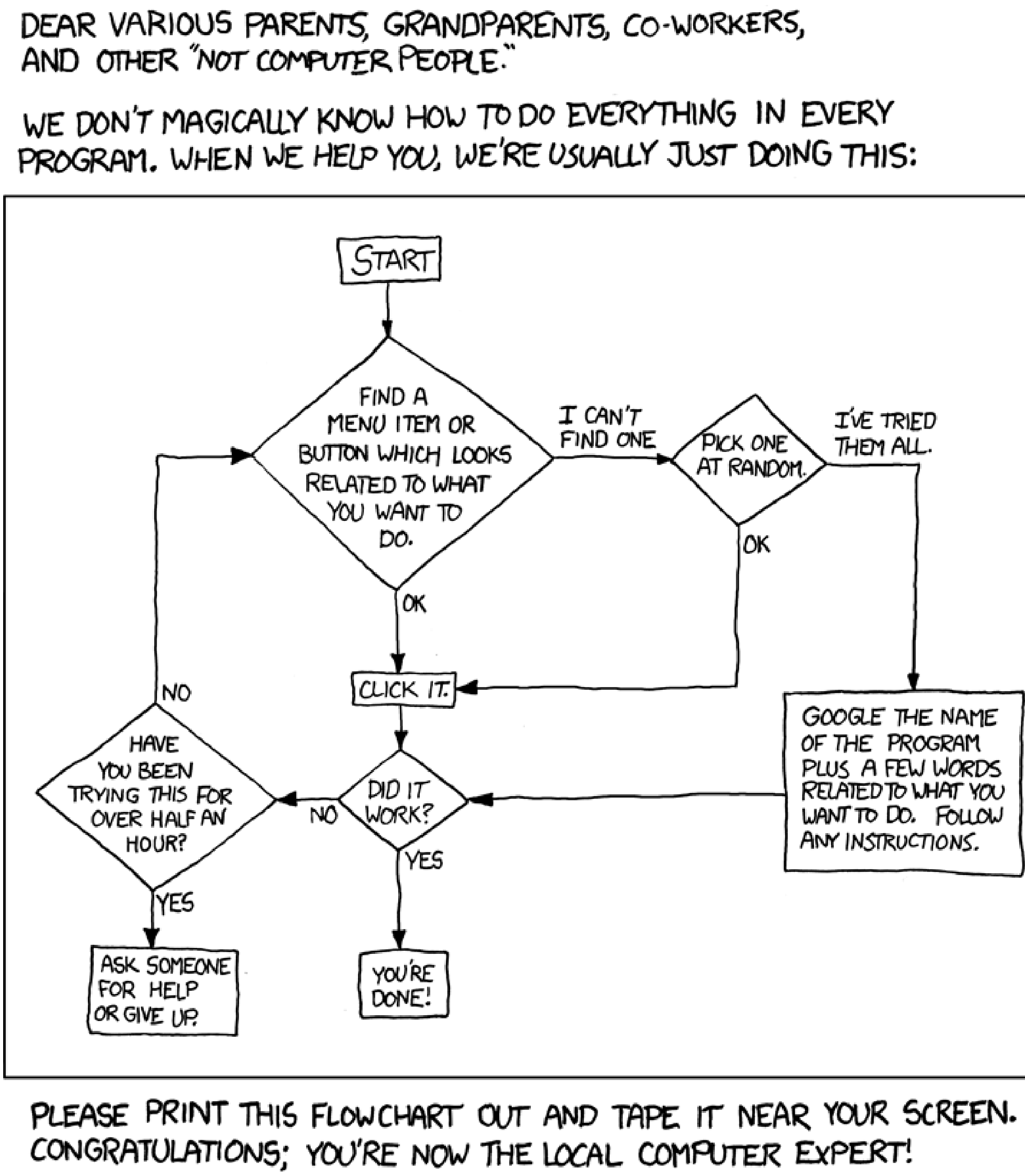
OCR Computing F452

Student Workbook


DEAR VARIOUS PARENTS, GRANDPARENTS, CO-WORKERS, AND OTHER "NOT COMPUTER PEOPLE."
WE DON'T MAGICALLY KNOW HOW TO DO EVERYTHING IN EVERY PROGRAM. WHEN WE HELP YOU, WE'RE USUALLY JUST DOING THIS:
START
FIND A MENU ITEM OR BUTTON WHICH LOOKS RELATED TO WHAT YOU WANT TO DO.
I CAN'T FIND ONE
PICK ONE AT RANDOM.
I'VE TRIED THEM ALL.
OK
OK
CLICK IT.
GOOGLE THE NAME OF THE PROGRAM PLUS A FEW WORDS RELATED TO WHAT YOU WANT TO DO. FOLLOW ANY INSTRUCTIONS.
DID IT WORK?
NO
YES
YOU'RE DONE!
HAVE YOU BEEN TRYING THIS FOR OVER HALF AN HOUR?
NO
YES
ASK SOMEONE FOR HELP OR GIVE UP.
PLEASE PRINT THIS FLOWCHART OUT AND TAPE IT NEAR YOUR SCREEN. CONGRATULATIONS; YOU'RE NOW THE LOCAL COMPUTER EXPERT!

http://xkcd.com/627/

Contents

Examples taken from the OCR Computing for A-Level book should be acknowledged where they appear. Other pictures should be sourced. If I have failed to acknowledge something, please tell me – laura.dixon@computingatschool.org.uk or @codeboom on Twitter.

Getting Started

Download Python

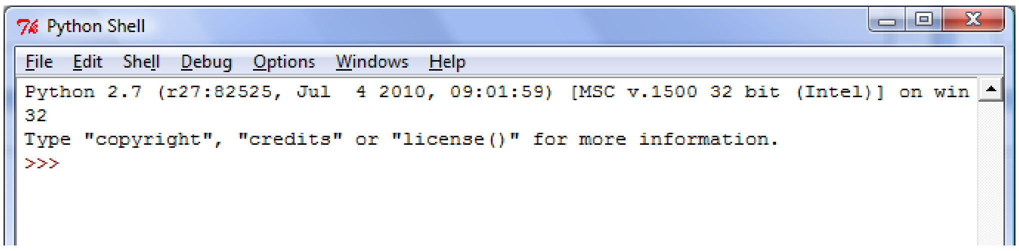
We are going to use a programming language called Python. You will need to download the interpreter from the Python website

http://www.python.org/download/

Click on the link that says ***Python 2.7 Windows installer Windows binary -- does not include source***

Then follow the wizard instructions to install it on your laptop.

Find the new menu item **Python 2.7** and open the program called **IDLE (Python GUI).** You should now be able to see the Python "shell" - it looks a little bit like Notepad. This is where we will be writing our programs.



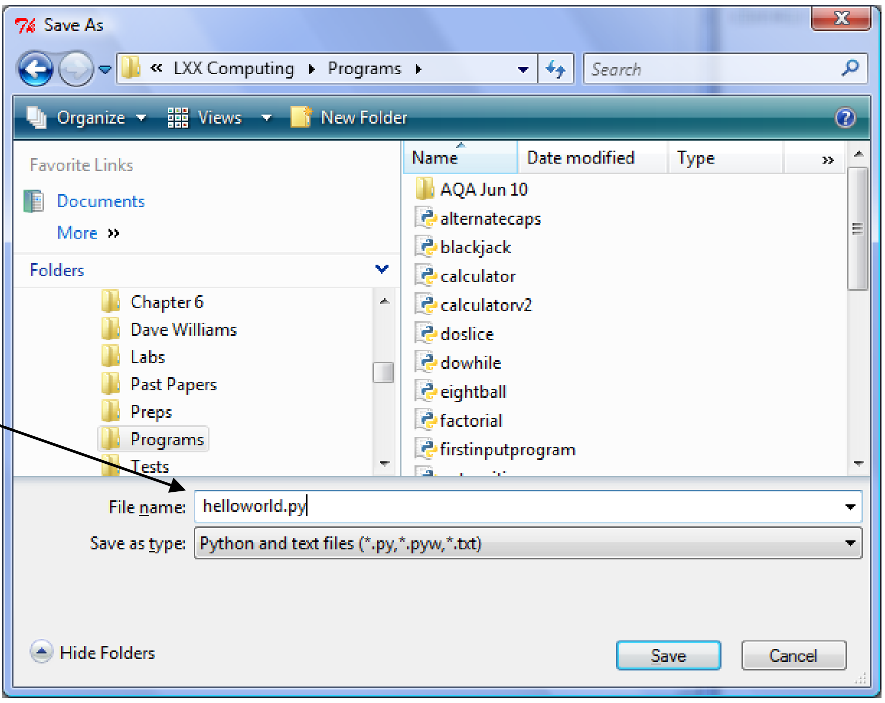
```
Python 2.7 (r27:82525, Jul  4 2010, 09:01:59) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>>
```

Creating a Python file

Sometimes we may want to write a program in a file rather than on the Python shell. To create a Python file, use

File > New Window

Then to save it, use
File > Save as
and make sure you save it as **.py** – this will ensure you get the syntax highlighting (where the program colours in your code for you).



File name: helloworld.py
Save as type: Python and text files (*.py,*.pyw,*.txt)

1 - Designing Solutions to Problems

a) The importance of interface design

After watching the Powerpoint presentation, fill in the table to reflect good and bad practice when designing interfaces.

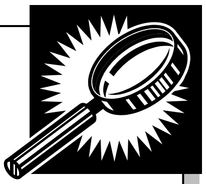
	What should you try to do, and why?	What should you try to avoid, and why?
The user		
Layout		
Order		
Validation		
GUI Objects		
Online help		

b) Designing layouts for input and output

Designing a layout

Design a data capture screen to allow parents to sign up for a school e-mail bulletin. This service will deliver news and information to the parent's email address. They will be able to choose which topics they would like to hear about (e.g. sports, arts, general news...) and how often they would like to receive an email.

You should label your design to show how you have taken into account the design considerations discussed in the previous topic.



Your favourite website?

Find a website which you think has a really good user interface.
Be careful with your selection – don't just pick a website where you like the content (e.g. your favourite football team) or where you like the colours.

Create a presentation to show the class, highlighting three key features of the interface which work well, explaining why they are following the principles of good interface design.

c) Data requirements of a program

All programs use data, but programs which store data need to have their requirements more carefully defined. Things such as **identifier names** and **data types** have to be stored somewhere - inside a **Data Dictionary**.

What is inside a data dictionary?

Fill in the table below with examples of items which could be found inside a data dictionary.

Table Name	
Field Names	
Data Type	
Field Length	
Default value of field	
Field validation	
Keys	
Relationships	
Indexes	
Access rights	

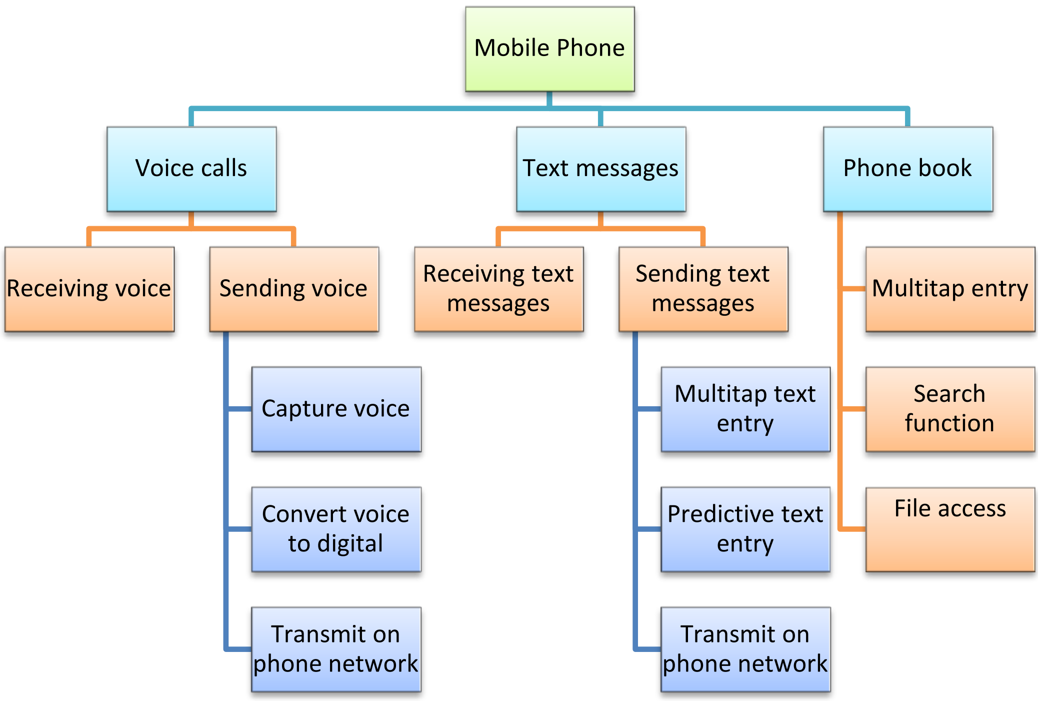
1. Why is it important to ensure an appropriate data type is chosen for a variable?

