CSCI 1100 — Computer Science 1
Homework 4
Loops and Lists; Passwords and Quarantine

Overview

This homework is worth **100 points** total toward your overall homework grade. It is due in **1 week** i.e. on Thursday, October 17, 2024 at 11:59:59 pm. As usual, there will be a mix of autograded points, instructor test case points, and TA graded points. There are two parts in the homework, each to be submitted separately. All parts should be submitted by the deadline or your program will be considered late.

See the handout for Submission Guidelines and Collaboration Policy for a discussion on grading and on what is considered excessive collaboration. These rules will be in force for the rest of the semester.

You will need the utilities and data files we provide in `hw4_files.zip`, so be sure to download this file from the `Course Materials` section of Submitty and unzip it into your directory for HW 4. Module `hw4_util.py` is written to help you read information from the files. You do not need to know how functions provided in `hw4_util.py` are implemented (but feel free to examine the code, if you are interested), you can simply use them.

Final note, you will need to use loops in this assignment. We will leave the choice of the loop type up to you. Please feel free to use `while` loops or `for` loops depending on the task and your personal preference.

Part 1 — Password Strength

Often when you create a password it is judged for its strength. The estimate of strength is compute by applying several rules — about the length of the password, the presence of certain types of characters, its match against common passwords, and even its match against license plates. In this part of the homework you will implement a few simple strength judgment rules and then determine if a password is to be rejected, or rated poor, fair, good or excellent.

Your program should start by asking for and reading in a password. The program should then evaluate the password based on the following rules. Each rule contributes to a numerical score (which starts at 0):

1. **Length:** If the password is 6 or 7 characters long then add 1 to the score; if it is 8, 9 or 10 characters long then add 2; and longer than 10 add 3.

2. **Case:** If it contains at least two upper case letters and two lower case letters add 2 to the score, while if it contains at least one of each, add 1 to the score.

3. **Digits:** If it contains at least two digits add 2 to the score and if it contains at least one digit then add 1.

4. **Punctuation:** If it contains at least one of `!@#$` add 1 and if it contains at least one of `%^&*` then add 1 (total possible of 2).

5. **NY License:** If it contains three letters (upper or lower case) followed by four digits, then it potentially matches a NY state license plate. In this case, subtract 2 from the score.

6. **Common Password:** If the lower case version of the password exactly matches a password found in a list of common passwords, then subtract 3 from the score.

Whenever a rule is applied that creates a change in the score, generate an explanatory line of output. After applying all the rules, output the score and then convert it to a final strength rating of the password:

- Rejected: the score is less than or equal to 0.
- Poor: the score is 1 or 2.
- Fair: the score is 3 or 4
- Good: the score is 5 or 6
- Excellent: the score is 7 or above.

Part 1 Notes

1. For this part and for part 2 you should write functions to keep the code clean, clear and easy to manage.
2. We have provided you with a number of examples that show output values and formatting. Please follow these examples closely.
3. The common passwords are extracted from a file. One of the utility functions we have provided reads this file and return a list of these passwords. To use this function, start by making sure that `hw4_util.py` and `password_list_top_100.txt` are in the same folder as your code. Then add the line

   ```
   import hw4_util
   ```

   into your program. Finally, call function `hw4_util.part1_get_top` with no arguments. It will return a list of strings containing 100 passwords for you to compare against.
4. Submit **only** your program file `hw4_part1.py`. Do not submit `hw4_util.py`.

Part 2 — COVID-19 Quarantine States

The NY State COVID-19 Travel Advisory at

`https://coronavirus.health.ny.gov/covid-19-travel-advisory`

requires that individuals who travel to New York from states that have significant community spread of COVID-19 must self-quarantine for 14 days. "Significant spread" in a state is measured as either

- a daily average of more than 10 people per 100,000 residents tested positive in the previous seven days, or
- a daily average of more than 10% of tests were positive in the previous seven days.

