CS 2316

Homework 4 – Employee Ranker

Due: Wednesday, September 21st, before 11:55 PM
Out of 100 points

Files to submit: 1. HW4.py

<u>This is an INDIVIDUAL assignment!</u>
Collaboration at a reasonable level will not result in substantially similar code. Students may only collaborate with fellow students currently taking CS 2316, the TA's and the lecturer. Collaboration means talking through problems, assisting with debugging, explaining a concept, etc. You should not exchange code or write code for others.

For Help:

- TA Helpdesk – Schedule posted on class website.
- Email TA's or use T-Square Forums

Notes:

- ***Do not 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.***
- ***Read the entire specifications document before starting this assignment.***

Premise

The Supervisor at the Aquatics department needs to schedule time slots to lifeguards every week according to their availability. To do this efficiently and fairly, they hired an Industrial Engineer to devise a ranking system for priority. He worked with them over the summer and suggested that each employee be given a score for ranking purposes. The IE who devised the system graduated after the summer, and now the Aquatics Department has hired you to write python code to rank all the employees from the CSV data they have. The IE left you a detailed report of the ranking scheme and how the program should be coded.

The ranking scheme is as follows:

- The Rank score of a lifeguard are based on 3 equally weighted categories:
  Position score, **Supervisor evaluation score**, **Peer evaluation score**.

- The employee has one of three positions (or titles): New Lifeguard, Lifeguard, and Head Guard.
  - New Lifeguards are employees that have been employed for less than a semester (16 weeks, or 112 days). <u>New Lifeguards gets 5 points</u> as their Position score.
  - Lifeguards are employees that have been employed for 1 semester or more (112 days or more). <u>Lifeguards get 15 points</u> as their Position score.
  - Head Guards are employees that have been employed for at least a year (365 days or more). <u>Head Guards get 20 points</u> as their Position score.

- Each employee has been evaluated by the supervisor and their peers over the summer, thus each employee now has associated to them a..:
    - <u>Supervisor evaluation score (scale: 0 to 10),</u> and
    - <u>Peer evaluation score (scale: 0 to 5)</u>

- The Rank score is calculated by simply adding up the 3 scores with equal weighting. [Rank score = Position score + Supervisor evaluation score + Peer evaluation score]

- The higher the score, better the rank. The employee with the highest score will be awarded rank 1, the next highest will be rank 2, and so on.

- There could be conflicts where two employees have the same score, but no two employees can have the same rank. To resolve conflicts between any two given employees with the same rank score,
    - Compare their date of employment first. The employee who was employed first (i.e. has an earlier employment date) has higher preference.
    - If they were employed on the same date, compare the Supervisor evaluation score. The employee with higher Supervisor evaluation score has preference.
    - If they were employed on the same day and have the same Supervisor evaluation score and Peer evaluation score, compare the GTID. The employee with a smaller GTID value has preference. [*Note that if they were employed on the same day, have the same Supervisor evaluation, and are in conflict, it means that they have the same Peer evaluation score, so we can't use the Peer evaluation score to de-conflict.*]

Program specifics

The data will be provided to you in CSV format and it will change every month as lifeguards are promoted and scored by their peers and supervisors. SE score is the Supervisor evaluation score, PE score is the Peer Evaluation score, and Days being the Days since employment, the CSV file will have data in the following format:

GTID, Last name, First Name, SE score, PE score, Days

An example record would be:

902345678, Burdell, George, 9, 5, 320

You have to parse the data into a <u>list</u> of <u>lists</u> in the following format:

[[(GTID, Last name, First name), SE score, PE score, Days], …]

An example of the inner list (representing each record) inside the list (that stores all records) would be:

[(902345678, “Burdell”, “George”), 9, 5, 320]

Remember the (GTID, Last name, First name) is a tuple inside each list record.

After you have calculated the Rank Score of each employee, you should append them to the end of the record. The records should now look like:

[(GTID, Last name, First name), SE score, PE score, Days, Rank score]

At this point, you should start ranking them, resolving the conflicts of any employees that might have the same Rank score according to the rules mentioned above. As you give ranks to the employees, they should be stored in a dictionary using the rank as the key and a list with the following information as the value:

[(GTID, Last Name, First name), Position]

This list has two items, a tuple that represents the lifeguard (gtid, lastname, firstname) and their title/position.

Example:

Dictionary[4] → [(902345678, “Burdell”, “George”), “Lifeguard”]

Note that you must convert the Days integer into a textual (string) position/title.

Once you have ranked them and created the dictionary, you should write the records to a new CSV file called “employeeRanks.csv” in the following format:

Rank, GTID, Last Name, First name, Position

Example record that you write out to the file would be:

4, 902345678, George, Burdell, Lifeguard

You will be writing the following functions:

1. parseRecord
2. readCSVFile
3. getPositionScore
4. rankEmployees
5. writeCSVData
6. employeeRanker

Plus any helper functions that you may decide to create and use. You are not required to use helper functions, but you are encouraged to do so if it makes your primary functions easier to understand.

Function Name: **parseRecord**

Parameters:
string – A string which contains one record (one line) from the file.
Return Value:
list – the data structure containing the employee information after adding the new record.

Description:
Write a function that will accept a string parameter which contains the contents of the one line from the file you are reading in the function readCSVFile(). This function will parse the given string from comma separated values into a list of the following format:
[(GTID, Last name, First name), SE score, PE score, Days]
Note that the GTID, SE Score, PE Score, and Days entries will need to be converted to an integer, while the Last name and First Name should stay as strings. You should strip any leading or trailing whitespace from the strings.
The GTID, Last name and First name are in a tuple. This list is a record of one employee and has to be returned by this function. If the data format of this line is invalid (missing needed data), you should raise a ValueError with a: `raise ValueError` statement. This value error will be handled by the employeeRanker function.