2. What could happen if a length was chosen which was too short?

d) Modular design – the advantages

Sometimes the software we need to write is extremely complicated, so it is easier for the designer to break the tasks down into sub tasks, and those sub-tasks into further sub-tasks if necessary.

Have a look at this **structure diagram** of mobile phone software, reproduced from the OCR textbook:



Fill in the following definitions:

Modular, top down design

Stepwise refinement

What are the advantages of using a top down, modular design?

e) Producing modular designs using stepwise refinement

Now that you know what a modular design is, try out the exercise from the OCR book, p63, on the page below:

f) Using algorithms to solve problems

An algorithm is considered to be the steps needed for a computer program to solve a given problem, whether these steps are expressed in a **flowchart,** in **pseudo-code** or **high-level code**.

Exam questions will ask you to create a variety of different algorithms to solve problems. An example of the type of question you may face is given below – you can answer in pseudo code!

The cost of delivery is calculated as follows:

- There is a basic delivery charge of £5 for all orders
- If the total weight of an order is more than 1kg, there is an additional charge of £0.50 for every extra 0.1kg
- If the total volume of an order is more than 1000cm^3, there is an additional charge of £0.50 for every extra 200cm^3.

Write an algorithm for a function which makes use of the global variables TotalWeight and TotalVolume and returns the cost of delivery.
(Taken from an OCR exam paper)

[7]

g) Program Flowcharts

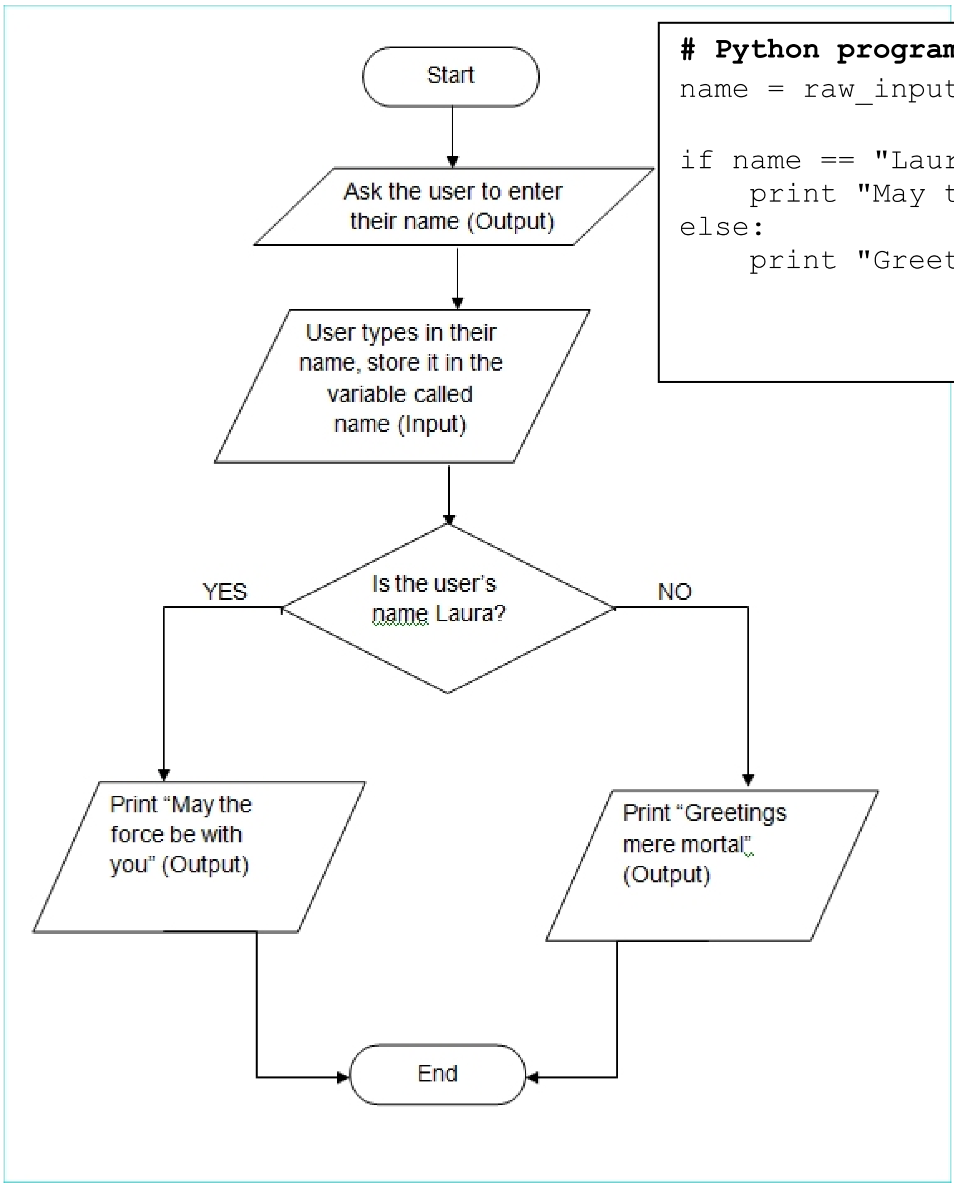
We will look at using flow charts as a method of planning a program. They are very useful for simple programs but obviously they get rather difficult to manage when your programs become bigger.

Circle - the start and end of the flow chart = the start/end of the program

Rectangle - a process = any calculation that happens in the program

Diamond - a decision/question = an IF/ELSE statement

Parallelogram - an input or output = raw_input() or print


Start
Ask the user to enter their name (Output)
User types in their name, store it in the variable called name (Input)
Is the user's name Laura?
YES
NO
Print "May the force be with you" (Output)
Print "Greetings mere mortal" (Output)
End

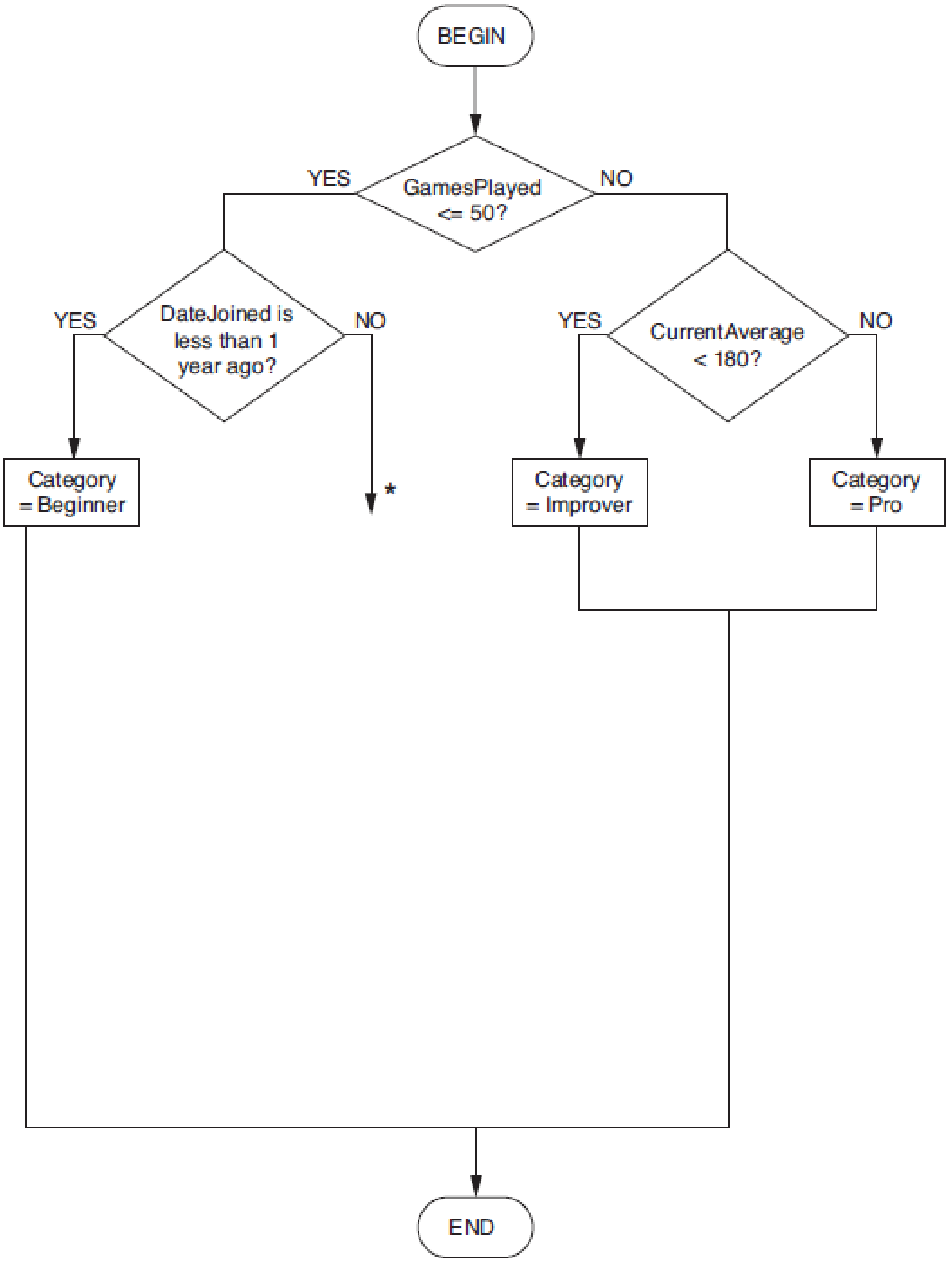
```
# Python program
name = raw_input("Enter your name: ")

if name == "Laura":
    print "May the force be with you"
else:
    print "Greetings mere mortal"
```

Here is an example of an exam question where you are asked to complete a flow chart

Complete the flowchart at the arrow marked * to show:

- If the CurrentAverage >= 200 then the category should be Pro,
- otherwise the category should be Improver.



[6]

h) Pseudo-code

Pseudo code allows you to specify the steps of an algorithm, but without worrying about the syntax of your code.

There are no standard conventions for pseudo-code, but the logic of your algorithms should be clear. You can do this by capitalising keywords, using English phrases rather than complex expressions and indentation to show the control structures. Example from OCR text book.

For example:

```
IF Character has reached end of platform
   Display "YOU WIN"
   REPEAT
      Play Music
   UNTIL any key is pressed
END IF
```

Usually, to make your pseudo code understand able, you should:

- Put any key words (e.g. IF, WHILE, REPEAT) in capital letters
- Use proper indentation
- Use quote marks for strings
- Use sensible identifier names

Write some pseudo code which would form part of a simple platform game. The code should describe that if the character touches an enemy, a noise should play and the message "Oops!" should appear on screen.

i) Evaluating algorithms, commenting on their efficiency, correctness and appropriateness

We have programmed a variety of different algorithms so far during the course. However, we have not really considered whether or not these algorithms are *efficient*.

Think back to this program from the start of the course:

Write a script that asks a user for a number. The script should add 3 to that number. It then multiplies the result by 2, subtracts 4, subtracts twice the original number, adds 3, then prints the result.

- On Python, write this program in the most <u>inefficient</u> way possible, but so that it still works
- What actions did you take to make this program more efficient?
- How could you improve it?

Competition
I want to write a program where the user must think of a number between 1 and 100 (an error message if this criteria is not met). The program should then make guesses as to what the number is.

