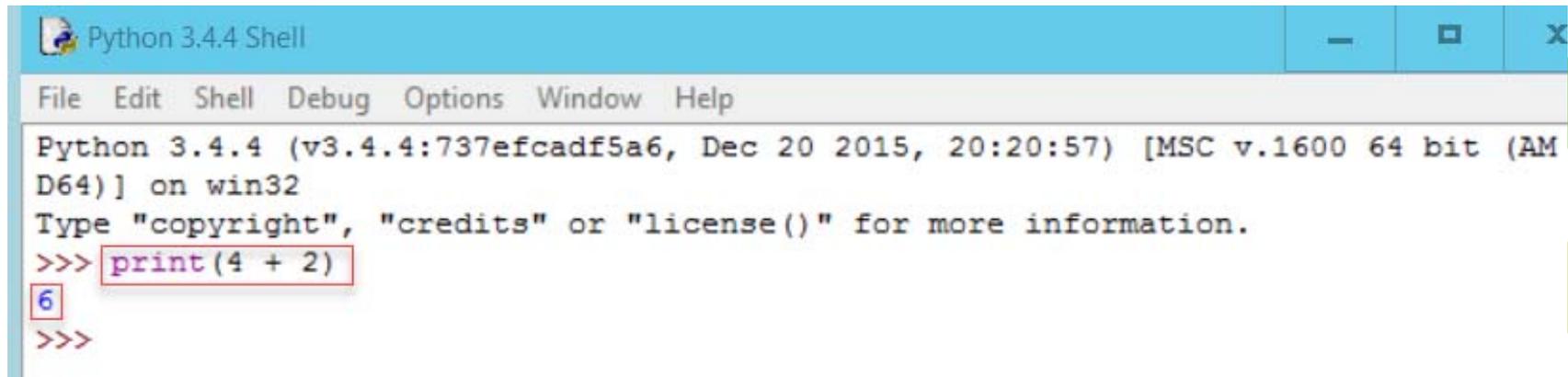
- Python uses an Interpreter.
- This combines the compile and execution steps together
- The interpreter takes you file of program code and produces output immediately
- The use of Interpreter, allows the creation of REPL systems
- REPL stands for Read, Execute, Print, Loop



Lesson 1 – How Programs are run

REPL –

- Python comes with a REPL environment called IDLE



```
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 first highlighted area is a Python statement
- The second is the output produced from executing the statement



Lesson 2

Using words and pictures



Lesson 2

Everyday tasks are often performed automatically. Writing instructions for others to follow can be more difficult.

Exercise – write instructions to do one of the following;

- 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.



Comments on the exercise

- It is natural to write the instructions as a list of things to do
- Although we recognise the tasks as singular, we naturally break them down into sub-tasks
- Linear lists do not make it easy to include decisions or choices.
- Typically, assumptions will be made. e.g.
 - gas or electric?
 - How long do you boil an egg?



Lesson 2 - Before you start coding

- Even before deciding on a programming language
- There are two techniques which can be used to help in the design of your program

- Pseudocode and
- Flowcharting

- You can use both, you should certainly use one.



Lesson 2 - Pseudo code

In general pseudo code consists of simple English phrases on separate lines.

There are however two key rules;

1. If a line is dependent on the line before, you indent the line.
2. Use a set of commonly accepted constructs for selections and actions.



Lesson 2 - Pseudo code

- A simple example

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

- The second and fourth lines are indented because they depend on the lines above
- The If...Else...EndIf construct is used to indicate a selection (do this or do that)



Lesson 2 - Pseudo code

- The commonly used constructs

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



Lesson 2 - Pseudo code 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
```



Lesson 2 - Flowcharts

- A flowchart is a diagrammatic way of representing your algorithm or program
- It is a graph with directed lines representing the edges
- And a series of symbols representing the nodes
- The Symbol used in some way represents the purpose of the node



Lesson 2 - Flowcharts

Some of the key symbols are shown here

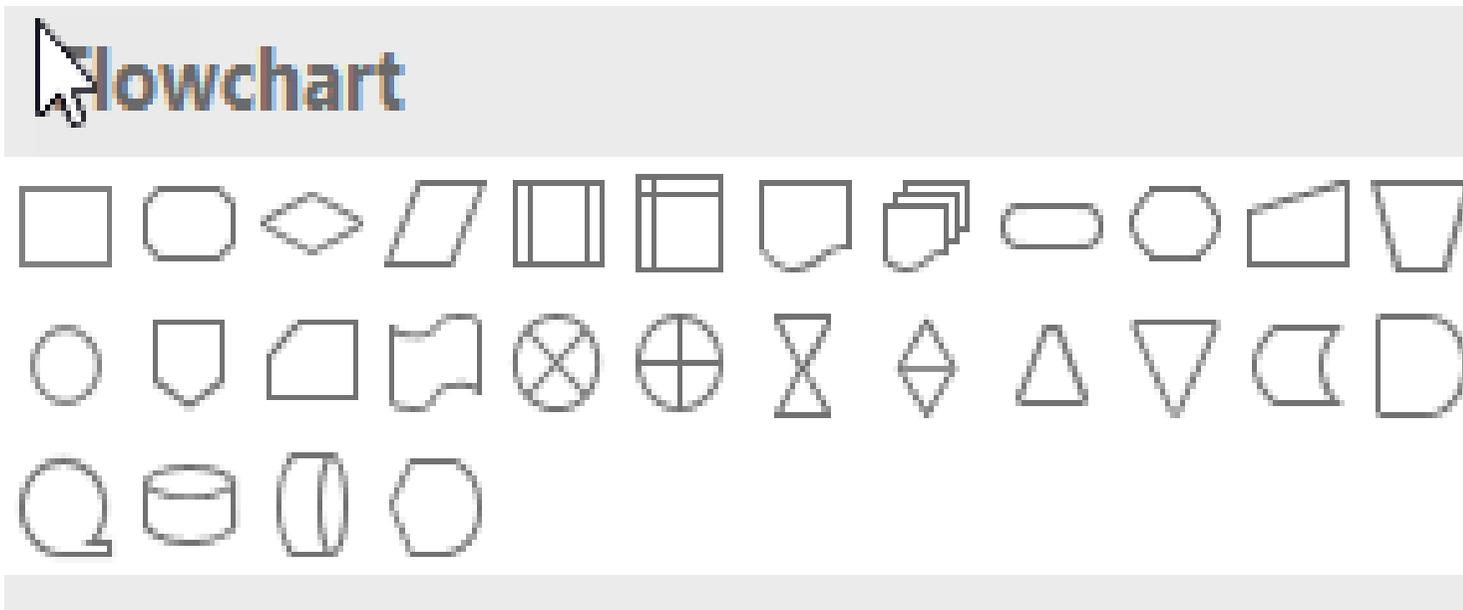
Flowchart Symbols

	Terminator (Start and Finish)
	Process
	<u>Subprocess</u>
	Decision
	Off page connector
	Manual Input



Lesson 2 - Flowcharts

- The flowcharting symbols are available from MS Word and other packages



- In practice it is normally more convenient to start your flowchart freehand (pen and paper!)

