CSE 231 Spring 2022

Programming Project #8

Assignment Overview

In this assignment, you will practice with dictionaries.

This assignment is worth 50 points (5.0% of the course grade) and must be completed and turned in **before 11:59 PM on Monday, 04/04/2022.**
A 3 pts extra credit will be awarded if submitted by 11:59 PM Sunday 04/03/2022.

Background

An unfortunate fact of the Covid pandemic is that in the US 40% of deaths have been diabetics. Diabetes is a chronic health condition that affects how your body turns food into energy. Most of the food you eat is broken down into sugar and released into your bloodstream. When your blood sugar goes up, it signals your pancreas to release insulin. Insulin acts like a key to let the blood sugar into your body's cells for use as energy. If you have diabetes, your body either doesn't make enough insulin or can't use the insulin it makes as well as it should. Around 5% to 10% of people with diabetes are Type 1. Type 1 diabetes is thought to be caused by an autoimmune reaction (the body attacks itself by mistake) that stops your body from making insulin. It's usually diagnosed in children, teens, and young adults. If you have type 1 diabetes, you'll need to take insulin every day to survive. Currently, no one knows how to prevent type 1 diabetes. In this project, we look into the prevalence of diabetes in countries and regions around the world—using data for people up to age 19 which effectively means type 1 diabetes. We will use data from the International Diabetes Federation: https://idf.org. Note that there are some anomalies/mistakes in the data from the International Diabetes Federation.

Requirements:

1. You will write a program that reads a CSV file, puts the data into a dictionary (actually, a dictionary of lists), and has some functions that will manipulate that `master_dictionary`. There will be a main program that calls the functions.
2. You have to use the following functions plus `main()` in the provided `proj08.py` skeleton. We also provide a `strings.txt` file which contains all the strings that your need for the input and print statements to match Mimir tests. You have to implement and use them as specified:

    a. `open_file()→ file pointer`
        i. Prompts for a file name and continues to prompt until a file is correctly opened. See the provided `strings.txt` file for prompts and error message strings. Hint: use `try-except`. There will be characters in the file python can't understand by default; pass `encoding="utf8"` as an argument to the `open()` function call to fix this:
        `fp = open(filename, encoding='utf-8')`
        ii. Parameters: none
        iii. Returns: `file pointer`
        iv. Displays: prompts & error messages

    b. `read_file(fp)→ dictionary of list of lists`
        i. Read the file referenced by the parameter. Use the `csv` module's `reader` method—see notes. Remember to skip the header line, hint: `next(reader,None).` Each line of the file is data for a region or country. The dictionary key will be the region (a string); the value will be a list of list of data; put the country's information into a `list` and then assign that list to your master dictionary. Remember to initialize the master dictionary to empty before the loop. When you encounter a new region the easiest thing

to do is to initialize that dictionary entry with an empty list so you have something to append your country list to. The country list is formatted this way.
`[country, diabetes, population]`
where `country` is a string and the remaining values are `float`. Note that population has commas that must be removed before converting to a number.
If any value is missing, skip that line. Hint: use `try-except`
Data in the file is in these columns:
`region` is at index 1
`country` is at index 2
`population` is at index 5 (remember to handle the commas)
`diabetes` is at index 9
Open the csv file in a spreadsheet such as Excel to observe what the file looks like.
The dictionary returned should have each list of countries in each region sorted alphabetically by country name (which will be the default because country is the first item in each country list). Hint: use the `.sort()` method rather than the `sorted()` function because the former changes the list and the latter creates a new list which will not be inside your dictionary.

ii. Parameters: `file pointer`
iii. Returns: `dictionary of sorted list of lists`
iv. Displays: nothing

c. `add_per_capita (master_dictionary)` → `master_dictionary`

i. Calculate diabetes per capita for each country and region by simply dividing the diabetes value by the population value (they are both in the same units—thousands). Append the per-capita value onto each country's list. Return the `master_dictionary`, the dictionary of lists. If there is a division by zero error, append `0.0`.
(You may wonder why this was not done in the `read_data()` function. Obviously, it could have been, but it is good practice to limit the complexity of functions—in this case we want the `read_data()` function to only read data, as the name implies. Also, it is a good exercise for you to have a separate function to do the task of calculating a value and adding it to your master dictionary.)
ii. Parameters: `dictionary of lists`
iii. Returns: `dictionary of lists`
iv. Displays: formatted data

d. `max_in_region(master_dictionary,region)` → `tuple`

i. Find the maximum per-capita diabetes in the region. Return the country name and per-capita diabetes value in a tuple.
(For an extra challenge do this in two lines.)
ii. Parameters: `dictionary of lists`
iii. Returns: `tuple (string, float)`
iv. Displays: nothing

e. `min_in_region(master_dictionary,region)` → `tuple`

i. Find the minimum per-capita diabetes in the region. The minimum should be higher than 0. Return the country name and per-capita diabetes value in a tuple.
(For an extra challenge do this in two lines.)
ii. Parameters: `dictionary of lists`
iii. Returns: `tuple (string, float)`
iv. Displays: nothing

f. `display_region(master_dictionary,region)` → `None`
   i. Display the summary data for the `region` (string) followed by a table with the data for each country. The summary data is in the list of country data—its "country name" is the region name. First extract the summary data and print it. Then sort the country data in decreasing order of per-capita diabetes (without the summary region data). See Notes on how to sort by values other than the first in the list. Print the sorted country data in a table. Finally, print the maximum and minimum per-capita values for the region—calling the appropriate functions. See the `strings.txt` file and the `output.txt` files as well as the end of this document for details. Hint: create a list of the country data to sort and then print.
   ii. Parameters: `dictionary of lists, string`
   iii. Returns: nothing
   iv. Displays: table of region data

g. `main()`:
   i. It's the main driver of the program. It doesn't have any parameters. It opens and reads a file and then it calls the other functions.
   ii. Here is the basic structure.
      1. Open and read a file into a `master_dictionary`, a dictionary of lists.
      2. Call `add_per_capita` to add per-capita to the `master_dictionary`
      3. Display the message `"Type1 Diabetes Data (in thousands)"`
      4. Print all the region data in a loop calling `display_region`
      5. Print `"-"*72` after each region
      6. `'\n Thanks for using this program!\nHave a good day!'`

3. Use Coding Standard 1-8

Deliverables

The deliverable for this assignment is the following file:

`proj08.py` -- your source code solution

Be sure to use the specified file name and to submit it for grading via Mimir before the project deadline.

