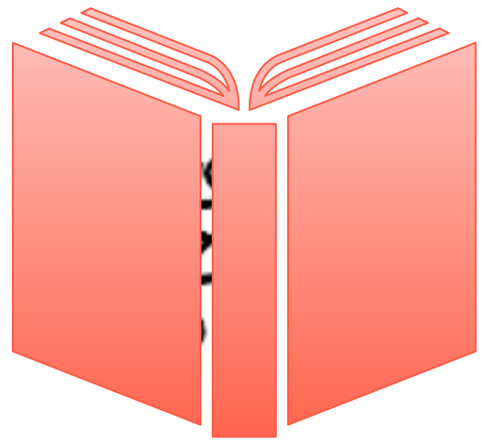
Reading and Writing to a Text File



Explanation

It is all very well being able to define a list, make changes and add new data, but if the next time the program is run it returns to the original data and your changes are lost then it is not a lot of use. Therefore, it is sometimes necessary to save data outside of the program and this way the data can be stored, along with any changes that are made.

The easiest place to start learning about writing and reading from an external file is with a **`text`** file.

When opening an external file you must specify how that file will be used within the program. The options are below.

Code	Description
`w`	**Write mode:** used to create a new file. Any existing files with the same name will be erased and a new one created in its place.
`r`	**Read mode:** used when an existing file is only being read and not being written to.
`a`	**Append mode:** used to add new data to the end of the file.

Text files are only used to write, read and append data. By the very nature of how they work it is not easy to remove or alter individual elements of data once it is written to the file, unless you want to overwrite the entire file or create a new file to store the new data. If you want to be able to alter the individual elements once the file has been created it is better to use a .csv file (see page 91) or an SQL database (see page 134).

Example Code

```
file = open("Countries.txt","w")
file.write("Italy\n")
file.write("Germany\n")
file.write("Spain\n")
file.close()
```

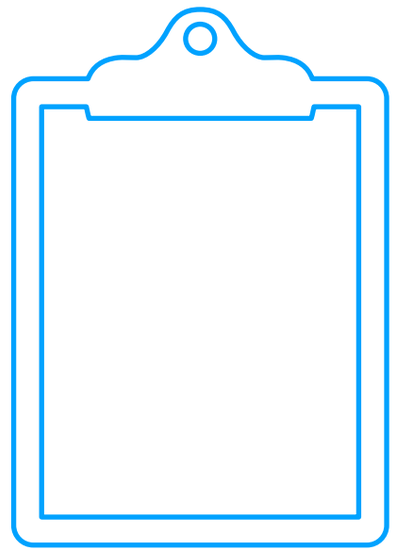
Creates a file called "Countries.txt". If one already exists then it will be overwritten with a new blank file. It will add three lines of data to the file (the \n forces a new line after each entry). It will then close the file, allowing the changes to the text file to be saved.

```
file = open("Countries.txt","r")
print(file.read())
```

This will open the Countries.txt file in "read" mode and display the entire file.

```
file = open("Countries.txt","a")
file.write("France\n")
file.close()
```

This will open the Countries.txt file in "append" mode, add another line and then close the file. If the **`file.close()`** line is not included, the changes will not be saved to the text file.



105

Write a new file called "Numbers.txt". Add five numbers to the document which are stored on the same line and only separated by a comma. Once you have run the program, look in the location where your program is stored and you should see that the file has been created. The easiest way to view the contents of the new text file on a Windows system is to read it using Notepad.

106

Create a new file called "Names.txt". Add five names to the document, which are stored on separate lines. Once you have run the program, look in the location where your program is stored and check that the file has been created properly.

107

Open the Names.txt file and display the data in Python.

108

Open the Names.txt file. Ask the user to input a new name. Add this to the end of the file and display the entire file.

109

Display the following menu to the user:

```
1) Create a new file
2) Display the file
3) Add a new item to the file
Make a selection 1, 2 or 3:
```

Ask the user to enter 1, 2 or 3. If they select anything other than 1, 2 or 3 it should display a suitable error message.

If they select 1, ask the user to enter a school subject and save it to a new file called "Subject.txt". It should overwrite any existing file with a new file.

If they select 2, display the contents of the "Subject.txt" file.

If they select 3, ask the user to enter a new subject and save it to the file and then display the entire contents of the file.

Run the program several times to test the options.

110

Using the Names.txt file you created earlier, display the list of names in Python. Ask the user to type in one of the names and then save all the names except the one they entered into a new file called Names2.txt.

Reading and Writing to a .csv File

Explanation

CSV stands for **Comma Separated Values** and is a format usually associated with importing and exporting from spreadsheets and databases. It allows greater control over the data than a simple text file, as each row is split up into identifiable columns. Below is an example of data you may want to store.

