CSC108 Files Lab

1 Text Files: the Basics

In this section, you will practice opening, reading, and writing text files. Before beginning, download the text files `data1.txt` and `data2.txt` from the Labs page.

1. Do the following **in the Python shell**:

   (a) Call `open` to open the file `data1.txt`. Since you are working at the shell, you will have to provide a complete path to the file (otherwise Python won't be able to find the file). Associate the open file with a variable called `input_file`.

   (b) Read all of the lines from the file using the `readline` method repeatedly. How many times did you have to call the method? What happened when the file was completely read?

   (c) Open the file again but use a `for` loop to read all of the lines from the file. In the body of the loop, print each line.

   Switch roles

2. Now we are going to write a program in Idle that starts by opening the file `data2.txt`. Save your new python file in the same folder as `data2.txt`. This way Python will be able to find `data2.txt` without you having to provide the path.

   Add a loop to your program that prints only lines from `data2.txt` that contain the string "lol" in any mixture of upper and lower case. (**Hint:** Use `str`'s `find` method to locate "lol", and consider converting text from the file to lowercase first.) Make sure that your output is single-spaced (there should be no blank lines printed).

   Switch roles

3. Modify your program so that it also opens a new text file named `output.txt` for writing. Instead of printing the "lol" lines to the screen, `write` them to the file `output.txt`.

2 File Processing

This section of the lab involves processing temperature data stored in a file. That file happens to be located on the web, but is still considered a file by Python. The only difference is in how you open it: once you open it, you can use familiar file-related methods.

Download `files.py` from the Labs page of the course website. It contains design recipes for functions that you need to implement. These functions analyze temperature data that is in a certain format.

1. Finish implementing the helper function `open_temperature_file`. It receives a URL as a parameter; use `urllib.request.urlopen` to open that URL.

   Call your function to open the following URL, which contains temperature data:
   `http://robjhyndman.com/tsdldata/data/cryer2.dat`
   This data file contains a description at the top ("Average monthly temperatures in Dubuque" etc.), before the actual data. As indicated by the description,, your function `open_temperature_file` should advance past that description in the file, so that when you next read from this URL, you will get actual data.

2. The data is organized so that each row represents data for one year. Each row therefore has 12 numbers corresponding to the months of that year (January, February, March, etc.) We have given you the function `avg_temp_march`. The function looks at the values in the column corresponding to March – run it and make sure that it produces the correct result for the data at the URL above.

   Switch roles

3. Next, implement function `avg_temp`. It will do the same thing as `avg_temp_march`, but it is more general: it finds the average temperature for *any* month, not just for March. Copy your `avg_temp_march` code to the `avg_temp` function and make a small change to it so that it works with any month.

4. Implement the function `higher_avg_temp`. Be careful: this function takes a URL (not a file). So your function should first open that URL.

   Switch roles

5. Implement the function `three_highest_temps`. Call the function with an open file for the URL above.

   Switch roles

6. Implement the function `below_freezing`. Call the function with an open file for the URL above. Show your TA that it works.

3 Analyzing Python Files: bonus material

Switch roles

There is lots to do in this lab, so you may not get to this section. If you do not finish this section you do **not** *need to go back and finish it on your own time; It is bonus material for those students who are finishing the labs quickly. In order to leave early, you must finish this material.*

Python files are just text files with a specific, detailed format. Whenever you run a Python program, the interpreter opens the file, reads it, and executes the commands in the file. The interpreter knows what to do because the Python language follows very strict grammar (or "syntax") rules, and the exact meaning of every type of statement has been defined. For example, the value to the right of a single `=` needs to be evaluated and then assigned the variable named to the left of the `=`. The process of reading a program and breaking it into recognizable pieces is known as parsing. This question asks you to parse a Python file, looking for specific Python structures.

1. Write a function `find_function_names` that accepts an open file as input and returns a list containing all of the function names in the file. Use a `for` loop to process each line in the file. Remember that each function definition is signaled by the keyword `def` and that the function's name is followed by a list of arguments surrounded by parentheses. **Hint:** You may find `startswith`, `find`, and slicing useful.

2. Write test code that calls `find_function_names`. Your test code should open the Python file `files.py` that you wrote for the previous section, call the function `find_function_names`, and then write the list of function names, one per line, to a new file named `function_names.txt`.

   Switch roles

3. Write a second function `get_comments` that accepts an open file as input and returns a list containing each comment or docstring in the file. This problem is tricky, since `'''` potentially indicates a multi-line docstring. Instead of using a for loop to read one line at a time, read the **entire** program into a `str` at the beginning of your function. Then, use `find` to locate all occurrences of `#` and `'''`. The text between a `#` and the end of the line is a comment. The text between two occurrences of `'''` is a docstring. (It could also be used for a multi-line string, but for this exercise assume that all `'''` strings are docstrings.)

4. Modify your test code to run the new function and place the output in a second new file named `comments.txt`.