Getting Started

- *Solve the problem using pencil and paper first.* You cannot write a program until you have figured out how to solve the problem. This first step can be done collaboratively with another student. Talk through each function and what it does. However, once the discussion turns to Python specifics and the subsequent writing of Python, you must work on your own.

- Use Anaconda Spyder to open the provided file (`proj08.py`).

- Write a simple version of the program, e.g. open, read the file, and print it so you can test on your own Spyder.

- Use the **Mimir** system to turn in the first version of your program. The tests are written so that if you start with the code we provide, functions will be tested individually.

- Next work on another function.

- Cycle through the steps to incrementally develop your program:
    - Edit your program to add new capabilities.
    - Run the program and fix any errors.

- Use the **Mimir** system to submit your final version.

Notes

1. Find display strings and format strings in the file `strings.txt`
2. Sorting using `itemgetter`. At the top of your program you need:

   ```
   from operator import itemgetter
   This is how you use it to sort by per_capita:
   list.sort(key = itemgetter(3))  # sorts on index 3 of the list
   ```

3. Print tuples using `*list_name`. Assume you have a tuple `L = (2,4,3)`

   ```
   print("{} {} {}".format(*L)
   ```

   is the same as

   ```
   print("{} {} {}".format(L[0], L[1], L[2])
   ```

4. Using the csv module

   ```
   import csv   # put this line at the top of your file
   fp = open("filename")
   reader = csv.reader(fp)
   header = next(reader, None)  #read one line and put it in a header
   variable
   for line_lst in reader:  # iterate through the file
       print(line_lst)      # each line is a list
   ```

Example Interaction

Test 1: one regions

```
Input a file: very_small.csv
Type1 Diabetes Data (in thousands)
Region                                  Cases   Population  Per Capita
South-East Asia                           244      563,552     0.00043

Country                                 Cases   Population  Per Capita
Maldives                                  0.1          134     0.00075
Sri Lanka                                 3.3        6,710     0.00049
Nepal                                     5.6       11,455     0.00049
India                                   229.4      485,465     0.00047
Bhutan                                    0.1          260     0.00038
Bangladesh                                5.9       59,227     0.00010
Mauritius                                 0.0          302     0.00000

Maximum per-capita in the South-East Asia region
Country                               Per Capita
Maldives                                 0.00075

Minimum per-capita in the South-East Asia region
Country                               Per Capita
```

```
Bangladesh                                0.00010

------------------------------------------------------------------------

 Thanks for using this program!
Have a good day!
```

Test 2: two regions

```
Input a file: small.csv
Type1 Diabetes Data (in thousands)
Region                                  Cases   Population  Per Capita
Europe                                    295      221,820     0.00133

Country                                 Cases   Population  Per Capita
Austria                                   3.6        1,756     0.00205
Belgium                                   4.3        2,621     0.00164
Bulgaria                                  1.1        1,337     0.00082
Bosnia and Herzegovina                    0.5          629     0.00080
Albania                                   0.5          680     0.00073
Armenia                                   0.5          791     0.00063
Azerbaijan                                1.9        3,032     0.00063
Belarus                                   1.1        2,085     0.00053
Andorra                                   0.0           14     0.00000

Maximum per-capita in the Europe region
Country                                 Per Capita
Austria                                    0.00205

Minimum per-capita in the Europe region
Country                                 Per Capita
Belarus                                    0.00053

------------------------------------------------------------------------
Type1 Diabetes Data (in thousands)
Region                                  Cases   Population  Per Capita
South-East Asia                           244      563,552     0.00043

Country                                 Cases   Population  Per Capita
Maldives                                  0.1          134     0.00075
Sri Lanka                                 3.3        6,710     0.00049
Nepal                                     5.6       11,455     0.00049
India                                   229.4      485,465     0.00047
Bhutan                                    0.1          260     0.00038
Bangladesh                                5.9       59,227     0.00010
Mauritius                                 0.0          302     0.00000

Maximum per-capita in the South-East Asia region
Country                                 Per Capita
Maldives                                   0.00075

Minimum per-capita in the South-East Asia region
Country                                 Per Capita
Bangladesh                                 0.00010

------------------------------------------------------------------------

 Thanks for using this program!
Have a good day!
```

Test 3: all regions

```
See output files online.
```

Scoring Rubric

Computer Project #8 Scoring Summary

General Requirements

__0__ (5 pts) Coding standard 1-9
(descriptive comments, function headers, mnemonic identifiers, format, etc...)

Program Implementation

__0__ (4 pts) Function Test open_file (no Mimir test)

__0__ (8 pts) Function Test read_file

__0__ (6 pts) Function Test add_per_capita

__0__ (5 pts) Function Test max_in_region

__0__ (5 pts) Function Test min_in_region

__0__ (5 pts) Test case 1: one region (effectively tests display)

__0__ (6 pts) Test case 2: two regions

__0__ (6 pts) Test case 3: all regions

Note:
1. hard coding an answer earns zero points for the whole project .
2. (-10) points for not using main()