**CSE 231** **Spring 2023**

# Computer Project #07

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

It is worth 65 points (6.5% of course grade) and must be completed no later than **11:59 PM on Monday, April 3, 2023).** After the due date, 1 pt will be deducted for every 1 hr late.
Note that you need to click on the submit button to be considered as submitted.

## Assignment Overview

In this assignment, you will practice with lists of lists and tuples and write a program to answer the questions described below. **You are not allowed to use any advanced data structures other than lists and tuples (e.g., Dictionaries, Sets, etc…). You are also not allowed to use any specialized packages/modules (e.g. statistics, collections, itertool, numpy, pandas, etc…) unless it is specifically specified that you can use it. You should only use the basic packages and what we learned in this class up until week 8 (i.e., up to Lists). The use of any other materials will result in a zero.**

## Assignment Background

In this project, we will continue to use lists to explore weather data from various cities. Building upon previous projects, you will need to use nested lists and navigate them by iterating, indexing, and adding to these lists whenever necessary. The program will require multiple csv files, and you will take these files to create lists of lists. From these lists you will extract information regarding temperature, precipitation, and snow fall. Finally, the main function will require you to make comparisons across all the datasets.

## Assignment Specifications

You will develop a Python program that has the following functions

### `open_files () → list of strings, list of file pointers:`

a. This function first prompts the user to enter a series of cities separated by comma and returns a list of cities and a list of file pointers that correspond to that list of cities. If a particular city is not found, print an error message and do not insert it in the list of file pointers. Note that the cities do not come with the "`.csv`" extension.
b. An example interaction is as follows:
```
Enter cities names: lansing,chicago,paris,fresno
Error: File paris.csv is not found
```
The function returns:
```
[lansing, chicago, fresno],[fp_lansing, fp_chicago, fp_fresno]
```
Where:
`fp_lansing` is the file pointer for `lansing.csv`
`fp_chicago` is the file pointer for `chicago.csv`
`fp_fresno` is the file pointer for `fresno.csv`

c. Parameters: nothing
d. Returns: `list of strings` and a `list of file pointers`
e. Display: prompts

**`read_files (cities_fp) → list of lists of tuples:`**

a. This function takes in a list of file pointers and reads in all the data into a list of lists of tuples. Each inner list corresponds to the data in a csv file. Each row of a csv file will be its own tuple and these rows will be added to the inner list. Then all the lists will be added to form a list of lists. Make sure to ignore the headers in each file (skip the first two lines). Whenever there is a missing value, make sure to substitute a `None` into the list, so that each of the lists inside the big list has the same length (length 6). Do not forget to close all the file after you read all the files.
Each tuple has the following format:
`(Date, TAVG, TMAX, TMIN, PRCP, SNOW, SNWD)`

The type of each element in the tuple:
`(string, float, float, float, float, float, float)`
So for example, if we have these sample 2 files with the following contents:

**lansing.csv**
```
"LANSING CAPITAL CITY AIRPORT, MI US (USW00014836)",,,,,,
Date,TAVG (Degrees Fahrenheit),TMAX (Degrees Fahrenheit),TMIN (Degrees Fahrenheit),PRCP (Inches),SNOW (Inches),SNWD (Inches)
1/1/1948,,26,22,0.84,3.1,3
1/2/1948,,27,22,0.09,0.9,5
11/18/2022,28,31,22,0.12,2.6,5
12/25/2022,17,18,13,0.01,0.4,9
```

**chicago.csv**
```
"CHICAGO MIDWAY AIRPORT, IL US (USW00014819)",,,,,,
Date,TAVG (Degrees Fahrenheit),TMAX (Degrees Fahrenheit),TMIN (Degrees Fahrenheit),PRCP (Inches),SNOW (Inches),SNWD (Inches)
11/17/2000,31,34,28,0,0.1,
11/18/2000,29,35,23,0,,
11/19/2000,33,43,23,0.02,,
11/20/2000,25,29,20,0,0.3,0
```

