Teaching skills and subject expertise

Key Stage 4 Computing

An introduction to programming using Python

Delegate handbook
Permission to reproduce all copyright materials have been applied for. In some cases, efforts to contact copyright holders have been unsuccessful and AQA will be happy to rectify any omissions of acknowledgements in future documents if required.
<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session 1 - Introduction to Python</td>
<td>4 - 5</td>
</tr>
<tr>
<td>Session 2 - Saving Python programmes</td>
<td>6 - 9</td>
</tr>
<tr>
<td>Session 3 - Interactive programmes</td>
<td>10 - 13</td>
</tr>
<tr>
<td>Session 4 - Using variables</td>
<td>14 - 15</td>
</tr>
<tr>
<td>Session 5 - Calculations</td>
<td>16 - 17</td>
</tr>
<tr>
<td>Session 6 - Data Types</td>
<td>18 - 20</td>
</tr>
<tr>
<td>Session 7 – Selection: The if statement</td>
<td>21 - 24</td>
</tr>
<tr>
<td>Session 8 - Iteration: Condition controlled loops</td>
<td>25 - 26</td>
</tr>
<tr>
<td>Session 9 – Iteration: Count controlled loops</td>
<td>27 - 28</td>
</tr>
<tr>
<td>Session 10 – Subroutines: Procedures and functions</td>
<td>29 - 30</td>
</tr>
<tr>
<td>Session 11 - Lists</td>
<td>32 – 35</td>
</tr>
<tr>
<td>Session 12 - String handling</td>
<td>36 - 37</td>
</tr>
<tr>
<td>Session 13 - File handling</td>
<td>38 - 40</td>
</tr>
</tbody>
</table>
Session 1 - Introduction to Python

Open the Python IDLE GUI which is the Integrated Development Environment (IDE) (called IDLE after Eric Idle). You will find it in your All Programs Menu under Python.

Using the IDLE in “Interactive Mode”

Open Python as described above and you will see this window:
Hello World

All programmers start with the program “Hello world”. It’s very much of a rite of passage.

Exercise 1

Copy and run the following program in Python in interactive mode (first Window in IDLE).

```python
print('Hello World')  # (case sensitive as are all commands in Python)
```

You will hopefully see the message **Hello World** as shown below

*Hint*

`print  (“xxx”)` will work as will `print  (‘xxx’)` (N.B. speech marks)

But `print (“xxx”)` won’t because Python does not accept capitals in commands such as print

---

Exercise 1a

Change the program such that it prints out:

**Pleased to meet you**

Exercise 1b

Write a program to display using apostrophes rather than speech marks ‘ ‘:

**This is John's first program**

The apostrophe makes this trickier than it first may seem.

*Hint:*

Use a combination of “ ” and ‘ ‘ characters try it and see.
Session 2 - Using IDLE and Saving Programs

When using the IDE (IDLE) in “Script Mode”, IDLE is in effect a word processor for Python programmers.
Using “interactive mode” is fine if you want to look at one line programs or use notepad to write programs and copy them into IDLE but for real programs you need to write and save your program and then run it.
Open Python and you will see the IDLE window

Now select File and then select New Window. A new window will open looking just like the one below.

It is slightly different from the interactive window as it is currently untitled and has no Shell menu tab. This window is called a “Script Window” and a new one is opened for each new program you want to write.

Now type into this new window: `print ("Hello New World")` and press enter.
Hopefully nothing will happen except the cursor moves to a new line.

Now go to the Run Menu and select Run Module (you can just press F5 at any time to run your program).

You will be asked to save the program. Save it into a folder as shown below. I saved it into a folder I made in My Documents that I called Python.

Note that the PC saves it as a type .py. This means the PC recognises it as a Python File. If you run this on your laptop at home you might have to add the .py yourself. You will know if you need to do this as your program will change from multi-coloured to all black if it doesn’t have the correct extension suffix.

Once you have saved it, if necessary press F5 again and the program will run. Below you should see what happens.
Exercise 2a

Try typing in the following program: Remember the syntax is important and be careful when you use capitals. Type in the following:

```python
print("This is")
print("a computer program")
print("that prints on several lines")
```

Run the program by using menu: Run – Run Module (or F5).