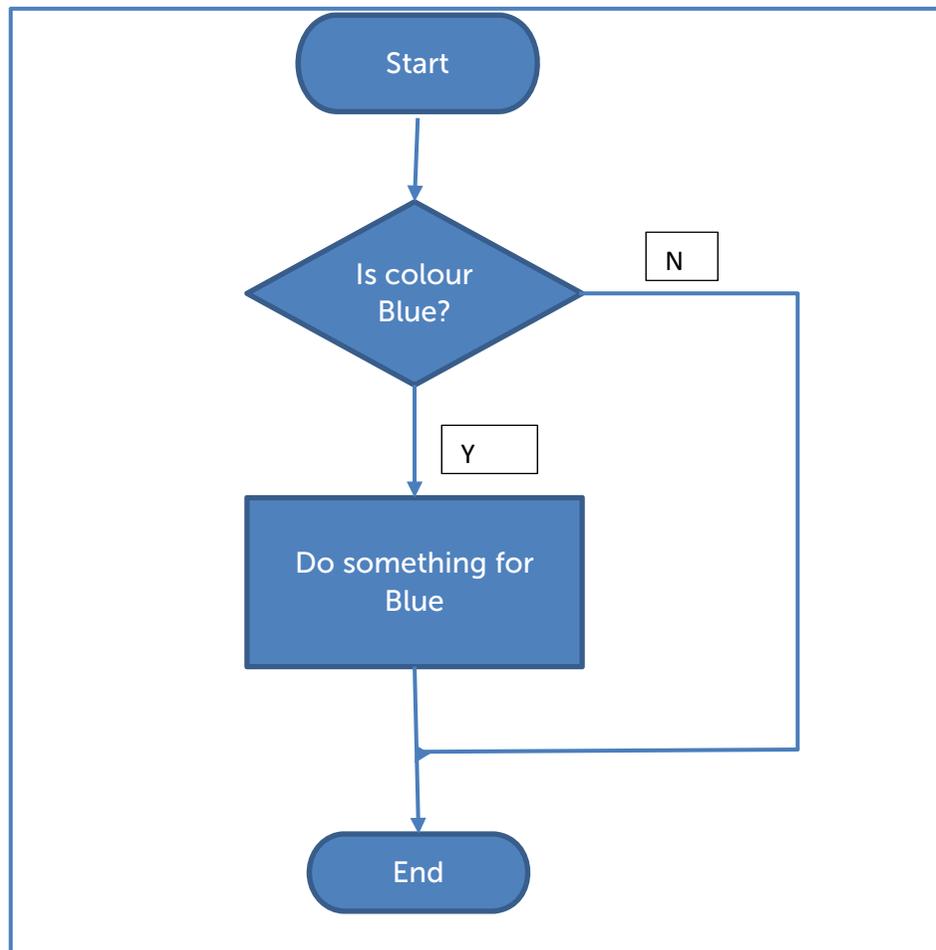


Lesson 2 - Flowcharts

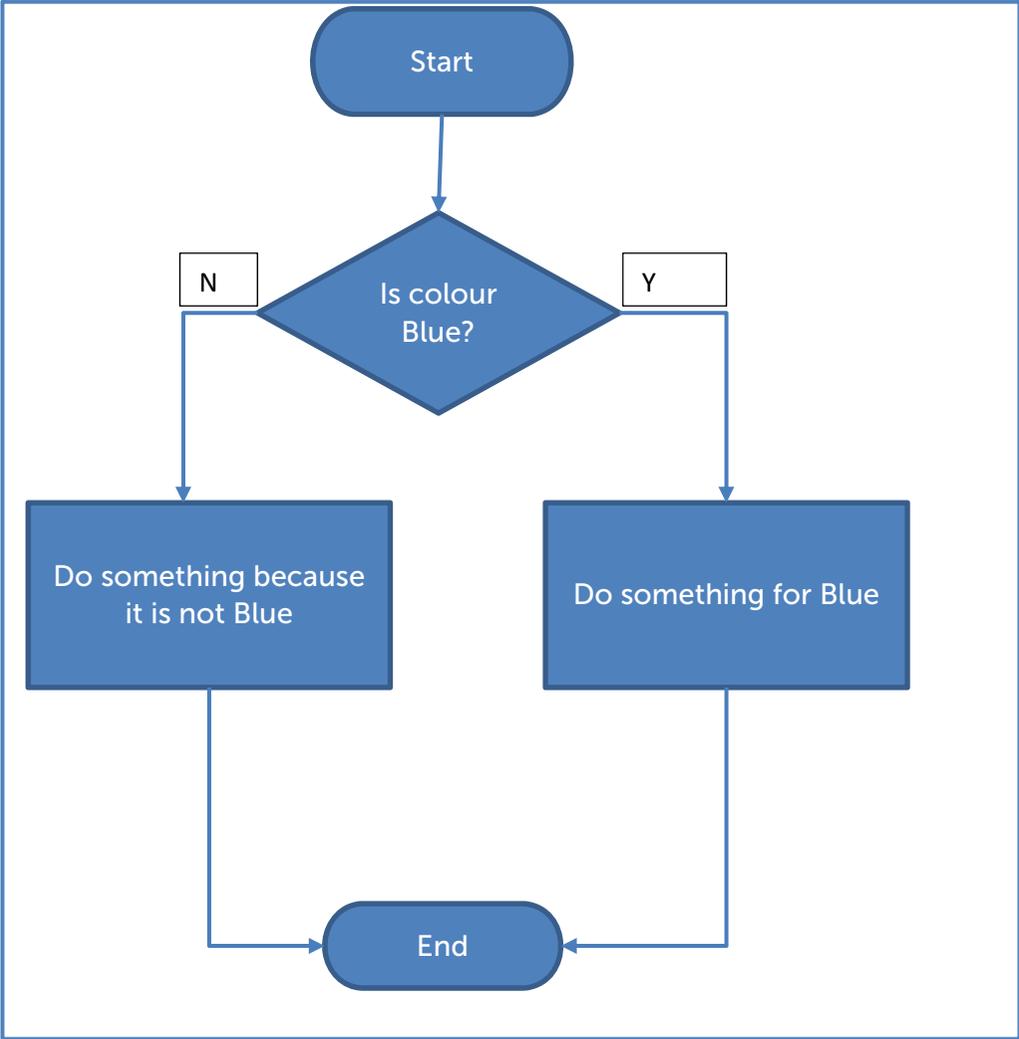
- The following slides illustrate how several of the selections and actions referred to in the pseudo code section are realised as flowchart segments.
- Remember however that a complete flowchart will consist of many of these segments together so as to represent the complete task.