The returned list of lists of tuples should look like this:
```
[
[('1/1/1948', None, 26.0, 22.0, 0.84, 3.1, 3.0),
('1/2/1948', None, 27.0, 22.0, 0.09, 0.9, 5.0),
('11/18/2022', 28.0, 31.0, 22.0, 0.12, 2.6, 5.0),
('12/25/2022', 17.0, 18.0, 13.0, 0.01, 0.4, 9.0)
],
[('11/17/2000', 31.0, 34.0, 28.0, 0.0, 0.1, None),
('11/18/2000', 29.0, 35.0, 23.0, 0.0, None, None),
('11/19/2000', 33.0, 43.0, 23.0, 0.02, None, None),
('11/20/2000', 25.0, 29.0, 20.0, 0.0, 0.3, 0.0)
]
]
```

b. You may need a helper function - you may name it whatever you like.
Important: the outer list is organized in order of `file pointers` in the `cities_fp` list.
c. Parameter: `list of file pointers`
d. Returns: `list of lists of of tuples`
e. Displays: nothing

**get_data_in_range(data, start_date, end_date)→ list of lists of tuples:**

a. This function takes the list of lists of tuples (`data`) read in `read_files`, extracts/filters the data based on the given dates. The dates for the data are the first element in each tuple and are `strings` in the form `month/day/year (e.g, '5/2/1999')`
   `start_date` and `end_date` are also strings of the form `month/day/year` that help specify the range of dates to look at in `data.`
b. When ranges are specified, keep all the tuples in `data` with dates in – between the start and end dates inclusive).
   For example, suppose `start_date = '1/1/2000' and end_date = '12/31/2001'.`
   We would keep any list with date in between January 1st 2000 and the 31st of December 2001.
   If the list is:
```
[
[ ('5/2/1999' …), ('3/1/2000'…), ('1/1/2000'…), ('12/31/2001'…), ('3/6/2002'…) ],
[ ('11/17/2000'…), ('11/18/2000'…), ('11/19/2000'…), ('11/20/2003'…)]
]
```
   the return would be:
```
[
[('3/1/2000'…), ('1/1/2000'…), ('12/31/2001'…)],
[('11/17/2000'…), ('11/18/2000'…), ('11/19/2000'…)]
]
```

c. You may need a helper function - you may name it whatever you like.
d. You may use the `datetime` python module. To compare dates using the module, you first need to convert the date strings to`.`Then compare these dates using any comparison operators. To convert a string of the form `month/day/year` to a variable of type `datetime:`

```
from datetime import datetime
date_str = '5/2/1999'
date = datetime.strptime(date_str, "%m/%d/%Y").date()
```

e. Parameters: `list of lists of tuples, tuple, tuple, tuple, int`
f. Returns: `list of list of tuples`
g. Display: nothing

**get_min(col,data,cities) → list of tuples:**

This function takes a column index `col`, the list of lists of tuples (`data`) read in `read_files`, and the list of cities returned from the `open_files` function and finds the minimum value of the corresponding column `col` for each city in `cities`. You are allowed to use the `min()` function. Then it returns a list of tuples of the following form:

`[(city,min_value),…]`

For example, suppose we have the following data:
```
data = [[('1/1/1948', None, 26.0, 22.0, 0.84, 3.1, 3.0), ('1/2/1948', None, 27.0,
22.0, 0.09, 0.9, 5.0), ('11/18/2022', 28.0, 31.0, 22.0, 0.12, 2.6, 5.0),
('12/25/2022', 17.0, 18.0, 13.0, 0.01, 0.4, 9.0)],
[('11/17/2000', 31.0, 34.0, 28.0, 0.0, 0.1, None), ('11/18/2000', 29.0, 35.0,
23.0, 0.0, None, None), ('11/19/2000', 33.0, 43.0, 23.0, 0.02, None, None),
('11/20/2000', 25.0, 29.0, 20.0, 0.0, 0.3, 0.0)]]
cities = ['lansing_small', 'chicago_small']
col = 2
```

the function will return:
```
[('lansing_small', 18.0), ('chicago_small', 29.0)]
```

a. You may need a helper function - you may name it whatever you like.
b. Parameters: `int, list of tuples, list of strings`
c. Returns: `list of tuples`
d. Display: nothing

