Intro Programming/CS Essentials I
56:198:500/56:121:530 (Fall 2022)

Homework:	**5**	**Professor:**	**Suneeta Ramaswami**
Due Date:	**11/30/22**	**E-mail:**	`suneeta.ramaswami@rutgers.edu`
Office:	**321 BSB**	**URL:**	`http://sites.rutgers.edu/suneeta-ramaswami`
		Phone:	**(856)-225-6439**

Homework Assignment 5

The assignment is due by 11:59PM of the due date. The point value is indicated in square braces next to each problem. Each solution must be the student's own work. Assistance should only be sought or accepted from the course instructor. **Please read the submission guidelines at the end of this document before you start your work.**

Important note: When writing each of the following programs, it is important that you name all the functions *exactly as described* because I will assume you are doing so when testing your programs. If your program produces errors because the functions do not satisfy the stated prototype, points will be deducted. See additional details below:

1. You will submit three modules for this homework assignment: `hw5problem1.py` for Problem 1, `hw5problem2.py` for Problem 2, and `hw5problem3.py` for Problem 3.

2. In addition to this homework document (hw5.pdf), you will also download data files provided for each of the three problems, as indicated on the Canvas page for the assignment.

3. Download the Jupyter notebook `hw5.ipynb`. Enter your name and collaboration declaration in the notebook. This notebook contains some instructions and code cells which import your modules and call functions from the module suitably. *You will not need to modify any of the code cells.* You may include additional markdown cells to explain something about your program, but you are not required to.

4. All three modules and their functions must be documented *succinctly* with docstrings. There will be a penalty for not including documentation.

5. **Please note:** You may not import any modules other than `math` for any of the problems in this assignment.

Problem 1 [35 points] Formatting a text file. The judges of an essay competition require all essays to be submitted electronically in a neatly formatted fashion. However, not all submissions follow the formatting rules, and so the judges would like a program that formats a file according to their formatting rules. The judges also require some statistics for each essay. In particular, they require a count of the number of non-blank lines, the number of words, and the average word length for each essay. In this problem, you are asked to write a program that does both things; that is, a program that formats the files, and then calculates the required statistics.

For the purposes of the program and the discussion, a word is simply a consecutive sequence of non-whitespace characters surrounded by white space. By this definition, a period or a comma by itself, surrounded by white space, would be counted as a word, but we'll live with that. The formatting rules for each essay are as follows:

- There should not be any blank spaces at the beginning of a line.
- Two or more blank spaces should not appear consecutively. That is, there should be only one blank space between consecutive words.
- Two or more blank lines should not appear consecutively. That is, there should be only one blank line between consecutive paragraphs.
- There should be at most 60 characters per line (including blank spaces). Also, words should not be broken across lines. This implies that in a paragraph, a newline character should appear after the last complete word that will fit in a 60-character line. You may assume that no single word is longer than 60 characters (seems like a reasonable assumption).

Format the essay file according to the rules described above and compute the required statistics. You should do this in three steps. For the purposes of the following discussion, assume that the essay is in a file called `essay.txt`.

1. First, remove all extra white space (extra spaces between words, extra spaces at the beginning of a line, and extra blank lines between paragraphs) from `essay.txt` and output the result into a file called `essay_neb.txt`. Keep in mind that only *extra* white space is removed. In particular, paragraphs should still be separated by one blank line.
2. Next, adjust the length of the lines in `essay_neb.txt` to 60 characters, and output the result into a file called `essay_final.txt`.
3. Finally, count the number of (non-blank) lines, the number of words, and the average word length for the text in `essay_final.txt`. Output the result, with the appropriate headings, into a file called `essay_stats.txt`. The *average word length* is simply the sum of the word lengths divided by the number of words.

Your program should contain the following functions:

- A function called `remove_extra_whitespaces` with two parameters: `infile`, the name of the file to be read from and `outfile`, the name of the file to be written to. Both these parameters are strings as they are file names. This function should open and close files as necessary and carry out the task described in step (1) above.
- A function called `adjust_linelength` with two parameters: `infile`, the name of the file to be read from *(this is the file from which all extra white spaces have been removed)* and `outfile`, the name of the file to be written to. Both these parameters are strings as they are file names. This function should open and close files as necessary and carry out the task described in step (2) above. *Keep in mind that words should not be broken across lines.* This implies that in the output file, a newline character should appear after the last complete word that will fit in a 60-character line. This should be true for every line of a paragraph. Furthermore, each line should contain the maximum number of complete words that can fit in that line. Also make sure that consecutive paragraphs continue to be separated by a blank line.
- A function called `essay_statistics` with two parameters: `infile`, the name of the file to be read from *(this is the file in which line lengths have been adjusted)* and `outfile`, the name of the file to be written to. Both these parameters are strings as they are file names. This function should open and close files as necessary and carry out the task described in step (3) above.

