CSE 231 **Spring 2020**

Computer Project #7

This assignment focuses on the design, implementation and testing of a Python program that uses lists and tuples.

It is worth 50 points (5% of course grade) and must be completed no later than **11:59 PM on Monday, March 23, 2020).**

Assignment Overview

In this assignment, you will practice with lists and tuples.

Assignment Background

The US Constitution requires a population count (census) every ten years to determine the apportionment of the 435 members of the US House of Representatives. This year (2020) is a census year so this assignment will implement the algorithm for apportionment. Every state must have at least one representative, so the focus of the algorithm is on subsequent assignments of seats to the house. We begin with an ordered list of which state gets the next representative. Then we go through that list assigning seats until all 435 seats are filled. The creation of the ordered list is the interesting part. It is called the *priority list* and it will have 385 items (385 = 435-50). The census web site describes the algorithm here, in particular, how to create the priority list: https://www.census.gov/population/apportionment/about/computing.html

Assignment Specifications

You will develop a Python program that reads in a census file and displays each state and the number of representatives for the state based on that census file. We provide two census files in csv (comma-separated value) format:

- data_2010.csv # state population from the 2010 census
- data_2019.csv # estimated state population for the year 2019

You can open a file to look at it using a spreadsheet program such as Excel. Each file has one header line that must be skipped. The files contain different data, but each has the information we need in the same columns so the same function can read both files.

- Index 1: state # string – remove quotation marks
- Index 2: population # int

Important:

1. The file data_2019.csv also contains population data for two entities with US citizens who do not get a seat in the house: the District of Columbia and Puerto Rico. Ignore the lines in the file with that data (hint: use `continue`).
2. As noted above the state name strings have double quotes in the file, and those must be removed before saving the name in a list.

`open_file () → file pointer:`

a. This function takes no arguments, prompts for a file name and returns the file pointer to that file. The function will keep asking until a file is found. Use `try-except` and `FileNotFoundError`.
b. Parameters: none
c. Returns : file pointer
d. The function displays a prompt and possibly an error message.

`calc_multipliers ()→ list:`

a. This function accepts no arguments and returns a list of multipliers, each calculated as

$$1/\sqrt{n(n-1)}$$

for values of *n* from 2 to 60 in that order (the list will have 59 values).
b. Parameters: `none`
c. Returns : `list of floats`
d. The function displays nothing

`calc_priorities (state, population, list of multipliers)→ list of tuples:`

a. This function accepts three arguments, a `state` (`str`), the state's `population` (`int`), and a list of floats (the multipliers calculated by the `calc_multipliers` function), and returns a list of priorities for the state, each is a tuple with the priority value and the state name:
`(priority value, state name)`
There is one priority value for each float in the list of multipliers so the returned list will be the same length, i.e. 59 values. The list should be sorted in decreasing value of priority.
Algorithm: multiply each multiplier by the state's population to get the priority and convert the value to an int.
b. Parameters: `state (str), population (int), list of multipliers`
c. Returns : `list of tuples (int, str)`
d. The function displays nothing

`read_file_make_priorities (fp,modifiers) → list, list:`

a. This function accepts a file pointer and a list of floats (returned from the `calc_multipliers` function) and returns two lists. It reads each state and its population from the file and creates two lists:
`state_reps`: a list of lists where each list is a state and a count of representatives for the state (initial value is 1), i.e. `[state, count]`. The length of this list is 50, one entry for each state, initialized to 1 because each state has at least one representative. Sort alphabetically on state name.
`priorities`: a list of priority tuples (priority, state) sorted in decreasing order of priority. The length of this list is 385, i.e. one entry for each representative to be added to the states. Hint: put all the states' priority tuples in one big list, sort the list and then use slicing to truncate the list.

b. Parameters: file pointer, list of floats
c. Returns : `list of lists, list of tuples`
d. The function displays nothing.

`add_to_state (state, list of states)→ None`

a. This function accepts as parameters a state (str) and the list of lists where each list is of the form `[state, count]`, representing a state's name and a count of its representatives. The purpose of this function is to find the state in the list of states and add one to its representative count.
b. Parameters: `state (str), list of lists [state, count]`
c. Returns : `None`
d. The function displays nothing.