**`get_max(col,data,cities) → list of tuples:`**

This function takes a column index `col`, the list of lists of tuples (`data`) read in `read_files`, and the list of cities returned from the `open_files` function and finds the maximum value of the corresponding column `col` for each city in `cities`. You are allowed to use the `max()` function. Then it returns a list of tuples of the following form:

`[(city,max_value),…]`

For example, suppose we have the following data:
```
data = [[('1/1/1948', None, 26.0, 22.0, 0.84, 3.1, 3.0), ('1/2/1948', None, 27.0,
22.0, 0.09, 0.9, 5.0), ('11/18/2022', 28.0, 31.0, 22.0, 0.12, 2.6, 5.0),
('12/25/2022', 17.0, 18.0, 13.0, 0.01, 0.4, 9.0)],
[('11/17/2000', 31.0, 34.0, 28.0, 0.0, 0.1, None), ('11/18/2000', 29.0, 35.0,
23.0, 0.0, None, None), ('11/19/2000', 33.0, 43.0, 23.0, 0.02, None, None),
('11/20/2000', 25.0, 29.0, 20.0, 0.0, 0.3, 0.0)]]
cities = ['lansing_small', 'chicago_small']
col = 2
```

the function will return:
```
[('lansing_small', 31.0), ('chicago_small', 43.0)]
```

a. You may need a helper function - you may name it whatever you like.
b. Parameters: `int, list of tuples, list of strings`
c. Returns: `list of tuples`
d. Display: nothing

**`get_average(col,data,cities) → list of tuples:`**

a. This function takes a column index `col`, the list of lists of tuples (`data`) read in `read_files`, and the list of cities returned from the `open_files` function and finds the average value of the corresponding column `col` for each city in `cities` rounded to 2 decimals. Then it returns a list of tuples of the following form:

`[(city,average_value),…]`

For example, suppose we have the following data:
```
data = [[('1/1/1948', None, 26.0, 22.0, 0.84, 3.1, 3.0), ('1/2/1948', None, 27.0,
22.0, 0.09, 0.9, 5.0), ('11/18/2022', 28.0, 31.0, 22.0, 0.12, 2.6, 5.0),
('12/25/2022', 17.0, 18.0, 13.0, 0.01, 0.4, 9.0)],
[('11/17/2000', 31.0, 34.0, 28.0, 0.0, 0.1, None), ('11/18/2000', 29.0, 35.0,
23.0, 0.0, None, None), ('11/19/2000', 33.0, 43.0, 23.0, 0.02, None, None),
('11/20/2000', 25.0, 29.0, 20.0, 0.0, 0.3, 0.0)]]
cities = ['lansing_small', 'chicago_small']
col = 2
```

the function will return:
```
[('lansing_small', 25.5), ('chicago_small', 35.25)]
```

b. You may need a helper function - you may name it whatever you like.
c. Parameters: `int, list of tuples, list of strings`
d. Returns: `list of tuples`
e. Display: nothing

**`get_modes(col,data,cities) → list of tuples:`**

a. This function takes a column index `col`, the list of lists of tuples (`data`) read in `read_files`, and the list of cities returned from the `open_files` function and finds the list of modes, or the most common value in the column `col` for each city in `cities.`
b. The returned list needs to be a list of tuples where each tuple consists of the city, the list of the most common values (sorted in ascending order) and the number of times they appear:

   `[(city,[modes],number_of_times),…]`

**Warning:** you are only allowed to use Lists (No other advanced data structure such as Dictionaries or Sets). You are also not allowed to use any specialized packages (such as statistics, collections, itertool, numpy, pandas, etc…).

