Object-Oriented Programming
50:198:113 (Spring 2022)

Homework:	**2**	**Professor:**	**Suneeta Ramaswami**
Due Date:	**2/21/22**	**E-mail:**	`suneeta.ramaswami@rutgers.edu`
Office:	**321 BSB**	**URL:**	`http://crab.rutgers.edu/~rsuneeta`
		Phone:	**(856)-225-6439**

Homework Assignment 2

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. Any violation of this rule will be dealt with harshly.

This assignment requires the use of dictionaries, lists, and file processing. In this and all future assignments, you are graded not only on the correctness of the code, but also on clarity and readability. Hence, I will deduct points for poor indentation, poor choice of object names, and lack of documentation. All modules and function definitions must use docstring (triple quote) documentation. For the remaining code, use a common sense approach. While I do not expect every line of code to be explained, all code blocks that carry out a significant task should be documented *briefly* in clear English.

Download the homework document (`hw2.pdf`), the module stub for Problem 2 (`problem2.py`) and all the text files containing data for the Problem 1 (`essay.txt`, `essay2.txt`) and Problem 2 (`balance.dat`, and `transactions.dat`). Place all these files in the same directory (or folder) when working on your assignment.

Important note: When writing each of the following programs, it is important that you name 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. In particular,

1. After importing the `problem1` module, I will type `python3 problem1.py` at the command line to test `problem1.py`, and
2. After importing the `problem2` module, I will type `python3 problem2.py` at the command line to test `problem2.py`.

For each problem, put all your function definitions in one file. *Do not create a separate file for each function.* The modules `problem1.py` and `problem2.py` should contain *all* the function definitions for Problems #1 and #2, respectively.

Problem 1 [25 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. After each of these function calls, 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`".
- Once again, so that I may run your program in the shell, include an appropriate call to the `format_essay` function in the body of the following `if` statement:

```
if __name__ == "__main__":
```

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 run of the program, obtained by running `python3 problem1.py` in the shell:

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

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

The formatted essay is in the file essay_final.txt
The essay statistics are in the file essay_stats.txt
```

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

Problem 2 [25 points] Bank Transactions. For this problem, you will write a program to do some basic file processing to carry out some banking transactions for the customers of a bank. Assume for our purposes that each customer has exactly one account at the bank. The bank maintains two files of data.

- One file contains account balance information for each customer of the bank. Each line of this file contains three data items: A customer's social security number (SSN), the bank account number, and the balance at the end of the previous business period (this might be a month, a week, or a day, but the actual length of the business period is not relevant for our purposes). The SSN is a string of digits in the usual format xxx-xx-xxxx, the bank account number is simply a three digit number (unique to each customer), and the balance is a floating point value.
- The second file contains all the bank transactions that have taken place in the current business period for all the customers at the bank. Each line of this file contains just two data items: a bank account number and the dollar amount of the transaction. If the transaction is a deposit, it is indicated as a positive value and if it is a payment

(withdrawal), it is indicated as a negative value. A customer's bank account number may appear several times in the file (if they had several transactions during the business period) or may not appear at all (if they did not carry out any transactions at the bank during the business period). A transaction is simply recorded at the time that it takes place, hence the account numbers may appear in any order in the file and all the transactions for a particular account may not appear together (that is, on consecutive lines) in the file.

The goal is to update each customer's bank balance information by taking into account all transactions (deposits as well as withdrawals) that have taken place in the current business period. If the resulting account balance is greater than $3000, a 2% interest (on the new account balance) is added on. If the resulting account balance is less than $100, a $10 penalty is applied to the account.

Your program must create two files. One file should contain transaction summaries for all the customers of the bank. It should contain, for each customer, their account number, SSN, the old balance, total of all deposits made, total of all withdrawals made, the interest amount added, the penalty added, and the final new balance. The second file is the file containing the new account balance information (the bank will eventually replace the old account balance file with this one). Each line of this file should contain the customer's SSN, bank account number, and the new balance at the end of the current business period.

All of the data in the output files should be formatted nicely. In particular, the bank account number and SSN should be printed left-justified in appropriately chosen field widths. All dollar amounts should be printed right-justified, with a precision of 2, in appropriately chosen field widths. For the purposes of formatting, you may assume that bank balances are always below $100,000. The columns in the output file should be given appropriate headings as well (see sample files below).

The above task will be accomplished by using a dictionary to keep track of a customer's transactions for the current business period, as specified in the following three functions:

1. Implement a function called `customer_dictionary` with a single parameter `balfilename`, the name of the file containing current account balance information for the customers of the bank in the format described above The function should open `balfilename` for reading. It should then create a dictionary with key:value pairs in which the key is the bank account number of a customer, and the value associated with a key is a list containing the SSN and the account balance for that customer. The function must **return** the dictionary. Remember to close the file *prior to* the return statement.
2. Implement a function called `update_customer_dictionary` with two parameters. The first parameter, `cdictionary`, is the customer dictionary described above. The second parameter, `transfilename`, is the name of the file containing all the transactions for all the customers at the bank in the format described above This function should open `transfilename` for reading, and then update the `cdictionary` to include the total of all the deposits and withdrawals for each customer, as well as the *maximum deposit* and the *maximum withdrawal*. Recall that the value associated with each customer is a list. Hence, the deposits, withdrawals, and maximum deposit and withdrawal for a customer can be recorded as additional entries in the list. Remember to close the file after it has been read completely.

3. Implement a function called `new_balance_files` to create the two output files described above. This function has three parameters: The first parameter, `cdictionary`, is the customer dictionary described above. The second parameter, `summfilename`, is the name of the file containing the transaction summaries for the customers of the bank. The third parameter, `newbalfilename`, is the name of the file containing the new account balance information for all the customers. As noted above, the output files created by this function should be neatly formatted.

An example is provided below. Consider the following example of the file containing account balance information:

```
123-45-6789         100            348.17
345-23-1782         300           3029.25
672-39-1929         700           1800.87
819-00-7810         900            -50.25
```

Suppose the file containing transactions data is the following file:

```
900 150.25
100 525.72
100 -150.75
300 -500.00
100 -27.38
300 -550.00
900 -120.00
300 2400.00
300 -300.00
```

Then the output file containing transaction summaries should look like this:

```
ACCT#  SSN            PREVBAL  DEPOSITS  WITHDRAWALS    MAXDEP  MAXWDRAW  INTEREST   PENALTY    NEWBAL
-------------------------------------------------------------------------------------------------------
100    123-45-6789     348.17    525.72       178.13    525.72    150.75      0.00      0.00    695.76
300    345-23-1782    3029.25   2400.00      1350.00   2400.00    550.00     81.58      0.00   4160.84
700    672-39-1929    1800.87      0.00         0.00      0.00      0.00      0.00      0.00   1800.87
900    819-00-7810     -50.25    150.25       120.00    150.25    120.00      0.00     10.00    -30.00
```

The output file containing the new account balance information should look like this:

```
123-45-6789      100         695.76
345-23-1782      300        4160.84
672-39-1929      700        1800.87
819-00-7810      900         -30.00
```

The file `problem2.py` contains function calls to all the functions you must implement. Insert your function implementations into this file. When you are ready, type 'python3 problem2.py' at the command line. Two output files (`transsummary.dat` and `newbalance.dat`) will be created. Make sure these files contain the correct output as described above.

Submission Guidelines

Implement the first problem in a file called `problem1.py`. Implement the second problem by inserting your functions into the `problem2.py` module stub that is provided. *Your name and RUID should appear as a comment at the very top of each file.* Test each of your programs thoroughly before submitting your homework. When you are ready to submit, upload your files on Canvas as follows:

1. Go to the "Assignments" tab of the Canvas site for this course.
2. Click on "Programming Assignment #2" under Homework Assignments.
3. Upload your homework files (`problem1.py` and `problem2.py`) when you are ready to submit.

You must submit your assignment at or before 11:59PM on February 21, 2022.