For example:

```
>> Welcome! Please think of a number but do not tell me what it is!
>> My guess is 32. Is it correct (press y) or incorrect (press n)?
>> n
>> My guess is 58. Is it correct (press y) or incorrect (press n)?
>> y
>> Yay! I guessed the number in 2 guesses.
```

We will measure this program's efficiency by measuring how many guesses it takes to correctly guess a number. Our test data for this program will be: 2, 45, 78, 100

The winner will be the person whose program takes the lowest amount of combined guesses to guess all four numbers of test data. Use comments in your algorithm to show how you are making it work efficiently

What makes an algorithm efficient?

Why is efficiency so important?

j) Rapid Application Development (RAD), prototyping and iterative development

Using the notes available to you in class, create a presentation to show the rest of the group which should cover: prototyping, iterative development, purpose of RAD, advantages and disadvantages.

Stick your presentation in to this booklet below.

2 - The structure of procedural programs

a) Procedural programming key terms

Fill in the following key definitions and ensure you learn them properly as they will be important throughout the course!

Sequence

Statement

Function

Subroutine

Parameter/Argument

Selection

Iteration

Procedure

Loop

b) Sequence, Selection and Iteration

Sequence

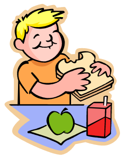
A **sequence** is ...

An example of a sequence in code:

c) Selection – IF and CASE-SELECT statements

Selection (if/elif/else)

Selection decides what will happen in the code, depending on a condition. We see conditions in everyday life, for example:



"If you eat all your greens, you can have some pudding..."

condition *what will happen if the condition is met*

The part after the 'if' is the **condition** which must be satisfied in order for the following **action** to happen. Sometimes we are given an **opposite** which will come into force if the condition is not met:

"If you eat all your greens you can have some pudding...else you will go to bed early!"

what will happen if condition is not met

This describes the condition, what will happen if the condition is met and what will happen if the condition is not met. In Python we can write some code that will have the same effect.

```
greensEaten = raw_input("Did you eat your greens?")
if greensEaten == "yes":
      print "Well done, you can have some pudding"
else:
      print "NAUGHTY! Go to bed now!"
```

Which part of this code is the....

Condition?

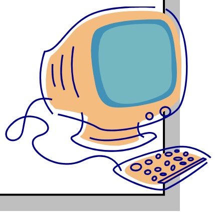
Action when the condition is met?

Action when the condition is not met?

Extra thought...

What happens if you type in "Yes" or "YES"? Why does this happen?

Programming Task
Ask the user to input the temperature of their water, measured in degrees centigrade. Depending on the number they input, output "Freezing" (0 or below), "Ambient" (1 - 99) or "Boiling" (100 or more).



Case-Select statements

Sometimes we may have a lot of different actions to perform depending on the outcome of a condition. We could write the above programming task using a case-select statement (sometimes called a switch) in pseudo code like this:

```
SELECT CASE
   CASE less than 0
      PRINT "Freezing"
   CASE 1 to 99
      PRINT "Ambient"
   CASE more than 100
      PRINT "Boiling"
END SELECT
```

Written task
Write some pseudo code to control a switch on a washing machine. The three possible values of the switch are:

- Off (set spin speed to zero)
- Slow spin (set spin speed to 50)
- Fast spin (set spin speed to 100).

Use a CASE-SELECT statement in your answer.

Why bother?
In some languages and programming environments, the use of a **case** or **switch** statement is considered superior to an equivalent series of *if-else* statements because it is:

- **easier to debug**
- **easier to read**
- **easier to understand** and therefore
- **easier to maintain**
- **executes much faster**

But...Python doesn't even have a CASE-SELECT statement – you can achieve the same result using ELIF.

d) Iteration - count-controlled loops (FOR-NEXT loops) and condition-controlled loops (WHILE-ENDWHILE and REPEAT-UNTIL loops)

Condition controlled loops (WHILE/REPEAT)

The last of the basic programming constructs is **iteration** which is usually called a *loop*.

Do you remember the song from when you were younger?

I know a song that will get on your nerves
I know a song that will get on your nerves
I know a song that will get on your nerves
And it goes like this....

We can print this using something called a *while loop* in Python:

```
1       timesPrinted = 0
2       while timesPrinted < 3:
3               print "I know a song that will get on your nerves"
4               timesPrinted = timesPrinted + 1
5       print "And it goes like this"
```

Questions

1. What data type is the variable `timesPrinted`?

2. What is the purpose of this variable?

3. What is the condition on line 2?

4. What is the purpose of line 4?

5. What do you think would happen if you deleted line 4 from the program?

6. How many times will line 5 be printed? Why do you think this?

Programming task

What about if we wanted to change the variable to be called `timesLeft` and use it to count down (instead of starting at 0 and counting up)? Can you write a program to do this?

Why is a WHILE loop called a condition controlled loop?

REPEAT-UNTIL loop

A REPEAT-UNTIL loop is also an example of a condition controlled loop. Here is the same code as on the previous page, re-written as a REPEAT-UNTIL loop in pseudo code.

```
timesPrinted = 0
REPEAT:
        print "I know a song that will get on your nerves"
        timesPrinted = timesPrinted + 1
UNTIL timesPrinted == 3
print "And it goes like this"
```

What is the difference between this and the WHILE loop version?

Count controlled loop (FOR)

The previous types of loop relied upon a condition being met for them to stop executing. A FOR loop is slightly different – it will execute the loop a certain number of times and then stop. For example:

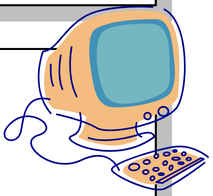
```
FOR i FROM 1 TO 4:
   PRINT "Howdy!"
END FOR
```

You may be wondering what **i** is – it's a counter, just like `timesPrinted` from the previous example. It is traditional to use i (or j) as a FOR loop counter variable name, probably because it is so short and easy to use.

Here is the same loop in actual Python code – even easier!

```
for i in range(4)
    print “Howdy!”
```

Why is a FOR loop called a count-controlled loop?



Loops Programming Tasks

1. Write any loop that runs exactly 9 times.

2. Write a <u>different</u> loop that runs exactly 9 times.

3. Write a program that asks the user for two numbers x and y. The program should print out x y number of times.
e.g.
x = 1
y = 3
The program will print 1 1 1 (it doesn't matter if they are on different lines)

Hard
4. Write a loop that will print the numbers 1 to 6. For each number, print next to it whether it is even or odd. Don't forget you can check this with the modulus operator

number % 2 == 0

e.g.
1 odd
2 even
3 odd
4 even
...

5. Write a program which will continuously ask the user to "talk to me". If the user types "bye" the loop ends and the program says "Cya later", otherwise the program responds "That's interesting, tell me more" and the user must type in something else.

e) Nested statements

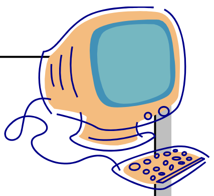
A nested statement is where a selection or iteration statement is contained within another selection or iteration statement.

e.g. (pseudo code)

```
IF gender EQUALS female
      IF name EQUALS Laura
            PRINT "Hooray!"
      ELSE
            PRINT "Hello"
      END IF
END IF
```

This IF statement is **nested** (completely inside) the other IF statement

Write your own nested statement pseudo code to print "Accepted" if someone is a member and over 18, "Not a member" if they are not a member or "Underage" if they are not 18.



Programming Tasks – rather tricky!

1. Create some code which will print a box that looks like this:

```
+ -- +
|    |
+ -- +
```

Hint: If you don't want to move on to the next line after printing something, put a comma e.g. `print "+",`

2. Now make some code that will use a loop to print a row of boxes. Here is a good opportunity to use a for loop, e.g. `for i in range(5)` should print 5 boxes

```
+ -- + -- + -- + -- + -- +
|    |    |    |    |    |
+ -- + -- + -- + -- + -- +
```

You need to be careful that you don't end up with two middle pieces to your box (see below) – how can you get around this?

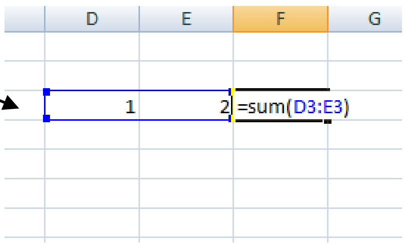
```
+ -- ++ -- +
|    ||    |
+ -- ++ -- +
```

3. Now see if you can use a nested loop to create a square of boxes. Again, careful that you don't end up with duplicates of the middle part, think of a way around this!

f) Subroutines - Procedures and Functions

A ***procedure*** is a set of instructions which are given a name and can be run at any time as a ***subroutine*** of the program you are writing.

Remember Excel? You were using subroutines like SUM without even realising it!



Defining a procedure

Here is how to define a procedure in Python:

```
1     def greeting():
2             print "Hello"
```

- The key word def tells Python that we want to define a procedure
- The name of the procedure is *greeting*
- In this procedure, we just have empty brackets (). More about this later...
- Line 2 is the instructions that will execute when the procedure is called. You can have more than one line of instructions.

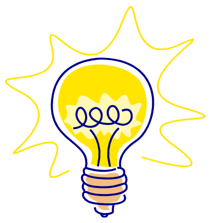
But...if you just put this code into a Python file and run it, NOTHING WILL HAPPEN! In order to use the procedure, we have to '**call**' it - we are telling the program we would like to run that set of instructions now please.

Calling a procedure

You can call the procedure by typing its name, with the brackets:

```
greeting()
```

You can call a procedure multiple times, but you can only define it once



Parameters and Arguments

Think back to the SUM() procedure in Excel. You had to tell it which of the cells to sum up, and you put that information inside the brackets, e.g. =SUM(D3:E3).

You can do a similar thing with Python! Look at this example:

```
1     def greeting(person):
2             print "Hello",person
```

On line 1, we have now added a variable (person) inside the brackets, which is known as a **parameter**. We can then use that variable wherever we want inside the procedure – here we are saying hello to a specific name.

When we call a procedure which has a parameter, we must specify a value for the parameter - this is called an **argument**. For example, I am using the argument “Laura” when I call the function.

```
greeting(“Laura”)
>>> Hello Laura
```

You can think of it as if the parameter is deciding on the name of a variable, and the argument is giving it a value:

Watch out! At the end of the procedure, the variable person ceases to exist.

Written Tasks

1. In the OCR textbook, read the section on Subroutines on page 80 - 81

2. Define the following terms (in context):
- Argument

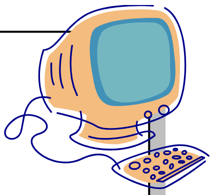
- Parameter

- Identifier

- 'Calling' a procedure

3. What is the difference between an argument and a parameter?

4. In the Python book, read page 24 - 28 (sections 3.5 - 3.8 inclusive)



Programming Tasks

1. Write a procedure that prints out "Hello World!" when it is called

2. Write a procedure that has `inputNumber` as a parameter. When the procedure is called with a number as an argument, it will print out that number squared. (e.g. if the argument given is 4, it will print 16). Make sure you give the procedure a sensible name.

3. See if you can translate this pseudo code into real Python code:

```
Define procedure
Higher( a, b )
If a is larger than b
     print a
otherwise
     print b
End of procedure
```

4. Can you make a similar procedure for `Lower( a, b )` which prints whichever number is lowest?

Functions

I have a slight confession to make. When I implied that =SUM() in Excel was a procedure, I lied – it is actually an example of a **function**. The difference between a procedure and a function is that a procedure has no return value, whereas a function does. That probably makes no sense at all, so let me explain:

```
def plusProcedure(a,b):
    c = a + b
    print c
```

```
def plusFunction(a,b):
    c = a + b
    return c
```

Here are two pieces of code – on the left is a procedure, on the right is a function. They do the same job, but with very slightly different results.

Once the **procedure** has ended, it prints out the result of adding up the two numbers. Happy days – we know the result and it is on the screen for us to see, right? Wrong. What if we didn't want to print out the result quite yet, or we wanted to store it for later, or we wanted to use it in another calculation? We can't....