- A function called `format_essay` without any parameters. This function should ask the user to enter the name of the file containing the essay, and then carry out steps (1)-(3) by calling functions `remove_extra_whitespaces`, `adjust_linelength`, and `essay_statistics`, respectively. The function should inform the user that the intermediate file has been created. For example, if the user enters "`essay.txt`" as the name of the file, then after calling `remove_extra_whitespaces`, inform the user that the file "`essay_neb.txt`" has been created, and so on for the other functions as well. Ideally, you should use intermediate file names that are derived from the original file name. For example, if the user enters "`rain.txt`" as the name of the file containing the essay, then the files created by your program should be "`rain_neb.txt`", "`rain_final.txt`", and "`rain_stats.txt`".

A sample set of files is shown below. Suppose `essay.txt` is the following file:

```
    Albuquerque is my     turkey and he's   feathered and    he's      fine, And    he
       wobbles       and he gobbles and       he's
absolutely mine


He's the best            pet           you            can get yet, better than
a dog or cat,        He's my          albuquerque turkey and i'm awfully proud of that



Albuquerque is my     turkey  and he's     happy in   his bed, 'Cause for
our thanksgiving        dinner        we'll have            sphagetti         instead
```

Then, the file `essay_neb.txt`, with extra white spaces removed, should look like this:

```
Albuquerque is my turkey and he's feathered and he's fine, And he
wobbles and he gobbles and he's
absolutely mine

He's the best pet you can get yet, better than
a dog or cat, He's my albuquerque turkey and i'm awfully proud of that

Albuquerque is my turkey and he's happy in his bed, 'Cause for
our thanksgiving dinner we'll have sphagetti instead
```

The final file `essay_final.txt` with line length adjusted to 60 should look like the following. Just for your information, the longest line (the first line of the second paragraph) is exactly 60 characters long. Note that no words are broken across lines.

```
Albuquerque is my turkey and he's feathered and he's fine,
And he wobbles and he gobbles and he's absolutely mine

He's the best pet you can get yet, better than a dog or cat,
He's my albuquerque turkey and i'm awfully proud of that

Albuquerque is my turkey and he's happy in his bed, 'Cause
for our thanksgiving dinner we'll have sphagetti instead
```

Finally, file `essay_stats.txt` should look like this (average word length has been truncated):

```
In the file essay.txt:

Number of (non-blank) lines: 6
            Number of words: 63
        Average word length: 4
```

Here is a sample output from calling `format_essay()`:

```
Essay Formatting Helper Program
-------------------------------

Enter the name (*.txt) of the file containing the essay: essay.txt

Three files have been created:

essay_neb.txt: the essay without extra whitespaces or blank lines
essay_final.txt: the formatted essay
essay_stats.txt: the essay statistics
```

Two sample files, `essay.txt` and essay2.txt, are provided for you to test your implementation.

Problem 2 [35 points] Calculating student grades. This problem requires you to read data from several files (containing numeric scores on homeworks, quizzes, and exams for students in a course), use a dictionary to collect information from the data in those files, and finally to write a combined course grade roster into an output file. Evaluation criteria for the course are described below:

1. Four homework assignments are handed out. Each homework is out of 100 points. The lowest homework score is dropped, and the remaining three scores determine the final homework score. The homeworks are worth 30% of the final grade.
2. Eight quizzes are given in class. Each quiz is out of 50 points. The lowest and the highest quiz scores are dropped, and the remaining six scores determine the final quiz score. The quizzes are also worth 30% of the final grade.
3. Two comprehensive exams are given. Each exam is out of 200 points. (No exam scores are dropped!) The exams are worth 40% of the final grade.