Name	Age	Star sign
Brian	73	Taurus
Sandra	48	Virgo
Zoe	25	Scorpio
Keith	43	Leo

A .csv file would store the above data as follows:

```
Brian, 73, Taurus
Sandra, 48, Virgo
Zoe, 25, Scorpio
Keith, 43, Leo
```

However, it may be easier to think of it as being separated into columns and rows that use an index number to identify them.

	0	1	2
0	Brian	73	Taurus
1	Sandra	48	Virgo
2	Zoe	25	Scorpio
3	Keith	43	Leo

When opening a .csv file to use, you must specify how that file will be used. The options are:

Code	Description
`w`	Creates a new file and writes to that file. If the file already exists, a new file will be created, overwriting the existing file.
`x`	Creates a new file and writes to that file. If the file already exists, the program will crash rather than overwrite it.
`r`	Opens for reading only and will not allow you to make changes.
`a`	Opens for writing, appending to the end of the file.

Example Code

```
import csv
```
This must be at the top of your program to allow Python to use the .csv library of commands.

```
file = open ("Stars.csv","w")
newRecord = "Brian,73,Taurus\n"
file.write(str(newRecord))
file.close()
```
This will create a new file called "Stars.csv", overwriting any previous files of the same name. It will add a new record and then close and save the changes to the file.

```
file = open ("Stars.csv","a")
name = input("Enter name: ")
age = input("Enter age: ")
star = input("Enter star sign: ")
newRecord = name + "," + age + "," + star + "\n"
file.write(str(newRecord))
file.close()
```
This will open the Stars.csv file, ask the user to enter the name, age and star sign, and will append this to the end of the file.

```
file = open("Stars.csv","r")
for row in file:
 print(row)
```
This will open the Stars.csv file in read mode and display the records one row at a time.

```
file = open("Stars.csv","r")
reader = csv.reader(file)
rows = list(reader)
print(rows[1])
```
This will open the Stars.csv file and display only row 1. Remember, Python starts counting from 0.

```
file = open ("Stars.csv","r")
search = input("Enter the data you are searching for: ")
reader = csv.reader(file)
for row in file:
 if search in str(row):
  print(row)
```
Asks the user to enter the data they are searching for. It will display all rows that contain that data anywhere in that row.

```
import csv
file = list(csv.reader(open("Stars.csv")))
tmp = []
for row in file:
 tmp.append(row)
```

A .csv file cannot be altered, only added to. If you need to alter the file you need to write it to a temporary list. This block of code will read the original .csv file and write it to a list called "tmp". This can then be used and altered as a list (see page 58).

```
file = open("NewStars.csv","w")
x = 0
for row in tmp:
 newRec = tmp[x][0] + "," + tmp[x][1] + "," + tmp[x][2] + "\n"
 file.write(newRec)
 x = x + 1
file.close()
```

Writes from a list into a new .csv file called "NewStars.csv".

111

Create a .csv file that will store the following data. Call it "Books.csv".

	Book	Author	Year Released
0	To Kill A Mockingbird	Harper Lee	1960
1	A Brief History of Time	Stephen Hawking	1988
2	The Great Gatsby	F. Scott Fitzgerald	1922
3	The Man Who Mistook His Wife for a Hat	Oliver Sacks	1985
4	Pride and Prejudice	Jane Austen	1813

112

Using the Books.csv file from program 111, ask the user to enter another record and add it to the end of the file. Display each row of the .csv file on a separate line.

113

Using the Books.csv file, ask the user how many records they want to add to the list and then allow them to add that many. After all the data has been added, ask for an author and display all the books in the list by that author. If there are no books by that author in the list, display a suitable message.

114

Using the Books.csv file, ask the user to enter a starting year and an end year. Display all books released between those two years.

115

Using the Books.csv file, display the data in the file along with the row number of each.

116

Import the data from the Books.csv file into a list. Display the list to the user. Ask them to select which row from the list they want to delete and remove it from the list. Ask the user which data they want to change and allow them to change it. Write the data back to the original .csv file, overwriting the existing data with the amended data.

117

Create a simple maths quiz that will ask the user for their name and then generate two random questions. Store their name, the questions they were asked, their answers and their final score in a .csv file. Whenever the program is run it should add to the .csv file and not overwrite anything.

Subprograms

Explanation

Subprograms are blocks of code which perform specific tasks and can be called upon at any time in the program to run that code.

Advantages

- You can write a block of code and it can be used and re-used at different times during the program.
- It makes the program simpler to understand as the code is grouped together into chunks.

Defining a subprogram and passing variables between subprograms

Below is a simple program that we would normally create without subprograms but have written it with subprograms so you can see how they work:

```
def get_name():
    user_name = input("Enter your name: ")
    return user_name

def print_Msg(user_name):
    print("Hello", user_name)

def main():
    user_name = get_name()
    print_Msg(user_name)

main()
```

This program uses three subprograms **`get_name()`**, **`print_Msg()`** and **`main()`**.

