Computer Laboratory 03

CSCI 1913: Introduction to Algorithms,
Data Structures, and Program Development

1 Essential information

- This assignment is due Tuesday February 18th at noon and will be turned in on gradescope
- The assignment is intended to be done in pairs, although you're allowed to work on your own if you want.
- You can start the assignment before your lab period, although I recommend not getting too far – the lab TAs often have useful hints, and I don't want you wasting too much time struggling with something that the TAs might announce in-class.
- For more rules see the lab rules document on canvas.

2 Introduction

This laboratory is intended to give you practice with a few core skills:

- Working with dictionaries
- Thinking about common algorithms in different ways
- Working with real-world data files
- Processing lists in various ways

Many of the functions you will be writing this lab are computations you have presumably coded before (finding the max of something, counting how many times something happens, etc) with a twist. For example, computing a sum **separately for each year in the dataset**. You already know how to take a sum, but you will need to think creatively to compute multiple sums at the same time. This will help you train your algorithmic thinking skills and help you think more generically about the core ALGORITHM behind these computations.

3 Files

This lab will involve the following files:

- (REQUIRED, INITIAL CODE PROVIDED) weather.py – you will program (most) of this. This will contain functions to perform basic processing on CSV files.
- (REQUIRED, PROVIDED) weather_tester.py – we will provide this. This file helps you test your solution.
- (OPTIONAL) original.csv – the data file our code is primarily designed to process.

4 Initial Steps

The initial steps for this lab can reasonably be done in collaboration with other groups – but you should make sure to stop before you get to actually coding a solution to the problem.

1. Create a new PyCharm project for this lab.
2. Download all of the provided files and move them into your PyCharm project folder.
3. Begin by looking at the provided data files. Make sure you open them with a text-editor (as the CSV file format can be opened by many different programs. (More info on this file is provided in a future section) Make sure you understand this file's format before moving on.
4. Look at the provided example file – this contains 2 pre-written functions which will (hopefully) help you understand how to manipulate CSV formatted data in python. Try to work out how the *pieces* of what you've seen work – that way you can use those *pieces* again in new and interesting ways! You may also find it useful to comment out some bits of code and run just the provided code.
5. Preview the required functions and work through a few examples using pencil/paper to make sure you understand what behaviors you've been asked to implement.

Remember – all of this can be discussed with other groups, not just your partner. Once you've understood all of this, you're ready to start working on the specific computations we want you to program.

5 Weather Data

For this lab we're going to work with files of historic weather data. This data is often collected for future use, and can be analyzed in many interesting ways. Our specific *primary data file* will be from `https://www.dnr.state.mn.us/climate/historical/daily-data.`

`html?sid=mspthr&sname=Minneapolis/St%20Paul%20Threaded%20Record&sdate=2010-01-01&edate=por`. Note I say "Primary" data file – you should always approach programming problems like this with multiple data files in-mind. Never program to only work for one specific file.

In this lab we will be working with data files stored in the ubiquitous CSV (comma separated values) data format. The CSV data format is one of the oldest formats for storing data tables. Newer formats certainly exist (Microsoft excel and google sheets being two example) but the CSV format is still the most common format for exchanging scientific data tables. An example CSV file is provided below, followed by the same data in a more conventional table layout.

```
"Date","Max_Temperature","Min_Temperature","Precipitation","Snow","Snow_Depth"
"2010-01-01","6.0","-9.0","T","T","9.00"
"2010-01-02","1.0","-15.0","0.00","0.00","9.00"
"2010-01-03","7.0","-14.0","0.00","0.00","9.00"
"2010-01-04","7.0","-10.0","0.00","0.00","9.00"
"2010-01-05","10.0","-9.0","0.00","0.00","9.00"
"2010-01-06","16.0","-4.0","0.00","0.00","9.00"
"2010-01-07","16.0","4.0","0.06","2.20","11.00"
```

This file encodes the following data:

Date	**Max_Temperature**	**Min_Temperature**	**Precipitation**	**Snow**	**Snow_Depth**
2010-01-01	6.0	-9.0	T	T	9.00
2010-01-02	1.0	-15.0	0.00	0.00	9.00
2010-01-03	7.0	-14.0	0.00	0.00	9.00
2010-01-04	7.0	-10.0	0.00	0.00	9.00
2010-01-05	10.0	-9.0	0.00	0.00	9.00
2010-01-06	16.0	-4.0	0.00	0.00	9.00
2010-01-07	16.0	4.0	0.06	2.20	11.00

We normally talk about this data as being organized by rows and columns, with each row representing one piece of data, and each column representing one attribute. For example, each row of this data file would describe one day's weather, with each column representing one attribute (the min and max temperature, or how much rain we got, etc.)