Function Name: **readCSVFile**

Parameters:
none
Return Value:
list – the data structure containing the employee information in the above mentioned format.

Description:
Write a function that will ask the user for a valid CSV file to read (for example, “employeeInfo.csv”). In the event that the user provides an invalid file name or the file doesn’t open, you should print a message such as “Invalid file name” and <u>prompt</u> the user to enter another valid file name. You should <u>force</u> the user to enter a valid filename until the file successfully opens. Your function will then read in this file and call parseRecord on each line to parse the information contained within that line. You must call parseRecord inside of readCSVFile to parse each line of the file. Create the new list of records that will be returned by this function. Each record that is returned (as a list) by parseRecord should be appended to this

list. The list that is to be returned should be a collection of lists that represent individual lifeguard records. Remember to close the file when you're done reading in the information. You <u>may not</u> use the CSV reader module; you must implement CSV reading yourself.

Function Name: **getPositionScore**

Parameters:
    int – Number of days the employee has been employed for.
Return Value:
    int – Position Score of the employee.

Description:
Write a function that takes in the days of employment of an employee and finds the corresponding position of the employee. The position of an employee can be one of the following: "New Lifeguard", "Lifeguard", and "Head Guard". New Lifeguards are employees that have been employed for less than 112 days and get 5 points as their position score. Lifeguards are employees that have been working for 112 days or more and get 15 points as their position score. Head guards have worked for 365 days or more and get 20 points as their position score.

Function Name: **rankEmployees**

Parameters:
    list – a data structure containing employee records.
Return Value:
    Dictionary – a data structure containing ranks and employee information.

Description:
Write a function that will accept the list that contains employee information as a collection of lists obtained after reading the file, as a parameter. Use the contents of each record in the list to calculate the Rank score of each employee. (remember that no two employees can have the same rank score!) The getPositionScore function should be helpful in calculating the Rank Score.

You must then append the Rank Score to the end of each record so that each record looks like:
[(GTID, Last name, First name), SE score, PE score, Days, Rank Score]
You should then rank all the employees. Be sure to resolve all conflicts where two employees have the same Rank Score according to the rules given above. No two employees can have the same rank. The details of how you want to resolve the conflicts and where you want to store the ranks are up to you.

You may use the dictionary while you are ranking employees, or transfer the data to the dictionary after you have finished ranking them. You are most welcome to use any helper functions that may assist you in helping rank the employees and resolve the conflicts.

Create a <u>new</u> empty dictionary with the key as Rank, in the following format:

Rank → [(GTID, Lastname, Firstname), Position]
Notice, (GTID, Lastname, Firstname) is still a tuple. Rank, which is a key in the dictionary, is an integer value starting with 1. You will need to use the Position score given in the list to compute the Position of employee that you need to store in the dictionary. Once your dictionary is complete, you should return it. Note that the Position is a textual label, not a position score (number).

Function Name: **writeCSVData**

Parameters:
Dictionary – the data structure that contains the ranks and the employee information.
Return Value:
None

Description:
Write a function that will accept one parameter that is a dictionary with key as Rank of each employee. The dictionary would have the following format:
Rank → [(GTID, Lastname, Firstname), Position]
Use this information to write all the records of this dictionary into a CSV file called "employeeRanks.csv" in the following format:
Rank, GTID, Last Name, First name, Position
Order the records starting at the employee with the smallest rank (1) and moving up in rank as you write the file until you have written out the details for all of the employees. Remember to close the file when you're done writing in the information.

Function Name: **employeeRanker**

Parameters:
None
Return Value:
Bool – True if everything went smoothly, or False if their was an error reading or writing files.

Description:
This is the main function or the driver function of this Ranking code that will make calls to some of the other functions. Your TA will call this function to test your code. This function essentially makes calls to readCSVData, rankEmployees, and then writeCSVData. There are no parameters, but you must return True if the code ran without any problems. If there was an error while reading the file, writing the file, or in the formatting of the given data, the code should not crash. Instead, you should use try and except at various points in this method to make sure all the possible errors are caught. If an error is encountered, return False. (You do NOT have to recover from an error other than if the user enters an invalid file name.)

Testing your Code:

Read from the attached file: “employeeData-Sep-14.csv”
Your output CSV file should look like the attached file: “employeeRanks-Sep-14.csv”

You may create your own CSV data and experiment with your code, as our test data may not exercise all possible situations.
Grading:

You will earn points as follows for each function that works correctly according to the given specifications.

parseRecord		**10**
Properly handles invalid strings by raising a value error	3	
Records correctly formatted data into the list with a tuple inside	7	
readCSVFile		**25**
Properly handles invalid files	5	
Properly reads multi-line data	5	
Properly passes each line into parseRecord function	5	
Formats and appends the records correctly into a list that is returned	5	
Closes file before exiting function	5	
getPositionScore		**5**
Returns the correct Position score	5	
rankEmployees		**30**
Correctly computes the ranks	5	
Appends the Rank Score to each record correctly	5	
Calculates the ranks correctly taking care of all conflicts	10	
Correctly creates and returns the dictionary of ranks	10	
writeCSVData		**20**
Properly opens the file to write	3	
Uses the correct filename	2	
Properly writes the data in the correct format	10	
Closes the file before exiting function	5	
employeeRanker		**10**
Properly makes all the required function calls	3	
Does not let the program crash	5	
Returns True or False correctly in the appropriate situations	2	