(See `https://coronavirus.health.ny.gov/covid-19-travel-advisory`.) We will refer to states having a significant spread as *quarantine states*. In this part of HW 4 you will use per state data downloaded from `https://covidtracking.com/` to answer queries about which states were quarantine states and when.

The data we obtained was downloaded (on October 5) in the form of a large "comma-separated value" file. This file contains one line per day per state, and there are many fields per line. We have condensed it to a form suitable for a CS 1 homework. The data are shared under the Creative Commons BY 4.0 license which means we can

- Share: copy and redistribute the material in any medium or format and
- Adapt: remix, transform, and build upon the material for any purpose, even commercially.

Note that there are other versions of the Creative Commons BY 4.0 license that are more restrictive in the uses they allow.

We have provided a simple utility to give you access to the condensed data. To use this (similar to Part 1) you must have the files `hw4_util.py` and `prob2_data.csv` in the same folder as your own code. You must then

```
import hw4_util
```

into your program. `hw4_util` has a function called `part2_get_week` that accepts a single integer argument, `w`, and returns a list of lists. Argument `w` is the index of a previous week, with `w==1` being the most recent week, `w==2` being the week before that, etc., up to `w==29` which corresponds to 29 weeks ago, all the way back to March 15. The returned list contains one list per state, plus the District of Columbia (DC) and Puerto Rico (PR) — 52 in all. Each state list has 16 items:

- Item 0 is a string giving the two letter (upper case) state abbreviation. These are correct.
- Item 1 is an integer giving the state's population estimate — from the 2019 Census Bureau estimate `https://www.census.gov/newsroom/press-kits/2019/national-state-estimates.html`.
- Items 2-8 are the number of positive tests for that state in each of the seven days of the specified week — most recent day first.
- Items 9-15 are the number of negative tests for the state in each of the seven days of the specified week — most recent day first.

For example, the list for Alaska for week 1 is

```
['AK',731545,189,147,128,132,106,125,118,3373,3819,6839,4984,6045,6140,1688]
```

Here is what you need to do for this assignment. Your program, in a loop, must start by asking the user to specify the index for a week, as described above. (You may assume an integer is input as the week number.) A negative number for the week indicates that the program should end. For non-negative numbers, if the data are not available for that week the function will return an empty list; in this case, skip the rest of loop body. Otherwise, after getting the list of lists back, the program should answer one of four different requests for information about that week. Answering the request starts by the user typing in a keyword. The keywords are `'daily'`, `'pct'`, `'quar'`, `'high'`. Here's what the program must do for each request:

- `'daily'`: Ask the user for the state abbreviation and then output the average daily positive cases per 100,000 people for that state for the given week, accurate to the nearest tenth.
- `'pct'`: Ask the user for the state abbreviation and then output the average daily percentage of tests that are positive over the week, accurate to the nearest tenth of a percent.
- `'quar'`: Output the list of state abbreviations, alphabetically by two-letter abbreviation, of travel quaratine states for the given week as discussed above. There should be ten state abbreviations per line — call `hw4_util.print_abbreviations` with your list of abbreviations to print the output as required. (Note: every week has at least one quarantine state.)
- `'high'`: Output the two-letter abbreviation of the state that had the highest daily average number of positive cases per 100,000 people over the given week, and output this average number, accurate to the nearest tenth.

Input key words and state abbreviations may be typed in upper or lower case and still match correctly. If a key word is not one of these four or if a state is not found (because its abbreviation is incorrectly typed), output a simple error message and do nothing more in the current loop iteration.

Part 2 Notes:

1. As always, look at the example output we provide and follow it accurately.
2. All reported values for numbers of positive and negative test results will be at least 0; however, some may be 0. You may assume, however, that there will never be a week where all days have 0 negative tests.
3. Compute the daily percentage of tests that are positives by summing the positive cases for the week, and summing negative cases for the week. If these sums are P and N respectively, then the percentage positive is $P/(P+N) * 100$. This is not exactly the same as the average of daily percentages for a week, but it is easier to compute.
4. Submit **only** your program file `hw4_part2.py`. Do not submit `hw4_util.py`.