Bridgewater State University – Computer Science Dept.

COMP390 – Software Engineering (Fall 2023)

Prof. Joseph Matta

Individual Coding Assignment #1

Overview and Project Requirements

For this assignment, you will write a custom application in the Python programming language. **NO EXTERNAL LIBRARIES/MODULES ARE ALLOWED. ALSO, NO PYTHON STANDARD LIBRARIES! THIS MEANS YOU SHOULD HAVE NO 'IMPORT' STATEMENTS AT THE TOP OF YOUR FILES (EXCEPT FOR IMPORTS OF MODULES YOU WRITE YOURSELF)**. The purpose of this application will be to filter a large data set. The target dataset is a collection of 45,716 data entries. Each entry has information about a single meteorite landing on Earth. The entries consist of the following twelve data fields:

Name id nametype recclass mass (g) fall year reclat reclong GeoLocation States Counties

This dataset is provided to you as a text file ('*meteorite_landings_data.txt*'). Each line of text in the file, *except for the first line*, is an entry describing a single meteorite landing. (Remember: each line ends with a newline character, '\n'.) Each meteorite landing is described by the twelve fields listed above. These fields are "tab-separated" on each line. Meaning that in between each field value string there is a 'tab' character ('\t'). As an example, look at the sample entry line shown below. (This entry was taken from the provided text file):

```
Allegan  2276  Valid  H5  32000  Fell  1899  42.53333  -85.88333 "(42.53333, -85.88333)"  50  429
```

Note that this data entry line consists of twelve values corresponding to the twelve data fields:

name: `Allegan`

id: `2276`

nametype: `Valid`

recclass: `H5`

mass (g): `32000`

…

(and so on) …

There are four important characteristics about these data entry lines to be aware of:

1) Each value is separated by a 'tab' character ('\t')
2) Each line ends with a 'newline' character ('\n')
3) Each field value is read in as a 'string' (including the numerical values)
4) Some field values in a data entry line may be empty. Instead of having a value, empty fields consist of an empty string ('').

Although we have a lot of good data in the text file, it is impossible to sort through by hand. Furthermore, we are not very interested in small meteorites or ones that fell *eleven or more* years ago. Your task is to filter the data separately on the following two parameters:

1) Meteorites that weigh *more than* 2,900,000 grams (*do not* include meteorites that weigh 2,900,000 grams *or less*)
2) Meteorites that fell within the last ten years (*do not* include meteorites that fell *before* 2013. Include meteorites that fell *in 2013 and after*.)

Your program should output (print to the terminal) two separate tables:

Table 1: Name and Mass of all meteorites weighing *more than* 2,900,000 grams

Table 2: Name and Year of all meteorites falling in the year 2013 *or after* (2013-2023, inclusive)

These tables should be **neatly formatted** and printed to the terminal. Each table line should be numbered on the left. *Partial* output examples for the NAME/MASS and NAME/YEAR tables are shown below. Only five (5) table lines are shown for the MASS table and only one (1) line is shown for the YEAR table. The complete tables will have more lines than the partial tables shown below. The purpose of these samples is to show how the tables should be formatted. **To receive full credit for this project both of your tables MUST have the column titles and three columns (table line number, meteor name, MASS|YEAR data.** ("…" indicates that there are more lines in the tables):

```
     NAME                          MASS (g)
=============================================
1    Jilin                         4000000
2    Sikhote-Alin                  23000000
3    Al Haggounia 001              3000000
4    Armanty                       28000000
5    Bacubirito                    22000000
```

…

```
     NAME                          YEAR
=============================================
1    Chelyabinsk                   2013
```

…

To output neatly formatted tables like the ones shown above, use format strings and alignment notation. For example, to get the 'NAME' and 'MASS (g)' column labels to neatly align as shown, use the following lines of code:

```
name_label = 'NAME'
mass_label = 'MASS (g)'
print(f'{name_label:<24}{mass_label:<20}')
```

(*The ": < 24" notation says: "Give me 24 characters worth of space and align the string on the left of that 24-character space."*)

Strategy

There are many ways to approach this assignment. Some are more efficient than others. However, for this assignment the only required output is the correct filter data displayed as two neatly formatted tables printed to the terminal **WITHOUT USING EXTERNAL PROGRAMS, LIBRARIES, OR MODULES**.