...unless....we use a **function**! With the example on the right, nothing actually gets printed. However, because we have returned the result, we can store it in a variable, like this:

```
result = plusFunction(1,2)
```

The variable result now has the value of 3, and we are free to do whatever we like with it.

g) Recursion

Recursion is essentially when a function calls itself. You may wonder why anyone would want to bother doing this, but it is actually extremely powerful. Observe this example from p65 of the Python book:

```
01    def factorial(n):
02      if n == 0:
03          return 1
04      else:
05          prev = factorial(n-1)
06          result = n * prev
07          return result
```

This is an example of a **recursive algorithm**, because on line 5 the function `factorial` calls itself. However, caution is needed because if a function calls itself it is possible to make **infinite recursion** and this will cause you to have to crash and burn your Python interpreter ☺

In order to avoid this happening, you need to have a **stopping condition**. In this case, the stopping condition is `n == 0` as seen on line 2. i.e. when n is equal to zero, we do something else instead of calling the function again.

However this by itself is not enough if we are always calling the function with the same value of n, so another crucial part is line 5 where we specify that we are calling the function `factorial`, but with the argument `n-1` i.e. n is gradually getting closer to 0, our stopping condition.

Read p67 of the Python book to find out the recursive algorithm for the Fibonacci numbers (0, 1, 1, 2, 3, 5, 8, 13, 21...). Write out the algorithm below and label the self call, the stopping condition, and how the algorithm gets closer to the stopping condition.

h) Tracing a recursive algorithm

To really get to the bottom of what is going on in a recursive algorithm, we need to be able to trace it. Tracing a recursive algorithm is a bit like this example:

I want to know Auntie Brenda's address, so I phone my mum

Mum doesn't know, so she puts me on hold while she phones Gran

Gran doesn't know so she puts Mum on hold and phones Uncle Tom

Uncle Tom tells Gran the answer

Gran takes Mum off hold and tells her the answer

Mum takes me off hold and tells me the answer

I now know Auntie Brenda's address

The key thing to note in this example is that each person is put on **hold**, until the answer is received, at which point they are taken off hold and the result is relayed back down the chain. The same thing happens with a recursive algorithm – see if you can follow and fill in the values below

```
3! = 3 x 2 x 1 = 6
```

Line	n	Prev	Result	Comment
01 def factorial(n):				
02 if n == 0:				Condition not met, move to line 4
04 else:				
05 prev = factorial(n-1)				We put this frame on hold while we call factorial(2)
01 def factorial(n):				
02 if n == 0:				Condition not met, move to line 4
04 else:				
05 prev = factorial(n-1)				We put this frame on hold while we call factorial(1)
01 def factorial(n):				
02 if n == 0:				Condition not met, move to line 4
04 else:				
05 prev = factorial(n-1)				We put this frame on hold too and call factorial(0)
01 def factorial(n):				
02 if n == 0:				Condition is met!
03 return 1				Return value of 1, control passes back
05 prev = factorial(n-1)				We now know that the value of prev is 1
06 result = n * prev				
07 return result				Return value of 1, control passes back
05 prev = factorial(n-1)				
06 result = n * prev				
07 return result				Return value of 2, control passes back
05 prev = factorial(n-1)				
06 result = n * prev				
07 return result				Here is the answer

Here is a trace table for the following recursive algorithm. This one behaves exactly the same way but looks slightly different as there is no return value – it's even easier to trace! Can you finish it off?

```
01  def countdown(n):
02      if n <= 0:
03          print "Blast off!"
04      else:
05          print n
06          countdown(n-1)
```

Line	**N**	**Output**	**Comment**
01 def countdown(n):	3		
02 if n <= 0:	3		Condition not met, move to line 4
04 else:			
05 print n	3	3	

i) Iteration or recursion?

"Every recursive algorithm can also be written as an iterative algorithm.".... *Ehh?*
All this means is that instead of using a function that calls itself, we could be using a loop instead. Hopefully that makes more sense!

Fill in this table so that you are sure you know the difference between the two:

	Iteration	**Recursion**
How does it repeat instructions?		
How many function calls are there?		
Is there a stopping condition?		
How does the program move towards the stopping condition?		
How many sets of variables are there, and why?		
How much memory is needed?		
How easy is the code to understand?		

3 - Data types and data structures

a) Data Types

In a programming language there are different types of values. In Python these are called:

- Integers
- Strings
- Floating Point numbers
- Boolean

Write down a definition and an example for each of these **data types**

Data Type	Definition	Example
Integer		
String		
Floating Point		
Boolean		
Character		

Have a guess at what **data type** the following are, and then use the Python shell to see if you were right. For example, to find out the data type of `'Hello World'` you should type:

```
>>> type('Hello World')
```

The shell will respond to you with the type.

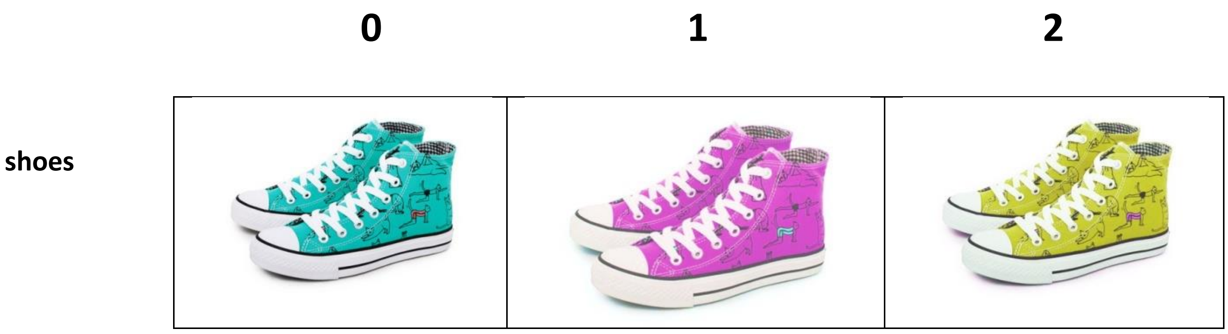
Data	Your guess	Actual data type
123		
'cat'		
3.142		
cat		
'123'		
1,000		

Important Question -Why is 123 different to '123'?

b) Arrays

An array is a special **data structure** which contains several items of similar data.

Imagine a fictitious shoe rack with three spaces for shoes:



We could have made three variables to store the colours of these shoes:

```
Shoes1 =”Green”
shoes2 =”Pink”
shoes3 =”Yellow”
```

But...that’s a bit long winded and there is a much better way of doing it using an **array**!

```
shoes = [“Green”, “Pink”, “Yellow”]
```

In this case, **shoes** is the name of the array, which has three values. You may have been wondering why my shoe rack was numbered 0, 1, 2 instead of 1, 2, 3. It’s not a mistake! Python starts numbering its **array indexes** (the position of each item in the array) at 0 not at 1.

So...if we want to get at the pink shoes and we know they are in position 1, we can use the code:

```
print shoes[1]
```

So what? Why bother?

There are several reasons (other than sheer tidiness) why arrays are so awesome! One is that if you need to add an extra pair of shoes, you can just **append** one on the end, without having to bother making a new variable or knowing which position the shoes will occupy:

```
shoes.append(“Black”)
print shoes[3]
```

Even more awesome than this is the fact that you can really easily go through every item in an array and print it, or do whatever you like with it!

Python code	Output

`for pair in shoes:`
`   print pair`
	`Green`
	`Pink`
	`Yellow`
	`Black`

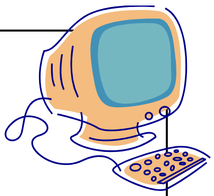
And...you can very easily check if an item is in an array:

Python code	Output

`if “Pink” in shoes:`
`    print “Stylish!”`
	`Stylish!`

Hip Hip.....Array!

...*groan*...



Now it’s your turn to make an array and see what it can do:

1. Create an array called alphabet, which contains all of the lower case letters of the alphabet

2. Loop through the alphabet array, printing out each letter

3. Write a program where the user is asked to input a letter. Test whether what they have input is in the array (i.e. test it with numbers and words too), and give appropriate messages depending on the outcome, e.g. “Thank you – you input L”

4. At present, if someone inputs “A” then it will not be found, even though “a” is in your array. See if you can improve on your program from part 3 by using the lower() method.

```
      theirLetter = theirLetter.lower()
```

...will convert the letter they input (stored in the variable theirLetter) to lower case. (If it is already lower case, not a lot happens.)

Scary things they might ask you to do with arrays in the exam include:

- Find the maximum/minimum/total/average/most frequent of the elements of an array
- Extract elements of an array that fit a given condition and placing them in another array
- Use a two-dimensional array as a look-up table

c) Advantages and disadvantages of different data types

The point of this section is to show that you can sensibly consider which **data type** or **data structure** to use when, and justify your choice with comments about the advantages of the chosen type - or disadvantages of another type!

Discuss with your partner and make notes on which data type you would use in the following situations, and why you chose this type:

Number of votes cast for each candidate in an election

Animal details to be stored by a computer program for vets

Photocopier usage data for A4 and A3 printing, both colour and black and white

Record of whether a customer has paid their gas bill or not

A telephone number

d) Design and implement a record format

Record Types

In some programming languages, it is possible to define your own data types.

1. Have a look at the **Pascal** program below, and comment it appropriately.
(We are using Pascal for this because it is rather tricky to do in Python...)

```
program Types;

Type
   StudentRecord = Record
      Number: Integer;
      Name: String;
   end;

var
   Student: StudentRecord;

begin
   Student.Number := 12345;
   Student.Name := 'John Smith';
end.
```

2. What is the name of the custom data type? ____________________

3. What are the fields inside this record called? ____________________________

4. For each field, list its data type and its value:

Data type ____________________ Value ____________________

Data type ____________________ Value ____________________

5. What is the name given to the instance of the custom data type? ___________________

6. Suppose you want to create a record type for a car having an MOT. Suggest some suitable fields in the table below.

Field	**Data Type**	**Size**

7. Now define the type from question 6 in **Pascal** using the example given above to help you, and then create an instance of it called CustomerCar.

e) File organisation - serial, sequential, indexed sequential and random

It’s all very well being able to make our own record types, but once the program ends, all of this data is lost. That’s not very useful to us, so we need to be able to store the data somewhere.

There is more than one way of organising the records in a file – in fact there are four:

	How is data stored? + Advantages/disadvantages of this format
Serial	
Sequential	
Indexed sequential	
Random	

Which of these types do you think would be the most efficient to use in the following situations? Discuss with a friend:

Files with a huge amount of records in them

Files with only a few records in them

When you need all of the records, one by one, in order of the key field

When the key you are searching for is near the end of the file

f) Store, retrieve and search for data in files

Opening a file

To be able to use a file, we need to open a **file handle** first. A file handle is a variable which is a way of referring to the file so we can perform various operations on it. A common name for a file handle variable is simply the letter f.

e.g.

```
      f = open('test.txt','w')
```

File Access Mode: w = writing, r = reading, a = appending (adding things on to the end)

Creating a file

If you try to open a file which doesn't exist, Python will create that file for you – very helpful.

Writing to a file

We can then **write** things to the file:

```
      f.write("Hello world!")
      f.write( str(4) )      # Cast numbers to string before writing
```

...and close the file:

```
      f.close()
```

Reading from a file

We can also **read** things from the file so we can use them. The `readLines()` method is really handy because it reads all of the lines of the file into an **array**, and then you can access them individually:

```
      f = open('test.txt', 'r')
      fileArray = f.readlines()
```

Note the r for read mode

e.g. if you wanted to now print the 3rd line (remember, 3rd item = array index 2)

```
      print fileArray[2]
```

Getting to a specific location

If you are using a direct access file where all the records are of fixed length, you may want to use these functions to read specific parts of the file:

```
      f.tell()         # tells you the current position in the file
      f.seek(10)       # moves you along 10 bytes in the file
      f.seek(10,20)    # moves you along 10 bytes, from position 20
      f.read(10)       # reads 10 bytes from your current position
```

Renaming a file

To rename a file, we need to import a special library called `os`. This is done by putting an import statement at the very top of our program

```
        import os
```

We are then free to use the functions provided by the `os` library, which can allow us to rename a file:

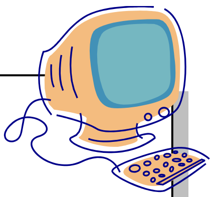
```
        os.rename(current_filename, new_filename)
```

Make sure the file is closed - f.close() – before trying to rename or delete it, otherwise you will receive an error and your program will terminate.