`display (list of states)→ None`

a. This function accepts as parameters a list of lists where each list is of the form `[state, count]`, representing a state's name and a count of its representatives. Print a header and then the state and its representative count. Print in alphabetical order of state name. Use this format string: `"{:<15s}{:>4d}"`
b. Parameters: `list of lists [state, count]`
c. Returns : `None`

`main():`

This function calls the above functions, first to calculate multipliers , open the file and read the data, then calculate the priority list. Each state in the state-list starts with one representative. Go through the priority list from highest priority to lowest priority, and for each state associated with the priority increment its representative count (using the function `add_to_state`). Then call the `display` function to display the states and their counts. (Note: almost all of the work is done in the functions. The instructor's version of `main()` is less than ten lines of code, not counting comments.)

Assignment Notes and Hints

1. The coding standard for CSE 231 is posted on the course website:

   http://www.cse.msu.edu/~cse231/General/coding.standard.html

Items 1-9 of the Coding Standard will be enforced for this project.

2. The program will produce reasonable and readable output, with appropriate labels for all values displayed.

3. We provide a `proj07.py` program for you to start with.

4. If you "hard code" answers, you will receive a grade of zero for the whole project. An example of hard coding is to simply print the approximate value of e rather than calculating it and then printing the calculated average.

Suggested Procedure

The last version of your solution is the program which will be graded by your TA.

*You should use the **Mimir** system to back up your partial solutions, especially if you are working close to the project deadline. That is the easiest way to ensure that you won't lose significant portions of your work if your machine fails or there are other last-minute problems.*

Assignment Deliverable

The deliverable for this assignment is the following file:

`proj07.py` – the source code for your Python program

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

Function Test: `calc_multipliers`
`Data: data_small.csv`

```
Instructor:
[0.7071067811865475, 0.4082482904638631, 0.2886751345948129,
0.22360679774997896, 0.18257418583505536, 0.1543033499620919,
0.1336306209562122, 0.11785113019775793, 0.10540925533894598,
0.09534625892455924, 0.08703882797784893, 0.08006407690254357,
0.07412493166611012, 0.06900655593423542, 0.06454972243679027,
0.06063390625908324, 0.05716619504750295, 0.05407380704358752,
0.051298917604257706, 0.048795003647426664, 0.046524210519923545,
0.044455422447438706, 0.04256282653793743, 0.040824829046386304,
0.03922322702763681, 0.03774256780481986, 0.036369648372665396,
0.03509312031717982, 0.033903175181040524, 0.032791291789197645,
0.031750031750047626, 0.03077287274483318, 0.029854071701326607,
0.028988551782622423, 0.02817180849095055, 0.02739983121755955,
0.026669037353133248, 0.025976216673306556, 0.025318484177091666,
0.02469323991623974, 0.024098134635593994, 0.023531040266750583,
0.022990024493585143, 0.022473328748774737, 0.0219793491131929,
0.021506619680967013, 0.021053798026662976, 0.020619652471058063,
0.020203050891044214, 0.019802950859533486, 0.019418390934515434,
0.01904848294398648, 0.018692405136401476, 0.018349396085439344,
0.018018749253911177, 0.017699808135119715, 0.017391961901349125,
0.017094641498783945, 0.016807316136320357]
```

Function Test: `calc_priorities`
`Data: data_small.csv`

```
State: Michigan
Population: 9911626
Multipliers: [0.7071067811865475, 0.4082482904638631,
0.2886751345948129, 0.22360679774997896, 0.18257418583505536,
0.1543033499620919]
Instructor:
[(7008577, 'Michigan'), (4046404, 'Michigan'), (2861239, 'Michigan'),
(2216306, 'Michigan'), (1809607, 'Michigan'), (1529397, 'Michigan')]
```