The final total score is simply the sum of the combined homework score (out of 30), the combined quiz score (out of 30), and the final exam score (out of 40), each calculated as described above. All scores are real values. The letter grade is determined according to the usual scale: A final total score of 90.0 or higher is an `A`, 85.0 or higher is a `B+`, 80.0 or higher is a `B`, 75.0 or higher is a `C+`, 70.0 or higher is a `C`, 60.0 or higher is a `D`, and anything lower than 60.0 is an `F`.

The data for this course is available in four separate input files:

1. A file called "`studentids.txt`" that contains simply the list of identification (ID) numbers for the students in the course with one entry per student. Each line of the file contains just an ID, which is a string of characters in the usual format xxx-xx-xxxx.

2. A file called "`hwscores.txt`" that contains a dump of all the homework scores for all the students in the course. Each line of the file contains simply an ID followed by a *single* homework score. A homework score is an integer value between 0 and 100 (inclusive). Observe that the same ID will, in general, appear on several lines in the file, once for every homework submission made by the student with that ID. The number of entries for a student is equal to the number of homework submissions made by that student. If a student submits fewer than 4 homeworks, the remaining homework scores are simply zero (keep this in mind when dropping the lowest score). The IDs may appear in any order in the file (in other words, you should not assume that the same ID will appear consecutively in the file).
3. A file called "`quizscores.txt`" that contains a dump of all the quiz scores for all the students in the course. Each line of the file contains simply an ID followed by a *single* quiz score. A quiz score is an integer value between 0 and 50 (inclusive). As above, the same ID may appear several times in the file, once for every quiz taken by the student with that ID. The number of entries for a student is equal to the number of quizzes taken by that student. If a student takes fewer than 8 quizzes, the remaining quiz scores are zero. Again, the IDs may appear in any order in the file.
4. A file called "`examscores.txt`" that contains a dump of all the exam scores for all the students in the course. Each line of the file contains simply an ID followed by a *single* exam score. Once again, the same ID may appear more than once in the file and the IDs appear in any order.

Your program should create a neatly formatted output file called "`graderoster.txt`" that contains one line of data per student in the format described below:

1. The first line should contain the column headings, which are `RUID`, `HW(30)`, `QUIZ(30)`, `EXAM(40)`, `TOTAL(100)`, `GRADE`. The width of the first column must be 14 characters long. The width of the second, third, and fourth columns should be 10 characters long, the width of fifth column should be 12 characters long, and the width of the sixth column should be 10 characters long. Print the first column header left-justified and the remaining ones right-justified.
2. The next line should be a series of hyphens ('-') (as seen in sample output file).
3. The rest of the file should contain one line of information per student. Each line should contain the ID in the first column (width 14), the final homework score out of 30 (width 10), the final quiz score out of 30 (width 10), the final exam score out of 40 (width 10), the final total score out of 100 (width 12), and the letter grade (width 10). The RUID must be left-justified, the scores must be right-justified, and the letter grade must be right justified as well. Each score should be printed as a real value with a precision of 2.
4. After the above has been printed, print a blank line, followed by the maximum, minimum, and average scores for the homeworks, quizzes, exams, and totals.

You are asked to accomplish the above task by implementing the following in the module `hw5problem2.py`:

1. A function called `create_dictionary` with four parameters: `idfilename`, `hwfilename`, `qzfilename`, and `examfilename`. These parameters are, respectively, the names of the files containing student IDs, homework scores, quiz scores, and exam scores (in the

format described above). The function should open and close all necessary files. This function **returns a dictionary** of key:value pairs in which the key is the student ID and the value is itself a dictionary containing the keys `"hw"`, `"quiz"`, and `"exam"`. The values associated with these keys are lists of length 4, 8, and 2 containing, respectively, the homework, quiz, and exam score data for that student. An example of a key:value pair in such a dictionary is shown below:

```
"123-45-6789":{"hw":[98,89,92,75], "quiz":[45,36,42,50,29,27,40,41], "exam":[175,157]}
```

2. A function called `create_graderoster` with two parameters: `sdata_dict` and `outfilename`. The first parameter, `sdata_dict`, is a dictionary of the type returned by the above function `create_dictionary`. The second parameter, `outfilename`, is the name of the output file in which a neatly formatted grade roster will be printed (exactly as described above). The function is responsible for opening and closing the file.

