Assignment 3

Lists, Files, Functions, and Ghosts

SUBMISSION REQUIREMENTS: Submit a single zip file called **assignment3.zip**. It must contain all of your question code, sample data, and sample output. Information regarding deductions for late submissions, invalid submissions, and grade disputes are available on the cuLearn assignment submission page.

This assignment has **50 marks (may change, check marking scheme for up to date information)**.
A full marking scheme is available on the cuLearn submission page (unavailable as of 10/26).
Bonus marks will not exceed 100% on this assignment.

MARKING NOTES: A full marking scheme with notes, deductions, and policies is available on the cuLearn assignment submission page.

(Please note: This is **not a real job posting**, just context for the assignment!)

Job Posting: Raven Ghost Hunting Society

Job Title: Paranormal Instrument Software Designer
Duties: Developing the software for ghost hunting equipment
Requirements: Experience with lists, strings, reading and writing files, and functions

Description: We're the Raven Ghost Hunting Society (RGHS) and we're looking for someone to help automate our sensor readings. We go to abandoned houses and take sensor data wth various machines to try and detect ghosts! Our sensors output their data to files, and each sensor uses a slightly different file format.

We need you to write a program that can read sensor data from our three different sensors, process that data, and output whether or not it's *abnormal*. If a room has abnormal readings in more than one sensor, we're dealing with some kind of entity!

Once you've written the software to analyze the sensor data, please generate a report to specify which types of entities are in different rooms. We'll provide you with some sample data to test against, but beware: the real data might have situations we didn't account for in the test data sent to you! You might wish to **modify the test data** to account for different situations.

Submission Notes:

- **You must make use of a main function**
- **You must have no global variables, except for global constants**
- **Your main function must be run correctly (if __name__ == "__main__": etc.)**
- Each question will be in a different function in the same file
- Your file should be named **assignment3.py**
- **BE AWARE:** You must follow naming and file conventions **precisely**. Capitalization, names of functions, and how the function work must **all work exactly as specified**, otherwise you will face large deductions

Information

- There is a lot of text for this assignment, but this is just to make sure everything is clear for you! Approach each question one at a time.
- **Flowcharts are required.** You do not need to submit the flowchart, but to receive help, you may be required to provide a flowchart.
    - One flowchart per function/question, not for the assignment as a whole
- Problems 1-3 of this assignment ask you to write a function that reads in data from a file line-by-line. Refer to the File I/O lecture material and sample code and the Tools Review lecture where we did a large employee database example with File I/O
- Sample data for each sensor has been provided on cuLearn

Tips

- Remember that most File I/O begins with the following lines to open a file:
```
filename = location + "rest.of.filename.txt"
with open(filename, "r") as f:
    for line in f:
        line = line.strip()
```
    - This opens the file, loops over each line of the file, and removes the newlines and spaces from either side of the line.
- You can check if a value is in a list with the **in** keyword
```
my_list = ["a", "b", "c"]
this_is_true = "a" in my_list
also_true = "d" not in my_list
```
- Any time you need to "remember" information, that sounds like you need a variable!
- Flowcharts are **incredibly useful** for this assignment. *Please* take the time to consider each function's flow before attempting to program it. Focus on solving one problem at a time, getting something small that works, and improving upon it.

Problem 1: Motion Sensor

Submission: Your code should be contained in a function called **read_motion(location_name)** and saved in your submission file as **assignment3.py** and add it to the zip submission.

Question Description:
For this question, you must create a function named **read_motion(location_name)** that takes in the string name of the location we're hunting in and returns a list of rooms with abnormal readings. The file you will read is **location_name.motion.txt**, replacing location_name with the value of the location sent to the function.

You will read the input file line by line, and if the sensor reading lists "detected", keep track of the name of the room in a list. After you've read every line, return the list of rooms where motion was detected. **Do not include the same room more than once.**

What: The motion sensor occasionally turns on to see if anything is moving in the room. It reports "detected" if something moved, or "undetected" if nothing moved.

