CSCI 248 **Homework #8**
Spring 2022 **Due: 10 pm Sunday, March 27**

This assignment contains both written problems (W) and programming problems (P). For written problems, you will **type** your answers into the output file: **`hw8-output.txt`**. Remember to put a blank line between Problem 1 and Problem 2, etc., with all of the problems in order.

Reminder: Make sure you review the academic honesty section of the course syllabus before working on any assignment. For each problem in which you write a program, make sure you include complete header comments, as specified in Homework #1.

Pair Programming:

As discussed previously, occasionally I will allow you to work on specified programming problems with a partner using a concept known as *pair programming*. With pair programming, two people collaborate on writing and debugging a program as follows:

- Both people work together throughout the **entire** development of the problem solution.
- Both people sit together in front of the **same** computer.
- Both people collaborate on the same program.
- The person doing the typing is known as the *driver*.
- The person who is guiding is known as the *navigator* and is an **active participant** (not just an observer) in the writing and debugging of the program.
- The driver and navigator trade places roughly every 30 minutes or when changing problems.

Studies have shown that pair programming leads to better code since two sets of eyes are watching for errors or omissions, and the discussion process leads to better understanding of the program.

In a course environment, the danger of pair programming is that one student in a pair may let the other student do all of the work. That is **not** part of the pair programming concept! Pair programming involves a *collaboration* of both people.

To make sure that everyone in this course masters the material, almost all homework problems must be done on an individual basis. As discussed in the course syllabus, the general rule is that all work on homework problems must be **entirely** your own. Problems specifically designated as pair programming problems are the **only** exception to this rule. However, if you violate the rules of pair programming on an eligible problem, you have cheated and will be penalized accordingly.

For this assignment, I will allow you to perform pair programming with a CSCI 248 partner on Problem 3 — and only that problem. You may **not** collaborate on the other problems.

The rules for pair programming in this course are as follows:

- You may use pair programming only if a programming problem specifically allows it.
  - Pair programming is not allowed for written or extra credit problems.
- You may work with only **one** partner who must be a current CSCI 248 student.
- **Both** names must go in the header comments for each program developed using pair programming.

- The work must be a true collaboration.
- By submitting a program with your name on it, you state that you participated *fully* in its **entire** development and understand **every** line of code
- For electronic submissions, ***one*** member of the pair submits the files electronically.
- For the output, **both** partners must put the program output in their output file.
  - The driver should email the output to the other person, so that **both** will have the output in their individual output files submitted for grading. (Be careful that the email/copy/paste process does not ruin the formatting of the output.)
  - **In the preface to the output for that problem**, both partners should put a comment that the output is the result of pair programming, and list both partners' names so it is easy for me to find the correct Python file in the electronic submissions.
- If a paired problem is late, both partners suffer the late penalty.
- Pair programming is entirely optional — you may develop the programs on your own as usual.

Note that I may revise or clarify these rules at any time. Please contact me if you have any questions as to what is or is not allowed in terms of collaboration.

Problem 1: (W) Given the function below, explain in plain English what it does. For example: "The function returns the sum of the elements in the list." Note that your description should be short and to the point. Do **not** give a line-by-line explanation of the code.

```
def mystery(lst):
    if len(lst) == 0:
        return

    first = lst[0]
    for i in range(len(lst)-1):
        lst[i] = lst[i + 1]
    lst[-1] = first
```

Hint: You may find it helpful to write out a sample list to see what the code does. Note that you want to practice reading code like this **without** having to run it.

NOTE: Problems 2 and 3 do NOT involve lists!!

Problem 2: (P) Reading San Antonio Spurs data

Copy the file **`~lwilson/python/homework/Spurs-19-20.txt`** to your current directory.

The file **`Spurs-19-20.txt`** contains information from the abbreviated 2019-20 season of the San Antonio Spurs. Each line contains the number of points scored by the team for a game. For example, the first lines of the file are as follows:

```
120
124
113
97
```

We want a program that **prompts** the user for minimum and maximum score values (inclusive). The program then will print the number of games in which the Spurs' score was in that inclusive range. For example, here is the last line of output from three **different** runs of the program:

```
The Spurs scored between 110 and 120 points 23 times.

The Spurs scored between 80 and 150 points 71 times.

The Spurs scored between 104 and 104 points 3 times.
```

Note that your program must **prompt the user** for the minimum and maximum values. It also must perform **input validation** to make sure that the minimum ≤ maximum. If the inputs are not valid, the program must keep prompting for **both** inputs until they are valid. For example, inputs of 120 and 110 are not valid, but inputs of 110 and 120 are valid.

