Object-Oriented Programming
50:198:113 (Spring 2019)

Homework:	**2**	**Professor:**	**Suneeta Ramaswami**
Due Date:	**2/26/19**	**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 11:55PM 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. You may seek or accept assistance only from the course instructor or the TA. Any violation of this rule will be dealt with harshly.

This assignment requires the use of lists, dictionaries, 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. For documentation, use the triple-quote method for the entire module, as well as each function in the module.

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 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 [18 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 `problem1.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.
3. So that I may run your program in the shell, include appropriate calls to the above functions in the body of the following `if` statement:

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

A set of sample input files is available on **the Sakai site** (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. Feel free to create additional data files to test your functions.

Problem 2 [32 points] Matrix Operations. This problem requires you to work with a list of lists. *You are expected to use list comprehension whenever suitable* in this module. Download module `problem2.py` from Sakai. Insert your implementation at the top of the module. This module contains some test code for you to test the implementation of your functions.

In this problem, you are asked to write several functions to manipulate *matrices*. An $m \times n$ *matrix* is a rectangular array of numbers consisting of m rows and n columns. If $m = n$, the matrix is said to be a *square* matrix. An example of a 2×3 matrix A and a 3×3 (square) matrix B is shown below.

$$A = \begin{bmatrix} 5 & 3 & -1 \\ 9 & 4 & 12 \end{bmatrix}$$

$$B = \begin{bmatrix} 6 & 9 & 12 \\ -8 & 6 & -4 \\ 7 & 11 & 13 \end{bmatrix}$$

For an $m \times n$ matrix A, the element of the matrix in the i-th row and j-th column is denoted by a_{ij}, where $1 \leq i \leq m$ and $1 \leq j \leq n$. Using this notation, we can now define a number of operations on matrices.

- The **sum** of two $m \times n$ matrices A and B is an $m \times n$ matrix C such that $c_{ij} = a_{ij} + b_{ij}$. We write this as $C = A + B$.
- The **difference** of two $m \times n$ matrices A and B is an $m \times n$ matrix C such that $c_{ij} = a_{ij} - b_{ij}$. We write this as $C = A - B$.
- The **product** of two matrices A and B is defined when A is an $m \times k$ matrix and B is a $k \times n$ matrix (that is, the number of columns in A is the same as the number of rows in B). In this case, the product C of A and B, written as $C = AB$, is an $m \times n$ matrix, where c_{ij} is obtained by multiplying the i-th row of A with the j-th column of B as follows:

  $$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + a_{i3}b_{3j} + \ldots + a_{ik}b_{kj}$$

  Note that the number of elements in the i-th row of A is equal to the number of elements in the j-th column of B.
- The **determinant** of a *square* matrix A, denoted $\det(A)$, is a function that calculates a real value from a matrix. It is defined as follows:

  $$\text{If } A = [a], \det(A) = a$$

  $$\text{If } A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \det(A) = ad - bc$$

  For all larger $n \times n$ matrices, let A_{1j} denote the $(n-1) \times (n-1)$ matrix obtained by deleted row 1 and column j from A. Then $\det(A)$ is defined recursively as follows:

  $$\det(A) = a_{11}\det(A_{11}) - a_{12}\det(A_{12}) + a_{13}\det(A_{13}) - a_{14}\det(A_{14}) + \ldots + (-1)^{n-1}a_{1n}\det(A_{1n})$$

Here are some examples to illustrate the above matrix operations. Let A and B be the example matrices shown on the previous page. In addition, let

$$C = \begin{bmatrix} 0 & -21 & -1 \\ 11 & 13 & 17 \end{bmatrix}$$

Then, we have

$$A + C = \begin{bmatrix} 5 & -18 & -2 \\ 20 & 17 & 29 \end{bmatrix}$$

$$A - C = \begin{bmatrix} 5 & 24 & 0 \\ -2 & -9 & -5 \end{bmatrix}$$

$$AB = \begin{bmatrix} 30 - 24 - 7 & 45 + 18 - 11 & 60 - 12 - 13 \\ 54 - 32 + 84 & 81 + 24 + 132 & 108 - 16 + 156 \end{bmatrix} = \begin{bmatrix} -1 & 52 & 35 \\ 106 & 237 & 248 \end{bmatrix}$$

$$\begin{aligned} \det(B) &= 6\det(B_{11}) - 9\det(B_{12}) + 12\det(B_{13}) \\ &= 6\det(\begin{bmatrix} 6 & -4 \\ 11 & 13 \end{bmatrix}) - 9\det(\begin{bmatrix} -8 & -4 \\ 7 & 13 \end{bmatrix}) + 12\det(\begin{bmatrix} -8 & 6 \\ 7 & 11 \end{bmatrix}) \\ &= 6(78 + 44) - 9(-104 + 28) + 12(-88 - 42) \\ &= 732 + 684 - 1560 \\ &= -144 \end{aligned}$$

