Python 3 Programming

OCR GCSE Computing

<u>Contents</u>

Introduction

Learning to program takes time and lots of practice but more importantly should be fun. On the training day we will be covering sections of this booklet but it would be unrealistic to expect to learn everything over a one day period.

After the training day go back over this booklet. If you are unsure I would recommend starting from the beginning and working through all the tasks. It is one thing to read through the exercises but this is no substitute for doing them yourself. Not only does this help embed the concepts being covered but also helps you get a feel for the sort of errors your students may come across. Each section contains three tasks a, b and c, each of increasing difficulty.

Once you are comfortable with the material in this booklet it is hoped you will have a strong enough grounding to develop your programming skills in the direction you want.

Remember your trainers will be available to contact after the day itself so please do not hesitate to email them if any questions arise.

Tip: If you are finding yourself stuck on a programming task it often pays to go and do something completely different for an hour. Watch TV, mow the lawn, walk the dog anything to take your mind off the problem. It's amazing how many silly mistakes you will spot when reviewing your code with a fresh pair of eyes.

Python
The OCR GCSE in Computing can be carried out using any (imperative) language. The fact this course is in Python isn't to say it is any better than the many alternatives out there but it is certainly a good language to teach with.

Python was designed specifically to be easy to understand and quick to build programs. It is used in real world companies such as Google where its creator Guido Van Rossum now works.

There are currently two versions of Python in use Python 2 and Python 3. Whilst they are both very similar there are significant differences. In this course we will be looking at Python 3.
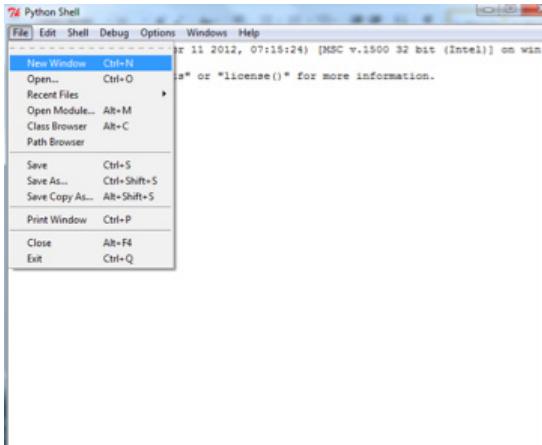
Conventions Used
In this booklet the code and the output of programs will be displayed in Courier New font `like this.`

IDLE

IDLE is the graphical user interface that comes with Python.

Once inside the IDLE shell select File and New Window to open the text editor.

From here you can enter your program code.

TIP: When saving your file remember to add the .py extension to your file. It will still run without it but by adding this, IDLE will use syntax highlighting which will make your code easier to read.
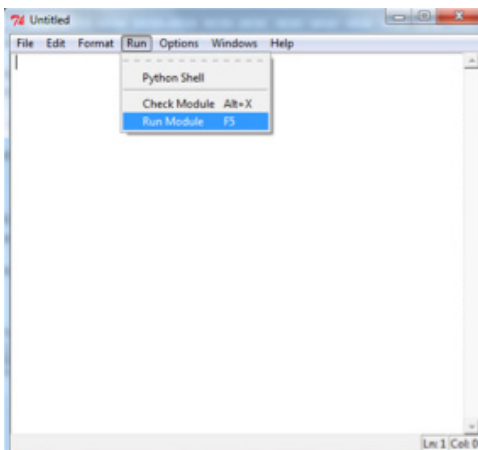
To run your code select **Run --> Run Module.**

1. Output to the screen

Hello World

All programmers start with the program hello world. It's very much of a rite of passage.

Task 1a

Copy and run the following program in Python.

```
print('Hello World!')
```

You will hopefully see the message `Hello World!`

Task 1b
Change the program so it says `Pleased to meet you.`

Task 1c
Write a program that says:

```
This is
a computer program
that prints on several lines
```

Challenge Exercise
Write a program to say

```
This is John's first program
```

The apostrophe makes this trickier than it first may seem (**Hint:** Find out about escape characters in Python).

2. Storing Data in Variables

We use variables in our programs to store data. Each variable has a name and stores data of a certain type (string, integer, real etc). In Python we do not have to explicitly state the data type being stored, it will work this out for itself when the variable is first used. In this section we will just use string variables. String is the programming term for text (i.e. letters, numbers and symbols).

