CMSC 201 Spring 2017
Lab 09 – File I/O

Assignment: Lab 09 – File I/O
Due Date: **During discussion**, April 10th through April 13th
Value: 10 points (8 points during lab, 2 points for Pre Lab quiz)

This week's lab will put into practice the new concepts you learned about file input so far: **`open()`**, **`read()`**, **`split()`**, **`strip()`**, and more.

(Having concepts explained in a new and different way can often lead to a better understanding, so make sure to pay attention as your TA explains.)

Part 1A: Review – Opening Files

Using files as input is a much quicker and easier way to get information from the user, especially for large amounts of data. Rather than having the user enter everything by hand, we can **read in the data from a file**.

To open a file for reading, we use the following command:

```
    myInputFile = open("theFile.txt", "r")
```

This line of code does three things:

1. It opens the file `theFile.txt`
2. The file is opened for **reading** (`"r"`) – as opposed to writing
    - Writing would use a “`w`” instead
    - If no second parameter is provided, the file is opened for reading
3. The opened file is assigned to the variable `myInputFile`

Part 1B: Review – Reading Information from Files

Once we have opened a file and assigned it to a variable, we can use that variable to access the file. There are four different ways to read in a file.

1. Read the entire file in as one enormous string (including newlines)
   ```
   myInputFile.read()
   ```
2. Read in a single line of the file
   ```
   myInputFile.readline()
   ```
3. Read the file in as a list of strings (each line being a single string)
   ```
   myInputFile.readlines()
   ```
4. Iterate over the file using a `for` loop, reading in a line each loop
   ```
   for singleLine in myInputFile:
       # singleLine contains a line from the file
   ```

Often, if we want to extract or examine data from a file, the last option (using a `for` loop to iterate over the lines of the file) is the most obvious choice.

On the next page, you can see an example where we read in from a file, printing only those lines that are *exactly* 36 characters long.

In this code, we read in from a file, printing only those lines that are *exactly* 36 characters long.

```
inputFile = open("road.txt") # Robert Frost's poem
for line in inputFile:
    line = line.strip()    # remove the newline (and
                           # any other whitespace)
    if len(line) == 36:    # choose the lines to print
        print(line)
inputFile.close()
```

When the file "road.txt" contains the poem "The Road not Taken" by Robert Frost, the output looks like this:

```
Two roads diverged in a yellow wood,
To where it bent in the undergrowth;
And having perhaps the better claim,
Though as for that the passing there
Had worn them really about the same,
In leaves no step had trodden black.
Yet knowing how way leads on to way,
Two roads diverged in a wood, and I—
```

Part 1C (Review) – String Manipulation

This is fine, but often we want to look at the *contents* of a line, and make a decision based on that, rather than on something trivial like the line length.

For example, we may have a file that contains information about our employees and how many hours they worked this week. Using this information, we want to be able to determine which employees are full-time (work 30 hours or more) and which are part-time.

If we know the format of the file we are reading in, we can take advantage of the `split()` function to assign each ***token*** in a line to individual variables. (A token is a set of characters – we don't call it a "word" because it may be numbers, letters, whitespace, or a combination of any of the three.)

If we take a look at the **totalHours.txt** file, we can see that each line is formatted the same: employee id, employee name, and the total hours worked that week. Since we know the format, we can directly assign each piece to a separate variable, and use those variables to help decide which employees are full-time.

```
totalHours.txt
123 Suzy 18.5
456 Brad 35.0
789 Jenn 39.5
101 Thom 28.6
```

One important thing to remember is that all of these variables will be strings to start off – so if we want to use them as integers or floats, we will need to first cast them to be that type.

```
workerHours = open("hours.txt")
for line in workerHours:
    # directly assign each token to a variable
    id, name, hours = line.split()
    # remember to cast to another type if needed
    if ( float(hours) >= 30):
        print(name, "is a full-time employee")
    else:
        print(name, "is a part-time worker")

# don't forget to close the file!
workerHours.close()
```

That code and the totalHours.txt file will give us the following output:

```
Suzy is a part-time worker
Brad is a full-time employee
Jenn is a full-time employee
Thom is a part-time worker
```

By default, the `split()` function uses all whitespace (spaces, newlines, tabs, etc.) as the ***delimiter***. The delimiter is the boundary between each token when the string is being split up. However, we can give it a specific character (or characters) to split on. Here's an example from class:

```
nonsense = "nutty otters making lattes"
nonsense = nonsense.split("tt")
print(nonsense)
# which will output this list of strings:
# ['nu', 'y o', 'ers making la', 'es']
```

This is a bit of a silly example — normally when we choose to split on something that isn't whitespace, we are instead using some other sort of separator character. Using commas, semicolons, and underscores are all common choices, as can be seen in the example code below:

```
courseInfo = "CMSC_201_Fall_2016_Sec_01"
infoList = courseInfo.split("_")
print(infoList)
# which will output this list of strings:
# ['CMSC', '201', 'Fall', '2016', 'Sec', '01']
```

Part 1D (Review) – String Clean-Up

When we use the `split()` function with no parameters, it splits on whitespace. This means that it automatically removes any trailing whitespace (like a newline character) from the end of the string; any leading whitespace is also removed from the start of the string.

