Analytics

CSV Files

CSV files

While text files have their uses, they are not the most suitable for storing data, especially large amounts of data.

A CSV file is essentially like a .txt file. It uses the same Unicode/ASCII structure.

CSV stands for “comma separated values”

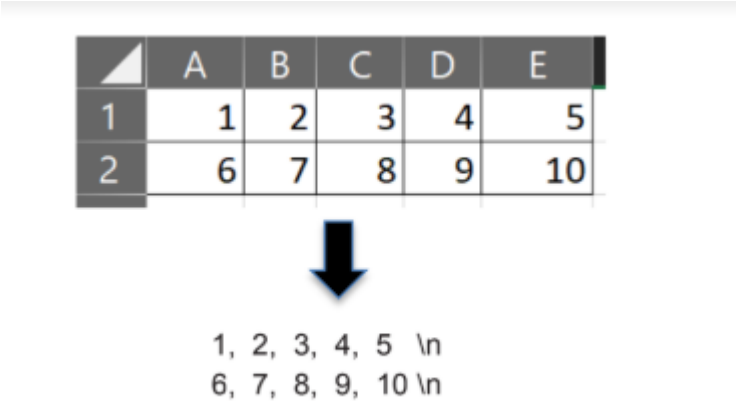
Computer Scientists like this format for the following reasons:

- Quick to read and write to in Python
- Excel has a row limit of 1,048,576 rows
- The file can be used even if you do not have excel on your computer.

CSV Structure

The structure of a CSV file is relatively simple but the format can cause some issues:

- Commas separate elements on a row. These are called delimiters.
- A new line special character “\n” represents the end of a row and the start of a new row.



Reading and Writing to CSV file

Reading and Writing to a csv file can be done the same way as for a text file. You just need to change the file extension.

```
file = open("myFirstCSVFile.csv", "w")
file.write("1,2,3,4,5")
file.close()
```

```
file = open("myFirstCSVFile.csv", "r")
dataIn = file.read()
file.close()
print(dataIn)
```

The output is:

```
1,2,3,4,5
```

The code in the previous slide prints a string for us. This is not really useful if we want to find a total or an average etc.

We know that every element is separated by a comma in a csv file.

We can use this to our advantage and write a loop to go through the string and take out elements with a comma in between them.

This sounds complicated but Python has a method for doing this - it can separate a string into a list based on any specified delimiter (in our case it’s a comma)

```
myList = dataIn.split(",")
print(myList)
```

The output is:

```
['1','2','3','4','5']
```

The .split() method takes the delimiter as an argument.

Notice that all the elements in the list are strings.

We can convert each element to an int so that we can do calculations on them. Python has an easy way to do this.

```
myList = dataIn.split(",")

myList = [int(element) for element in myList]

print(myList)
```

The output is (now all integers):

```
[1,2,3,4,5]
```

Using the csv library

The first step when we want to use any library is to import the library. In this case the library is simply called csv.

```
import csv
```

The next step is learning how to write to a file and then read from the file. The code snippet below will write a set of headings to a file called patients.csv.

```
header = ["firstName","lastName","phoneNum","dob","age"]
file = open("patients.csv","w")
db = csv.writer(file)
db.writerow(header)
file.close()
```

The first line creates a list of the items that we want to add to the file. Using the csv library requires that we enter a list of items. The order of these items is important and needs to be in the same order as any other data entered into the file.

Opening the file is the same as before. Note that I used “w” to open the file, removing any text that was there before.

The following two lines are new. The first line creates a connection to the file that we will be using. db stands for database or where we are going to store our information and this is just a variable name.

The second line is writing our data to the patients.csv file. In this case we are adding all of the headers that we need.

```
db = csv.writer(file)
db.writerow(header)
```

Finally, we close our file, when we are finished with it.

Appending to a csv file

```
record1 = ["Joan", "Byrne", "0981 45877","2/2/75","45"]
record2 = ["Gideon","Jones","098376800","4/7/59","61"]
file = open("patients.csv", "a")
db = csv.writer(file)
db.writerow(record1)
db.writerow(record2)
file.close()
```

Note: use of ‘a’ instead of ‘w’

Order of the attributes are important - they should match order of headings

Reading from a csv file

```
file = open("patients.csv","r")
records = csv.reader(file)

next(records)  #skip the header row
for r in records:
    print(r)

file.close()
```

IMPORTANT:
The csv file MUST be stored in the same location as your python file.

```
#Show specific part only - in this case Name
file = open("patients.csv","r")
records = csv.reader(file)

next(records)  #skip the header row
for r in records:
    print("Firstname:", r[0])

file.close()
```

```
Firstname: Joan
Firstname: Gideon
```

```
#Format the information
file = open("patients.csv","r")
records = csv.reader(file)

print("First Name\t Last Name\t Phone Number\t Date of Birth\t Age")
print("-------------------------------------------------------------------------")

next(records)  #skip the header row
for r in records:
    print(r[0],"\t\t",r[1],"\t\t",r[2],"\t",r[3],"\t",r[4])

file.close()
```

```
First Name        Last Name        Phone Number     Date of Birth    Age
-------------------------------------------------------------------------
Joan              Byrne            0981 45877       2/2/75           45
Gideon            Jones            0938376800       4/7/59           61
```

Appending to a csv with user input

```
#Asking user to input record
fn = input("Enter firstname: ")
ln = input("Enter lastname: ")
ph = input("Enter phone number: ")
dob = input("Enter date of birth in format dd/mm/yy")
age = input("Enter age: ")
db.writerow([fn,ln,ph,dob,age])  #note the square brackets
```

Task 1

- Add three more patient records to the patients.csv file
- Check all the information prints out
- Print out just the first name and last name of the patients

Task 2

1. Create a new file called students.csv
2. Write Python code to add the name, grade and age of three students to the file.
3. Ask the user to input a new student record.
4. Display all the information in the students.csv file in a user-friendly way.