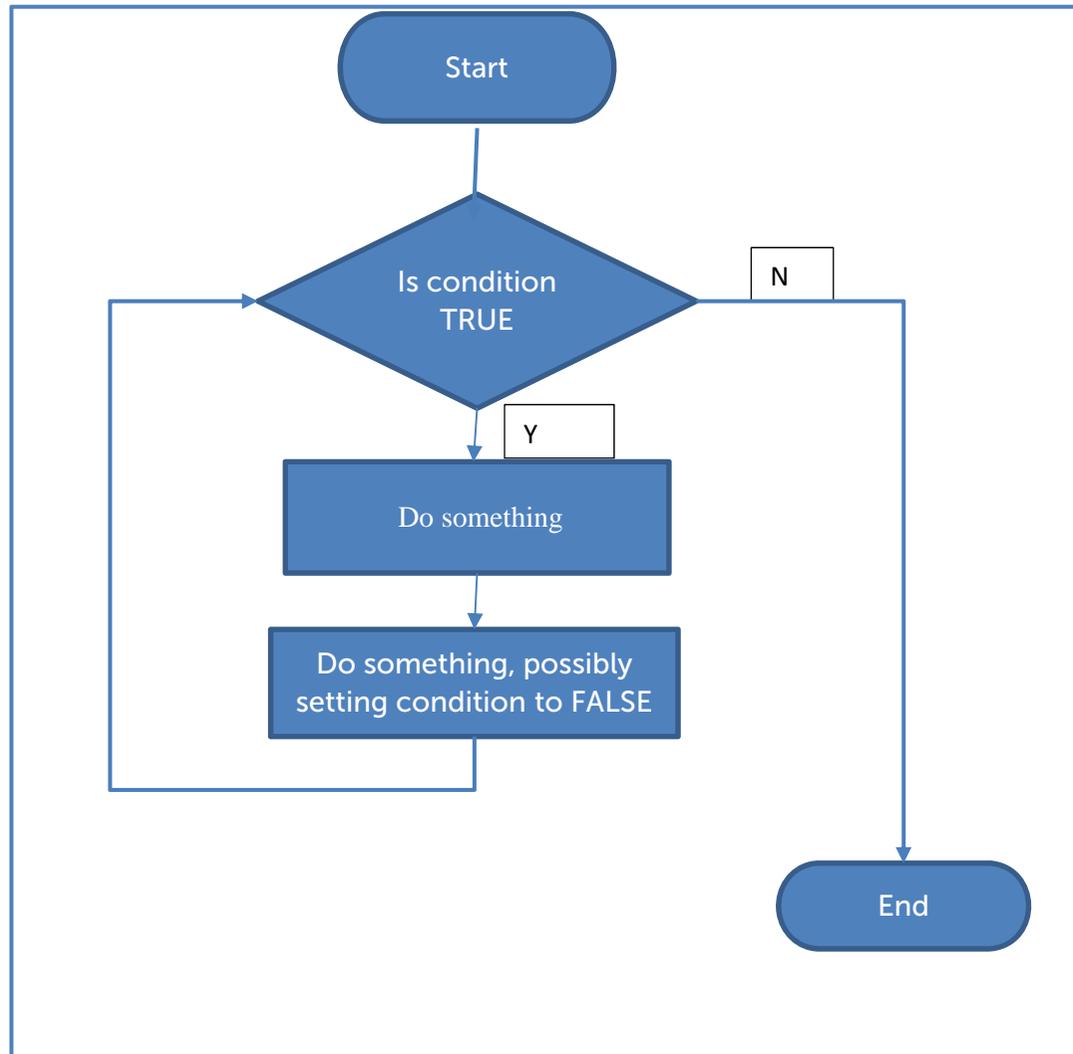
Lesson 2 – Flowcharts If...Endif



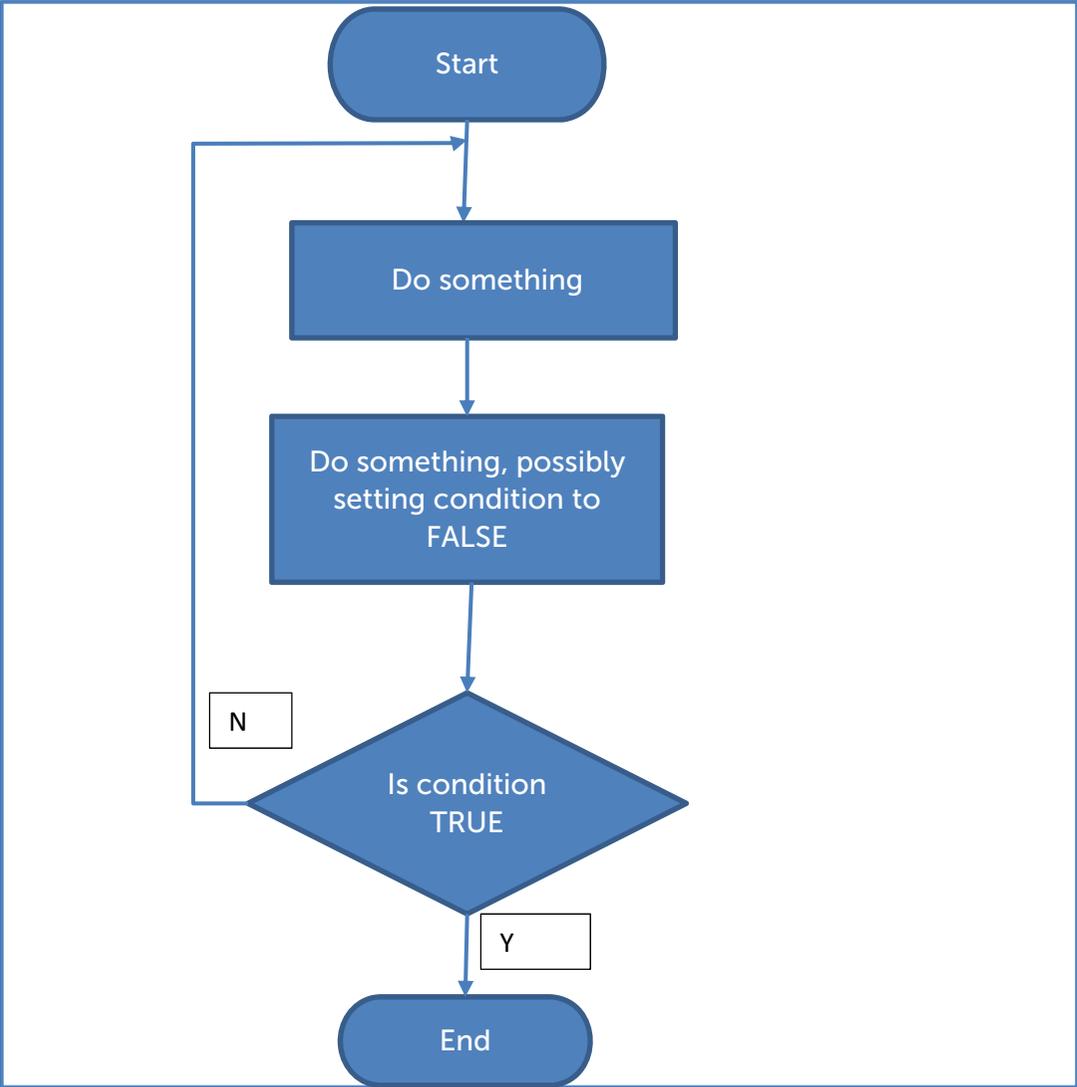
Lesson 2 – Flowcharts If...Else...Endif



Lesson 2 – Flowcharts Do While...EndDo

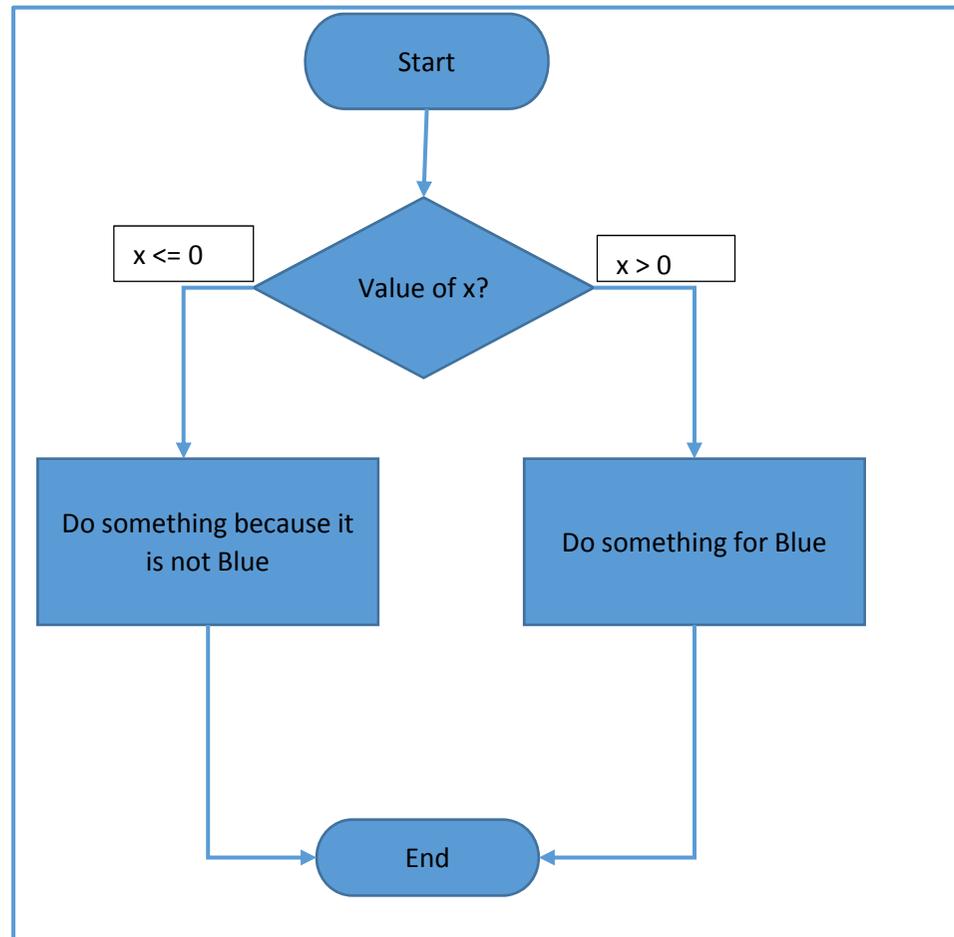


Lesson 2 – Flowcharts Do Until...EndDo



Lesson 2 - Flowcharts

- The label from the condition don't have to be Y and N



Lesson 2 - Exercise 1

- Convert your English task into pseudo-code and a flowchart
- Don't forget:
 - Gas or Electric?
 - How do you like your eggs boiled?
 - Do you take sugar?



