11. FILE HANDLING

None of the programs we have written will store any data permanently; the data only exists while the program is running. In order to save data from our programs to use again, we need to write the data to a text file and save it in secondary storage.

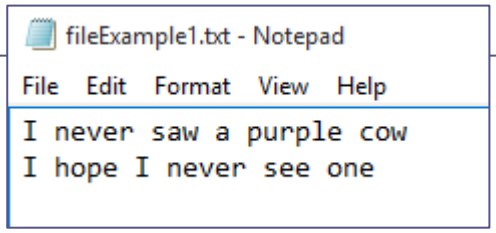
At GCSE level, this means saving the text file in the same directory as the Python program we have written.

11.1 WRITING TO A FILE

When we want to write to a file in Python we need to create a 'file object' using the built-in function: open(). The open() function has two arguments, the file name and the MODE. The mode determines how the file is read from or written to.

Example 1:

```
def write_file():
    """write data to a file"""
    myFile = open('fileExample1.txt','w') # writing mode
    myFile.write("I never saw a purple cow")
    myFile.write("\n")
    myFile.write("I hope I never see one")
    myFile.write("\n")
    myFile.close() # ensure that the file is closed
```



The text file has been created in the same folder as the program file.

Common modes for reading from and writing to a file:

Mode	Meaning
'r'	Used when the file is to be read into your program
'w'	Used when writing to a file; any existing file with the same name will be deleted
'a'	Used when opening an existing file to write data; the data is automatically appended to the end of the file

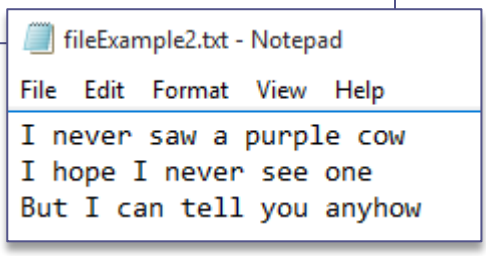
In the method used in Example 1 above, you must add the line of code to close the file object after you have finished reading or writing data.

There is a second method we can use which will **automatically close the file object** for us; it also makes error handling much easier when we are writing robust code.

Example 2:

This method automatically closes the file so we don't need to add the line of code.

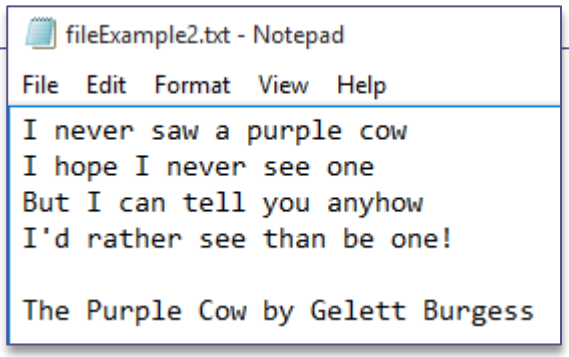
```
def write_file2():
    """write data to a file using with statement"""
    with open('fileExample2.txt','w') as myFile:
        myFile.write("I never saw a purple cow")
        myFile.write("\n")
        myFile.write("I hope I never see one")
        myFile.write("\n")
        myFile.write("But I can tell you anyhow")
        myFile.write("\n")
```



Example 3:

This example shows the **append mode** of writing to an existing file:

```
def write_file3():
    """write data to a file using with statement"""
    with open('fileExample2.txt', 'a') as myFile:
        myFile.write("I'd rather see than be one!")
        myFile.write("\n")
        myFile.write("\n")
        myFile.write("The Purple Cow by Gelett Burgess")
        myFile.write("\n")
```



We have been looking at writing to text files; there is another type of file that you may need to understand how to use in your programming project, a CSV (Comma Separated Value) file. This is the standard file format to import and export data to and from a database or spreadsheet. A CSV file will contain data in rows, separated by commas into columns.

When we want to read from or write to a CSV file format we need to import the CSV module.

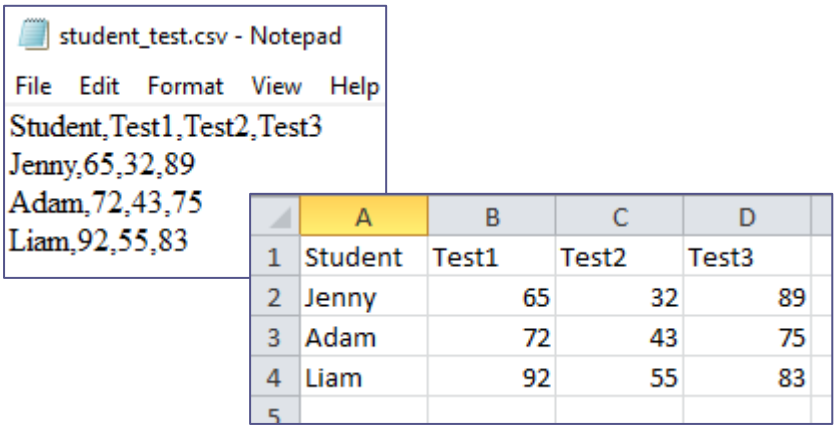
```
import csv

def write_csv_file():
    """write list to csv format"""
    student_details = [['Jenny', 65, 32, 89], ['Adam', 72, 43, 75], ['Liam', 92, 55, 83]]
    with open ('student_test.csv', 'w', newline = '') as myFile:
        writer = csv.writer(myFile)
        writer.writerow(['Student', 'Test1', 'Test2', 'Test3'])
        for student in student_details:
            writer.writerow(student)

def main():
    """runs all functions"""
    write_csv_file()

main()
```