For example, suppose we have the following data:

```
data = [[('1/1/1948', None, 26.0, 22.5, 0.84, 3.1, 3.0), ('1/2/1948', None, 27.0,
22.0, 0.09, 0.9, 5.0), ('11/18/2022', 28.0, 31.0, 28.0, 0.12, 2.6, 5.0),
('12/25/2022', 17.0, 18.0, 13.0, 0.01, 0.4, 9.0)], [('1/1/1948', None, 26.0,
22.0, 0.84, 3.1, 3.0), ('1/2/1948', None, 27.0, 23.2, 0.09, 0.9, 5.0),
('12/25/2022', 17.0, 18.0, 13.0, 0.01, 0.4, 9.0), ('12/25/2020', 17.0, 18.0,
22.4, 0.01, 0.4, 9.0),('11/18/2022', 28.0, 26.4, 13.17, 0.12, 2.6, 5.0)],
[('11/17/2000', 31.0, 34.0, 28.0, 0.0, 0.1, None), ('11/18/2000', 29.0, 35.0, -
26.0, 0.0, None, None), ('11/19/2000', 33.0, 43.0, 20.0, 0.02, None, None),
('11/20/2000', 25.0, 29.0, -26.0, 0.0, 0.3, 0.0), ('12/25/2020', 17.0, 18.0,
22.4, 0.01, 0.4, 9.0),('11/18/2022', 28.0, 22.0, -26.47, 0.12, 2.6,
5.0),('12/25/2022', 17.0, 18.0, -26.32, 0.01, 0.4, 9.0),('11/25/2022', 17.0,
18.0, -25.97, 0.01, 0.4, 9.0)]]
cities = ['FL', 'CA', 'MI']
col = 3
```

the function will return:

```
[('FL', [], 1), ('CA', [13.0, 22.0], 2), ('MI', [-26.47], 5)]
```

**Algorithm to find modes of list:**

To find how many times a value appears by sorting first and then looping through and counting streaks. A tolerance has been provided for this purpose (TOL = 0.02). The steps are as follows:

`Column = [22.0, 23.2, 13.0, 22.4, 13.17]`

**Step 1:** sort the list in ascending order: `[13.0, 13.17, 22.0, 22.4, 23.2]`

**Step 2:** iterate through the sorted list while counting streaks (i.e, streaks are successive floats that are equals). The lowest float in the streak is considered the representative of the streak. A float `N2` is considered equal to the reference float `N1` if the absolute relative difference is below `TOL` (as provided in the starter code). The condition is defined as follows:

$$\begin{cases} \left|\dfrac{N1 - N2}{N1}\right| \leq TOL, & N1 \neq 0 \\ \text{False}, & N1 = 0 \end{cases}$$

So for example, `N1 = 22.0` and `N2 = 22.4`, then the absolute relative difference is:

`|(22.0 - 22.4)/ 22.0| ≈ 0.0182 < 0.02` which means `N1` and `N2` are equal and the representative of the streak is `22.0`. I repeat checking until I check every number in the list. For example, I created a tuple with the count and the float for each streak to keep track of the streaks.
`[(2,13.0), (2, 22.0), (1,23.2)]`
**Step 3:** filter the floats with the highest occurrence values. List comprehension is useful, but not necessary. A set of data may have one mode, more than one mode, or no mode at all (all values in the data are unique). If there are no modes, then the list of modes for that city is an empty list and the occurrence is 1.

```
column = [13.0, 13.17, 22.0, 22.4, 23.2]--→ [13.0, 22.0], 2
column = [2,2,3,3,4,5,6,6,7]--→ [2,3,6], 2
column = [2,2,3,3,4,6,6,6,7]--→ [6], 3
column = [2,3,4,5,6,7]--→ [], 1
```

c. Parameters: `int, list of list of tuples, list of strings`
d. Return: `list of tuples`
e. Display: nothing