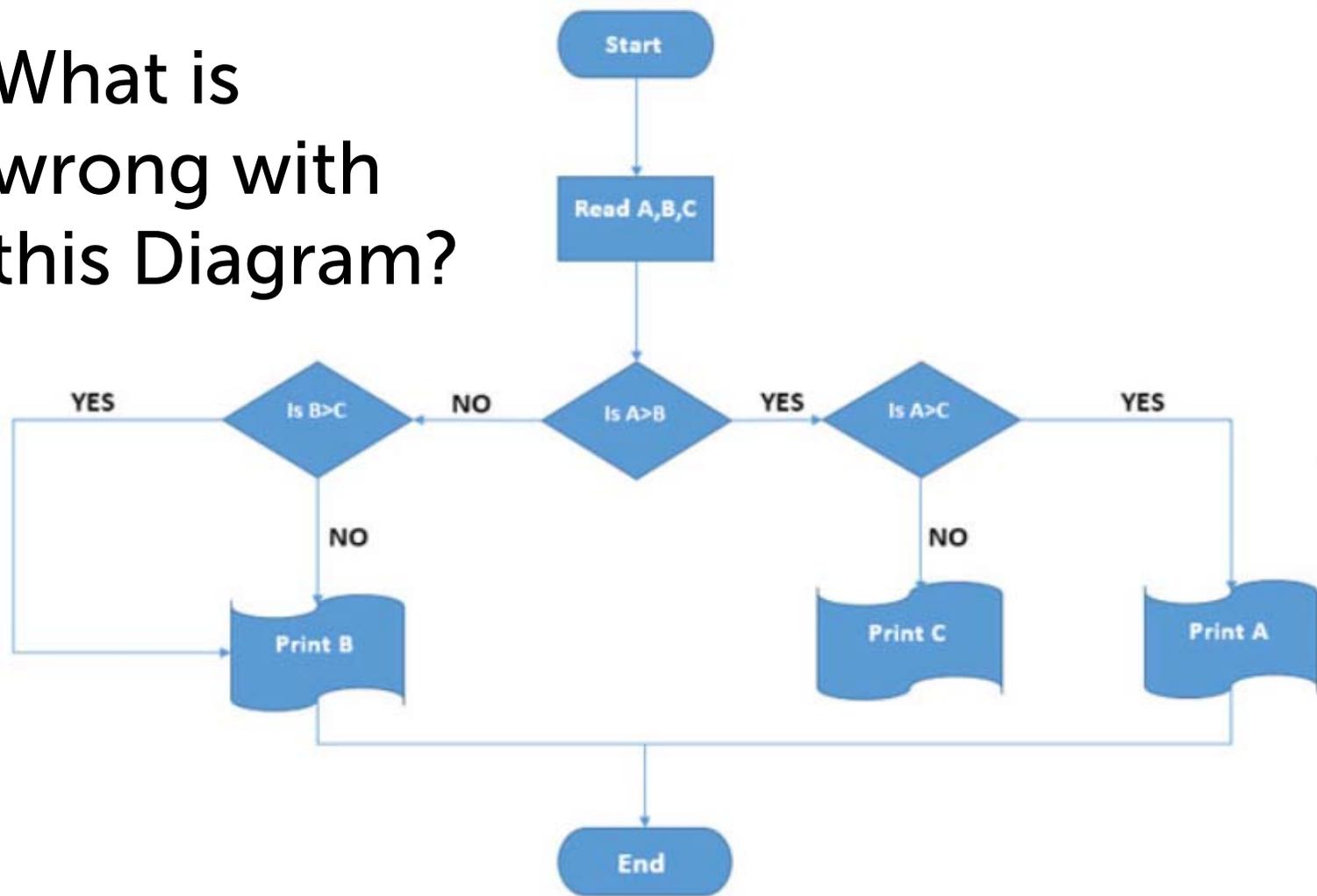
Lesson 2 - Exercise 2

- The diagram on the next slide is meant to represent a flowchart to decide the largest of 3 (not equal numbers)
- Find 2 things wrong with the flowchart and correct them



Lesson 2 - Exercise 1

What is wrong with this Diagram?



Lesson 3

Documentation



Lesson 3

- All computer programs should be documented
- There are two types of documentation
 - External documentation and
 - Internal documentation



Lesson 3 - External documentation

- Examples
 - Program Title
 - Program specification or description
 - What the program does
 - How it is to be used
 - Indication of inputs and outputs
 - Pseudo-code / Flow charts



Lesson 3 – Internal documentation

- All programming languages allow the programmer to include comments in the file of program code.
- Different languages have different ways of allowing the programmer to write comments. In Python, there are two different ways
- You can use whatever is the more appropriate
- All programs should have a block comment at the beginning of the file
- In line comments should be used as required but sparingly – you don't need to comment the obvious



Lesson 3 – Internal documentation

Example

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

"""



Lesson 3 – Internal documentation

Block comment at the beginning of the file

```
"""  
Program Name :  
Author       :  
Date written :  
Description  :  
  
Inputs       :  
Outputs      :  
  
Calls        :  
Is called by :  
"""
```



Lesson 4

Python programming constructs



Lesson 4 – About Python

- Python is a very popular general purpose language which is used extensively for data manipulation and data analysis.
- The official Python website is at www.python.org
- From here you can download different versions of Python
- Complete documentation is also available from the site
- If stuck, you can google; ‘how do I ... in python’
 - There will be plenty of responses
 - Answers from ‘stackoverflow’
 - Links to official documentation etc.

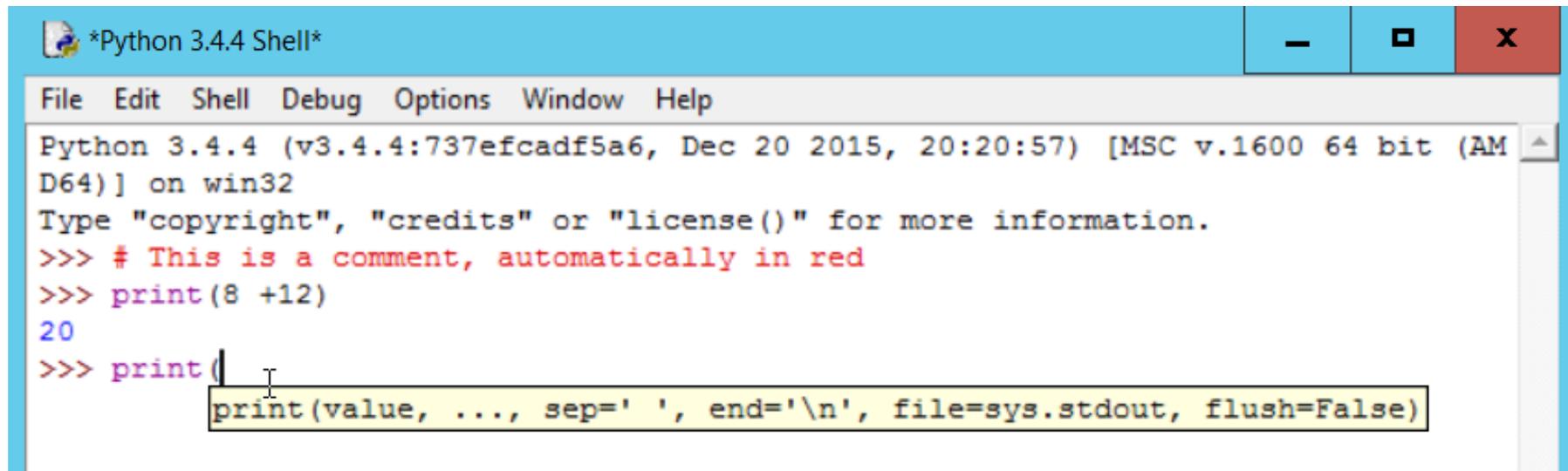


Lesson 4 – Running Python code

- Python code is just text, you can use any text editor
 - You then need to explicitly run the code
 - This is rarely done
 - And almost never during development
- When you install Python you will get a small IDE (Interactive Development Environment)
- This includes the REPL environment we saw in Lesson 1
- This is what we will be using for this workshop
- There are most sophisticated environments such as;
 - PyCharm
 - Jupyter notebooks