Like most real-world data, this information is *messy* – for example, note that the precipitation for January First 2010 is listed as "T" – according to the source of the data, this means "Trace" as in "there was definitely rain, but not enough to be measured". This is one of 5 possible "special" values they report (M–missing, and S and A for "multi-day values" and blank for unknown). We will have to account for these special values when processing the data. Likewise, you should note that all of these columns are stored as strings (which kind of makes sense, we're reading them from a text-formatted file) – that too will need to be accounted for in our code.

5.1 Reading CSV files

The CSV file format has lasted so long because it is brutally simple. This makes it easy to read in any programming language. Python even has a special software library JUST for reading CSV files:

```
import csv
reader = csv.DictReader(open("friends_file_1.csv"))
for row in reader:
    if row["Min_Temperature"] < 32:
        print("It was freezing on", row["Date"])
```

The csv module (a python built-in module) gives you many different ways to read a CSV file. We will use the `csv.DictReader` class – this allows you to read a CSV file as if it was a list of dictionaries. The above example, for instance, prints the dates where the low temperature was below freezing.

Some notes on this:

- A DictReader can be looped over just like a list, but it is not a list. All it supports is looping – no other behavior. As seen in our example file, it's often useful to simply read the DictReader into a list before other computations to make things easy.
- A single DictReader can only be used once – that is, you get one loop out of it before you need to reopen the file (recreate the reader)
- Each "row" object in the reader is a specialized dictionary, called an OrderedDict – it's a dictionary that remembers what order it's keys go in. You can use it just like it was a dictionary.
- The keys of the row dictionary are the column names found on the first line of the file.
- The values of the row dictionary are the string values for this row (so `row["Date"]` is the a string storing the date of the row, etc.)
- All keys and values returned by DictReader are strings – if your file has non-text data you will need to parse that yourself.

BEFORE YOU BEGIN THE LAB ITSELF Use the provided code in weather.py file to make sure you understand how to work with the data file with this syntax. Not only does this file provide some good examples you can use to make sure you understand, and remind you of some core syntax, but you can also use it as a "playground" to test out different pieces of code. This is something you can work on with other students, even those in other groups, so don't be afraid to ask around or share how you're thinking about this. (Of course the TAs will be there to help too!) Remember, while it can be intimidating to learn a new module, learning a new software module is a common task for a practical programmer – treat this like any other skill-building task.

6 The Work of the Lab

Once you have a sense of how the files work, you should begin the code-generation part of the lab. This must be done strictly in your lab partnerships, with no collaboration between groups. Remember – the maximum group size is 2.

You should take a few steps here:

1. Copy the files from canvas into your workspace.
2. Add empty function definitions for each required function. Now might also be a good time to write docstrings (many programmers find writing function documentation *before* writing the code makes writing the code easier!)
3. Validate that the tester file runs and can load your tests – it won't pass tests yet, but that's OK.
4. Review the functions and think if any one should be programmed first – as labs get longer, later functions may require calling earlier functions to make them work.
5. Program the required functions (each will be detailed below):
   (a) `F_to_C(f_temp)`
   (b) `F_to_C_file(file_list)`
   (c) `clean(file_list, column)`
   (d) `average(file_list, column)`
   (e) `total_rain_by_year(file_list)`

6.1 F_to_C

The temperatures in our data file are stored in Fahrenheit, but many scientists prefer to store data in Celsius. This function is a simple conversion function that takes numbers (measurements) in Fahrenheit and converts them to Celsius using the following equation:

$$C = \frac{5(F - 32)}{9}$$

- The input to this function will always be a number
- The output of this function is a number
- If you find yourself having trouble with this one, talk it over with a TA, this one should be easy and you might be over-thinking it.

6.2 F_to_C_file

The input of this function is a List containing dictionaries as might be returned by the provided "load" function. (I.E. it's a CSV file, stored row-by-row into dictionaries, and then those dictionaries are packed into a list)

This function should modify the list given to it so that all temperature in the file are converted from Fahrenheit to Celsius. This function will have no return value.