Variable names can contain letters and numbers but should always start with a letter.

We assign a value to a variable using =

Task 2a
Copy and run the following program:
```
name='Bob'
print('Hello '+name)
```

Task 2b
Start a program with:
```
language='Python'
print('I am learning to program in' [REMAINDER OF PROGRAM HERE]
```

Complete the program so it uses the variable language to print
```
I am learning to program in Python
```

We can change the value of a variable at any point. The code:
```
name='Bob'
name='Liz'
print('Hello '+name)
```

Will print `Hello Liz` (as the variable name has been overwritten with the value Liz).
We can also change the contents of a variable referring to itself:

```
name='Bob'
name=name+'by'
print('Hello '+name)
```

This will print `Hello Bobby`
Whereas
```
name='Bob'
name=name+name
print('Hello '+name)
```

Will print
```
Hello BobBob
```

We can add spaces just like any other letter:

```
name='Bob'
name=name+' '+name
print('Hello '+name)
```

Will print

```
Hello Bob Bob
```

Task 2c
Complete the following program so it uses the variables to print out `the cat sat on the mat.` Remember you will need to add spaces.
E.g.

```
word1='the'
word2='cat'
word3='sat'
word4='on'
word5='mat'
[REMAINDER OF PROGRAM HERE]
```

3. Inputting Data

We can now look at starting to make our programs interactive. We are going to take in input from the user using the `input` function.

```
name=input('What is your name?')
print('Hello '+name)
```

Task 3a
Write a program that asks for the town you live in and then replies I love visiting *[town name]*

e.g.
```
Where do you live?  London
I love visiting London
```

Task 3b
Write a program that asks for your first name, then asks for your last name and finally greets you with your full name.

e.g.
```
What is your first name? John
What is your last name? Smith
Hello John Smith
```

Task 3c
Write a program that asks for your name then prints it out 5 times

e.g.
```
What is your name? Trevor
Trevor Trevor Trevor Trevor Trevor
```

4. Calculations

We can carry out calculations in Python. The arithmetic operators we use to do this are:
+ addition
- subtraction
* multiplication
/ division

Task 4a
Copy and run the following program
```
print(1+6)
print(7-5)
print(3*9)
print(7/2)
```

Task 4b
Try changing the calculations to new ones.

We can put the results of calculations into variables. These variables will not be string but integer (whole numbers) or float (decimal).

```
a=10
b=2
c=a/b
print(c)
```

Task 4c
Complete this program so it uses addition on the two variables make c equal to 15 then prints it.

```
a=7
b=8
c= [REMAINDER OF
 PROGRAM HERE]
```

5. Data Types

Every variable has a given data type. The most common data types are:

String - Text made up of numbers, letters and characters.

Integer - Whole numbers. (e.g. 1, 78, 0 and -54)

Float - Decimal numbers (e.g. 3.5683, 98.74634, -6.3). Float comes under the umbrella of what we often refer to as *Real* numbers.

Boolean - True or False

In some languages we have to tell the computer what data type a variable is going to be. Python, on the other hand, is able to decide the data type of the variable according to what value it is first given (or to use the correct term what it is *initialised with*). This can cause problems when we try to do something with the wrong data type.

Task 5a

Copy and run the following program.

```
a=input('Enter number 1:')
b=input('Enter number 2:')
c=a+b
print('Adding your numbers together gives:'+c)
```

If you enter the numbers 5 and 4 it will output 54. This is because Python treats anything received through the input function as a string. We need to tell Python we want to convert this string to an integer before putting it into the variable. This is done using *type casting*.

To cast data to an integer we use `int()`

Try the program now:

```
a = int(input('Enter number 1:'))
b = int(input('Enter number 2:'))
c=a+b
print('Adding your numbers together gives:'+c)
```

This time you will get an error in line 4:

TypeError: Can't convert 'int' object to str implicitly

This is because a and b are now integers. This means c is now an integer (as it takes in the sum of to integers which in itself is an integer). The problem arises when we then try and add c to a sentence: 'Adding your numbers together gives:' We can concatenate strings with strings but in Python we can't concatenate strings with integers.

The solution is to cast c back to a string when we use it using str().

```
a = int(input('Enter number 1:'))
b = int(input('Enter number 2:'))
c=a+b
print('Adding your numbers together gives:'+str(c))
```

Now the program should work.