Input: Each line of the motion detector contains the room, the time it turned on, and whether it detects anything or not. These three pieces of data are separated by commas and no spaces. Example file - bolded is not in the file itself:

File name: house.motion.txt
bathroom,220543,undetected
livingroom,220622,detected
bathroom,220731,undetected

Process: First, create an empty list of rooms to keep track of rooms with motion. Then open the file, and for each line, split the line apart by commas. This gives you a list - the first element is the name of the room, the second is the timestamp - which we don't need, and the third is whether it is detected or not. If motion is detected, the third element will read "detected"

Output: Return a list of strings for each room in the house where motion was detected. There should be no duplicates in the list. Example for the above file:

```
>> read_motion("house")
['livingroom']
```

Problem 2: EMF Sensor (Electromagnetic Fields)

Submission: Your code should be contained in a function called **read_emf(location_name)** and saved in your submission file as **assignment3.py** and add it to the zip submission.

Question Description:
For this question, you must create a function named **read_emf(location_name)** that takes in the string name of the location we're hunting in and returns a list of rooms with abnormal readings. The file you will read is **location_name.emf.txt**, replacing location_name with the value of the location sent to the function.

You will read the input file line by line. If the line is a positive integer - which can be checked with the string's **isdigit()** function, it is a sensor reading for the most recent room we've read in. If it is not a positive integer, it is the name of the room that we are now taking readings from.

You will read in the name of a room and an unspecified number of sensor readings. Keep track of the **current sum** of sensor readings and the **number** of sensor readings for that room. Once the line is no longer a sensor reading, you should take the average of all of the previous room's sensor readings using the sum and number of readings. **If the average sensor reading is strictly greater than 3 (not equal to three),** it is considered abnormal and added to the list of rooms to return. Remember to reset your sum and count to zero when you switch into a new room!

What: The EMF provides integer readings of 0 to 5 to indicate the strength of electromagnetic fields in the area. Of course, this could just be wires, lights, or camera equipment, but if the average EMF reading in a room is over 3, we're *totally* dealing with a ghost of some kind.

Input: Each line is either the name of the room for all subsequent sensor readings *or* a sensor reading. A sensor reading is a positive integer, a room name isn't. Example file - bolded is not in the file itself:

File name: house.emf.txt
```
bathroom
1
2
1
livingroom
5
4
4
```

Problem 2: EMF Sensor (continued)

Process: First, create an empty list of rooms to keep track of rooms with abnormally high EMF readings. Read each line of the EMF data. If it's a positive integer, track the information we need to take an average. If it's not, we should take the average of our data, see if it's abnormal (average > 3), and if it's abnormal, add it to our result list.

Output: Return a list of strings for each room in the house where a >3 average EMF reading was detected. There should be no duplicates in the list. Example for the above file:

```
>> read_emf("house")
['livingroom']
```

Problem 3: Temperature Sensor

Submission: Your code should be contained in a function called **read_temp(location_name)** and **is_valid_temp(val)**, saved in your submission file as **assignment3.py** and included in the zip submission.

Question Description:
For this question, you must create a function named **read_temp(location_name)** that takes in the string name of the location we're hunting in and returns a list of rooms with abnormal readings. The file you will read is **location_name.temp.txt**, replacing location_name with the value of the location sent to the function.

Temperature files are formatted the exact same as EMF - except that each temperature reading is a **positive or negative float**. **isdigit()** only lets us check a string for positive integers. We'll make our own function, **is_valid_temp(val)** that takes in a string and returns **True** if the string is a positive or negative float value, and **False** if it is not.

is_valid_temp(val):
To do this, we'll use **string functions.** There are other ways to accomplish this, but for this assignment, you are expected to use **val.strip()** and **val.split()**.

val looks like "42.5" or "-10.2" if it is a floating value. First, we should remove the **"-"** from the front. Use the **strip()** function for this.

Next, we need to reduce it to only the number before the decimal. By using **split()**, we can get a list of values on either side of the **"."**. The number will be the [0] element of this list.

If the number at position 0 of our split list isdigit(), we know it's a valid number! Return True. Otherwise, return False. Now we can use this to check if a line is a temperature or a room name.

read_temp(location_name):
You will read the input file line by line. If the line is a valid temperature - which can be checked with your **is_valid_temp()** function, it is a sensor reading for the most recent room we've read in. If it is not, it is the name of the room that we are now taking readings from.

A room is considered abnormal if we read **five consecutive temperatures < 0.00** in the same room. Of course it could be a draft or a cold piece of metal, but it's *definitely a ghost.* You need to read line by line and track **how many consecutive lines (one after another) are < 0.00**. If this number is greater than or equal to five (5) for a room, it's abnormal! Track it in our result list. Finally, like all of our other functions, return the list of abnormal rooms.