Lesson 4 - IDLE



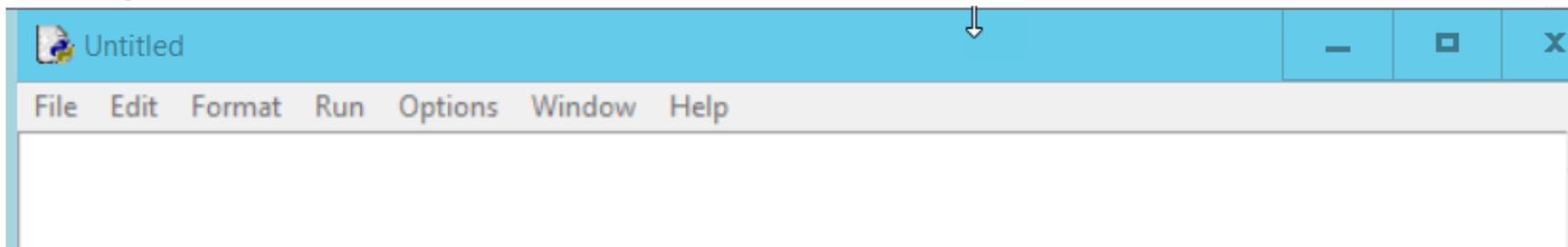
```
*Python 3.4.4 Shell*
File Edit Shell Debug Options Window Help
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 20:20:57) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> # This is a comment, automatically in red
>>> print(8 +12)
20
>>> print(
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

- Notice the colour coding for the different elements
- When you type the open bracket for a function, IDLE automatically provides some help
- IDLE has many other useful features.



Lesson 4 – more IDLE

- You can use IDLE to create a file of code and then run the code as a complete program
 - This is what you most commonly do when developing
- You create a new file (not a new REPL window) by selecting File | New File (or ctrl+N) from the REPL window
- This opens a new ‘notepad’ like window with extra options in the menubar



Lesson 4 – more IDLE

- You can also open an existing file of Python code using File | Open
- Many of the examples for the coding constructs we wish to look at have been pre-written for you in the files you downloaded from GitHub.



Lesson 4

Before we start on the constructs, a little bit about **types, variables and operators** in Python.



Lesson 4 – simpletypes.py

Types

Int => -2, 0, 1, 200001

Float => -1.0, 0.001, 7689.3452

String => "Hello", "", 'either quotes are OK'

Boolean => True or False

Print(type(variable)) # print the type of the variable

Print(type(variable)) is int # test to see if variable is

of type int



Lesson 4 – simpletypes.py

```
# 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")
```



Lesson 4 – simpletypes.py

- We will only be using integers (a), floating point numbers (b) and strings (c).
- The `type()` function will tell you what type a variable is
- If you are checking the type in code you use the `'is'` operator not the `'=='` operator
- 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



Lesson 4

Variables

`x = 1`

`x = 3.4`

`x = "Some string value"`

- Python will change the type of `x` depending on how you use it
- Give variables reasonable name (not like above)



Lesson 4

Arithmetic operators

Usual suspects : + - / *

Also

** (raise to the power) and

% (mod)

Comparison operators

<, >, <=, >=, ==, !=



Lesson 4 – print.py

```
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)
```



Lesson 4 – input.py

```
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))
```



Lesson 4 - Input

- Input values are stored as strings. 42 becomes '42'
- If you want a numeric value, the programmer has to convert it using the `int()` or `float()` functions
- Can be a problem if you are given "Three" and not "3"
- Use the 'try...except' construct to check



Lesson 4 – tryExcept.py

```
# 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")
```

I



Lesson 4 – if.py



```
# if .. End If

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



Lesson 4 – if.py

- The colon ':' at the end of the 'if' line. Missing this out is a common error.
- 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
- The equivalent of the 'EndIf' is removing the indent.



Lesson 4 – ifelse.py

```
# 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)
```



Lesson 4 – ifelse.py

- 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.



Lesson 4 – ifelif.py

```
# 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)
```



Lesson 4 – ifelif.py

- Each 'Elif' clause has its own test expression.
- You can have as many 'Elif' clause as you need
- 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.
- Notice that in Python the operator used to check equality is '=='



Lesson 4 – for.py

```
# 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)
```



Lesson 4 – for.py

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

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

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



Lesson 4 – for.py

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’.



Lesson 4 – for.py

- The ‘:’ at the end of the ‘for’ statement. This is required and if you are using IDLE will automatically indent the next line.
- ‘<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.
- The list in the 3rd example has a mixture of data types. This is more a function of lists than the ‘for’ loop



Lesson 4 – for.py

- 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]`.
- 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.
- In the last example, the 3rd parameter is a step value, so in this case only every second value in the sequence will be used.



Lesson 4 – while.py

```
# 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)
```



Lesson 4 – while.py

- 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**.
- The clause can be made more complex by use of parentheses and and and or operators
- The statements after the while clause are only executed if the condition evaluates as **True**.
- Within the statements after the while clause there should be something which potentially will make the condition evaluate as **False** next time around.