Task 5b

Write a program that asks for a length and width and outputs the area of a rectangle.

E.g.
```
Please enter width: 9
Please enter height: 5
The area is: 45
```

Task 5c

The formula for the volume of a cylinder is: $\pi r^2 h$

And for its surface area is: $2\pi r^2 + 2\pi r h$

Where r is the radius and h the height. π is the value 3.14159 (5d.p.)
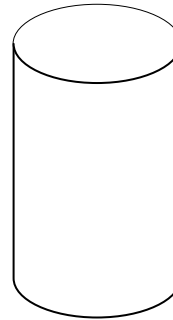
Assume that the user may enter the radius and height as real numbers. To cast these you will need to use `float()`

Write a program that asks you for the radius and height of a cylinder then calculates the volume and area.

6. Selection with IF

Sometimes we only want a program to execute code under certain circumstances. We call this *selection*. The most commonly used way of doing this is using if. Here we say *if* a condition is true execute some code.

First we need to make sure we are happy what a condition is. A *boolean condition* is one that can have only two values **true** or **false**.

The boolean comparison operators are as follows:

Operator	Meaning
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

So:
```
letter = 'a'
if letter=='a':
    print('This prints if letter is a')
```

We can add as many statements we want inside the if. Each should be indented.
```
letter = 'a'
if letter=='a':
    print('This prints if letter is a')
    print('This prints as well')
```

When we want to stop the if we remove the indentation.

```
if letter=='a':
    print('This prints if letter is a')
    print('This prints as well')
print('This prints out whatever the letter is')
```

We can tell the computer to do something different when the condition isn't true using the `else` keyword.

```python
if letter=='a':
    print('This prints if letter is a')
    print('This prints as well')
else:
    print('This prints if the letter is not an a')
    print('As does this line')
print('This prints out whatever the letter is')
```

```python
if letter=='a':
    print('This prints if letter is a')
    print('This prints as well')
elif letter=='b':
    print('This prints if the letter is b')
else:
    print('This prints if the letter is neither a nor b')
print('This prints out whatever the letter is')
```

A Note on Indentation
When you start a new line after a colon your code (if writing it inside IDLE) you should find your code indented). Lines will keep indenting until you press backspace. Python knows that anything indented is inside the structure with the colon - in this case the if. Indenting is considered good practice in all programming languages as it makes code easier to read but in Python it is even more important as it affects the code's meaning.

For more complex conditions we can use **and**, **or** and **not**. When you are confident conditions with if should try to find out about these to make your code more efficient.

```python
x=int(input('Enter a number between 1 and 100: '))
if x>=1 and x<=100:
    print('You have entered a valid number')
else:
    print('Your number is not valid')
```

A common mistake would be to write the if line as if `x>=1 and <=100` missing out the second `x`. This is wrong and will generate an error.

Task 6a

Copy and run the code below.

```
print('What is the capital of France?')
if city== 'Paris':
    print('Well done')
elif city=='Lyon':
    print('Right country, wrong city')
elif city=='F':
    print('Terrible joke and wrong answer.')
else:
    print('Sorry wrong answer')
```

Change the question so it asks for the capital of England and gives appropriate answers to different entries.

Task 6b

Below is a program that asks for three numbers and outputs SNAP if they all match. Use your knowledge of the **and**, **or** and **not** operators to make the program more efficient.

```
one=int(input('Please enter number 1: '))
two=int(input('Please enter number 2: '))
three=int(input('Please enter number 3: '))
if(one==two):
    if(two==three):
        print('SNAP!')
    else:
        print('They do not all match')
else:
    print('They do not all match')
```

Task 6c

The grade boundaries for a test are:

U - 0
D – 40
C – 50
B – 60
A – 70

Write a program that asks for a mark and then says what grade it is worth.

7. Making Programs Easier to Read

Last section we looked at indentation as a way of making your code easier to read.
In the development section of A453 one of the descriptors for the top mark band is:

"The code will be well organised with meaningful variable names and detailed annotation indicating the function of each section."

Annotation in your code would usually be in the form of commenting. A comment is purely for the person reading it. The computer completely ignores comments but they make your program much more understandable to anyone trying to follow your code. The key to good commenting is to assume the reader is a competent programmer but doesn't know what your program is meant to do. There is a fine art to being able to strike the right balance between under and over commenting.