Save to your Python folder when prompted and you should see what is shown below.

```
>>> This is a computer program that prints on several lines
```
Exercise 2b

Try this the \n command at any point in a line of text will force a new line:

```python
print("This is \n")
print("\n")
print("a computer program")
print("\n")
print("that prints on several lines")
```

You should see something like this:

```python
>>> This is

a computer program

that prints on several lines
```
Session 3 - Introduction and storing variables

Storing Data in Variables

We need a mechanism for tracking values and changes of values in a program. We do this in high level programming by using what we term as "variables". A typical variable usage would be:

\[ a = 6 \]

which assigns the value of 6 to the variable "a"

In programming we try and assign meaningful variable names, e.g., name, age, postcode.

*NB*

Be a little careful here as I have seen variable names that are so complex in order to simplify understanding they make the program hard to follow. e.g. student_initial_id_parameters which is neither meaningful nor easy to use.

Another issue with variables when teaching students is they forget that a variable is only a name, i.e., it has no meaning to a computer. e.g. a variable called date could be written dote or dite.

To the program it is just a variable and has no special significance but students can forget this and try and read special meaning or believe they must use specific names for variables. This is particularly a problem explaining a concept and using variables such as "result" or "value".

Data Types

Every variable has a name, a data type, and a value. 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/real** - 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 programming languages we have to tell the computer what data type a variable is going to be.

E.g. in VB we say DIM name as string = “ “ or DIM age as Integer = 0

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*).

E.g. **age = 23** will automatically initialise a variable called **age** as an **integer variable** and assign a value 23 to **age** and define the variable as an integer.
Exercise 3a

Open Python and you will see the IDLE window

Now select File and then select New Window. A new window will open looking just like the one below.

Copy in the following program note variables are not normally given capital letters in Python e.g. we use name not Name. This is just a convention and has no effect on the way the program runs.

```python
name='Bob'
print("hello " + name)
```

NB.

*Do not copy and paste from MS Word as it will not work because MSWord/IDLE mismanage the apostrophes. Students will always try and do this so feel free to try later to see examples of the error messages.*

Run the program(F5) and when prompted save it to your Python Folder and it should look like the example below.
NB
IDLE puts the colours in automatically so if your program does not run check the syntax by comparing the colours on the screen shot and the colours in the code you have written.

Exercise 3b

Try another program

Language='Python'
Print("This program is written in " [YOU FINISH OFF THE PROGRAM AND RUN IT]

Exercise 3c

We can change the value of a variable within the program. Try running this program this and explain the result:

Exercise 3d

Here is another one
Now explain the outcome (Hint: the secret is in the use of the + symbol in the program which is used to connect the two strings together. This joining is called concatenation in programming)

Whereas if we use the program:

```python
name='Fred'
name=name+name
print("Your name is "+name)
```

Now explain difference between the outcomes of the two programs (Hint: again the + symbol is the key).

Finally using the following variables

```python
word1='the'
word2='cat'
word3='sat'
word4='on'
word5='mat'
```

write a program to Print the cat sat on the mat.

Hint:

The trick is the use of spaces in the print command. You don’t want a space in the variable so you need it in the command. For example:

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

will output `Hello Bob`

I have drawn an arrow between the spaces. As you can see the spaces are introduced before the speech marks in the print command.
Session 4 - Interactive Programming

We have run some simple programs now we will look at how to make the programs interactive. By interactive we mean making a program wait for a user to input some data. The first command we will look at is the "input" command (note this is still lower case). Copy in the following program note variables are not normally given capital letters in Python e.g. we use name not Name. This is just a convention and has no effect on the way the program runs.

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

I have done this below with some comments using the # sign.

Comments are used to allow a programmer to add notes to the program so that others will understand what he is doing. Students will be asked to add comments to their programs when you do the controlled assessment. The example below also demonstrates that too many # comments can actually make a program harder to read.