The **`get_name()`** subprogram will ask the user to input their name and then it will return the value of the variable "user_name" so that it can be used in another subprogram. This is very important. If you do not return the values, then the values of any variables that were created or altered in that subprogram cannot be used elsewhere in your program.

The **`print_Msg()`** subprogram will display the message “Hello” and then the user name. The variable “user_name” appears in the brackets as the current value of the variable is being imported into the subprogram so it can be used.

The **`main()`** subprogram will get the user_name from the **`get_name()`** subprogram (using the variable user_name) as this was returned from the **`get_name()`** subprogram. It will then use that user_name variable in the **`print_Msg()`** subprogram.

The last line “**`main()`**” is the actual program itself. All this will do is start the **`main()`** subprogram running.

Obviously, there is no need to create such a convoluted way of performing what is in fact a very simple program, but this is only used as an example of how subprograms are laid out and variables can be used and passed between the subprograms.

Please note: Python does not like surprises, so if you are going to use a subprogram in a program, Python must have read the “**`def subprogram_name()`**” line before so it knows where to go to find it. If you try to refer to a subprogram before Python has read about it, it panics and will crash. When calling a subprogram, the subprogram must be written **above** the section of code you use to call it. Python will read from the top down and **run** the first line it comes across that has not been indented and does not start with the word def. In the program above this would be **`main()`**.

Example Code

The following examples are all part of the same program and would be displayed in the order shown here.

```
def get_data():
 user_name = input("Enter your name: ")
 user_age = int(input("Enter your age: "))
 data_tuple = (user_name, user_age)
 return data_tuple
```

Defines a subprogram called "get_data()" which will ask the user for their name and age. As we want to send more than one piece of data back to the main program for other parts of the program to use, we have combined them together. The return line can only return a single value, which is why we combined the user_name and user_age variables into a tuple (see page 58) called data_tuple.

```
def message(user_name,user_age):
 if user_age <= 10:
  print("Hi", user_name)
 else:
  print("Hello", user_name)
```

Defines a subprogram called message() which uses two variables that have previously been defined (user_name and user_age).

```
def main():
 user_name,user_age = get_data()
 message(user_name,user_age)
```

Defines a subprogram called main() which obtains the two variables from the get_data() subprogram. These must be labelled in the same order as they were defined in the tuple. It then calls the message() subprogram to run with the two variables.

```
main()
```

Runs the main() subprogram.

Challenges

118

Define a subprogram that will ask the user to enter a number and save it as the variable "num". Define another subprogram that will use "num" and count from 1 to that number.

119

Define a subprogram that will ask the user to pick a low and a high number, and then generate a random number between those two values and store it in a variable called "comp_num".

Define another subprogram that will give the instruction "I am thinking of a number…" and then ask the user to guess the number they are thinking of.

Define a third subprogram that will check to see if the comp_num is the same as the user's guess. If it is, it should display the message "Correct, you win", otherwise it should keep looping, telling the user if they are too low or too high and asking them to guess again until they guess correctly.

120

Display the following menu to the user:

```
1) Addition
2) Subtraction
Enter 1 or 2:
```

If they enter a 1, it should run a subprogram that will generate two random numbers between 5 and 20, and ask the user to add them together. Work out the correct answer and return both the user's answer and the correct answer.

If they entered 2 as their selection on the menu, it should run a subprogram that will generate one number between 25 and 50 and another number between 1 and 25 and ask them to work out num1 minus num2. This way they will not have to worry about negative answers. Return both the user's answer and the correct answer.

Create another subprogram that will check if the user's answer matches the actual answer. If it does, display "Correct", otherwise display a message that will say "Incorrect, the answer is" and display the real answer.

If they do not select a relevant option on the first menu you should display a suitable message.

121

Create a program that will allow the user to easily manage a list of names. You should display a menu that will allow them to add a name to the list, change a name in the list, delete a name from the list or view all the names in the list. There should also be a menu option to allow the user to end the program. If they select an option that is not relevant, then it should display a suitable message. After they have made a selection to either add a name, change a name, delete a name or view all the names, they should see the menu again without having to restart the program. The program should be made as easy to use as possible.

122

Create the following menu:

```
1) Add to file
2) View all records
3) Quit program

Enter the number of your selection:
```

If the user selects 1, allow them to add to a file called Salaries.csv which will store their name and salary. If they select 2 it should display all records in the Salaries.csv file. If they select 3 it should stop the program. If they select an incorrect option they should see an error message. They should keep returning to the menu until they select option 3.

123

In Python, it is not technically possible to directly delete a record from a .csv file. Instead you need to save the file to a temporary list in Python, make the changes to the list and then overwrite the original file with the temporary list.

Change the previous program to allow you to do this. Your menu should now look like this:

```
1) Add to file
2) View all records
3) Delete a record
4) Quit program

Enter the number of your selection:
```