Lesson 4 – dountil.py

- 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



Lesson 4 – dountil.py

```
# do ... until

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

There is an error in the code above – What is it?



Lesson 4 – dountil.py

- 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'.
- The if clause is used to check some other condition and if (when) it becomes True, the Break statement is executed.



Lesson 4 – dountil.py

- The break statement has the effect of exiting the complete while loop.
- 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.
- 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



Lesson 5 - Coding environments

- This lesson is just a summary a some of the different types of coding environments you might come across



Lesson 5 - Notepad

- Advantages
 - Always available
- Disadvantages
 - Provides no help whatsoever
 - The code file has to be run separately after it has been created and saved



Lesson 5 – Notepad++

- Notepad++ (and others like Atom etc.) is a more sophisticated text editor. It can be told that you are writing Python code
- Advantages
 - You get colour coding of the Python syntax
 - You get help with function names and autocomplete
- Disadvantages
 - No debugging facilities



Lesson 5 – IDLE

- Advantages
 - Gets installed automatically with Python
 - Provides Syntax colour coding
 - Autocomplete of function names
 - Can run code directly from the REPL or save the code and run immediately
 - Has debugging facilities
- Disadvantages
 - Not as sophisticated as some other freely available IDEs



Lesson 5 – Jupyter Notebooks

- Jupyter Notebooks is a web browser based system
- Advantages
 - Syntax colour coding
 - Debugging
 - Integrated Documenting system
 - Can store code and the results of running the code across sessions
- Disadvantages
 - No built-in help with syntax



Lesson 5 – PyCharm

Pycharm is a more substantial IDE than IDLE

Advantages

- Same as IDLE

- Can run Jupyter notebooks as well

- Can manage the installation of packages

- More Extensive debugging facilities

Disadvantages

- More complex to learn and use



Lesson 6

Debugging



Lesson 6



<http://www.anomalies-unlimited.com/Science/Grace%20Hooper.html>



Lesson 6

92

9/9

0800 Antman started
 1000 " stopped - antman ✓
 13°C 103W MP-MC ~~1.130476415~~ ~~2.130476415~~ } 1.2700 9.037847025
 033 PRO-2 2.130476415 } 9.037846995 correct
 correct 2.130676415 } 4.615925059 (-2)

Relays 6-2 in 033 failed speed speed test
 in relay 11.000 test

Relays changed

1100 Started Cosine Tapc (Sine check)
 1525 Started Multi-Address Test.

1545  Relay #70 Panel F
 (moth) in relay.

First actual case of bug being found.

~~1630~~ Antman started
 1700 closed down.



Lesson 6 - Type of Errors



There are four basic types of errors



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



Lesson 6 - Causes of Errors



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



Lesson 6 Causes of Errors

- Common causes of errors
- Misspelling of variable names
 - Python is case sensitive
 - The name Myvar is not the same as myvar
- Incorrect logic
- Not anticipating what can go wrong
 - Trying to do arithmetic on string values



Lesson 6

What does an error look like?

```
>>> num1 = 23
>>> num2 = 0
>>> num3 = num1 / num2
```

```
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    num3 = num1 / num2
ZeroDivisionError: integer division or modulo by zero
>>>
```

You may get many line of cryptic messages. Generally it is only the last one which is of interest to you – hopefully holding a clue to the problem



Lesson 7

More Python



Lesson 7 - Lists

- A list is a group or sequence of items of any type.
- We used lists when we were looking at the 'for' loop construct



Lesson 7 – lists.py



```
#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 ]
```



Lesson 7 – lists.py

```
# 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")
```



Lesson 7 – lists.py

```
# 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]
```



Lesson 7 – lists.py

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



Lesson 7 – files.py

```
# 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()
```



Lesson 7 – Reading and Writing files

- 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)
- If you say you are going to write to a file and it doesn't exist, then it will be created for you.
- If you say you are going to write to a file and it does exist, then you will overwrite any existing content



Lesson 7 – Reading and Writing files

- 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
- The `open()` function returns a value of type `file`. This is often referred to as a file handle.
- 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



Lesson 7 – Reading and Writing files

- 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.
- 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.



Lesson 7 – Reading and Writing files

- If you just wanted to read a single record from a file you could use a statement like;
 ‘line = fr.readline()’
- When you have finished processing a file you should close the file using the close() procedure of the file handle (e.g fr.close())



Lesson 7 - String functions

- There is a whole variety of string functions available in Python. A full list is provided in the official documentation for v3.x here <https://docs.python.org/3/library/string.html#string-functions>.
- We have already seen the len() function when we looked at lists
- We will now look at the very important 'split' function
- Unlike the len() function where the string is passed as a parameter, the split function is built-in to every string
- You use it by writing something like;
`mysplitupstring = mystring.split(",")`



Lesson 7 - split.py

```
# string.split()

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

mylistofitems = mystring.split(",")
for item in mylistofitems :
    print item
```

- split is a function which returns a list of strings.
- The single parameter to split is the character that you want to split the string on. A ',' being quite typical, for a csv file format, but it could be any single character.



Lesson 7 - Functions & Procedures

- The difference between a function and a procedure is that a function returns a value and a procedure does not.
- Print() is a procedure because it return a value, it just does something
- Len() is a function because it returns the length of a string, which you will typically assign to a variable



Lesson 7 - functions.py

```
# functions
```

```
myString = "Hello World"
```

```
print(myString)
```

```
x = len(myString)
```

```
print(x)
```



Lesson 7 - Writing your own 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)
```



Lesson 7 - Writing your own Functions

- 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.
- 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.
- Within the function, the parameters behave just like variables whose initial values will those that they were given when the function was called.



Lesson 7 - Writing your own Functions

- 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)
- 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.



Lesson 8

Introduction to Testing



Lesson 8 - Testing

“If your program works first time, it probably wasn’t worth writing”

(anon)



Lesson 8 - 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



Lesson 8 - Testing

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).



Lesson 8 – Testing

- All programs should be tested. Whether they appear to be working or not
- How much depends on size and complexity
- How do we know when we have done enough testing?



Lesson 8 - Testing

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’



Lesson 8 - Testing

Testing approach

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



Lesson 8 - Testing

- Start test plan construction as soon as possible
 - Some tests can be designed before the code is written
- You should aim to test for all reasonable values or ranges of input variables
 - So you need to know what they are,
 - Their types
 - Their valid ranges
- And combinations of these



Lesson 8 – Testing - Example tests

- For numerical variables test for
 - Either side of the min and max points
 - A central point
 - Real where integer expected
 - -ve if only +ve expected (and vice versa)
- For string variables test for
 - Empty string
 - Overlength strings
 - Strings on in pre-defined list
 - Upper and lower case combinations



Lesson 8 – Testing - Example tests

- For boolean variables test for
 - All combinations of True and False
 - 0 and 1 (make sure you know which is which)
 - 0 and -1 (make sure you know which is which)
 - All combinations of Y, y, N, n, Yes, yes, No, no etc.
 - Empty string



Lesson 8

In addition to testing variables individually, you need to test them in combination with others

e.g. if $0 \leq x < 20$, then $0 \leq y < 8000$

It is the testing of the variables in combination that allows you to test the various branches in the code



Lesson 8 – The Test plan

- Test plans and Test results should always be recorded
- Use a test sheet to record:
 - What the test is testing
 - The values that the variables are to be given
 - Most importantly – The expected results
 - The actual results