A set of sample input files is available on Canvas (these are the files `studentids.txt`, `hwscores.txt`, `quizscores.txt`, and `examscores.txt`) along with the output file created for that data (this is the file `SRgraderoster.txt` that was created by my Python code for this problem). The output file created by your `create_graderoster` function should look similar (it's okay if the order of the listing varies, but the numbers/grades should look the same). Feel free to create additional data files to test your functions.

Problem 3 [30 points] Correlating Stock Data Correlation measures the relationship between two variables in terms of strength and direction. Another way to state it is to say that correlation measures the tendency of two variables to increase or decrease at the same time. For example, average test scores in a school are positively correlated with graduation rates (as one increases so does the other), whereas the Dow Jones Industrial average and the price of gold are negatively correlated (as one increases the other decreases). The measure of correlation is sometimes referred to as the correlation coefficient. There are several algorithms for calculating a correlation coefficient. Here, we consider the *Pearson correlation coefficient*, which is calculated as shown below:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{(n-1)S_xS_y}$$

where $x_1, x_2, \ldots, x_n$ are n values of the x variable, and $y_1, y_2, \ldots, y_n$ are n values of the y variable, $\bar{x}$ and $\bar{y}$ are the means of the two variables, and S_x and S_y are the standard deviations. Recall that the standard deviation is defined as follows:

$$S_x = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}$$

Note that the value of r lies between -1 and 1. When the value of r is close to 1, the two variables are strongly positively correlated; when r is close to -1, the two variables are strongly negatively correlated; when r is close to 0, the two variables have no correlation.

In this problem, you are asked to determine the correlation between stock prices for two companies. This question is of interest because it could be a factor in making investment decisions. If two stocks are positively correlated, and one of them is increasing, it may suggest that it is a good time to invest in the other. You are given stock data for a company in a CSV (comma separated value) file. Some additional details about the file are given below:

- Each line of the file has 7 data values: the date, the opening stock price, the high price of the day, the low price of the day, the closing price for the day, the adjusted closing price, and the volume (number of shares traded).
- The date is the date on which the trade occurred. They appear in increasing chronological order in the file.
- Note that the very first line of the file contains value headers and can be ignored.
- For this problem, we are interested in only two data values: the date (the first value on each line) and the closing price of the day (the fifth value on each line).

You are provided with stock data over a 5-year period for Apple in the file called `AAPL.csv` and for Coke in the file `COKE.csv`. (Historical stock data is available at `finance.yahoo.com` and and can be downloaded from there.) Use these files to test your code.

Implement the following functions in the module `hw5problem3.py`. As always, minimize code repetition and use an existing function to carry out a task whenever possible.

- A function called `closingprices_list` with a single parameter called `stockfilename`, which is the name of a file containing stock data in the format described above. The function returns a list of 2-tuples, where each 2-tuple contains the closing price and the date. There are as many 2-tuples in the list as the number of entries in `stockfilename`. The 2-tuples should be listed in order by date (note that the entries in `stockfilename` are ordered by date). Remember to close the file before the function returns.
- A function called `best_days` with two parameters: `stockfilename` (the name of a file containing stock data) and `n` (a positive integer). The function returns a list of 2-tuples containing the n best days (that is, the n highest closing prices) and their associated dates. For example, `best_days("AAPL.csv", 5)` should return the 5 best days of Apple stock in the given file.
- A function called `worst_days` with two parameters: `stockfilename` (the name of a file containing stock data) and `n` (a positive integer). The function returns a list of 2-tuples containing the n worst days (that is, the n lowest closing prices) and their associated dates.
- A function called `correlation_coeff` with two parameters: `stockfilename1` and `stockfilename2`, both of which are names of files containing stock data for two different companies. The function returns the correlation coefficient (calculated as shown above) for the two companies' stock prices. You should obviously use `closingprices_list` when implementing this function. You may assume that both files have data for the same date ranges, which means that both will have exactly the same number of lines. (If you wish, you can check for this condition and raise an exception if it is not satisfied.)

Submission Guidelines

- For this homework assignment, you are asked to submit a **single zip file** containing your four homework files: `hw5problem1.py`, `hw5problem2.py`, `hw5problem3.py`, and `hw5.ipynb`. Please make sure that the files are named exactly as specified and that your zip file contains exactly these (and only these) four files.
- Once you are ready to submit, upload the (single) zip file on Canvas.

You must submit your assignment at or before 11:59PM on November 30, 2022.