Function Test: `read_file_make_priorities`
`File: data_small.csv`

```
Multipliers:
[0.7071067811865475, 0.4082482904638631, 0.2886751345948129,
0.22360679774997896, 0.18257418583505536, 0.1543033499620919]
Instructor states:
[['Alaska', 1], ['California', 1], ['Kansas', 1], ['Mississippi', 1],
['Texas', 1], ['Wyoming', 1]]
Instructor priorities:
[(28240069, 'California'), (20840059, 'Texas'), (16304411,
'California'), (12032014, 'Texas'), (11528960, 'California'), (8930294,
'California'), (8507918, 'Texas'), (7291554, 'California'), (6590205,
'Texas'), (6162488, 'California'), (5380880, 'Texas'), (4547673,
'Texas'), (2113726, 'Mississippi'), (2057933, 'Kansas'), (1220360,
'Mississippi'), (1188148, 'Kansas'), (862925, 'Mississippi'), (840147,
'Kansas'), (668418, 'Mississippi'), (650775, 'Kansas'), (545761,
'Mississippi'), (531356, 'Kansas'), (519017, 'Alaska'), (461252,
'Mississippi'), (449077, 'Kansas'), (400947, 'Wyoming'), (299655,
'Alaska'), (231486, 'Wyoming'), (211888, 'Alaska'), (164127, 'Alaska'),
(163686, 'Wyoming'), (134009, 'Alaska'), (126790, 'Wyoming'), (113258,
'Alaska'), (103524, 'Wyoming'), (87493, 'Wyoming')]
```

Function Test: `add_to_state`

```
States before:
[['Alaska', 1], ['California', 1], ['Kansas', 1], ['Mississippi', 1],
['Texas', 5], ['Wyoming', 1]]
Adding to states of Texas and Alaska.
Instructor states after:
[['Alaska', 2], ['California', 1], ['Kansas', 1], ['Mississippi', 1],
['Texas', 6], ['Wyoming', 1]]
```

Test 1:

```
Enter filename: data_2010.csv

State          Representatives
Alabama          7
Alaska           1
Arizona          9
Arkansas         4
California      53
Colorado         7
Connecticut      5
Delaware         1
Florida         27
Georgia         14
Hawaii           2
Idaho            2
Illinois        18
Indiana          9
Iowa             4
Kansas           4
Kentucky         6
Louisiana        6
Maine            2
Maryland         8
Massachusetts    9
Michigan        14
Minnesota        8
Mississippi      4
Missouri         8
Montana          1
Nebraska         3
Nevada           4
New Hampshire    2
New Jersey      12
New Mexico       3
New York        27
North Carolina  13
North Dakota     1
Ohio            16
Oklahoma         5
Oregon           5
Pennsylvania    18
Rhode Island     2
South Carolina   7
South Dakota     1
Tennessee        9
Texas           36
Utah             4
Vermont          1
Virginia        11
Washington      10
```

```
West Virginia     3
Wisconsin         8
Wyoming           1
```

Test 2:

```
Enter filename: data_2019.csv

State          Representatives
Alabama           6
Alaska            1
Arizona          10
Arkansas          4
California       52
Colorado          8
Connecticut       5
Delaware          1
Florida          29
Georgia          14
Hawaii            2
Idaho             2
Illinois         17
Indiana           9
Iowa              4
Kansas            4
Kentucky          6
Louisiana         6
Maine             2
Maryland          8
Massachusetts     9
Michigan         13
Minnesota         7
Mississippi       4
Missouri          8
Montana           2
Nebraska          3
Nevada            4
New Hampshire     2
New Jersey       12
New Mexico        3
New York         26
North Carolina   14
North Dakota      1
Ohio             15
Oklahoma          5
Oregon            6
Pennsylvania     17
Rhode Island      1
South Carolina    7
South Dakota      1
Tennessee         9
Texas            39
```

```
Utah              4
Vermont           1
Virginia         11
Washington       10
West Virginia     2
Wisconsin         8
Wyoming           1
```

Grading Rubric

```
Computer Project #07                                Scoring Summary
General Requirements:
  ( 5 pts) Coding Standard 1-9
      (descriptive comments, function headers, mnemonic identifiers,
      format, etc...)

Implementation:
 ( 4 pts)  open_file function (No Mimir tests)

 ( 6 pts)  calc_multipliers function

 ( 5 pts)  calc_priorities function

 ( 9 pts)  read_file_make_priorities function

 ( 5 pts)  add_to_state function

 ( 8 pts)  Test 1  (uses data_2010.csv)

 ( 8 pts)  Test 2  (uses data_2019.csv)


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