CSCI 1133, Spring 2023
Homework 09
Due: 11:55 pm, Tuesday March 28, 2023

Due Date: Submit your solutions to Gradescope by 11:55 pm, Tuesday, March 28. Do not upload anything to Canvas and PLEASE be sure to use proper naming conventions for the file, classes, and functions. We will NOT change anything to run it using our scripts.

Late Policy: Submissions will be accepted up to 24 hours late (11:55 pm on Wednesday) for a 1 point penalty, up to 48 hours late (11:55 pm on Thursday) for a 2 point penalty, or up to 72 hours late (11:55 pm on Friday) for a 3 point penalty. Submissions past 11:55 pm on Friday will not be accepted.

Unlike the computer lab exercises, this is not a collaborative assignment. See the syllabus and read section "Academic Dishonesty" for information concerning cheating. Always feel free to ask the instructor or the TAs if you are unsure of something. They will be more than glad to answer any questions that you have. We want you to be successful and learn so give us the chance to help you.

Purpose: The purpose of this assignment is to introduce user-defined functions and documentation, essential concepts for building larger programs in a reasonable and organized manner.

Instructions: This assignment consists of 3 problems, worth a total of 30 points. You must create a Python file called `hw09.py`, and submit it to Gradescope. If you have questions, ask!

Because your homework file is submitted and tested electronically, the following are very important:

- Submit the correct file, hw09.py, through Gradescope by the due date.
- Your program should run without errors when we execute it using Python 3.

Submit your work to Gradescope under the Homework 09 assignment. Make sure that your file is called hw09.py: if you don't match the file name exactly then the testing script won't work and you'll lose points.

The following will result in a score reduction equal to a percentage of the total possible points:

- Incorrectly named/submitted source file or functions (20%)
- Failure to execute due to syntax errors (30%)
- Bad coding style (missing documentation, meaningless variable names, etc.) (up to 30%)

Some of the points in this assignment are given based on the Gradescope autograder, so doing something that breaks the gradescope tests (naming your file incorrectly, naming your function incorrectly, having syntax errors, or using the input() function somewhere the autograder doesn't expect) are likely to lose you all of those points.

Documentation:
Use the following template for EVERY function that you write. The docstring below should be placed inside of the function, just after the function signature.

```
'''
Purpose:
Parameter(s):
Return Value:
'''
```

Download the hw09files.zip folder from Canvas, extract the sample files. You'll want to create your hw09.py file within the extracted hw09 folder.

Problem A. (10 *points*) **Total Time**

Write a function `total_time(fname, employee)` that takes in a string `fname` representing the name of a file in the same folder which contains data on how many hours various employees reported working in a given week, and a string `employee` representing the name of one of the employees. The file will be formatted as follows: Each line will have an employee name, then a space, then the number of hours worked.

It is possible that a name will appear multiple times in a file, since some employees may report hours worked more frequently than once a week. If this happens then the value returned should be the sum of all hours reported by the employee.

You must return a float representing the total number of hours worked by the designated employee.

- If the employee does not report any hours during the week, the function should return 0.0.
- If the file requested does not exist within the current folder, the function should return -1.0, without causing any runtime errors.
    - Use a try-except block to avoid a FileNotFoundError in the case that the specified file does not exist.
- Be sure to close the file before you return from the function.

Examples (assumes that fakefile.txt does not exist within your current directory):

```
>>> total_time('hours1.txt', 'Leslie')
22.0

>>> total_time('hours2.txt', 'Jeff')
24.0
```

```
>>> total_time('hours3.txt', 'Tian')
0.0

>>> total_time('fakefile.txt', 'Nobody')
-1.0
```

Problem B. (10 *points*) **Most Populous City**

In this problem, we'll use a data set containing information about the twelve most populous cities for a given year. Take a look at the `cities.csv` file.

Write a function `most_populous(year, region)` that takes in a string `year` that has the year and another string `region` that has the region to look for. The function should read through the cities.csv file and return a list of all the cities that were from the given region in the given year.

- Regions named in the data are "East Asia", "Middle East", "South Asia", "Europe", "North America", and "Latin America" (odd choices, we know, the data set is compiled from other sources.)
- Only the twelve most populous cities for a given year are in the data set, so a city that is among the most populous one year might not appear in a later year.
- You may assume that cities.csv exists in the folder you're running Python from: no need to use a try-except.
- It might be helpful to open the cities.csv file with a text editor program (e.g. VSCode, Notepad, TextEdit) rather than something like Excel. A text editor will show you the data in the same format that Python will read it.

```
>>> most_populous('1500', 'East Asia')
['Beijing', 'Guangzhou', 'Hangzhou', 'Nanjing']

>>> most_populous('1673', 'Europe')
['Istanbul', 'London', 'Naples', 'Paris']

>>> most_populous('1499', 'South Asia')
[]

>>> most_populous('1500', 'USA')
[]
```

Problem C. (10 *points*) **Temporal Stasis**

In the year 2489, the mage Witzidrema avoided defeat at the battle of New Duluth by transporting herself and her forces hundreds of years into the past, into an underground vault beneath a major city. There she froze herself and her allies in time, waiting until the opportune moment to alter the course of history.

You've discovered the location and year that Witzidrema jumped to, and are responsible for updating the `cities.csv` spreadsheet to account for this change. Since Witzidrema brought about 3000 people with her to the past, this means the population of the city she teleported to needs to be increased by 3000 for that year and every year after (even if those people are frozen in time, they are still technically alive and residing within city limits).

Write a function `stasis(jump_year, jump_city)` that takes in a string `jump_year` that has the year and another string `jump_city` that has the name of a city to look for.

The function should produce a new CSV file called `fixed_cities.csv`, which is identical to the original, except that the population of the specified city has been increased by 3000 for all years greater than or equal to `jump_year`. **Note that because the population column specifies the population in thousands, this means that you're actually increasing the number in that column by 3, not 3000.**

The function should return an integer representing the number of lines that needed to be altered.

- You can ignore years where `jump_city` does not appear in the spreadsheet.
- Don't change the population of the specified city for years before `jump_year`.

For example, if we ran `stasis('1973', 'London')`, then `fixed_cities.csv` would have contents identical to `cities.csv`, except for the following three lines (and the function would return the integer 3):

```
1973,London,United Kingdom,7758,Europe
```
would become
```
1973,London,United Kingdom,7761,Europe
```

```
1974,London,United Kingdom,7744,Europe
```
would become
```
1974,London,United Kingdom,7747,Europe
```

`1975,London,United Kingdom,7730,Europe`
would become
`1975,London,United Kingdom,7733,Europe`

No lines before the year 1973 would change (because Witzidrema didn't arrive until 1973), and no lines after the year 1975 would change (because past that year London no longer appears in the spreadsheet).

Examples:

Note: For this problem, getting the correct return value is not sufficient: **you must also match the expected output file**. See the sample output files (`correct_1973_London.csv`, `correct_1450_Edirne.csv`, `correct_1798_Suzhou.csv`) in your hw09files.zip folder to see what your fixed_cities.csv file should look like after running each of the tests.

```
>>> stasis('1973', 'London')
3

>>> stasis('1450', 'Edirne')
53

>>> stasis('1798', 'Suzhou')
10
```