Problem 3: Temperature Sensor (continued)

What: The temperature reader will output all of its temperatures over time and the room it's operating in.

Input: Each line is either the name of the room for all subsequent sensor readings *or* a sensor reading. A sensor reading is a **valid temperature (positive or negative float, checked with your is_valid_temp(val) function that you wrote)**, and a room name isn't. Example file - bolded is not in the file itself:

File name: house.temp.txt

```
bathroom
25.3
24.3
26.1
livingroom
25.6
23.1
22.3
10.7
gameroom
30.1
20.5
-5.1
-2.4
-10.1
-5.0
-20.1
-9.1
10.4
26.5
```

Process: First, create an empty list of rooms to keep track of rooms with cold spots. Read each line of the temperature data. If it's a valid temperature (remember to write your **is_valid_temp** function for this), it's a sensor reading. If the reading is strictly less than zero, increase the number of consecutively cold readings. If we have 5 or more consecutive readings, add the current room to the list. If the current line is greater than or equal to 0 degrees, reset the number of consecutive readings. If it's a new room, reset the number of consecutive readings.

Output: Return a list of strings for each room in the house where at least 5 consecutive low temperatures were detected. There should be no duplicates in the list. Example for the above file:

```
>> read_temp("house")
['gameroom']
```

Problem 4: Generate a Report

Submission: Your code should be contained in a function called **generate_report(location, motion, emf, temp)** saved in your submission file as **assignment3.py** and included in the zip submission. Include your **ghost_report.location.txt and sample data**

In this problem, you will detect what type of ghost is in each room of the house. There might be more than one ghost in the house, but only at most one ghost in a room at a time.

You will write a function named **generate_report** that takes in one string and three lists: The location name, the list of rooms with abnormal motion detector readings, the list of rooms with abnormal EMF readings, and the list of rooms with abnormal temperature readings.

You will open a file named **ghost_report.location.txt** in **write** mode - replacing the location in the filename with the correct location name. Your task is to compare the lists and see which rooms each list has in common. There are multiple acceptable ways to do this. A simple way is to look at each room in a list and check if that room is **in** another list using the **in** keyword.

Note: If you're interested in using ghost tales from your own culture, feel free! Just include a comment to let the TAs know which ghosts align to which evidence.

In the resulting file, write out which ghosts were in which rooms.

- Motion Detector + EMF + Temperatures = **Poltergeist**
- Motion Detector + EMF = **Oni**
- Motion Detector + Temperatures = **Banshee**
- EMF + Temperatures = **Phantom**

Your report should include the name of the location and the name of the ghost in each room. It does not need to be exactly like this, but should look something like:
File name: ghost_report.house.txt
== Raven Ghost Hunting Society Haunting Report ==
Location: house
Poltergeist in bathroom
Oni in gamesroom

Recap

- Your cuLearn submission should be a single file, **assignment3.zip**
- Your zip file should contain **one Python file, assignment3.py**
- Your zip file should contain the sample data used to test your program
- Your zip file should contain the output of your Question 4 - **ghost_report.ravensnest.txt**
    - You may also include other sample data you write yourself
- Late submissions will receive a 2.5%/hour deduction up to an 8 hour cut-off period
- Invalid submissions (incorrect name, incorrect function names) will receive a 10% deduction immediately
- It is your responsibility to verify the submitted files work correctly - redownload and try them again
- Your submission should contain the following functions:
    - main()
        - Used by you to run and test your functions
    - read_motion(location_name)
        - Question 1, input is a string with the name of the location, output is a list of strings for each room with abnormal readings(no duplicate rooms)
    - read_emf(location_name)
        - Question 2, input is a string with the name of the location, output is a list of strings for each room with abnormal readings (no duplicate rooms)
    - is_valid_temp(val)
        - Question 3 Part 1, input is a string, output is True if the string is a positive or negative number, false if it is not
    - read_temp(location_name)
        - Question 3 Part 2, input is a string with the name of the location, output is a list of strings for each room with abnormal readings (no duplicate rooms)
    - generate_report(location, motion, emf, temperature)
        - Question 4, input is a string with the location name and three lists representing the list of rooms where abnormal activity occurred using each of the three sensors. This function returns nothing, and instead writes a report to a file named **ghost_report.location.txt**, where location is replaced by the location name.