Commenting in Python is done using the hash # character. A comment can be put at the end of a line or on a line by itself.

```
#Question to ask user the capital of France
print('What is the capital of France?')
if city== 'Paris':
    print('Well done')
elif city=='Lyon': #Handle common wrong answer
    print('Right country, wrong city')
elif city=='F':
    print('Terrible joke and wrong answer.')
else: #any other wrong answer.
    print('Sorry wrong answer')
```

The final thing you should consider is your variable names. Names like a, b and c are fine in trivial 4 line programs but as your programs get bigger this becomes quickly confusing. Meaningful variable names help make your code much more readable.

8. Iteration - Count Controlled Loops

One of the things that makes computers so useful is the speed at which they do things. This makes them useful for when we want them to do something over and over. We call this *iteration* or looping. We will look at two types of loop, the first of which is a count controlled loop. This is when a program carries out a set of instructions a certain number of times. To do a count controlled loop we use **for**. Notice how anything inside the loop is indented.

```
for i in range(0,5):
    print('looping')
```

Will output

```
looping
looping
looping
looping
looping
```

In the loop `for i in range(x,y)` x is the starting value of i and y is the one above what it will get to. So above it runs with i equal to 0,1,2,3 and 4

Traditionally loops tend to use the variables i,j and k and you may see lots of code that conforms to this convention. There is, however, no reason why you can't use any other variable name especially if it makes your code more readable.

Task 8a
Write a program that outputs the word `computing` 15 times.

We can use the variable `i` inside the loop.

```
for i in range(0,5):
    print(str(i)+' looping')
```

Outputs

```
0 looping
1 looping
2 looping
3 looping
4 looping
```

Task 8b

Write a program that takes in a letter and number then prints out that letter that many times.

E.g.
```
Please enter a letter: T
Please enter a number: 6
TTTTTT
```

Task 8c

Write a program that asks for a number then outputs it's 10 times table.

E.g.
```
Please enter a number: 7
1 times 7 is 7
2 times 7 is 14
3 times 7 is 21
4 times 7 is 28
5 times 7 is 35
6 times 7 is 42
7 times 7 is 49
8 times 7 is 56
9 times 7 is 63
10 times 7 is 70
```

9. Iteration - Condition Controlled Loops

We have looked at count controlled loops, we will now look at condition controlled loops. These are loops which continue to repeat code **while** a condition is true.

These are exactly the same type of conditions we looked at when using `if`

```
x=0
while x<5:
    print('looping')
```

If we run the above code (don't) you will find it goes on forever printing looping. that is because x remains 0 and so is always less than 5. Let's fix it:

```
x=0
while x<5:
    print('looping')
    x=x+1
```

Now try running the above code.

Task 9a
Write a program that asks for a password and keeps asking until the correct password, `apple` is entered and then says `Accepted`

Task 9b
The sequence 1,4,9,16,25 is made up of square numbers (i.e.$1=1^2$, $4 = 2^2$, $9=3^2$ etc.). Write a program that writes out all the square numbers under 5000.

Task 9c
The following code will create an integer x that is a random number between 1 and 100.

```
import random
x=random.randint(1,100)
```

The line `import random` has to be put at the very top of the program you can use `random.randint(1,100)` wherever you want after that.

Write a program in which the computer thinks of a number between 1 and 100 (i.e. picks number at random). It should then ask the user to guess what number it is thinking of. It should then say whether the number the computer is thinking of is higher or lower than the one guessed. If the user guess correctly it should say well done and say how many guesses it took them, if not it asks them to guess again.

```
I am thinking of a number between 1-100.  Can you guess what it is? 50
No, the number I am thinking of is higher than 50.  Can you guess what it is? 80
No, the number I am thinking of is lower than 80.  Can you guess what it is? 60
No, the number I am thinking of is higher than 60.  Can you guess what it is? 64
Well done! The answer was 64 and you found it in 4 guesses.
```

10. Subroutines: Procedures and Functions

It is possible to write a whole program as one big block but where possible we want to break it down into subroutines. There are several advantages to this:

- The program becomes easier to read.
- The program becomes easier to test. If you have tested a subroutine you don't have to worry about it when you subsequently use it.
- Subroutines can often be reused meaning code does not need to be rewritten.