**`high_low_averages(data, cities, categories)→ List of list of tuples:`**

a. This function gets the list of lists of tuples (`data`) read in `read_files`, and the list of cities returned from the `open_files` function, and a list of categories. It will return a list of list of tuples of the highest and lowest average across all cities for each category. To do this, you need to first find the column index that corresponds to each of the categories (Hint: we provide a COLUMNS in the starter code which can be used to find the column index). Then call previously defined function to get the average of the column for each city. Find the cities with highest and lowest average values for each category.
b. The returned list needs to be a list of lists of tuples where inner list corresponds to a category and each tuple consists of the cities with the lowest and highest average. The format for the list is as follows:

```
category_list = [(city, lowest_average), (city, highest_average)]
returned_list = [Category1_list, Category2_list, …]
```

c. Suppose my input looks like this:

```
cities = ['LA', 'NY']
categories = ['low temp', 'snow']
data  =[[('1/1/1948', None, 26.0, 22.0, 0.84, 3.1, 3.0), ('1/2/1948', None,
27.0, 22.0, 0.09, 0.9, 5.0), ('11/18/2022', 28.0, 31.0, 22.0, 0.12, 2.6, 5.0),
('12/25/2022', 17.0, 18.0, 13.0, 0.01, 0.4, 9.0)],
[('11/17/2000', 31.0, 34.0, 28.0, 0.0, 0.1, None), ('11/18/2000', 29.0, 35.0,
23.0, 0.0, None, None), ('11/19/2000', 33.0, 43.0, 23.0, 0.02, None, None),
('11/20/2000', 25.0, 29.0, 20.0, 0.0, 0.3, 0.0)]]
```

For low temp, the highest average among the two is NY at 23.5, and the lowest average goes to LA at 19.75. So we have a list `[('LA', 19.75), ('NY', 23.5)]`
For snow fall, we have `[('NY', 0.2), ('LA', 1.75)]`
Therefore, our return is:
`[[('LA', 19.75), ('NY', 23.5)], [('NY', 0.2), ('LA', 1.75)]]`

d. Parameters: `list of list of tuples, list of strings, list of strings`
e. Return: `list of list of tuples`
f. Display: nothing

## **display_statistics(col,data,cities) → None:**

g. This function takes a column index `col`, the list of lists of tuples (`data`) read in `read_files`, and the list of cities returned from the `open_files` function and displays the summary statistics for each city. You will need to call previously defined functions to get the minimum, maximum, average, and list of modes for all the cities. Display the city first, then the min, max and average. Then it displays the most common repeated values (Modes). The modes need to be separated by commas. For example:

```
lansing:
Min: -7.00 Max: 84.00 Avg: 49.77
Most common repeated values (112 occurrences): 70.0,80.0

anchorage:
Min: -8.00 Max: 72.00 Avg: 40.59
Most common repeated values (131 occurrences): 56.0

lansing_small:
Min: 18.00 Max: 31.00 Avg: 25.50
No modes.
```

h. Parameters: `int, list of list of tuples, list of strings`
i. Return: nothing
j. Display: yes

## **main():**

This function would read all the files and create the main lists of lists of tuples. It would give users 7 different options to select from:

```
1. Highest value for a specific column for all cities
2. Lowest value for a specific column for all cities
3. Average value for a specific column for all cities
4. Modes for a specific column for all cities
5. Summary Statistics for a specific column for a specific city
6. High and low averages for each category across all data
7. Quit
```

Depending on the option selected it would prompt the user for specific inputs and display the result based on the selection. The function would keep prompting the user for option until the user quits.

1- For option 1, prompt the user to enter start and end dates, extract the data that falls within those dates. Prompt for a desired `category` (the program should accept lower and upper cases, e.g. Average vs average), if the `category` is valid (i.e., it is one of the categories in the `COLUMNS` defined in the starter code), display the maximum of that category for each city. Otherwise, display an error message and reprompt for a category again.

```
average temp:
Max for lansing: 84.00
Max for anchorage: 72.00
```

2- For option 2,(1) prompt the user to enter start and end dates, (2) extract the data that falls within those dates, (3) prompt for a desired `category` (the program should accept lower and upper case), (4) if the `category` is valid (i.e., it is one of the categories in the `COLUMNS` defined in the starter code), display the minimum of that category for each city. Otherwise, display an error message and reprompt for a category again.

```
average temp:
Minimum for lansing: -7.00
Minimum for anchorage: -8.00
```