Run the program (F5) and when prompted save to your Python Folder; it should look like the example below. I used the name John when asked, you can use what you like.
Exercise 4a

Write a program that asks for your favourite type of music and then replies I love {your choice} as well.

e.g.

```python
>>> ================================
>>> What is your favorite music? jazz
I really like jazz
``` 

Exercise 4b

Write a program that asks for your first name, then asks for your last name and finally prints out a greeting including your full name.

e.g.

```python
>>> ================================
>>> What is your first name? Fred
What is your surname? Smith
Hello Fred Smith
``` 

There are two ways to achieve this one uses a blank space “ “ the other involves investigating the difference between using the + (concatenating) and the , to separate the variables. Try both of these now. We will look into the both these methods in more detail in later lessons.
Session 5 – Calculations and different variables

Calculations

We can carry out calculations in Python. The arithmetic operators we use to do this are:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
</tr>
</tbody>
</table>

Run the following program:

```
print(3+7)
p
```
Exercise 5b

Complete this program below so that it adds the two variables `a` and `b` to make `x` equal to 16 then prints out the result:

```
a=9
b=7
x= #You put the rest of the program in
```
Session 6 Initiating Variables

Every variable has a name, a data type and a value. In Python when a value is assigned to a variable, Python automatically assigns the data type.

Data Types used in Python

The most common data types are:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Text made up of numbers, letters and characters.</td>
</tr>
<tr>
<td>Integer</td>
<td>Whole numbers (e.g. 1, 78, 0) and negative numbers (e.g.-54)</td>
</tr>
<tr>
<td>Float</td>
<td>Decimal numbers (e.g. 3.5683, 98.74634, -6.3). Float are also often referred to as Real numbers.</td>
</tr>
<tr>
<td>Boolean</td>
<td>True or False (meaning the variable is assigned a value of either 1 or 0)</td>
</tr>
</tbody>
</table>

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).

As mentioned earlier for x = 3, Python will make (assign) x as an integer variable.

and for a = 1.34, Python will make (assign) a as an float variable.

Finally of course or for b = “Fred”, Python will make (assign) b as a string variable.

This can cause problems when we try to do something with the wrong data type. See example below:

Exercise 6a

Copy and run the following program:

```
File Edit Format Run Options Windows Help
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 the program 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 a something called **type casting** (we use the term cast but it just means change or turn).

To turn (cast) the strings 4 and 4 into an integer we use the `int()` function.

There are two ways of doing this:

```python
a = input('Enter number 1:')
a = int(a)
```

or

```python
a = int(input('Enter number 1:'))
```

Which does the same thing and is the way we will normally do it in our programs.

**Exercise 6b**

Try this program:

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

**Exercise 6c**

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

**Example:**

```
Please enter width: 12
Please enter height: 6
The area is 72
```

**Exercise 6d** Converting integers into real numbers (float)

The formula for the volume of a cylinder is: \( \pi r^2 h \)
And for its surface area is: \(2\pi r^2 + 2\pi rh\).  
Where \(r\) is the radius and \(h\) the height. \(\pi\) is the value 3.14159 (to 5 dec. places)

Assume that the user may enter the radius and height as real numbers. To turn (cast) these into real numbers for the calculation you will need to use \texttt{float()} command.

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

Hint: Include the value of \(\Pi\) in your program so that you do not have to enter it when the program is running. Here is my result after running the program:

```python
>>> enter radius: 3
>>> enter height: 5
The surface area of the cylinder is: 150.796319999999998
```

If you want to print out the result with 2 decimal places you can use the \texttt{round()} command. I used a variable “a” for the area and this line of code to round down or up just before printing the output:

e.g. \(a = \text{round}(a, 2)\)

try this:

```python
a = 2.873
a = \text{round}(a,2)
p\text{rint}(a)
```

will print out 2.87

Whereas the \texttt{int()} command round down any floating point number. In actual fact Python merely truncates the number back to its decimal point.

```python
a = 2.899
a = \text{int}(a)
p\text{rint}(a)
```

will print out 2.
Session 7 – Selection: The if statement

You may remember the IF function in Excel. In all high level programming languages (including Python) there is an IF function to test for certain conditions.

A **Boolean condition** is one that can have only two values **true** or **false** (1 or 0, yes or no).

The boolean comparison operators used in python are as follows:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>==</code></td>
<td>Equal to</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>(&lt;&gt; is out of date now) Not equal to</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>Less than</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>Greater than</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>Less than or equal to</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>Greater than or equal to</td>
</tr>
</tbody>
</table>

The indent and the colon are important as Python uses them to manage the *if* statement. This program doesn’t really do much it’s just an example.

**Note** we are using the apostrophe `''` rather than speech mark `""` as we said earlier they are interchangeable but can cause problems if you are using apostrophes in your print e.g. can't.

Run the program above and save when prompted.

We can add as many statements we want inside the if statement. Each should be indented.

At the end of an *if statement* we remove the indentation.
when writing `if` statements, the colon after the `if` statement starts the actual testing for the `if` statement. The indents tell the program that this is still a part of the `if` statement and the removal of the indentation tells the program that the `if` statement has finished and to stop testing. See below:

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

So when writing `if statements`, the colon after the `if` statement starts the actual testing for the `if statement`. The indents tell the program that this is still a part of the `if` statement and the removal of the indentation tells the program that the `if statement` has finished and to stop testing. See below:

```python
if letter=='a':  # the colon starts the if
    print('This prints if letter is a')  # indent
    print('This prints as well')  # indent
print('This prints out whatever anything')  # indent removed
```

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 always prints out at the end of the if')
```

You can extend the IF statement by using an abbreviation of the ELSE IF Function in Python this is written `elif`.
A Note on Indentation

When you start a new line after a colon (if writing it using 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.

```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.

Exercise 7a

A slightly more complex task; enter the code below and run the program

```python
city = input('What is the capital of France?')

if city == 'Paris':
    print('Well done')
else:
    print('Wrong country, wrong city')
else:
    print('Wrong country, wrong city')
```
Exercise 7b

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.

```python
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')
```
8 – Iteration: Condition 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 condition controlled loop.

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.

If we run the above code you will find it goes on forever printing looping. that is because x remains 0 and so is always less than 5. You can try this if you wish but you will need to go to File/exit to stop it.

Let’s fix the program

Exercise 8a

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

This will revise your knowledge of the input command. Try and use a sensible variable (e.g. pass) when it is running use the print command to print meaningful hints like "Wrong Password" and "Correct"

Exercise 8b

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.

Hint: You have done squares in an earlier task keep doing the square and test for greater or equal to 5000. Print out the squares and print finished when completed.
Python: Importing libraries and the Random function

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

```python
import random  # This tells Python to use a subroutine called Random
x = random.randint(1,100)  # find a random number between 0 and 100
```

**Random** is a library function used to generate random numbers and the **import** command is used to bring external libraries into your program. The line **import random** has to be put at the very top of the program but does not need putting in the program again.

You can use the **random.randint ()** command wherever you want after that but remember to assign it to a variable e.g. **numb = random.randint (1,10)**. Which will assign a variable between 1 and 10 into the variable numb.

**Exercise 8c**

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.

*It might help to write this in rough using Pseudo code (almost Python but in plainer English) this will help sort out the sequence. Pseudo code is another form of algorithm. We are not worried about elegance, just getting into the habit of writing complex problems in a simpler form before writing the program.*

**Required Output**

```
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.
```
Session 9 - Iteration : Count Controlled Loops

As we said in the last task 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 the second type of loop, 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 the for command.

Notice how anything inside the loop is indented.

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

Will output

```
>>> looping
looping
looping
looping
looping
```

In the loop `for i in range(x,y), 0` is the starting value of `i` and `5` is the amount of times it will loop that is to say the final value of `i`.

So above it runs with `i` equal to 0,1,2,3 and 4 (not 5 because the loops stops when `i = 5`)

Traditionally loops tend to use the variables `i, j` and `k` and you will 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.

Try this program:

```python
for i in range(0,5):
    print('loop number',i)
```

As with the if statement the indent and the colon are important as Python uses them to manage the for statement.
It will print out `looping` for each value of `i` as the program goes around the loop and adds 1 to `i` each time.

```python
>>> loop number 0
loop number 1
loop number 2
loop number 3
loop number 4
```  

**Exercise 9a**

A slightly more complex task; enter the code below and run the program.

Change the program below so that it outputs the word `computing` 15 times.

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

**Exercise 9b**

Using the input command you learnt earlier, write a program that asks for a number then outputs it’s 10 times table.

**Example Output.**

```
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
```
Session 10 – Subroutines : Procedures and Functions

We are in new territory now as Procedures and Functions are very much aspects of higher level programming. It is possible to write a whole program as one big block but where possible we want to break it down into subroutines – chunks of code. 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 re-used meaning code does not need to be rewritten.

We will start off with a procedure (subroutine) that prints out “This is a subroutine” 3 times.

In Python for ease of reading the subroutines come before the main program.

