CS 1301 Homework 9

Homework – Functional Programming and Dictionaries!
Due: Wednesday November 30th, before 11:55pm

THIS IS AN INDIVIDUAL ASSIGNMENT!

You should work individually on this assignment. You may collaborate with other students in this class. Collaboration means talking through problems, assisting with debugging, explaining a concept, etc. Students may only collaborate with fellow students currently taking CS 1301, the TA's and the lecturer. You should not exchange code or write code for others. For individual assignments, each student must turn in a unique program. Your submission must not be substantially similar to another student's submission. Collaboration at a reasonable level will not result in substantially similar code.

Scored out of 100 points

Files to Submit:

hw9.py ***(make sure to complete all 4 parts!)***

If you need help, we have several resources to assist you successfully complete this assignment:
- The TA Helpdesk – Schedule posted on class website.
- Email the TA's
- Jay's office hours

Notes:
- **Don't forget to include the required comments and collaboration statement (as outlined on the course syllabus).**
- **Do not wait until the last minute to do this assignment in case you run into problems.**
- If you find a significant error in the homework assignment, please let a TA know immediately.

Part 1 – Averaging Robot Light Values

For the second part of the homework, you will be writing a function to compute an average of the robot light values for a certain period of time. Your function should be named *avgLightValues* and should take in an integer parameter, representing the number of seconds that the following procedure will run (you will need to use timeRemaining).

For the time specified by the given parameter, you must repeatedly perform the following task

- Get the light value
- Move the robot forward for 1 second at a **random** (between 0 and 1, exclusive) speed.

While performing this task, you must keep track of each individual light value for the left, center, and right light sensors in a separate list. Once the time for gathering the light values is finished, you must use the **reduce** function to help determine the average light value for the left, center, and right light values. Hint, it will be quite helpful to keep count of how many times you gathered the light values.

Your function must return a tuple representing the average light values for the left, center, and right light sensors respectively, as floats, over the period of time specified by the parameter.

You may write any additional helper functions to assist with this part of the homework.

Hint: The Random Module

To generate a random number, take a look at the Random Module.

Sample Input/Output

If after running your function the light values were found to be

```
leftValues = [1300, 1240, 1401]
centerValues = [1470, 1403, 1501]
rightValues = [1200, 1402, 1550]
```

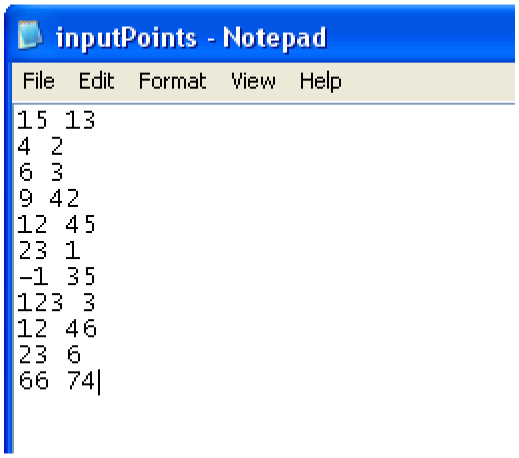
Then your function would return (1313.6666667, 1458.0, 1384.0)

Note, this is only an example showing the output of your function... in actuality you will probably have much more than three light values stored.

Part 2– Calculating Hypotenuse Values

For this part of the homework, you will be finding the hypotenuse of several right triangles and writing the results you find to a new file.
Write a function named *hypCalculator* that takes in two parameters, an input file name and an output file name.

The input file will always have the following format:



```
15 13
4 2
6 3
9 42
12 45
23 1
-1 35
123 3
12 46
23 6
66 74
```

Where each line contains a set of numeric values *(b , h).*The value *b* is the base of the right triangle while the value *h* is the height of the right triangle.

Recall that the formula to find the hypotenuse of a right triangle is
Sqrt($b^2 + h^2$)

Your function must read all the lines of the input file splitting each line into a list, and appending that list (which results from the split) to a "points list" containing all points. For you to correctly implement the map function, you will probably want your "points list" to look something like this:

[[“15”, “13\n”], [“4”, “2\n”], … [“66”, “74\n”]].

You can then apply the distance formula, using the **map** function for each item in the master list.

Successfully applying the map function on the list shown above would yield the following list:

[19.849433241279208, 4.4721359549995796, … 99.156442049924323]

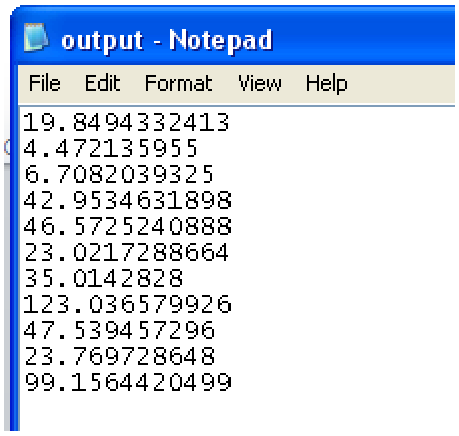
Once you have successfully used the **map** function to calculate the distance between each set of points, you need to write each result to a file that has a name corresponding to the second parameter passed in. Be sure to include a newline between each result.