We will start off with a function that prints out `This is my subroutine` 3 times. Notice how the subroutines come before the main program.

```
#Subroutine that prints out sentence 3 times
def myFirstSubroutine():
    for i in range(1,3):
        print('This is a subroutine')

#Main Program
print('Start of main program')
myFirstSubroutine()
print('End of program')
```

This would output:
```
Start of main program
This is a subroutine
This is a subroutine
This is a subroutine
End of program
```

We can use parameters to make subroutines even more useful. The parameters in the subroutine below are text and times.
```
#Subroutine that prints out sentence 3 times
def myFirstSubroutine(text, times):
    for i in range(0,times):
        print(text)

#Main Program
print('Start of main program')
myFirstSubroutine('Sample text',5)
print('End of program')
```

The subroutines we have looked at so far are procedures. The other type of subroutine we use is a function. A function is a subroutine that returns a value. That means we can use it within other statements. We use the `return` keyword to send the value back to the main program.

```
#Function to double a number
def double(number):
    twicenum=2*number
    return twicenum



#Main Program
a=6
print('Double a is')
print(double(6))
```

Task 10a

Write a function called `circleArea` that takes in a float representing the radius and returns the area of a circle.

(Area of a circle is πr^2 where r is the radius)

Call your function from this main program:

```
print(circle(1.0))
print(circle(2.0))
print(circle(3.0))
```

Task 10b

Write a procedure called triangle that takes in a number and then prints out a triangle of that height. so
```
triangle(4)
```
would print out:

```
   *
  ***
 *****
*******
```

Call your function from this main program:
```
triangle(2)
triangle(3)
triangle(4)
```

it should output:
```
   *
  ***
   *
  ***
 *****
   *
  ***
 *****
*******
```

11. Lists

If we think of a variable as a box we can think of a list as a collection of boxes containing data. Each "box" has an address starting at 0.

The following code produces a list called names.

```
names=['Alf','Betsy','Charlie','David']
```

names:

Alf	Betsy	Charlie	David
0	1	2	3

We can then access any element of the list by giving its index in square brackets.

```
names=['Alf','Betsy','Charlie','David']
print(names[2])
```

Will print `Charlie`

We can also alter the contents of the list by referring to the elements by their index.

```
names=['Alf','Betsy','Charlie','David']
names[2]='Bob'
print(names[2])
```

Changes the list to:

Alf	Betsy	Bob	David
0	1	2	3

So it will print `Bob`
If we start off with our list as empty we can create it with just two empty square brackets.
`names=[]`

To add entries to the list we then call the append function.

```
names=[]
names.append('Betsy')
names.append('Anita')
names.append('Murshed')
names.append('Hamish')
print(names)
```

Will output `['Betsy', 'Anita', 'Murshed', 'Hamish']`

To remove an item from the list we use the pop method if we want to remove by index

```
names=[]
names.append('Betsy')
names.append('Anita')
names.append('Murshed')
names.append('Hamish')
names.pop(1)
print(names)
```

Will output `['Betsy', 'Murshed', 'Hamish']`

If we know the item we want to remove we can use the remove function which deletes the first instance of it.

```
names=[]
names.append('Betsy')
names.append('Anita')
names.append('Murshed')
names.append('Hamish')
names.remove('Anita')
print(names)
```

Will also output `['Betsy', 'Murshed', 'Hamish']`

To get the number of items in a list we use `len`.

```
x=len(names)
```

Makes x the number of names in the list.

We can cycle through a list with a for loop.

```
names=['Anita','Betsy', 'Murshed', 'Hamish']
for i in names:
    print(i)
```

to print

```
Anita
Betsy
Murshed
Hamish
```

Arrays vs Lists

Most languages use arrays rather than lists so you may come across the term array being used in a similar context. (Python does have arrays available but lists are much more commonly used.)

The two main differences between the two are:

•Lists can hold different sorts of data whereas an array (usually, depending on the language) only holds one data type.

•Lists change size as new items are added where as in most languages arrays have their size declared at the start.