Deleting a file

This also uses the os library and is fairly straightforward:

```
        os.remove(filename)
```



Programming Exercise

<u>Fortune Cookies</u>

When the program runs you are presented with three options. It will continue running until you choose to exit.

1 – Add a new fortune
2 – Tell my fortune
3 – Exit

Option 1 should allow the user to type in a 'fortune'. This should be appended to the end of a file called fortune.txt. The user should then be taken back to the options.
Option 2 should read a random line from the file fortune.txt and tell the user their fortune. They should then be taken back to the options. Use the `random` library to do this.
Option 3 will exit the program.
The code should be appropriately commented.

g) Estimating the size of a file

Data files often contain a number of records, each one relating to a different object or person. It is easy to estimate the size of a file if you know the size of one record in bytes, and the number of records in a file.

Look at p94-95 of the textbook and make sure you know how the example works, and then try to find out the size of the following files **in kilobytes**.

1. A vet's database storing animal records as Name (String, 20) OwnerName(String, 20), Age(int, 2) with 320 animals on record.

2. A library storing book records as Title (String, 30), AuthorSurname(String, 20), AuthorForename(String, 20), ISBN(String, 15) with 659 books on record.

h) File operations (opening, reading, writing, updating, inserting, appending and closing)

Large programming task
Imagine you are a teacher, and you wish to have a program to keep a record of your students' grades in a file. The start of the data file looks something like this:

```
00,smitje,a,b,b,b
01,jonefj,e,d,c,d
02,blogja,c,c,b,a
```

1. Recreate this data file by typing in the data and saving as `markbook.txt` Be careful to ensure that you type it in carefully, with no spaces!

2. Start a program which has a menu where the user must type in a number to choose an option. The options at the moment will be:

```
1 – Add a new student
9 – Exit the program
```

The program should run forever until 9 (or an invalid input) is typed. (Hint: Use a loop!)

3. If the user types 1, they are prompted to enter the student's number and username. These are then written to the end of the file `markbook.txt` like this example:

```
04,dixoll,x,x,x,x
```

There is a big slip up to make here! To get things to write to the file properly, you will need to use the format operator, as seen below:

```
toWrite = "%s,%s,x,x,x,x\n" % (studentNumber, studentName)
```

The format we want to end up with. %s is a placeholder for where a string will go. If you want to save a space for an integer, use %d or for a decimal use %g.

\n means move on to a new line of the file – important!

The strings we want to substitute in to the placeholders (%s) – in the right order.

4. Add a new option to the menu:

```
2 – Search for student
```

The user should be prompted to type in a student number, and the program will print out the record for that student.

5. Add a new option to the menu:

```
3 – Delete student
```

The user should be prompted to type in a student number, and the program will delete the student *in the manner used by serial and sequential files*.
i.e. here is the pseudo code:

```
OPEN FILE markbook.txt in READ MODE
READ FILE INTO fileArray
CLOSE markbook.txt
OPEN FILE markbook2.txt in WRITE MODE
FOR i IN RANGE (LENGTH OF  fileArray)
     IF i DOES NOT EQUAL the line to delete
          WRITE fileArray[i] TO markbook2.txt
END FOR
CLOSE markbook2.txt
REMOVE markbook.txt
RENAME markbook2.txt TO markbook.txt
```

Optional extras

If you are feeling very confident you can have a go at these tasks – be warned, they are tricky and will require some thinking and possibly research!

a) In option 2, if they entered a student id that doesn't exist, give a message saying the student was not found rather than a Python error.
Hint: Use `len()` to find out how many records there are in the array, as in part 5.

b) In option 3, check first whether the student exists before trying to delete him/her.

c) In option 2, instead of just printing the whole line, try to print it more "prettily" like this:

```
Student 01: smitje
Assignment 1: A
Assignment 2: B
Assignment 3: B
Assignment 4: B
```

(Hint: use the method `split()` to split up the line by comma).

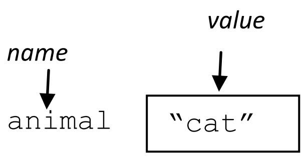
d) When you delete a student, ensure that all numbers remain in sequence, i.e. if you delete student 3 and there are 5 students, the subsequent numbers should be moved back.
-- This is really difficult ;)

4 - Common facilities of procedural languages

a) Assignment statements

One of the most important features of any programming language is the ability to store data inside **variables**. A variable is like a storage box – you can store something inside it, and you can give it a name by which you can refer to it later.

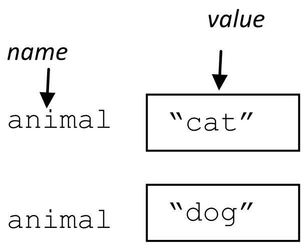
In this example, I have created a variable with the **name** animal, and **assigned** it the **value** “cat”



Real Python code

```
animal = “cat”
```

Here I am changing the value of the variable called animal to become “dog”. As you can see , the old value is overwritten and lost forever.

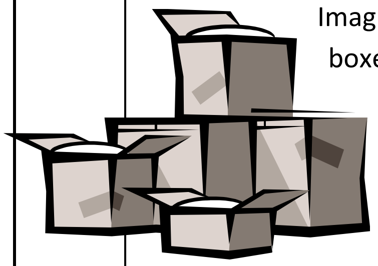


```
animal = “dog”
```

This is called an **assignment statement**

We can give variables *almost* any name at all*. You create a variable like this in the Python shell:

```
>>> message = "Hello World"
>>> pi=3.1415
>>> age=17
```



Imagine you are moving house. When you put your things in boxes, you labelled the boxes with sensible descriptions such as “Kitchen” or “Dad’s tools” so that you would easily be able to find things when you needed them again. Take the same care when labelling your variables!

Rubbish variable names - `variable1, myNumber, result4....`

Use a sensible name that describes the contents of what is inside the variable, so that if you need to find a bug or if someone else needs to alter your program, life is a lot easier!

* Watch out! There are some names you can’t use because they are **reserved words** – they are the words already being used by Python. For a full list of names you can't use, see p13 of the Python book.

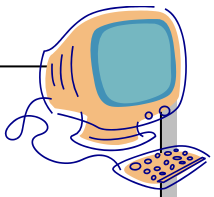
To show the value of a variable you can ask Python to print it by using the key word print, then the name of the variable, like this:

```
    print nameofvariable
```

e.g.

```
  message = "Hello world!"
  print message
```

Code Breaker Challenge



1. Use assignment statements to create some variables:

A is 2, B is 1, C is 7 and D is 4.

2. Now do some calculations to find out what the following letters are:

Hint: Can you remember how to show the value of a variable?

e = (a * b) + b

f = c + b

g = (d / a) – b

h = c * a

i = a + b + d

j = c - a

k = c – d * f

l = c + a

m = (c * a) – b

n = a * d

o = a + d – b

p = (c * d) – a(b + d)

q = a * (c + (d – b))

r = (d * d) - b

s = r – f - g

3. Using the values of the variables, work out what this word is:

6	18	7	8	3

The answer is: ___________________________________

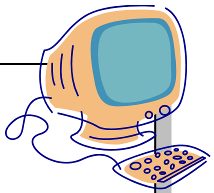
An assignment statement is....

An example in Python code would be....

b) Arithmetic operators including operators for integer division (+, -, *, /, MOD and DIV)

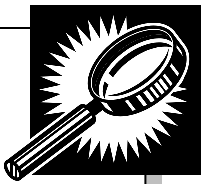
1. Basic calculations using the Python Shell
Type in 2 + 2 next to the arrow prompts and press enter - what happens?

```
>>> 2 + 2
```

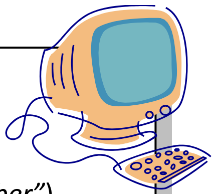


Use the Python shell to find the answer to these problems

1. 5 plus 500
2. 1237 minus 686
3. 12 times 12
4. 9 divided by 3
1. 14 divided by 9 – is the result what you expected? Why do you think this is?



Look on page 46 of the Python book and find out what the **modulus operator** - % - does.



Programming Task
Write a program which asks the user to input an integer number.

- Tell them whether the number they have input is an odd number *("3 is an odd number")* or an even number *("4 is an even number")*.
- If they entered zero, print *"You entered zero".*
- If they did not input a number (for instance if they entered some text) print a suitable error message.

What is the difference between / and DIV?

c) Relational operators, e.g. =, <, <=, >, >= and <>

Find out what the following operators mean from the Python book, page 47. Some of them you know already, some you will recognise from Maths and one is not even used in Python but might show up in the exam.

=

==

>

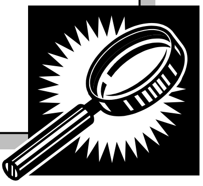
<=

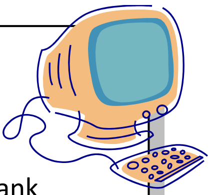
<

>=

<>

!=





Programming Tasks

1. Write a Python script which asks the user to input a number less than or equal to 20. If the user tries to enter a number larger than 20 print an error message, otherwise say thank you.

2. Modify your program from part 1 so that the program prints a count up from 1 to that number

e.g. if the user enters 5, the program will print

```
1
2
3
4
5
```

Hint: This will require a loop!

Evaluate the expressions below and write True or False (or error) next to them

	True, False or error?		**True, False or error?**
`4 > 5`		`2 < 1`	
`5 >= 4`		`9.8 <= 9.9`	
`5 == 5`		`9.0 = 9`	
`5 = 5`		`9.0 == 9`	
`3 != 2`		`a = 5`	

This one's difficult ;)

d) Boolean operators (AND, OR and NOT)

Think back to the last time you went to a theme park, and the biggest scariest ride there was in that theme park. You probably had to be over a certain age and over a certain height to ride that rollercoaster.

Let's say there are two conditions you have to meet:

- You must be over 18
- You must be taller than 150cm

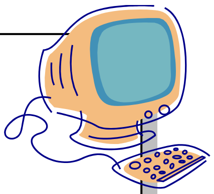
We can write these in an if statement using the boolean operator **AND**

```
# Boolean operators
# Get them to type in their age and height
age = raw_input("Enter your age: ")
height = raw_input("Enter your height in cm: ")

# Convert what they input into integers so we can compare
age = int(age)
height = int(height)

if age > 18 and height > 150:
    print "You can go on this ride"
```

Notice that we are comparing two different things - we are checking that age is greater than 18 AND that height is greater than 150.



Can you add some more elif and else statements which do the following:

1. If the person is old enough but too short, print "You are too short"
2. If the person is tall enough but too young, print "You are too young"
3. If the person is too short and too young, print "You are too short and too young"

e) Operator precedence & parentheses

From your GCSE Maths lessons, you probably already know about BIDMAS:

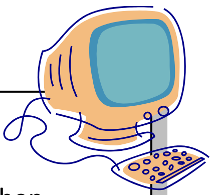
Brackets
Indices
Divide / **M**ultiply (and **M**od)
Add / **S**ubtract

Exactly the same applies when you do a calculation in Python – the operators are evaluated in this order, this is known as **operator precedence**.

So, the result of these calculations is different:

```
5 * 2 + 10
5 * (2 + 10)
```

If you want the addition to happen first, you need to put it inside brackets.

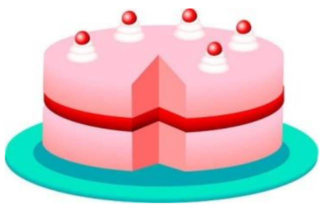


Programming Task
Write a script that asks a user for a number. The script should add 3 to that number. It then multiplies the result by 2, subtracts 4, subtracts twice the original number, adds 3, then prints the result.

Logical Operators
In programming, we also have some other operators to contend with, the logical operators which are NOT, AND and OR. They are evaluated in that order.

An easy way to remember this is to imagine you are a small child and there is cake and tea!

1. Mum says you can NOT have cake
2. You want to have cake AND tea
3. She lets you have cake OR tea



Mmm...cake.

Evaluate the following expressions. If operators have equal precedence, the leftmost is first:

1. x= 17 / 2 * 3 + 2
2. x = 2 + 17 / 2 * 3
3. x = 19 % 4 + 15 / 2 * 3
4. x = (15 + 6) – 10 * 4

f) Evaluating expressions

When we use a *selection* (if statement), we are asking the computer to evaluate whether a *condition* is true or false. You may be asked to evaluate a condition yourself, in the exam.