Sample Input/Output

Assuming the above text file was used as an input file, calling

```
>>> hypCalculator("inputPoints.txt", "output.txt")
```

Would create a file named *“output.txt”* containing:



```
19.8494332413
4.472135955
6.7082039325
42.9534631898
46.5725240888
23.0217288664
35.0142828
123.036579926
47.539457296
23.769728648
99.1564420499
```

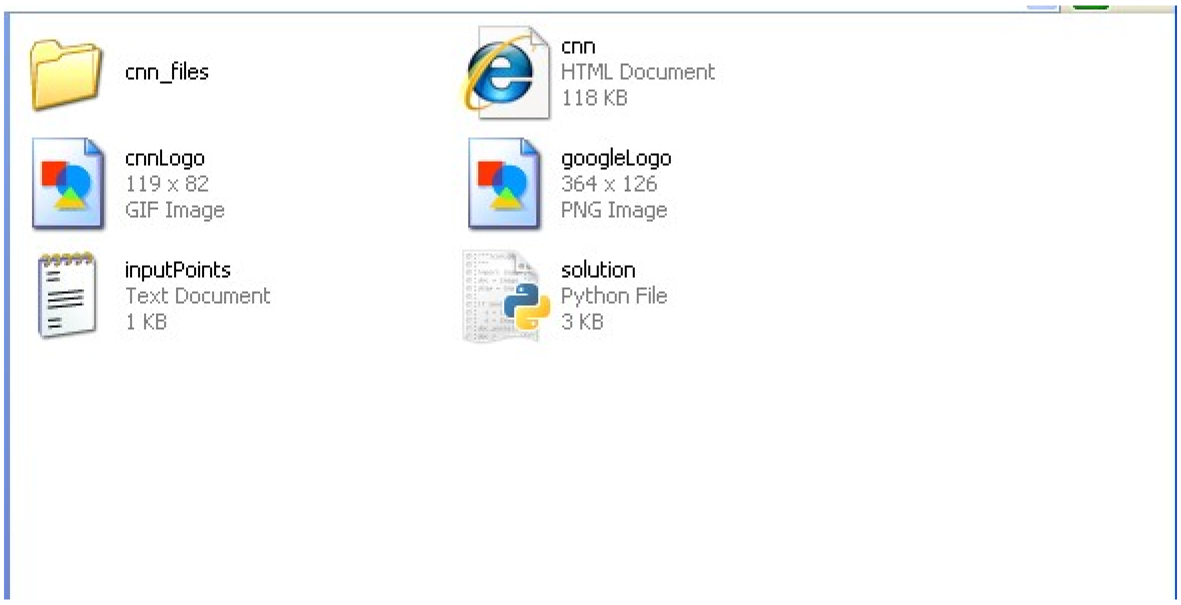
Part 3 – Finding All Images in a Directory
For this part of the homework, you write a function named *findImages*. This function will take in a single parameter, a string containing the path to a directory on the local machine. Your function must obtain a list of every file in the directory and return a list of all the file names that are of image type. You must use Python's built-in **filter** function in order to receive credit. You may also write any additional helper functions as needed.

You can assume the following image extensions are the only ones we will look for: ".jpg", ".gif", ".bmp", and ".png".

Hint: The OS Module
To help with this function, take a look at the OS Module. This module will provide interfaces to operating system features such as files, a clock, and other useful things.

- Entire OS Module Documentation
- Specific OS Module Documentation (which includes the function needed for this part of the assignment).

Sample Input/Output
Assume the following folder exists with the file path
"C:\Documents and Settings\sasghari3\Deskop\myFolder"



Running *findImages* on the specified folder path yields the following result.

```
>>> findImages("C:\Documents and Settings\sasghari3\Desktop\myFolder")
['cnnLogo.gif', 'googleLogo.png']
>>>
```

Part 4 – Extending Finding All Image Files

For this part of the homework, you will be writing a function that takes in a string as a parameter. The parameter should represent a path to a directory on your local machine. Your function will return a dictionary where the keys are the sub-directories inside of the directory represented by the parameter and the associated value for each key is the list image names inside of the the sub-directory. You should also have the directory that is passed in as a parameter be a key in your dictionary. Note that a sub-directory may have additional directories inside of it. Name your function *imagesWithSubdirectories*.

You can assume the following image extensions are the only ones we will look for: ".jpg", ".gif", ".bmp", and ".png".

Sample Input/Output

As an example, imagine the following directory structure (where "/folderX" is a directory and everything else is a file):

```
/folder1
    img1.jpg
    img2.jpg
    img3.gif
    /folder2
        solution.py
        notAnImage.doc
        /folder3
            img4.png
```

Passing "/folder1" as the parameter, you should return:

{"/folder1" : ["img1.jpg", "img2.jpg", "img3.gif"], "/folder1/folder2/folder3" : ["img4.png"]}

Hints

Once again, look at the OS module. You will find the *listdir*, *isdir*, and other functions quite helpful. You may also want to implement a recursive solution to handle sub-directories.

Grading Rubric:

Averaging Robot Light Values (25pts)

• robot moves forward using a random speed (random module)	5pts
• successfully applies reduce function on the right type of data	10pts
• returns a tuple containing the correct average of the light values	5pts
• properly iterates for the right amount of time and successfully grabs light values	5pts

Calculating Hypotenuse (25pts)

• successfully opens/closes, and reads/writes the correct information from the input and output files	10pts
• successfully applies the map function calculate the hypotenuse for each line in the input file.	15pts

Finding All Images (20pts)

• function works for specified image types (.jpg, .png, .bmp, .gif)	5pts
• function correctly uses filter function to find correct file names	10pts
• function returns a list containing only the names with the specified image extensions	5pts

Finding All Images, Extended (30pts)

• function returns dictionary with correct output, following key-value scheme of {"path": [imgPathNameList]}	5pts
• returned dictionary does not return a key for (sub)directories not containing any image files	5pts
• function finds all sub-directories inside the given path	10pts
• function works for specified image types (.jpg, .png, .bmp, .gif)	5pts
• function uses the os module	5pts