Task 11a

Write a program that keeps asking for names until the word END is entered at which point it prints out the list of names.

```
Please enter a name: Alfred
Please enter a name: Bradley
Please enter a name: Connor
Please enter a name: David
Please enter a name: Emily
Please enter a name: END
You have entered 5 names.
['Alfred', 'Bradley', 'Connor', 'David', 'Emily']
```

(Keep your code safe for this answer you will need it in Task13b)

Task 11b

Write a program that asks the user to enter 5 names which it stores in a list. Next, get it to pick one of these names at random and declare that person as the winner. (**Hint:** You will need to generate random numbers as in Task 9c)

E.g.
```
Please enter name 1:
Sarah
Please enter name 2:
Nathan
Please enter name 3:
Aniela
Please enter name 4:
Safiya
Please enter name 5:
Bob
Well Done Bob you are the winner!
```

Task 11c

The Sieve of Eratosthenes is an algorithm to find prime numbers. The algorithm goes as follows:

Eratosthenes invented a method for efficiently constructing tables of prime numbers. This method, the "Sieve of Eratosthenes", It goes like this. First, write down a list of integers beginning with 2 and ending with some number, say, 20:

```
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

Then mark all multiples of 2:

```
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
    x   x   x    x     x     x     x     x     x
```

Move to the next unmarked number, which in this case is 3, then mark all its multiples:

```
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
    x   x   x x x    x     x x x     x     x
```

Continue in this fashion, marking all multiples of the next unmarked number until there are no new unmarked numbers. The final result is in fact the last table given, so the primes not exceeding 20 are

```
2 3 5 7 11 13 17 19
```

---

http://www.math.utah.edu/history/eratosthenes.html

Write a program that prints out all the primes below 10,000.
Hint: you will need to use `%` which gives the remainder of a division operation.
So `x=11%5` would give **1** as `11/5=2` remainder **1**

12. String Handling

Often we want to perform operations on strings. This may involve cycling through letters in the string. This is particularly easy in Python as we can do it in a for loop.

Task 12a

Copy and run the following program.

```
sentence = 'The cat sat on the mat.'
for letter in sentence:
    print(letter)
```

Sometimes we might just want part of the string. We can do this by treating the string as a **list** of characters.

S	a	n	d	w	i	c	h
0	1	2	3	4	5	6	7

We can then refer to letters by their index (remembering that a list starts at index 0). So

```
text='Sandwich'
print(text[2])
```

will output the letter `n`.

If you want a group of letters this is done using the following notation.

```
Stringname[position of first letter:position after last letter]
```

So

```
text= 'Sandwich'
print(text[0:4])
```

produces `Sand`

and

```
text= 'Sandwich'
print(text[1:5])
```

produces `andw`

If we want the length of a string we use `len` just like we did with lists. So:

```
text= 'Sandwich'
a=len(text)
print(a)
```

outputs

8

Task 12b

Change the previous program so it counts the number of occurrences of the lowercase letter a.

Task 12c

Write a program that takes in a word and says whether or not it is a palindrome.
(A palindrome is a word that is the same backwards as forwards like **noon** and **radar**)

Specimen Exercise

A good exercise to test your understanding of string handling is the password strength exercise in the Specimen A453 tasks.

Here is a reminder of the task:

Design, code test and evaluate a system to accept and test a password for certain characteristics.
•It should be at least 6, and no more than 12 characters long
•The system must indicate that the password has failed and why, asking the user to re enter their choice until a successful password is entered.
•A message to indicate that the password is acceptable must be displayed.
•Password strength can be assessed against simple criteria to assess its suitability; for example a password system using only upper and lower case alphabetical characters and numeric characters could assess the password strength as:

    WEAK if only one type used, eg all lower case or all numeric
    MEDIUM if two types are used
    STRONG if all three types are used.

For example
hilltop, 123471324, HAHGFD are all WEAK,
catman3 and 123456t are MEDIUM and
RTH34gd is STRONG
 A message to indicate the password strength should be displayed after an acceptable password is chosen

---

It would be excellent practice to give this a go. On the next page is one version of what the solution could look like in Python.

Worked Password Example

NB This is just the code section. To score full marks in A453 you would need design, testing and evaluation as appropriate.