As you can see in this example, I am writing a 2D list called studentDetails to the CSV file; this method uses the csv.writer() built-in function to manage the process of writing the data in the CSV file format. The writerow() built-in function is then used to write the rows of data into the file, and this is the result:



	A	B	C	D
1	Student	Test1	Test2	Test3
2	Jenny	65	32	89
3	Adam	72	43	75
4	Liam	92	55	83
5				

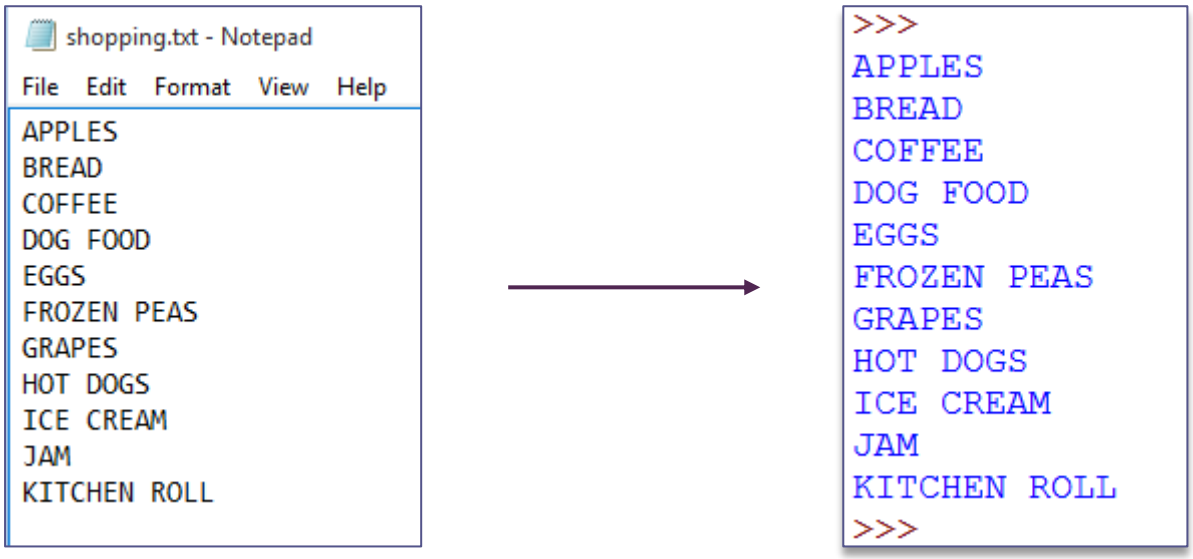
The data can be opened in a text editor, spreadsheet or database program.

11.2 READING FROM A FILE

When we read in a file, we also need to create a 'file object', just as we did when writing to a file. This time there are several methods we can use to read from the file, which each work in a slightly different way.

Example 1:

```
def read_file1():
    """reads in file line by line"""
    with open('shopping.txt', 'r')as myFile:
        line = myFile.readline()
        while len(line) != 0:  # while there is data in the line
            print(line, end='')
            line = myFile.readline()# read the next line
```



When we read in a text file, we will generally want to do some other type of processing with it rather than just print it out.

Example 2:

This example uses readlines(), which creates a list of the string values in the file but includes the '\n' newline code in each list element.

```
def read_file2():
    """reads in file line by line"""
    with open('shopping.txt', 'r')as myFile:
        items = myFile.readlines()
        print(items)
```

```
>>>
['APPLES\n', 'BREAD\n', 'COFFEE\n', 'DOG FOOD\n', 'EGGS\n', 'FROZEN PEAS\n',
'GRAPES\n', 'HOT DOGS\n', 'ICE CREAM\n', 'JAM\n', 'KITCHEN ROLL\n']
>>>
```

This can cause problems when you are trying to compare a string entered by the user with a string in the list; they do not match!