In our Python program, we will represent an $m \times n$ matrix as a list of m elements (one for each row), where each element is itself a list of n numbers. In other words, a matrix will be stored as a *list of lists*. For example, the above matrix A will be represented as `A = [[5, 3, -2], [9, 4, 12]]`. Note that in the mathematical matrix notation, the row numbers go from 1 to m and the column numbers go from 1 to n. Hence, row i of the matrix is the $(i-1)$-th element of `A`, and column j of the matrix is the $(j-1)$-th element of each list in `A`. This means that the element a_{ij} is stored in `A[i-1][j-1]`.

In order to implement the above matrix operations in Python, you are asked to write the following functions. **Important:** Do not import any modules when implementing or testing this program.

1. *(1 points)* A function called `dimension` with a single parameter, a matrix `M`. The function returns the number of rows and the number of columns in `M`. The returned object is thus a 2-tuple.
2. *(1 points)* A function called `row` with two parameters, a matrix `M` and a positive integer `i`. The function returns the `i`-th row of `M`. Observe that the returned object is a list.
3. *(3 points)* A function called `column` with two parameters, a matrix `M` and a positive integer `j`. The function returns the `j`-th column of `M`. Observe that the returned object is a list.
4. *(3 points)* A function called `matrix_sum` with two parameters, a matrix `A` and a matrix `B`. If `A` and `B` have the same dimensions, the function should return the matrix sum of `A` and `B`. If `A` and `B` do not have the same dimensions, the function should print an error message.
5. *(2 points)* A function called `matrix_difference` with two parameters, a matrix `A` and a matrix `B`. If `A` and `B` have the same dimensions, the function should return the matrix difference of `A` and `B`. If `A` and `B` do not have the same dimensions, the function should print an error message.
6. *(7 points)* A function called `matrix_product` with two parameters, a matrix `A` and a matrix `B`. If `A` and `B` are product compatible (that is, if the number of columns in `A` is equal to the number of rows in `B`), then the function should return the matrix product of `A` and `B`. If they are not not product compatible, the function should print an error message. Use functions `row` and `column` to implement this function.

7. *(4 points)* A function called `reduce_matrix` with three parameters: a matrix `M`, a positive integer `i`, and a positive integer `j`. The function returns the matrix obtained from `M` by removing the `i`-th row and `j`-th column of `M`. Note that the row and column dimensions of the returned matrix are one less than the row and column dimensions of `M`. *Important:* Do not modify `M` itself when computing the reduced matrix. Create a *new* matrix with the reduced dimensions and return that.
8. *(7 points)* A function called `determinant` with a single parameter, a matrix `M`. If the matrix is *not* a square matrix, the function should print an error message. If it is a square matrix, the function returns the determinant of `M`. Implement the function recursively; the function `reduce_matrix` will come in handy here.
9. *(4 points)* A function called `pretty_print` with a single parameter, a matrix `M`. The function should print the matrix in a neatly formatted way (use the usual row-wise order); use string formatting with field widths. We use this function to print the matrix in a readable format, since printing it out as a list of lists is not easy to read, particularly for large matrices.

Submission Guidelines

Implement the first problem in a file called `problem1.py` and the second one in a file called `problem2.py` (*download this file from Sakai*, as it contains test code for your functions). **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 Sakai as follows:

1. Use your web browser to go to the website `https:sakai.rutgers.edu`.
2. Log in by using your Rutgers login id and password, and click on the `OBJECT-ORIENTED PROG S19` tab.
3. Click on the 'Assignments' link on the left and go to 'Homework Assignment #2' to find the homework file (`hw2.pdf`), the data files for Problem 1, and the module stub `problem2.py` for Problem 2.
4. Use this same link to upload your homework files (`problem1.py` and `problem2.py` when you are ready to submit.

You must submit your assignment at or before 11:55PM on February 26, 2019.