```python
#Program to assess strength of password

#Ask for password until one of correct length is entered
passwordValid=False
while passwordValid==False:
    password=input('Please enter your password ')
    if len(password)<6:
        print('Password too short - must be 6 or more characters')
    elif len(password)>12:
        print('Password too long - must be 12 or fewer
characters')
    else:
        print('Password Accepted')
        passwordValid=True

#Cycle through the password character by character counting
#the number of each 'type' of character.
lowerCount=0
upperCount=0
numberCount=0

for letter in password:
    if letter>='a' and letter<='z':
        lowerCount=lowerCount+1
    elif letter>='A' and letter<='Z':
        upperCount=upperCount+1
    elif letter>='0' and letter<='9':
        numberCount=numberCount+1

#Count up the types of character
typesOfChar=0
if lowerCount>0:
    typesOfChar=typesOfChar+1
if upperCount>0:
    typesOfChar=typesOfChar+1
if numberCount>0:
    typesOfChar=typesOfChar+1

#Print out the strength of the password
if typesOfChar==1:
    print('This is a WEAK password')
elif typesOfChar==2:
    print('This is a MEDIUM password')
else:
    print('This is a STRONG password')
```

13. Reading and Writing to Files

Writing to Files

Most real world programs need to be able to read and write data to files. Fortunately Python makes this process extremely easy to do.

To get Python to write to a file we first open it, write what we want to it and finally close it.

```
myFile = open('example.txt', 'wt')
myFile.write('I have written to a file.')
myFile.close()
```

Copy and run this code. If you look in the same directory where you saved your program you should now see the file example.txt. When writing to a file Python creates it if it does not yet exist.

If we want multiple lines we can use the \n newline escape sequence.

```
# Write a file
myFile = open('example.txt', 'wt')
myFile.write('I have written to a file.\nIt now has three lines\nThe third being this one')
myFile.close()
```

Alternatively we can spread it out over several lines (but note we still need the \n)

```
# Write a file
myFile = open('example.txt', 'wt')
myFile.write('I have written to a file.\n')
myFile.write('It now has three lines.\n')
myFile.write('The third being this one.\n')
myFile.close()
```

Reading From Files

Reading from files is equally as straight forward.

Make sure you have run one of the above programs so you have the file example.txt containing:

I have written to a file.

It now has three lines.

The third being this one.

Copy and run the following code:

```
myFile = open('example.txt', 'rt')
contents = myFile.read()
print(contents)
myFile.close()
```

If all has worked the contents of your file should appear in your programming console.

rt/wt

The rt and wt that appear in the open command stand for read text and write text. They tell Python the type of file you are opening and what access you need to it.

Sometimes we want to read a file line at a time. We do this using the `readline` command. If we want it to read all the lines in the file we can use a while loop. We tell the program to `readline` *while* we are not reading in an empty line (and so the end of the file). Of course you could, of course, change this to keep reading until it reaches any other string of your choice.

```
myFile = open('example.txt', 'rt')
line = myFile.readline()
while line!='':
    print(line)
    line = myFile.readline()
myFile.close()
```

An more elegant solution is to use a for loop.

```
myFile = open('example.txt', 'rt')
for line in myFile:
    print(line)
myFile.close()
```

You will notice with both of these options there are blank lines in between each line of text. To prevent this change `print(line)` to `print(line,end='')` *[NB two single quotation marks not one double]*

Task 13a

Write a program that reads a list of numbers from a file then outputs the average.
So if your file contained

```
3
45
83
21
```

Your program would output: `38`

Task 13b

In task 11a you had to write a program that takes in names and stored them in a list until END is entered. This time you are going to change it so it stores the names in between times the program is run. Adapt the program so that when it loads it looks for the file names.txt and reads any names into the list. The user should then be able to enter new names as before. When END is typed it should output the new list (i.e. loaded names and entered names) then save it to the file names.txt

Task 13c

When you are confident with using read and write why not try the specimen A453 task:

High scores database
Design, code and test a system to store and manage user names and their highest score.
The system must be able to
•create a file
•add data to a file
•locate data in the file by name and their highest score
•delete an item and its associated data from the file
•locate and update a high score for a user
The system need only cater for 10 items

Useful Resources

The Official Python Website http://docs.python.org/py3k/
Here you will find tutorials and official reference materials.

Learn Python http://www.learnpython.org/
An excellent set of online Python tutorials that cover the topics in this booklet and beyond.

Computing At School http://www.computingatschool.org.uk/
CAS is an organisation that seeks to promote the teaching of computing in schools. It is an active and friendly community with a thriving forum that is always a good source of help and ideas.

The British Informatics Olympiad http://www.olympiad.org.uk/
The British Informatics Olympiad is an annual programming competition open to students 18 and under. The site has a collection of past papers with questions that vary from challenging to extremely difficult.