Example 3:

```
def read_file2():
    """reads in file line by line"""
    with open('shopping.txt', 'r')as myFile:
        items = myFile.readlines()
        print(items)
    if 'BREAD' in items:
        print("Toast for breakfast")
    else:
        print("Add bread to shopping list")
```

```
>>>
Add bread to shopping list
>>>
```

The solution is to use another method to read in the file to a list which also allows us to 'strip' out the newline character.

```
def read_file3():
    """reads in file line by line and strip \n"""
    with open('shopping.txt', 'r')as myFile:
        items = myFile.read().split('\n')
        print(items)
```

```
>>>
['APPLES', 'BREAD', 'COFFEE', 'DOG FOOD', 'EGGS', 'FROZEN PEAS', 'GRAPES',
'HOT DOGS', 'ICE CREAM', 'JAM', 'KITCHEN ROLL']
>>>
```


Example 4:

```
def read_file2a(file):
    """reads in file to an array"""
    shopping_list = []
    with open('shopping.txt', 'r') as myFile:
        for line in myFile:
            line = line.strip('\n')
            shopping_list.append(line)
        print(shopping_list)
```

The text file can also be read straight into an array, as shown in Example 4. This time the newline character is stripped from the end of the string before it is appended to the list. This is an alternative option to using the split() built-in function; the result is the same.

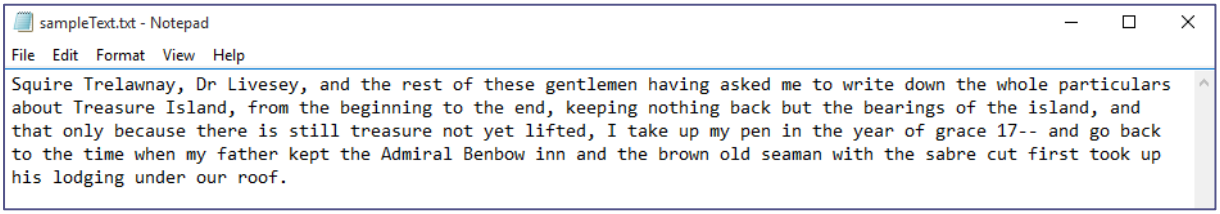
```
['APPLES', 'BREAD', 'COFFEE', 'DOG FOOD', 'EGGS', 'FROZEN PEAS', 'GRAPES', 'HOT
DOGS', 'ICE CREAM', 'JAM', 'KITCHEN ROLL']
```

Example 5:

```
def read_file4():
    """reads in file into a string """
    with open('sampleText.txt','r')as myFile:
        text = myFile.read()
        print(text)
```

When the text file contains a string of characters, we can use this method.

Original file:



The text string that has been read in and printed out:

```
>>>
Squire Trelawnay, Dr Livesey, and the rest of these gentlemen having asked me to
 write down the whole particulars about Treasure Island, from the beginning to t
he end, keeping nothing back but the bearings of the island, and that only becau
se there is still treasure not yet lifted, I take up my pen in the year of grace
 17-- and go back to the time when my father kept the Admiral Benbow inn and the
 brown old seaman with the sabre cut first took up his lodging under our roof.
>>>
```


Again, you may also need to read in a text file in to your program which is saved in CSV format. Import the CSV module, which deals with reading in the rows and columns of data. Look at this example:

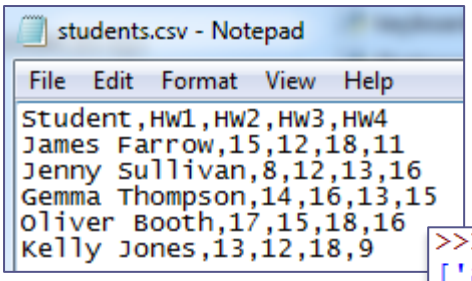
```
import csv

def import_file():
    """reads in the file"""
    with open ('students.csv', 'r') as myFile:
        reader = csv.reader(myFile)
        student_list = list(reader)
    return student_list

def main():
    """runs all functions"""
    student_list = import_file()
    for x in student_list:
        print(x)

main()
```

This is the CSV file which has been saved using a simple text editor. As you can see, the data is in rows and each data item is separated by commas.



The data in the text file has now been read into a 2D list of lists; each nested list contains the data for each student and a list of column headings. This data could be displayed in a more user-friendly way; look at ***Printing out a 2D List*** in Chapter 8 for more detail.

```
>>>
['Student', 'HW1', 'HW2', 'HW3', 'HW4']
['James Farrow', '15', '12', '18', '11']
['Jenny Sullivan', '8', '12', '13', '16']
['Gemma Thompson', '14', '16', '13', '15']
['Oliver Booth', '17', '15', '18', '16']
['Kelly Jones', '13', '12', '18', '9']
>>>
```

EXERCISE 28: FILE HANDLING

1. Create a text file with the first names of 12 students in your class and save it as 'students.txt'. Read in the file into a list and sort it into order, write the list back to the file and check the names are now in alphabetical order.

2. Create a text file with some items of grocery shopping, save as 'shopping.txt'. Write a program that will:
    a. Read in the file into a suitable data structure
    b. Ask the user for three additional items to add to the list
    c. Check whether the item is in the list, if not add it to the list
    d. Print out the new list with the three additional items