```python
# Subroutine that prints out sentence 3 times
def mySub():
    for i in range(1, 4):
        print('This is a subroutine')
```

The line: `def mySub():` defines the subroutine; the name is just for show and convenience in understanding. The brackets are a part of Python Syntax.

Having defined the subroutine we now go to the main program

```python
# Main Program
print('Start of main program')
mySub()  # This line "calls" (branches) to the subroutine
print('End of program')
```

Now we put this all together as a program
The # comments are optional but good practice.

We can use parameters to make subroutines even more useful. Parameters are a way of taking some values (numbers or strings of text) to be used by the subroutine. For example the parameters in the subroutine below are text and times:

```python
#Subroutine that prints out sentence 5 times
def mySub(text, times):
    for i in range(0, times):
        print(text)

#Main Program
print('Start of main program')
mySub("My name is Jack", 5)
print('Printout finished')
```

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 single value. That means we can use it within other statements. We use the return keyword to send the value back to the main program.

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

#Main Program
a=int(input("input a number"))
print('Double a is')
print(double(a))
""" the above line jumps to the function called double with the value a. The function returns the value a*2 which is printed out."""
```
Exercise 10a

Write a function called `convert` that converts degrees Celsius to Fahrenheit. Where

\[ ^\circ F = ^\circ C \times \frac{9}{5} + 32 \]

In the main program input a value and then call the Function

If you feel really keen use `input` and `if` to offer the choice between \(^\circ F\) to \(^\circ C\) or \(^\circ C\) to \(^\circ F\). There are quite a few ways to do this try writing a flow diagram first.

To help here is the re-arranged formula

\[ ^\circ C = (^\circ F - 32) \times \frac{5}{9} \]
Session 11 - ARRAYs and LISTS

Arrays are common to all programming languages but in python the restrictions that normally apply to arrays have been removed and the result is referred to as lists rather than arrays. 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 (called an index) starting at 0. For example a list called **Names** might look like this:

<table>
<thead>
<tr>
<th>Contents</th>
<th>Alf</th>
<th>Betsy</th>
<th>Charlie</th>
<th>David</th>
</tr>
</thead>
<tbody>
<tr>
<td>Box Number (Index)</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

The following snippet of Python code produces the list called Names.

```python
Names=['Alf','Betsy','Charlie','David']

#"""The [] bracket tells Python that the variable is in fact a list"""
```

<table>
<thead>
<tr>
<th>Names</th>
<th>Alf</th>
<th>Betsy</th>
<th>Charlie</th>
<th>David</th>
</tr>
</thead>
<tbody>
<tr>
<td>Box Number (Index)</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

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

```python
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:

<table>
<thead>
<tr>
<th>Names</th>
<th>Alf</th>
<th>Betsy</th>
<th>Bob</th>
<th>David</th>
</tr>
</thead>
<tbody>
<tr>
<td>Box Number (Index)</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

and the program will print: **Bob**.

If we start off with our list empty we can create it with just two empty square brackets.

```
names=[]
```

To add entries to the list we then call the **append** function which will automatically manage the index and put the names in order in the list.

```
names=[]
names.append('Sam')
names.append('Sebastion')
names.append('Quincy')
names.append('Susan')
print(names)
```

Will output

```
['Sam', 'Sebastion', 'Quincy', 'Susan']
```
To remove an item from the list we use the `pop` method if we want to remove by index:

```python
names = []
names.append('Sam')
names.append('Sebastion')
names.append('Quincy')
names.append('Susan')
names.pop(2)
print(names)
```

Will output

```python
[['Sam', 'Sebastion', 'Susan']
```

(The first item is index 0 so item 1 is ‘Sebastion’)

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

```python
names = []
names.append('Sam')
names.append('Sebastion')
names.append('Quincy')
names.append('Susan')
names.remove('Quincy')
print(names)
```

Will also output

```python
[['Sam', 'Sebastion', 'Susan']
```

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

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

Assigns to `x` the number of names in the list.
We can cycle through a list with a for loop as shown below:

```python
names = ['Sam', 'Sebastian', 'Quincy', 'Susan']

for i in names:
    print(i)
```

Will print:

```
Sam
Sebastian
Quincy
Susan
```

**Exercise 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 as shown below.