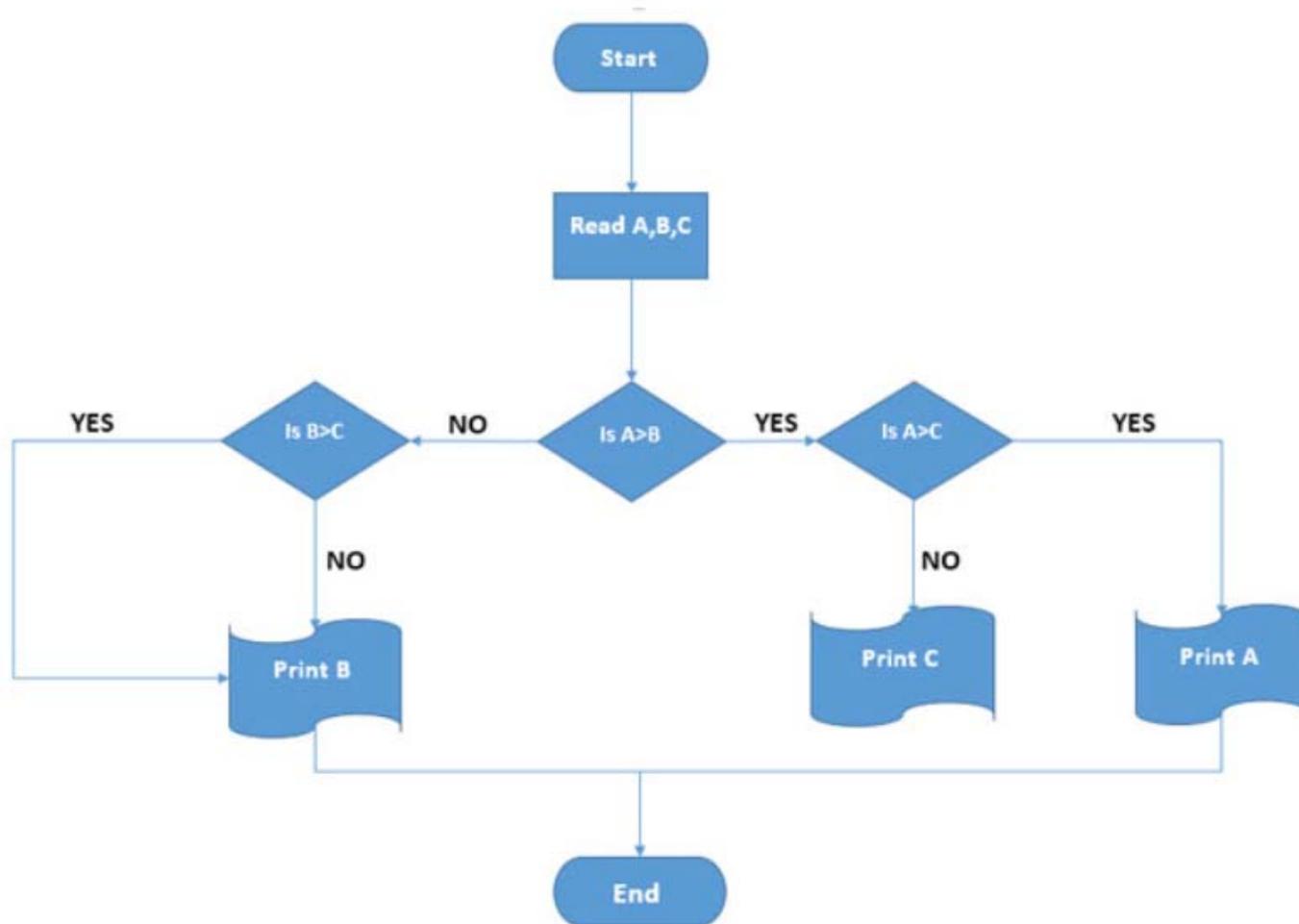


Lesson 8 – Testing example

- As an Example we will re-consider our flawed flowchart from lesson 2
- The program it represents is intended to find the largest of three (all non-equal integers) variables A, B and C
- As the variables are arbitrarily named and each represents an integer we can use values of 1, 2 and 3 and assign them in all combinations to A, B, and C
- This will result in a set of 6 tests shown on the next but one slide



Lesson 8 – Testing example



Lesson 8 – Testing example

Test No.	1	2	3	Expect result	Actual Result	Pass / Fail
1	A	B	C	C	B	Fail
2	A	C	B	B	B	Pass
3	B	A	C	C	C	Pass
4	B	C	A	A	A	Pass
5	C	A	B	B	B	Pass
6	C	B	A	A	A	Pass



Lesson 8 – Testing - VVT

- VVT stands for Verification, Validation and Testing.
- Testing we have already described as checking if anything wrong with the program code.
- Validation considers; Have we written the right program? Does it match the specification we were given?
- Verification considers; Have we written the program right? Does the programming logic we have used result in getting the right answers?
- Verification and Validation can require tests of their own



Lesson 9

Python packages



Lesson 9 - Python Packages

- Python is an extensible language
- There are 100's of additional 'packages' which can be added
- Included in the Python installation is a program called 'pip'
- Pip is a command line program used to install packages



Lesson 9

- 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



Lesson 9

- You use the import command to import a package
- You prefix the package function with the package name

```
import math
```

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



Lesson 9

- The `dir()` command lists the functions in the package
- The `help()` command will give a description of the function

```
import math
```

```
dir(math)
```

```
help(math.sin)
```



Lesson 10

Your Turn!



Lesson 10 - Exercise Hints

Break the problem into 2 parts.

Part 1

- User input to get the two country codes
- Read the file to find the two records relating to the country codes
- Extract the Gold, Silver and Bronze medal counts
- Keeping track of which is which
- You need to test whether
 - Both of the country codes exist
 - They were supplied in the correct order
 - These can effectively be done as you read the file



Lesson 10 - Exercise Hints

Part 2

- Calculate the differences of Gold, Silver and Bronzes between the two countries
 - The difference in the Golds has to be ≥ 0
 - For Silvers and Bronzes the difference could be any value.
- Draw a flowchart which illustrates what can happen if the difference in bronzes is either < 0 or ≥ 0 . For these two branches consider the situation for Silver and for these four branches consider the situation for Gold
- You could do the flowchart starting with Gold and work down. The resulting eight options will be the same
- You may find it useful to create a test table of the possibilities similar to what we did in Lesson 8.



Questions

Peter Smyth

Peter.smyth@manchester.ac.uk

ukdataservice.ac.uk/help/

Subscribe to the UK Data Service news list at
<https://www.jiscmail.ac.uk/cgi-bin/webadmin?A0=UKDATASERVICE>

Follow us on Twitter <https://twitter.com/UKDataService>
or Facebook <https://www.facebook.com/UKDataService>