Notes

- When initially loaded all data in the file is stored as strings. This includes the temperatures. Therefore you should expect the input temperatures to be stored as strings (I.E. "32" not numbers 32.0). Your code will need to parse these strings into floats in order to then convert the values into Celsius.
- Your code **should not** convert the resulting Celsius numbers back into strings.
- It will be treated as a code problem to avoid using the F to C function here. You should be using that function not copying it's equation.
- The special values such as "T" should not appear in the temperature columns – you do not have to account for them in this function.
- If you're struggling with this function you might be having trouble understnding the input format – Feel free to add print statements to "pull apart" this structure – remember, it's ultimately a list full of dictionaries. Try to make sure you know how to pull a dictionary out of the list, and then pull data out of the dictionary before you get too far into your coding.

6.3 clean

This function takes two inputs:

1. The first input of this function is a list containing dictionaries as might be returned by the provided "load" function.
2. The second input of this function is a string indicating one of the columns.

The clean function will process the file and return a new list containing only "clean" rows of the data. That is, any row of data in which the specified column contains a special value (defined by our data source as "T", "M", "S", "A", or blank ("")) should be removed.

- This function should return a modified copy of the original list, and should not modify the input list.
- Removing elements from a dictionary while you loop over it is notoriously hard to get right. You may find it easier to approach this problem by appending the rows you want to keep to a new list.

- It is OK if this function's return-list and the input list both reference the same dictionaries for each row (I.E. you do not have to also make separate copies of each row dictionary)
- You can assume that the column label given is valid – it is Ok if your code crashes when that column label is invalid.

6.4 average

This function takes two inputs:

1. The first input of this function is a list containing dictionaries as might be returned by the provided "load" function.
2. The second input of this function is a string indicating one of the columns.

The average function should return a mathematical average of the provided column.

- You will need to begin by cleaning the data file using the clean function as some columns may include special values such as "T"
- You cannot assume the list is not empty. If given an empty list (or a list that is empty after being cleaned) return 0.0[1]
- You can assume the column label given is valid – it is OK if your code crashes when given an invalid label.
- You should handle the situation where columns are still in String format. The `float` function can easily handle this, since `float("3.4")== float(3.4)== 3.4` (I.E. the float function can handle strings and numbers without issue).

6.5 total_rain_by_year

The input of this function is a List containing dictionaries as might be returned by the provided "load" function. (I.E. it's a CSV file, stored row-by-row into dictionaries, and then those dictionaries are packed into a list)

The return value of this function should be a dictionary. The returned dictionary should have **ints** as keys, and **floats** as values. The keys will be years, as seen in the data file (so 2010, 2011, etc.) The values will be a sum of rain (Precipitation) over all days in the data file from that year.

Hint the data is in a format like: "2023-02-07" if you have that in a string `date` then `year = date[:4]` will be "2023". You will still need to convert this to an int, but that should be easy. If you're curious how this works – I recommend looking up python slice syntax, it's useful but rarely **required**.

[1] it's important to return specifically 0.0 (the float) not 0 (the int) for how some of our testers work...

7 Grading

Like past labs, this lab will be 50% automated tests and 50% manual checks. The automated tests will be based on the tests seen in the tester file. The manual grading will include 10 points for code style, and 40 points for manual review.

The code style points will follow the same rubric listed in past PDFs – required elements are docstrings for each function, and a comment with your name in each file. Other than that, code style issues will be determined subjectively by your grader, and subject to professor review. Use good variable names, comment tricky lines, and don't abuse whitespace and you'll probably get most or all of these points.

The manual review points will be 4 points for c_to_F and 9 point each for the rest, and will include checking for requirements that are missed (which may not have been caught by the tester), code which clearly fails to meet the requirements and is written to fool the tester, good use of helper functions rather than repeating code, good problem-solving approaches, and partial credit for code that fails tests.

If you have further questions about the grading, please double-check the more detailed grading descriptions in the past labs, and if you are still confused, reach out to course staff over email.

8 Submission

Submitting this assignment will be the same as previous assignments:

- Double check that the code works as required and passes tests. (You will find it easier to debug testing issues on your computer)
- Double check that the required files all contain the name of yourself, and anyone who contributed to the code in that file in a comment at the top. This includes partners you worked with *briefly* but ultimately did not finish the assignment with
- Check that the file is named correctly (weather.py)
- Have a conversation with your partner about doing a joint submission, or separate submissions. If you and your partner are going to turn in the exact same file, do a joint submission on gradescope: `https://help.gradescope.com/article/m5qz2xsnjy-student-add-group`
- Submit the file on gradescope before the deadline.
- Remember that you can resubmit as many times as you want, but only the final submission will be graded.

As always, if you do not finish the assignment in-lab, remember that you are still responsible for completing the lab on your own time and submitting it before the deadline.