I suggest using the following strategy when writing your code:

1) Create a separate Python file in your project called 'meteor_data_class.py'
2) In 'meteor_data_class.py', write a class called 'MeteorDataEntry' that can hold the twelve data field values contained in an entry line. Within the 'MeteorDataEntry' class's __init__ method, make twelve instance variables to hold the data field values.
3) In 'main.py': Open the *meteorite_landings_data.txt* file in READ mode
4) Read in each line of the text file one line at a time (*file_obj.readline()*). (*The very first line of the text file consists of the data field headings. Read this line in first to advance the file reader to the second line. The second line of the text file is the first meteorite data entry.*)
5) For each incoming text line:
    a. Strip the newline character from the end of the incoming line along with any other leading or trailing whitespace (use the string method *strip()*)
    b. After stripping the whitespace, use *split('\t')* to create a Python *list* containing the data field values of the entry line. Use the tab character ('\t') as the parameter for the 'split' method because the data values in the entry line are tab separated. *Remember, fields with no value will read as empty strings*. As a result, elements in the list returned from the 'split' method that have no value will be empty strings. You should now have a list where each element corresponds to a data field value of the incoming line read from the file ([*name_value, id_value, nametype_value, recclass_value, mass (g)_value*…]).
    c. Now check to see if the meteorite being described by the entry line qualifies for one of your filtered lists:
        i. Does the meteorite have a mass greater than 2,900,000 grams?

<u>Yes</u> → create a 'MeteorDataEntry' object and fill its instance variables with the current line's data field values. Then add the newly created object to a list that will hold only 'MeteorDataEntry' objects describing meteorites with masses greater than 2,900,000 grams. Then move on to check the year…

<u>No</u> → move on to the check the year…

ii. Did the meteorite fall in the year 2013 or later?

<u>Yes</u> → create a 'MeteorDataEntry' object and fill its instance variables with the current line's data field values. Then add the newly created object to a list that will hold only 'MeteorDataEntry' objects describing meteorites that fell in 2013 or later. Then move on to the next data entry line in the text file.

<u>No</u> → move on to the next data entry line in the text file.

6) Loop through the list of 'MeteorDataEntry' objects describing meteorites with masses greater than 2,900,000 grams. Extract the meteorite name and mass from each 'MeteorDataEntry' object and print a neatly formatted table to the terminal with the table line number, name, and mass data.
7) Loop through the list of 'MeteorDataEntry' objects describing meteorites that fell in 2013 or later. Extract the meteorite name and year from each 'MeteorDataEntry' object and print a neatly formatted table to the terminal with the table line number, name, and year data.

Submission and Additional Requirements

1) Make a 'README.md' file and add it to your project folder. The README **MUST** include the following information (this is a project requirement!):
    a. your first and last name
    b. your COMP390 section. For example, if I am in section 001, I would write: COMP390-001
    c. a brief overview of what your project does (don't go deep into implementation details here, but provide a complete and succinct summary of your project)
    d. a brief description of what does *not* work in your project and/or what you did not complete (if you completed all of the project requirements, simply write "all project requirements completed" for this part)
2) Create a new GitHub repository named "COMP390_Individual_Project1" (** Make sure your repository is set to PRIVATE **)
3) Push your project to your "COMP390_Individual_Project1" repository
4) Make me (Prof. Matta) a collaborator on your repository. Use my email (j1matta@bridgew.edu) to search for my GitHub account to make me a collaborator. My GitHub username is 'j1matta'. My GitHub profile picture is shown below. Look for this picture when searching for my profile to further confirm that you are indeed making *me* a collaborator and not someone else.

Note: you should push changes made to your project to your GitHub repository often while you are developing your code, not just when you have completed the project. GitHub repositories are a great way to back up your work.

This project is worth 100 points. Please check Blackboard under 'Course Content' for your section's submission deadline. Late submissions will be accepted up to three days past the deadline. However, ten (10) points will be deducted from the total project grade for each day past the submission deadline. After three days, you will receive zero (0) points for the project.

~~~~
~~~~