3- For option 3, (1) prompt the user to enter start and end dates, (2) extract the data that falls within those dates, (3) prompt for a desired `category` (the program should accept lower and upper case), (4) if the `category` is valid (i.e., it is one of the categories in the `COLUMNS` defined in the starter code), display the average of that category for each city. Otherwise, display an error message and reprompt for a category again.

```
        average temp:
        Average for lansing: 49.77
        Average for anchorage: 40.59
```

4- For option 4, (1) prompt the user to enter start and end dates, (2) extract the data that falls within those dates, (3) prompt for a desired `category` (the program should accept lower and upper case), (4) if the `category` is not valid (i.e., it is not one of the categories in the `COLUMNS` defined in the starter code), display an error message and reprompt for a category again. If valid, display the most common repeated values (Modes) for the category for each city. The modes need to be separated by commas.

```
average temp:
Most common repeated values for lansing (112 occurrences): 70.0, ,80.0

Most common repeated values for anchorage (131 occurrences): 56.0
```

5- For option 5, (1) prompt the user to enter start and end dates, (2) extract the data that falls within those dates, (3) prompt for a desired `category` (the program should accept lower and upper case), (4) if the `category` is not valid (i.e., it is not one of the categories in the `COLUMNS` defined in the starter code), display an error message and reprompt for a category again. If valid, display the summary statistics for each city: the minimum, maximum, average, and list of modes for all the cities for the category. The modes need to be separated by commas.

```
        average temp:
        lansing:
        Min: -7.00 Max: 84.00 Avg: 49.77
        Most common repeated values (112 occurrences): 70.0,80.0

        anchorage:
        Min: -8.00 Max: 72.00 Avg: 40.59
        Most common repeated values (131 occurrences): 56.0
```

6- For option 6, (1) prompt the user to enter start and end dates, (2) extract the data that falls within those dates, (3) prompt for desired `categories separated by comma` (the program should accept lower and upper case), (4) displays the cities with the highest and lowest average across all cities for each category. For the `categories` that are not valid (i.e., it is not one of the categories in the `COLUMNS` defined in the starter code), display an error message.

```
High and low averages for each category across all data.

        low temp:
        Lowest Average: anchorage = 32.77 Highest Average: lansing = 39.90

        price category is not found.

        snow:
        Lowest Average: lansing = 0.12 Highest Average: anchorage = 0.18
```

7- For option 7, quit the program and displays a goodbye message.

**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 `enumerate()` function is useful to iterate through these data structures to get both the city or category and its index because the index can be used to extract the data for a particular `city` or a column index, depending on the data structure.

```
    for i, tup in enumerate(lst):
```

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

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

5. 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 a value rather than calculating it and then printing the results.

**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 before the project deadline.

**Function Tests:**
Look in the folder Function Tests if you want to test your function locally. Make sure that your code is in the same folder as the function tests and the csv files in your computer.

**I/O Test:**
Look in the folder "I_O_Tests" for Tests 1-7

We provide a zip file with all the files so it is easy to download all the files in your computer.

## Grading Rubric

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

Implementation:
 ( 4 pts)  open_files function (No Coding Rooms tests)

 ( 6 pts)  read_files function

 ( 5 pts)  get_data_in_range function

 ( 4 pts)  get_min function

 ( 4 pts)  get_max function

 ( 4 pts)  get_average function

 ( 8 pts)  get_modes function

 ( 6 pts)  high_low_averages function

 ( 4 pts)  display_statistics function (No Coding Rooms tests)

 ( 3 pts)  Test 1  (option 1)

 ( 2 pts)  Test 2  (option 2)

 ( 2 pts)  Test 3  (option 3)

 ( 2 pts)  Test 4  (option 4)

 ( 2 pts)  Test 5  (option 5)

 ( 3 pts)  Test 6  (option 6)

 ( 3 pts)  Test 7  (errors)


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