1. Assuming that the variables x and y have the values stated, evaluate the following statements to decide if they are true or false. One is done for you as an example:

x	y	Condition	True or False?
5	10	x > y	False
5	10	x < y	
5	10	x <= y	
5	5	x == y	
5	5	x >= y	
5	5	x > y	
3	1	x != y	

2. Now try these slightly more difficult ones – they use the Boolean operators AND, OR and NOT (the ! sign)

x	y	Condition	True or False?
5	-	x > 0 and x < 10	
5	-	x < 0 and x > 10	
5	10	x > 0 and y < 10	
5	10	x > 0 and y <= 10	
1	2	x > 1 or y > 1	
1	2	x >=1 or y >=1	
1	2	x <= 0 or y < 0	
Fred	Bob	x != "Jane" and y != "Jane"	
1	2	x == 1 or y == 1	

3. These are really difficult, see if you can figure them out!

x	y	z	Condition	True or False?
True	False	5	y == False and (z < 10 or y == True)	
True	False	5	(x == True and y == True) or z != 6	
True	False	5	(x == True and y == True) and z <= 6	
True	False	5	(x == True and z < 10) or (y == True and z > 10)	

g) String manipulation functions

Slice a string
It is possible to do all sorts of things with strings you are given. Imagine you are writing a program where you want to find out the first letter of each string. Have a look at this code which has been typed into the Python shell:

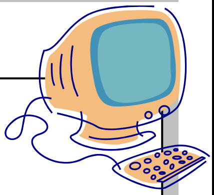
```
>>> meal="spam and eggs"
>>> meal[5:8]
'and'
>>> meal[3:]
'm and eggs'
>>> meal[:5]
'spam '
```

1. Describe in your own words how the slicing operator [start:end] works.

2. What effect does leaving out the first number have on the slice (e.g. `meal[:5]`)?

3. How would you specify a slice that prints 'egg'?

4. So how would we use this to find out the first letter of any given string?



Programming Task
Write a function called `firstLetter` which has one parameter called `givenString`. The function should print (or return) the first letter of any string it is given as an argument.

Length of a string

People often ask “how long is a piece of string” as an impossible-to-answer question. Happily, we have a really simple function to tell us how long a Python string is!

```
1     meal = “roast dinner”
2     print len(meal)
>>> 12
```

As you can see, we have defined a string variable called meal on line 1, and then used this variable as an argument for the len() function on line 2. We can see the result – the string is 12 characters long.

You can also call the len function with an actual string as an argument, e.g.

```
print len(“Curry”)
>>> 5
```

Finding characters in a string

Sometimes you might want to know whether a particular character or characters (known as a **substring**) is in a string. Use the find() method which will tell you the position of the first occurrence of your substring (needle) in the string you are searching in (haystack).

```
      haystack = “abcde”
      needle = “b”
      position = haystack.find(needle)
      print “Found it at ”,position
```

This code will print out “Found it at 1” because b is the 1-th letter (remember we start at 0).

If the needle is not found in the haystack, -1 will be returned, as in this example:

```
      haystack = “team”
      needle = “i”
      position = haystack.find(needle)
      print position
```

You obviously don’t have to call your variables needle and haystack for it to work, but it might help you to remember which variable contains the item you are searching for, and which variable contains the data you are searching in!

Comparing strings

We have already used these abilities a lot in checking whether strings are equal or not equal to each other. For example:

```
      if “apple” == “banana”:
            print “That’s weird....”
```

Surprisingly, we can also use the >, <, >= and <= operators on strings. More on this in the next section...

Concatenating strings

Concatenating is a big word. Essentially it just means adding up, but for strings. Somewhat logically, we can concatenate strings using the + operator too!

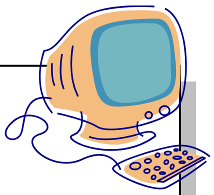
```
rainbow = "red"+"orange"+"yellow"+"green"+"blue"+"indigo"+"violet"
```

If you printed this, the output may not be what you had expected:

```
      redorangeyellowgreenblueindigoviolet
```

If you wish to have spaces, you must put them in!

```
      rainbow = "red "+"orange "+"blue etc."
```



Programming Task

Write a procedure in python code that takes three arguments -

doSlice(start, end, theString)

It should take two numbers (a start and an end) and a string and it should print out a slice of theString which begins at start and ends at end.

h) Relational operations on alphanumeric strings

We know from the previous section that we are allowed to use all of the relational operators (==, !=, >, <, >=, <=) on strings. Whilst the uses of equal and not equal are obvious, it may not be as obvious why we may want to use the greater than and less than operators.

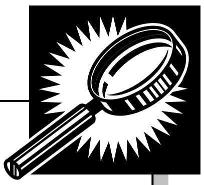
We have seen this before:

```
name = "laura"
if name == "Laura":
      print "Come in!"
```

A human reading this code would be able to recognise that "laura" and "Laura" are meant to mean the same thing, but a computer cannot, and so this expression evaluates to False.

Computers have absolutely no idea what a letter "l" is – to them it's just a number – the decimal number 108 in fact. The character "L" is decimal number 76. It should now make much more sense why "laura" is not equal to "Laura". (There is a whole list of ASCII character codes right at the back of this booklet.)

Use this character code sheet to figure out the answers to these questions:



Why is 'XYZ' < 'abc'?

Why is '2' > '17'?

Why is '3' <> '3.0'?

i) Input and validate data

It is really useful to be able to ask the user to type in some data which will be stored and perhaps used later on in your program. Here is some Python code which does exactly that:

```
name = raw_input("Please enter your name: ")
print name
```

This code:

- Creates a variable called `name`
- Gives the instruction "Please enter your name: "
- Waits until the user types something in, and then stores it inside the variable `name`
- Prints out the value of the variable `name`

Which data type do you think name will be?

Use the raw_input() function to try to read in an integer, for example:

```
age = raw_input("Please enter your age: ")
type(age)
```

Run your program and input your age.

Which data type do you think age will be?

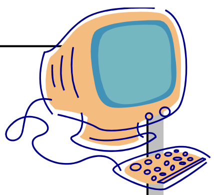
Discuss with a friend
Why do you think it is a problem that numbers typed in by the user are seen by Python to be strings?

Casting
If we want to be able to use numbers typed in from the raw_input() function in calculations, we must make sure Python sees them as numbers and not as strings.

To convert (or *cast*) to an integer, we can add this function to the code:

```
age = raw_input("Please enter your age: ")
age = int(age)
```

Incidentally, you can also convert to `float()` or even back to `str()` ...



Programming tasks

1. Write a program that asks two people for their names; stores the names in variables called name1 and name2; then says hello to both of them.

2. Write a program to take an amount of money in Pounds Sterling (£), convert it to Euros (to do this, divide it by 0.68), and then output both values.

3. Write some **pseudo code** for a program which asks the user to enter a username. The program should then check whether the username is valid.

The criteria for valid usernames are:
- The username must be longer than 1 letter
- The username must be shorter than 10 letters
- The first character in the username must be a letter

Use page 67 of the OCR book to help with the pseudo code.

4. Try to convert your pseudo code from part 3 into real Python code. You may need to use the `.isalpha()` method.

Validation

It is usually a good idea to do some form of check on the data the user types in, to ensure that it is in a suitable format for the program – we have done this in many programs but perhaps without you realising ☺

Validation can **never** 100% ensure that the data is correct, however it can make sure it is reasonable, sensible and in an acceptable format.

Some functions you can use to help you validate data are:

`len()` - to check the length of what has been entered
`str(), int(), float()` – to change the data type of what has been entered
`.isalpha()` – to check whether the string is alphabetical (i.e. no numbers)
`.isdigit()` – checks whether the string contains digits
`.isupper()` – checks whether the string is uppercase
`.islower()` – checks whether the string is lowercase

`.isdigit()` checks whether a STRING contains digits. This does NOT mean it is a number, it is a string containing digits e.g. "123"

j) Output data

We already know several ways to output data to the screen:

Printing

Concatenation

Format operator

How else might we be able to output information to the user?

5 - Writing maintainable programs

a) Programming terms: variable, constant, identifier, reserved word/keyword

Make sure you are familiar with all of the following definitions, and are able to apply them correctly to examples of code.

Word	Definition	Example
Variable		
Constant		
Identifier		
Reserved word (Key word)		

b) Program writing techniques

See how many methods you can think of that will make a program easier to maintain:

Ways to make a program easier to maintain

Example exam question

Good programmers always make sure they use helpful names for their variables. State two other things you could do to make your program easy for another person to understand. [2]

Computer task

Write a guide to good programming practice. Your guide should include and refer to:

- Initialising variables
- Selecting variable (identifier) names
- Indentation
- Formatting
- Comments
- Constants

c) Variables, constants and scope

Quick Quiz

Decide whether the statements below relate to a variable or a constant, and write V or C in the boxes next to them.

	A name used to refer to a particular memory location that is used to store data.
	A name used to refer to a fixed value.
	The main purpose is to make code easier to read and maintain.
	The value is set when the code is written and cannot be changed while the program is running.
	The value can change while the program is running.
	The main purpose is to allow the programmer to write code that manipulates the data, without knowing yet what the data will be.

Scope

Variables can be either **global** or **local** in scope. A global variable can be used anywhere in the code, including inside subroutines. A local variable is created within a subroutine. It only exists when that subroutine is called, and it is destroyed once the subroutine ends.

To help you imagine what this means, think of the school rules – they are **global** because they apply everywhere in the school, including all of the year groups. In addition, some year groups have **local** rules – rules which apply within that year group only.

Here is some code with both global and local variables. For each variable, say whether it is global or local:

```
target = "Bonjour"
letter = raw_input("Enter a letter: ")

def findChar(needle, haystack):
    for char in haystack:
        if needle == char:
            return True

    return False

result = findChar(letter,target)
```

Global or local?

target ___________________

letter ____________________

needle ___________________

haystack _________________

char ____________________

result ___________________

d) Identifier names and conventions

Good programmers will always use meaningful identifier names, because it makes the code easier to read. If you don't believe me, here is exactly the same function from the previous page, but with ridiculous variable names:

```
orange = "Bonjour"
grapefruit = raw_input("Enter a letter: ")

def findChar(banana, apple):
    for pear in apple:
        if pear == banana:
            return True

    return False

blueberry = findChar(grapefruit,orange)
```

confused.com

Confused?? This is what your code may be like to another person (or to you, if you don't look at it for a few months) if you don't pick your variable names sensibly.

Conventions

These are the ways of naming variables which are commonly used, and which you should pick your favourite from to use, but never mix and match two types within the same program.

camelCase	if an identifier has multiple words, all words after the first begin with a capital letter. e.g. getResult, numberOfTimesPrinted etc.
PascalCase	Almost the same, except the first letter is also capitalised. e.g. GetResult, NumberOfTimesPrinted etc.
Hungarian	A prefix is added to the start of the variable name to denote the data type e.g. intAge, strName etc.
Underscore	Words are separated using underscores, all letters are lowercase e.g. get_result, number_of_times_printed

When identifiers go bad...

Some identifier names you simply can't use, and some you shouldn't use because they are confusing.

You **can't use** any identifier names that are key words (e.g. if, else, def, while, for, import etc.)

You **shouldn't use** identifier names that are single letters (e.g. a, b, c) UNLESS you are using i or j as loop counters, or f as a file pointer.

e) Declaring constants

Declaring constants in Python is oddly simple:

```
VAT_RATE = 0.20
buy_price = 4.00
tax_paid = buy_price * VAT_RATE
```

In the example above, VAT_RATE is a constant, and all other identifiers are variables. We know that VAT_RATE is a constant because it is in capital letters.

Quick Questions

1. What is a constant?

2. How is a constant different to a variable?

3. Why do we bother with constants?

f) Initialising variables

We are lucky that in Python, when we create a variable we must also use an **assignment** statement to give it a value. This is called **initialising** a variable - rather than simply assuming it has a particular value (e.g. assuming an integer defaults to 0).

The Python interpreter won't let you be naughty and not initialise your variables, but you can slip up and forget to do this if you are writing pseudo code. Make sure in the exam that if you use a variable in your code, it is initialised – preferably at the start of the program if it is a global variable.

g) Creating modular programs

Because we are all very well behaved programmers and have learnt that designing programs with a **top-down modular** approach is a sensible idea, it will make sense to split our programs into smaller parts or **modules**.

