Introduction to Programming/ Week 9: Input and output from users and files/ File handling

File handling

To read in or write out to a text file in Python, you use the `open` function.

Text file input

The example below opens a text file for reading using the function `open`. The `open` function is passed the parameter `"r"` to tell it you want to read from the file. First save in.txt (right click, 'Save as') to the same folder as your Python file (alternatively, make your own file called `in.txt` which has multiple lines of text). Then run this code.

```
f = open("in.txt", "r")
for line in f:
    print(line)
```

Did you notice there is a blank line between each line of the file when it is output this way? Do you know why? This is because each line of the file ends in a newline, and by default `print` outputs the string with a newline at the end also, meaning there are two. To avoid this, we can override the default behaviour of `print` by passing a parameter `end`, as in the following example.

```
f = open("in.txt", "r")
for line in f:
    print(line,end="")
```

Alternatively, you can strip the whitespace (in this case a new line character) from each line before printing it, and let the new line in the output come from `print()`. The following code does this.

```
f = open("in.txt", "r")
for line in f:
    line = line.rstrip()
    print(line)
```

There are a few related functions for removing characters from the ends of a string: `strip()` removes any whitespace from both ends of a string, while `lstrip()` does this for the start (left-hand end) of a string and `rstrip()` does it for the end (right-hand end) of a string. Note that these strip all whitespace (space, tab, newline, etc.), not just the newline character, so if you have spaces or tabs at the ends of your string and don't want to lose them, take care.

If you try to open a file that does not exist, your script will halt with an error. Of course, you can check before trying to open the file whether it exists, but the more straightforward thing to do is to `try` to open the file, and then catch any exceptions that result from a file not being found. Run the following code in a folder that does not contain a file called `peter.txt`

```
try:
    f = open("peter.txt", "r")
    for line in f:
        print(line,end="")
except FileNotFoundError:
    print("Could not find the file peter.txt.")
```

CSV

CSV stands for Comma-Separated Variables, and is a common plain text file format for data which contains one row per line with cells separated by commas.

The following code imports a CSV file, if it exists, removes the newline character from each line and uses the function `split()` to split the string by commas and return the comma-separated parts as a list. Download the file films.csv and put it in the same folder as a Python file containing the following code and run it.

```
try:
    f = open("films.csv", "r")
    for line in f:
        line = line.rstrip('\r\n')
        parts = line.split(",")
        print("{} was released in {}.".format(parts[0],parts[1]))
except FileNotFoundError:
    print("Could not find the file films.csv.")
```

Did you notice that the first line of output was `Film was released in Release Date.`? This comes from reading the header row of the CSV file and is not ideal. You can skip the first line by moving past it before the `for` loop using `next()`.

```
try:
    f = open("films.csv", "r")
    next(f)
    for line in f:
        line = line.rstrip('\r\n')
        parts = line.split(",")
        print("{} was released in {}.".format(parts[0],parts[1]))
except FileNotFoundError:
    print("Could not find the file films.csv.")
```

Dealing with CSV code can become more complicated, for example if the cells are likely to contain commas or multiple lines. Python has a `csv` module you might want to investigate.

Text file output

A similar method is used to write to text files. This time the `open` function is passed the parameter `"w"` to tell it to write to the file. In the code below, if the file `out.txt` doesn't exist, it is created. If it does exist, it is overwritten, so take care. Notice the use of a line break `\n` to add a new line to the file. (You may need to close and reopen the file to see the change, depending on your text editor.)

```
f = open("out.txt","w")
f.write("Hello, World!")
f.write("\nA second line")
f.close()
```

It is also possible to append to (add to the end of) an existing file. Here, the `open` function is passed the parameter `"a"` to tell it to append to the file. Run the code above so that `out.txt` is created. Then run this code to append an extra line.

```
f = open("out.txt","a")
f.write("\nAn extra line added later")
f.close()
```

Outputting data example

The following example generates some random data and then outputs it to a file called `out.csv`. Run it, then open `out.csv` in Excel and see what you have. Study the code and make sure you know what it is doing.

```
from random import randint

# This block generates some random integers and stores them in a multi-dimensi
stuff_to_ouput = []
for i in range(0,10):
    a = []
    for j in range(0,8):
        a.append(randint(0,100))
    stuff_to_ouput.append(a)

# This block outputs the list as a multi-line file using comma-separated varia
f = open("out.csv","w")
for line in stuff_to_ouput:
    for item in line:
        f.write("{},".format(item))
    f.write("\n")
f.close()
```

Exercise

1. Download the file Bring_Recycling_Sites.csv and put it in the same folder as a Python script that imports the file, splits each line by comma and outputs the address of every recycling site. You might like to load the CSV file into Excel to have a look at the format of the data, but please avoid letting Excel save over the file.
2. Adapt your script so that it just prints out the addresses of recycling sites which accept clothes for recycling (hint: look for sites where the columns headed `clothes`, `clothes_shoes` and `clothes_books` are not zero).