Project 3

Due Sunday, March 21 by 11:59pm

For this project you will create a single file that, when run in Python, will perform the following actions in order. Please read **all instructions** before beginning the project.

Below is the list of actions/functions your code should perform/define, in order. However, if you cannot get your code to perform an action then please skip that action and have your code move on to the next action. Partial code that causes an error can be left in your file for partial credit, but should be commented out so that your code will run. Code that doesn't run (i.e. causes an error) will not receive as much partial credit, even if it is "close" to running.

For this project, you will practice interacting with text files as well as dictionaries, as well as function definitions. Often in programming, text files are used to store data which then later can be retrieved and/or analyzed. For this project, you must process some data from a particular type of data file. Take a look at the text file that accompanies this project (the one called t_data.txt). This data file, obtained from an online database, happens to contain the highest recorded temperatures by month and annually for dozens of US cities. You can see that the data is recorded in a specific format. On the left column is the name of each city accompanied by some sort of ID number. After that is an extra bookkeeping identifier, followed by the highest recorded temperatures by month from JAN to DEC, followed by the highest temperature overall in the last column ANN. This is not the only file of this format, others exist as well showing data from different years, lowest temperatures instead of highest temperatures, and so on. Your goal in this project is to create a series of functions that will, in the end, allow the user to parse these specific types of data files, compute averages, and save those averages to new data files.

FUNCTIONS

- Define a function called `func_1` that takes as input the name (as a string) of a single text file. You can assume this text file has the same format as t_data.txt (but you can't assume it has the same name). This function should extract the monthly data contained in the file as a dictionary. The keys of the dictionary should be the city names (you can leave the leading ID number attached), and the values should be the list of all 12 monthly data values. Note that the annual ANN data as well as the extra bookkeeping identifier should NOT be stored (simply skip over them). This dictionary should then be returned.

  In extracting the data, you have many options available at this point in the course and more than one correct solution exists. You might use .read or .readline or .readlines to read the file in the first place, and then you might use string methods (like .split) or string slicing or some other techniques to extract the monthly data. You may use the fact that these data files specifically reserve the first 37 characters of each line for the city name/id, the next 13 characters are reserved for the bookkeeping identifier (which we don't need), and then each 6 characters thereafter are reserved for the monthly and annual data. You may also or instead use the fact that the city names always contain a comma followed by the two-letter state abbreviation, and there is always a dash in the middle of the bookkeeping identifier (and no other dashes or commas ever appear in these data files).

  As an example, printing the output of `func_1` for the given data file t_data.txt should look like this:

```
>>> print(func_1("t_data.txt"))
{'13876BIRMINGHAM,AL                  ': ['    81', '    83', '    89', '    92',
'    99', '   106', '   107', '   105', '   102', '    94', '    88', '    80'],
'03856HUNTSVILLE,AL                   ': ['    77', '    83', '    88', '    92',
'    96', '   106', '   105', '   105', '   101', '    92', '    88', '    79'],
...etc.
```

NOTE: if desired, your code does NOT have to include the extra numbers and whitespace accompanying the city names. It might be easier, however, to leave them in.

- Define another function called `func_2` that takes as input the name (as a string) of a single text file, once again assumed to be a data file in the same format as t_data.txt. This function should compute a new dictionary with keys being the city names and values being the AVERAGE of all 12 monthly data values. This function should make use of `func_1`, which should save you a LOT of coding.

  As an example, printing the output of `func_2` for the given data file t_data.txt should look like this:

```
>>> print(func2("t_data.txt"))
{'13876BIRMINGHAM,AL                   ': 93.83333333333333,
'03856HUNTSVILLE,AL                   ': 92.66666666666667,
'13894MOBILE,AL                       ': 93.66666666666667,
...etc.
```

  NOTE: again, if desired you can omit the extra numbers and whitespace accompanying the city names.

- Define another function called `func_3` that takes as input the names (as strings) of both an input text file and an output text file. The input text file is once again assumed to be a data file in the format of t_data.txt. The output text file should be a file that does not yet exist, but rather this function should create a text file with the specified name and write all the averages computed in the previous function into the text file. Again, notice that this function should make use of and build upon the previous function. Please format your text file as in the example below, and be sure to label the two columns `City` and `Average`.

  As an example, the output file created by `func_3` for the given data file t_data.txt should look like the text file output.txt that is also attached to this project. The first few lines are shown here for convenience:

```
City                                  Average
13876BIRMINGHAM,AL                    93.83333333333333
03856HUNTSVILLE,AL                    92.66666666666667
13894MOBILE,AL                        93.66666666666667
...etc.
```

  NOTE: again, if desired you can omit the extra numbers and whitespace accompanying the city names. Also, keep in mind that in the end your code should work for ANY data file with the same format, and the grader may test your code with different inputs to make sure it works as intended.

Lastly, a few rules that apply to all projects.

- Your project must be submitted to Gradescope as a `.py` file (for instance, `Proj1.py`)
- **Name, UIC email, and Signature:** Please include your name and UIC email at the top of your submission as a comment. You must also copy and paste the following statement as a comment at the top of your file (make sure to put your name at the end):

  > I hereby attest that I have adhered to the rules for quizzes and projects as well as UIC's Academic Integrity standards. Signed: [[your full official name here]]

- ADD COMMENTS to clarify your choice of variables and to indicate what is happening at different steps of the program. The rule of thumb you should use in this course is 1 line of comments for every 3 lines of code. Your comments will be checked to see that you understand how your code is structured and what it is doing in the big picture. Code lacking in comments will be penalized!
- Even without comments, your code should be readable and clear. Avoid unneeded or slow or overly convoluted calculations. For instance, in Python it is often unnecessary to use `range(len(...))`.
- While this won't be strictly enforced, the PEP 8 style guide for Python code gives really really good advice for following proper coding conventions. Take a look at it!