All this means is that we should endeavour to break our program down into functions which do specific tasks, rather than having a Mary Poppins bag of a program which contains absolutely everything.

Mary Poppins – good nanny, bad programmer

h) Commenting code

Perhaps the single most useful thing you can do to help other people understand your code is to add comments to it. A comment is a written line which helps to explain what the code is doing. It is entirely for the benefit of humans reading the code, as it is completely ignored by the computer.

To add a comment, simply put a hash (#) symbol, followed by the comment:

```
# prints a greeting
print "Hello!"
```

Of course, this comment is entirely pointless as it is obvious what the code below it does. You do not need to comment every single line, only in places where there may be ambiguity.

Read p116-17 of the OCR textbook for an example of correct commenting style.

You should use sensible comments in all code you produce for this course, and you should encourage others in the class to do the same. From time to time we will swap code and critically evaluate the code others have written.

i) Indentation and formatting

Indentation is the need to use tabs and/or spaces to show where different parts of the code begin and end, for example indenting lines of code which should happen within an if statement.

Once again, we are lucky that Python forces us to indent properly and will complain if we don't – some programming languages don't do this!

Formatting exercise

1. Write the following code out again, with proper indentation, comments and appropriate variable names.

```
Keep talking = True
input = raw_input("Talk to me! ")
while Keep talking == True:
input = raw_input("That's interesting, talk some more! ")
if input == "bye":
print "Cya later!"
Keep talking = False
```

2. What does this code do?

6 - Testing and running a solution

a) Errors -syntax, logic and run-time

Now that you are writing larger programs, you will encounter more errors. Getting rid of errors is called **debugging**. There are three types of error:

Syntax error –

Logical error –

Runtime error –

See if you can categorise the following problems into the three types of error. Write the number in the table below.

1. Missing a colon at the end of an if statement
2. Running out of space to execute a loop because it runs so many times
3. Printing two lines in the wrong order
4. Not indenting something you wish to only happen as part of an if statement
5. `5 = age`
6. Writing a condition as being >0 when you wanted 0 to be valid
7. An infinite loop
8. Using an else without an if
9. Calling a function but missing one of the arguments
10. Writing a calculation wrong, e.g. mean = total / count instead of mean = count / total
11. Dividing by zero
12. Importing a library (a library is like import random) that does not exist
13. Trying to store a variable that is too big (e.g. 989048329478392742389478923432)
14. Using the wrong keyword, e.g. using foreach in Python, when this is a PHP keyword

Syntax	**Logical**	**Runtime**

b) Identifying why/where an error may occur in an algorithm

Having errors in your program can be incredibly frustrating. However, you can minimise the amount of errors you encounter in two ways:

When you are writing code:

- Write a small chunk of the program
- Think about what *should* happen when you run the program
- Run the program, and check if what you expected happens
- If it works, add on another chunk
- Else debug
- Repeat until you have written the whole thing!

What to do if your program doesn't work...

- ALWAYS fix the first error message first – this error may be causing the others!
- Only fix one problem at a time
- READ THE ERROR MESSAGE and think about what it is saying
- Print values – print out the values of variables at relevant places, to check whether things are happening as you expected. If they are not, go back in your code until you find out why!
- Comment bits out – put a # in front of lines you think might be dodgy to try and find the place where the program goes wrong

I have lost count of the people who have thrown their hands in the air, whined
"MISS!! MY PROGRAM DOESN'T WORK!!"
... and yet they have not even bothered to read the error message given to them!

In the exam, you may be given a piece of pseudo code which contains errors and asked to debug it. Here is an example of the type of exam question you may encounter on this topic:

The design for a game contains the following pseudo-code:

```
IF Character has reached the end of the platform
   Display "You Win"
   REPEAT
      Play music
END IF
   UNTIL any key pressed
```

i) Explain why this pseudo code contains an error [2]

ii) State the type of error the pseudo-code contains and when the error would be detected if implemented [2]

Type of error:

When detected:

iii) Name and describe one other type of error which can occur in a program, stating when it would be detected. [3]

c) Testing strategies including white box testing, black box testing, alpha testing, beta testing and acceptance testing

Programmers use many different strategies to try to eliminate as many errors and bugs as possible from their programs.

White box testing

Black box testing

DIY: White box testing

Look at this program which inputs a year as an integer and outputs whether or not it is a leap year.

```
01    BEGIN PROGRAM
02    INPUT Year
03    IF Year is not an integer greater than 0 THEN
04       OUTPUT "Invalid input"
05    ELSE
06       IF Year is divisible by 100 THEN
07          IF Year is divisible by 400 THEN
08             OUTPUT "LEAP YEAR"
09          ELSE
10             OUTPUT "NOT A LEAP YEAR"
11          END IF
12       ELSE
13          IF Year is divisible by 4 THEN
14             OUTPUT "LEAP YEAR"
15          ELSE
16             OUTPUT "NOT A LEAP YEAR"
17          END IF
18       END IF
19    END IF
20    END OF PROGRAM
```

We can **white box test** this algorithm using several items of test data.

Input data: 2000

Path of Execution: 01, 02, 03 (FALSE), 05, 06 (TRUE), 07(TRUE), 08, 11, 18, 19, 20

Output: LEAP YEAR

Now you try it with these inputs:

Input data: cheese

Path of Execution:

Output:

Input data: 2009

Path of Execution:

Output:

Alpha (α) testing

Beta (β) testing

Acceptance testing

Example exam question

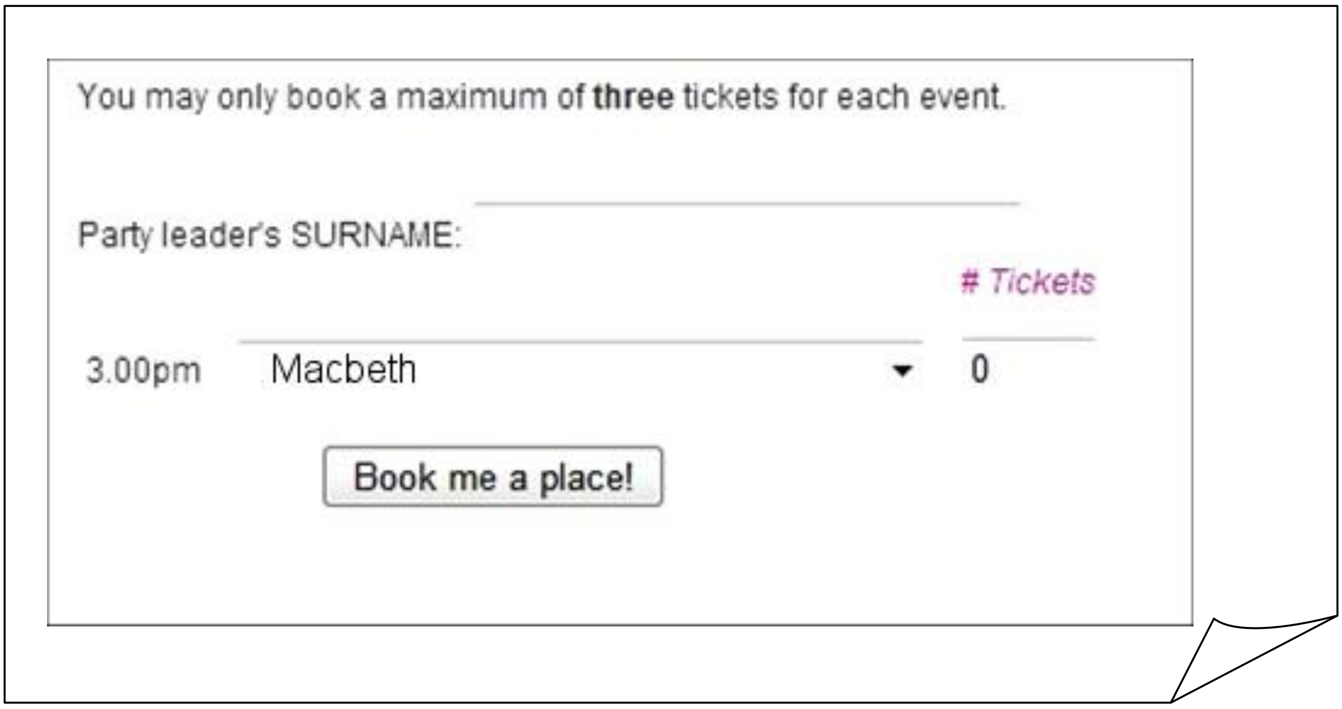
The program is tested using beta-testing and acceptance testing. Explain the difference between beta-testing and acceptance testing. [4]

d) Test data - normal, borderline and invalid data

Testing a system is important because it ensures:

- ...that the software meets the design specification
- ...that users have faith in the system
- ...that the system actually works!

You have been asked to produce a test plan for part of a ticket booking website. Below is the screen where parents can book their tickets. Using the test plan template on the next page, devise seven tests you could use to test whether this part of the system is working correctly.



You may only book a maximum of **three** tickets for each event.

Party leader's SURNAME:

# Tickets

3.00pm Macbeth 0

Test Plan for ticket booking website

Test #	Description of test	Type of test	Data used	Expected result

Normal data

Extreme data

Erroneous data

e) Dry running algorithms

Dry running an algorithm is where you go through a program line by line as if you were the computer, noting down the values of variables at each stage and the actions that are taken in a **trace table**.

Here is an algorithm which calculates the largest common factor of two integer numbers. See if you can complete the trace table when X = 24 and Y = 30.

```
01    INPUT X, Y
02    WHILE X > 0
03       N = X
04       X = Y MOD X
05       Y = N
06    END WHILE
07    OUTPUT Y
```

Line	N	X	Y	Output	Comment
01 INPUT X, Y		24	30		

1. Turn to p86 in the OCR textbook and look at exercise 2b i). Create a trace table for this algorithm using the inputs given (A = 8, B = 2).
2. Create a trace table for exercise 3 Module 1 where X = 1.
3. Create a trace table for exercise 3 Module 1 where X = 3.

f) Debugging tools - translator diagnostics, break points, stepping, and variable checks

Debugging

There are various ways the computer can help the user to find bugs in a program. The most common one is called (1)_________ (2)___________. This is the proper name for the messages which pop up when the user attempts to run the program. It is particularly useful for finding (3)________ errors as the program cannot run if one of these is present. It may also help with finding logical errors, for example if a (4)_______ is used but is not initialised.

Programmers can also do a (5)___ (6)____ on their program. This is where they write down a table which contains each line of the code, and the (7)______ of the variables at each point. It is then easy to see if the logic of the code is working as intended.

A way of getting the computer to do this is called using break points. A break point is when a marker is added to a line of code, and the program will (8)_____ when it gets to that line. The programmer can then (9)_____ through the code line by line just as in a dry run. You can also do variable (10)______ along with break points, either stopping the code when the (11)_____ of a variable changes, or simply looking at the values of the variables when the (12)____ (13)______ is reached and the code stops (14)_______.

g) Installation routines

Installation
The code that we write in Python and other languages is called **source/object** code. The computer does not understand this code so it needs to **translate/alter** it into **source/object** code. The programmer can do this because he has the correct program on his computer, e.g. we have the Python translator.

Most **programmers/users** do not have a **translator/compiler** program on their computer. This means that for the program to run successfully on a different computer it must be translated into a file which can be **executed/translated** on its own – sometimes this is an .exe file. This might include a particular file structure, and any **documents/libraries** which may be required need to be included too, for example the program may import random. This is called **installing/copying** the program on the user’s computer.

Sometimes there is a lot of work to do if there are a lot of parts to the program. In this case we need an **installation routine/experienced programmer**. This will **translate/copy** the executable program to the user’s computer, make sure that any data files are in the correct location and configure shortcuts such as **windows/icons**. Many video games use this method of **corruption/installation** in order to ensure the game is set up correctly.

Assignment 1

Due in: ....