Note carefully that the same value **may** be entered for both the minimum and maximum; in that case, the user wants to know how many games matched that **exact** value in terms of the score. (This is not a special case; correct code will work as long as min ≤ max.)

Remember to use clear, efficient conditions. As is true for all file I/O problems, your code must handle the appropriate exceptions.

Problem 3: (P) Reading records from a data file

Copy the file **`~lwilson/python/homework/scores.txt`** to your current directory.

The file **`scores.txt`** contains records from the recent Hogwarts video game tournament. For example, consider the first six lines of the file.

```
Hermione Granger
39537
Sirius Black
36200
Ron Weasley
27659
```

These tell us that Hermione had a score of 39,537, Sirius had a score of 36,200, and Ron had a score of 27,659. The file continues with entries for the other players.

Now, be aware that this problem involves file I/O but **not** lists. I say that because you are starting to learn about lists and might assume that lists are involved, but lists are **not** involved in this problem or Problem 2!

Your task is to write a program that displays the name and score for all players who achieved a minimum score of 30,000 points.

- Use a **defined constant** for the minimum score since we might want to update that in the future. Remember to use it in all appropriate places, including the instructions.
- Use the **`while`** loop approach for reading records from a file.
- Your program must process data with the appropriate type(s), and handle I/O and conversion errors appropriately as we did in class.
- Your program must display output in nicely-formatted columns as shown below:

```
This program displays information from the Hogwarts
video game tournament. All players with a score of
30000 or higher are listed below.

Hermione Granger       39537
Sirius Black           36200
Regulas Black          35634
Lavender Brown         35896
Albus Dumbledore       34388
Lee Jordan             30440
Remus Lupin            33753
Harry Potter           35073
Severus Snape          31898
Katie Bell             30214
```

Remember to **use** the constant here

Nicely-aligned columns

Hints: In the Chapter 6 class examples, **`read_records.py`** shows how to read each part of the record **before** attempting to process the record. It is important that we read *entire* records when moving through the file. Note that we will assume that the file contains complete records, meaning that if the first line of the record is present, the remaining lines also will be present. Remember to use **`rstrip`** appropriately.

Problem 4: (P) Writing a list function

Copy the file **`~lwilson/python/homework/alt_sum.py`**, to your current directory.

Write a function called **`alternating_sum`** that takes one parameter called **`num_list`** representing a list of integers, and add the function to the file. The function must return the *alternating sum* of the elements in the list. For example, if **`num_list`** represents [1, 4, 9, 16, 9, 7, 4, 9, 11], the sum returned by the function must be –2 since 1 – 4 + 9 – 16 + 9 – 7 + 4 – 9 + 11 = –2. Your function must include a suitable **docstring**.

Hint: My solution accesses the list elements **by index**.

Run the program to test your function. Remember to add complete header comments before submitting your work.

Challenge Problem #6[1]: (P) Processing name data

Copy the following files to your current directory.

`~lwilson/python/homework/boys_names2000s.txt`

`~lwilson/python/homework/girls_names2000s.txt`

These two files contain the top 200 boys' and girls' names, respectively, from the Social Security Administration for 2000-2009. The first line in each file represents the most popular name, so Jacob and Emily were the most popular names in that decade. Each file is ordered in terms of popularity, so the last name in each file ranked #200.

[1] Remember that Challenge Problems are numbered throughout the course, which is why this is Challenge Problem #6.

Write a user-friendly program that prompts the user for a name and then searches **both** files for the name. (Some names can be given to boys or girls, which is why you must search both files.) Your program then will print the results in the following format.

```
Jordan was #38 for boys’ names.
Jordan was #75 for girls’ names.

Taylor did not appear in the list of boys’ names.
Taylor was #17 for girls’ names.
```

Hint: Since you will have two input files, call them **`in_file1`** and **`in_file2`**, even though your program will read them one at a time. That is, you will have one loop for **`in_file1`** and another loop for **`in_file2`**.