```
Please enter a name: Sebastian
Please enter a name: Quincy
Please enter a name: Sam
Please enter a name: Susan
Please enter a name: Julie
Please enter a name: END

You have entered 5 names.
['Sebastian', 'Quincy', 'Sam', 'Susan', 'Julie']
```

**Exercise 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 using `import random` and `random.randint(1,5)` e.g.

```
Please enter a name: Sebastian
Please enter a name: Quincy
Please enter a name: Sam
Please enter a name: Susan
Please enter a name: Julie

Well Done Susan you are the winner!
```
Session 12 - String Handling

There are many ways of handling text (called strings) in high level programming languages and Python is no exception. String handling is a large subject and we are only introducing a few concepts to give an idea of what is possible.

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.

```python
text = 'Python is my favorite programming language.'
for chara in text:
    print(chara)
```

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

<table>
<thead>
<tr>
<th>E</th>
<th>x</th>
<th>a</th>
<th>m</th>
<th>p</th>
<th>l</th>
<th>e</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

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

```python
text='Example'
print(text[2])
```

will output:

```
>>> a
```

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

**Stringname [position of first letter : position after last letter]** For example:

```python
text='Example'
print(text[2:5])
```
Outputs:

```
>>> amp
```

and

```
text='Example'
print(text[0:3])
```

Outputs:

```
Exa
```

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

```
text='Example'
x=len(text)
print(x)
```

Outputs:

```
>>> 7
```

**Exercise 12a**

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

**Exercise 12b**

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`).
Session 13 - File Handling

Files are for permanent storage of data

What are files and why do we need them?

Around the average classroom you will see piles of folders with all sorts of information in them. Some will be just piled in a corner in no particular order perhaps just by class, some will be in order (perhaps alphabetical and class) and some will be in a nice cabinet in order and clearly identified for easy retrieval of information.

All of these are filing systems in the real world some being more sophisticated than others. In computers we can also store information in folders and files with the level of sophistication being dependent on how organised we are on our PC.

However in high level computer programming languages (and Python is no exception) we have the capability to construct sophisticated filing systems (a bit like a virtual filing cabinet) and then add and retrieve information from the filing system.

Variables and Lists allow you to store and access data when a program is running, however the data is lost when you close the program.

If you want to store data and access it at a later date, you need to save the data in such a way that it can be retrieved later. We do this in Python by using storing the data in a text file.

To create or open a file

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. The clever thing about Python is that you do not need to build your virtual filing cabinet first. As soon as you write to a file and give it a name Python will automatically build you virtual filing system and start putting files of documents in it.

```
myfile=open("subjects.txt","w")
```

This line of code creates a logical file called myfile. Python will create this and save it as an actual text file with the name “subjects” “w” means we are allowing the program to write to this file write to the file. If the file already exists any details already in the file will be overwritten(replaced).
If there is no such file as myfile stored as subjects.txt Python will create an empty file which we can write to later in the program.

**Writing to our file**

```python
myfile.write('english\n')
myfile.write('science\n')
myfile.write('maths\n')
myfile.write('computing\n')

"""This will write 4 lines to the file called myfile (\n is a "newline character")""

myfile.close() #YOU MUST CLOSE THE FILE WHEN FINISHED
```

**When you are finished with a file you must close it.**

**Checking to see if your new file exists**

Check to see if this text file exists and what it contains by looking in the folder where your Python programs are saved. Now open the file using any word processor e.g. Notepad, WordPad, MSWord and you will see the contents of that file.

**Reading data from a file**

```python
myfile=open('subjects.txt','r')
```

“r” tells Python you are going to read the file

```python
myfile.read() # reads the whole file
myfile.readline() # reads the current line
```

Try these lines of code and you should see the contents of the files
Adding to a file (appending)

Opening the file:

```python
myfile=open('subjects.txt','a')
```

'a' means we are going to add (append) to the file as opposed to overwriting a file.

Files only contain text

Txt files can only contain text so any numeric data has to be converted to text by using `str`. Either using the same name or a new name:

```python
height=str(height)
```

-or-

```python
newheight=str(height)
```