Python program building blocks

Create a poster with details of the blocks you know how to do so far:

- Assignment - making a variable and giving it a value
- Sequence - instructions that are run one after another
- Selection - deciding what instructions to run - if, else, elif
- Iteration - repeating instructions a fixed amount of times - while loop
- Input - getting information from the user - raw_input function
- Casting - changing the data type, e.g. int(), float()

For each building block, your poster should include information on

- Name of block
- What it does
- A working, **commented** example of what it can be used for - check the example works by running it on the Python shell

Mark:

Comment:

Assignment 2

Due in: ....

a) Programming constructs determine the way in which statements in a program are executed. Three types of programming constructs are sequence, selection and iteration. Describe what is meant by each of these.

Sequence

....

....

....

....

[2]

Selection

....

....

....

....

[2]

Iteration

....

....

....

....

[2]

b) A computer program contains the following instructions

```
X = 5
Y = 7
X = Y
OUTPUT X
```

(i) State which of the constructs in part (a) has been used.

....

[1]

(ii) State the value which will be output.

....

[1]

Mark:

Comment:

Assignment 3

Due in: ....

Writing a simple calculator program

1. Set up your program with comments so that there are two clear sections - the functions section and the program section.

2. In the functions section, write functions to perform all of these tasks:
    - Add two numbers
    - Subtract one number from another
    - Multiply two numbers
    - Divide one number by another

3. In the program section, set up three variables – **firstNumber**, **operator** and **secondNumber** and allow the user to type in values for each of these variables. (The operator is the action they want to perform, i.e. +, -, / or *)

4. Set up a selection statement so that if the user enters "+" as the operator, the function add() is called, if the user enters "-" as the operator, the function subtract() is called etc...

5. If the user does not enter one of +, -, / or * display a sensible error message and end the program

The hard bit – not compulsory but extra Brownie points!

6. If you know that your input is going to be a number, but you don't know what type of number, try using the **input()** function instead of the **raw_input()** function. The **input()** function will read in an integer with an integer data type, a float with a float data type etc. but it will cause a problem with strings.

7. Add a loop so that the program keeps asking you to enter sums instead of asking once and then finishing. Hint: you will need to make a variable called **endProgram** which begins as False

Mark:
Comment:

Assignment 4

Please do this on paper or on a Word Processor. You may use Python to help you figure out the answer, but **DO NOT** write the answers in Python.

1. What is the difference between a procedure and a function? [2]

2. For the following examples, state for each whether it is a procedure or a function, and explain why [2]

Example a

```
def add(x,y):
        a = x + y
        print a
```

Example b

```
def multiply(t,u):
        v = t*u
        return v
```

3. What is the difference between defining a function and calling a function? [2]

4. What is wrong with this code, and how should you fix it? [2]

```
print_lyrics()

def print_lyrics():
        print "Hello hello baby you called, I can't hear a thing"
```

5. How many times can you define the same function in a program? [1]

6. How many times can you call the same function in a program? [1]

7. What is the argument in this function, and why is it needed? [2]

```
def canDrink(age):
      if age >= 18:
            return True
      else:
            return False
```

8. A void function is a function which does not return anything (also known as a procedure). What data types do the following functions return? [3]

```
# Function a
def isEven(number):
      if number % 2 == 0:
            return "Even"
      else:
            return "Odd"

# Function b
def divide(x,y):
      z = x / y
      return z

# Function c
def divide(x,y):
      x = float(x)
      z = x / y
      return z
```

The hard bit – not compulsory but extra Brownie points!

9. `rollDice()` is a function which returns a random number between 1 and 6. It is being used here inside a different function. Assume `rollDice()` is already defined and that the random library has been imported. Why doesn't this code work? [1]

```
def luckySixes():
      rollDice()
      if number == 6:
            print "Winner!"
      else:
            print "Sorry, try again"
```

10. Here is a function to roll two dice, also using the same rollDice() function from before.

```
def rollTwice():
      roll1 = rollDice()
      roll2 = rollDice()
      return roll1 + roll2
```

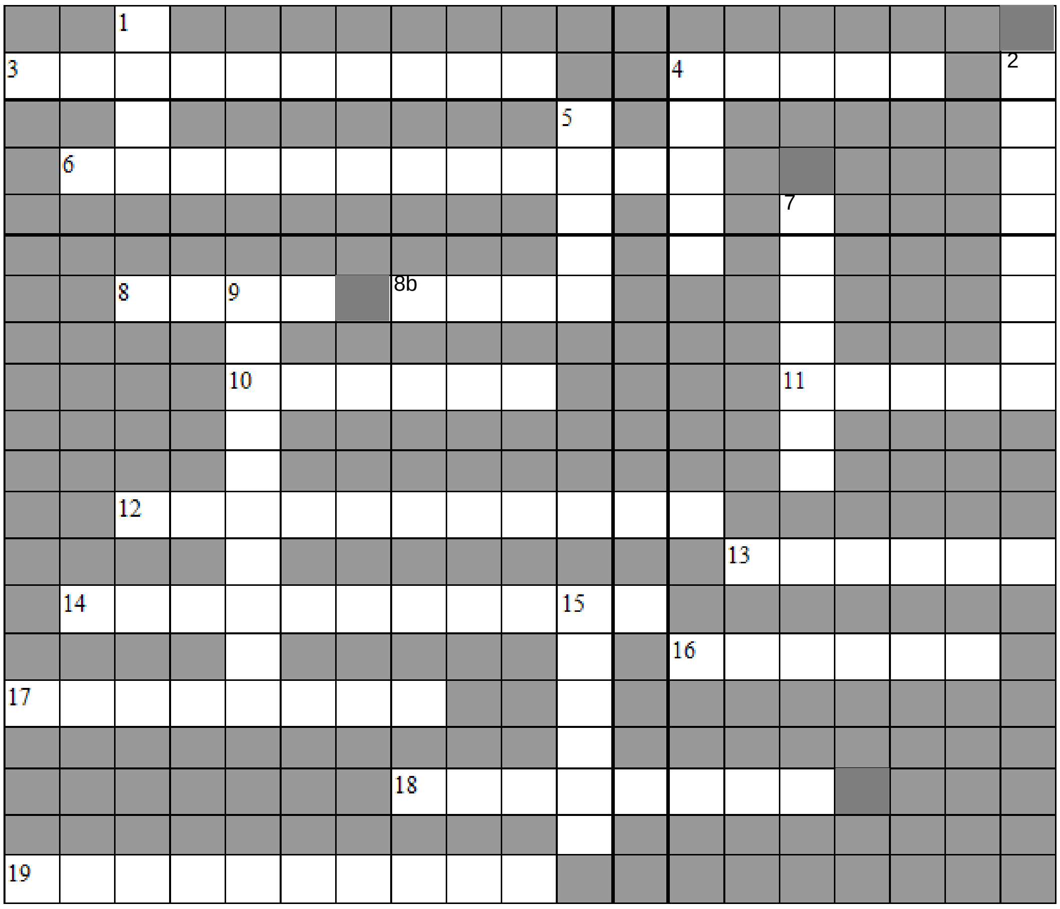
a) What arguments are used here [1]

b) How would we call this function? [1]

c) Re-write this function to do the same thing without using any variables? [1]

d) What data type does this function return? [1]

Computing Crossword 1



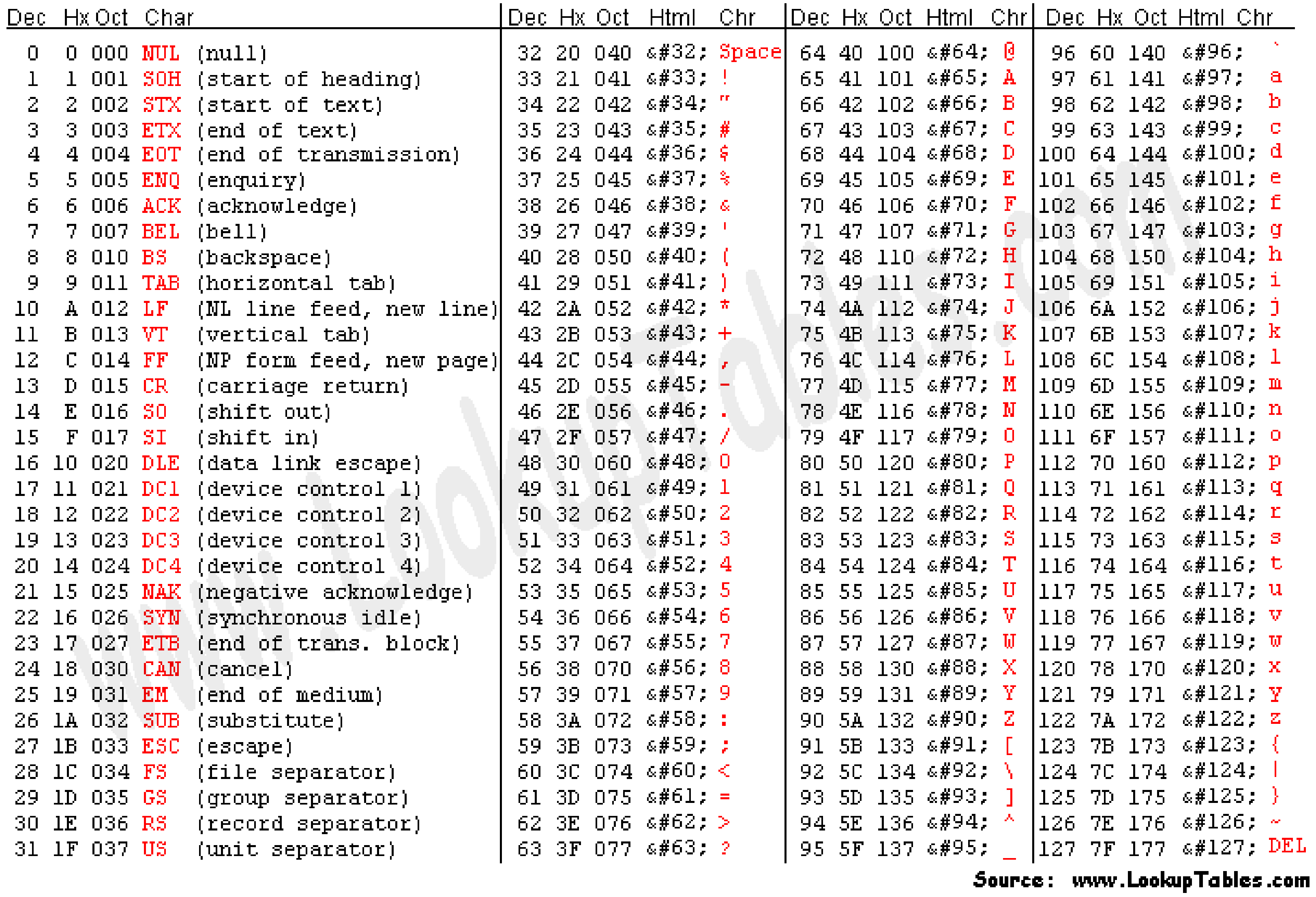
ACROSS

3. The name of a variable can also be called (10)
4. Error occurring when the programmer has not written what they intended to write (5)
6. Textual data such as characters, digits and punctuation (12)
8. A file access mode (4, 4)
10. Error occurring when a part of the program breaks the rules of the language (6)
11. The position of each item in the array (5)
12. Join two strings together (11)
13. A variable that is available throughout a program (6)
14. The layout of the code which allows different parts to be distinguished (11)
16. The code the programmer writes (6)
17. A fixed value within a program (8)
18. Testing concerned with the inputs and outputs of the program (5,3)

DOWN

1. A floating point number (4)
2. Testing concerned with the workings of the algorithms in the code (5,3)
4. A variable that is only available within a particular module or subroutine (5)
5. Contains several items of data (5)
7. Error occurring when data being processed in the program is not as expected (3,4)
9. Giving a value to a variable (10)
15. Type of code which is understood by the computer (6)

Appendix A – ASCII Character Codes



Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH	(start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX	(start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX	(end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT	(end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ	(enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK	(acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL	(bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS	(backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB	(horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT	(vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR	(carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO	(shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI	(shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE	(data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1	(device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2	(device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3	(device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4	(device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK	(negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN	(synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB	(end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN	(cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM	(end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB	(substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC	(escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS	(file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS	(group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS	(record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US	(unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Mark Sheet

Assignment	Mark	Comment

u unnerstand internetz nao?
or need moar teechings?
ICANHASCHEEZBURGER.COM