If we simply want to remove trailing and leading whitespace, and don't need to use the `split()` function, we can use the `strip()` function instead. It removes all of the whitespace from the start and end of a single string, but leaves all of the interior whitespace intact.

The code below shows the difference between the **split()** and **strip()** functions, and how they behave on a string. (We've printed out underscores on either side so you can "see" the exterior whitespace more easily.)

```
ride = "\tMerry go\t round\n\n"
print("Basic: _" + ride + "_")
print("Stripped: _" + ride.strip() + "_")
print("Split:", ride.split() )
```

This outputs:

```
Basic: _        Merry go         round


_
Stripped: _Merry go          round_
Split: ['Merry', 'go', 'round']
```

Notice that the **strip()** function left the interior tab character alone, but that it removed the tab character from the front, and both of the newline characters from the end. The **split()** function split the string into tokens by removing the interior whitespace, but it also removed all of the leading and trailing whitespace as well.

Part 2: Exercise

In this lab, you'll be writing a program to read in and process a file of information. The information must be read in and printed out in a specific way.

Tasks

Starting:

- ☐ Copy the **`showData.txt`** input file from Dr. Gibson's **`pub`** directory
- ☐ Open the file and examine the contents and the way they're formatted

Programming:

- ☐ Open the file and read in its contents
- ☐ Determine whether the show is confirmed or unconfirmed for renewal
- ☐ Print out information in a formatted table, following the sample output

General:

- ☐ Run and test your code as needed
- ☐ Show your work to your TA

Part 3A: Downloading the Input File

First, create the **`lab9`** folder using the **`mkdir`** command -- the folder needs to be inside your **`Labs`** folder as well.

Next, copy a file into your **`lab9`** folder using the **`cp`** command.

```
cp /afs/umbc.edu/users/k/k/k38/pub/cs201/showData.txt .
```

This will copy the file **`showData.txt`** from Dr. Gibson's public folder into your current folder.

Part 3B: Writing the Program

This program reads in data from the “showData.txt” file, which contains (in this order, and separated only by commas):

- Name of the show
- Year of premier
- Number of seasons already aired
- “yes” or “no” indicating if show has been renewed for another season

Your program will need to read this information in, process whether the show is “confirmed” or “unconfirmed” for a new season, and print the information out for each show.
The information is printed out in a different order than it is read in, so make sure to follow the sample output.

Here is some sample output of the completed program.
There is no user input, and yours should look very similar once it’s done.

```
bash-4.1$ python popularShows.py
Year     # Ssn   New Season?    Show Title
2011     7       confirmed      Shameless
2011     6       confirmed      Game of Thrones
2010     7       confirmed      The Walking Dead
2005     9       unconfirmed    How I Met Your Mother
2016     1       confirmed      Westworld
2008     5       unconfirmed    Breaking Bad
1994     10      unconfirmed    Friends
2005     12      confirmed      Criminal Minds
2013     4       confirmed      Orange Is the New Black
```

Part 4: Completing Your Lab

Since this is an in-person lab, you do not need to use the **`submit`** command to complete your lab. Instead, raise your hand to let your TA know that you are finished.

They will come over and check your work – they may ask you to run your program for them, and they may also want to see your code. Once they've checked your work, they'll give you a score for the lab, and you are free to leave.

Starting:

- ☐ Copy the **`showData.txt`** input file from Dr. Gibson's **`pub`** directory
- ☐ Open the file and examine the contents and the way they're formatted
    - ☐ Show title, year premiered, number of seasons, renewed or not

Programming:

- ☐ Open the file and read in its contents
- ☐ Determine whether the show is confirmed or unconfirmed for renewal
- ☐ Print out information in a formatted table, following sample output

General:

- ☐ Run and test your code as needed
- ☐ Show your work to your TA

IMPORTANT: If you leave the lab without the TA checking your work, you will receive a **zero** for this week's lab. Make sure you have been given a grade before you leave!