Object-Oriented Programming
50:198:113 (Spring 2017)

Homework Assignment 3

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. Assistance should be sought or accepted only from the course instructor or the TA. Any violation of this rule will be dealt with harshly.

The first problem of this assignment requires you to write some recursive functions. The second problem requires the implementation of several functions for matrices (these are defined in Problem 2), and the third problem gets you started with object oriented programming, where you are asked to create a Matrix class. As in past (and 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, all functions, classes, and methods should have appropriate docstring documentation. For the rest of your 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.

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.

Problem 1 [20 points] Recursive Functions. In this problem, you are asked to write three recursive functions. All three functions should be defined in a module called problem1.py. In each of these functions, you may not use any built-in functions other than len, and the operators [], [:], and + (string or list concatenation). You also may not use any loops (the built-in function in counts as a loop).

1. [6 points] Write a recursive function called replace_char with two parameters: a string astr, a character old_char, and another character new_char. The function should return a string in which every occurrence of old_char in astr is replaced with new_char. For example, replace_char("how now brown cow", 'w', 'o') should return the string "hoo noo broon coo". Once again, your implementation should not contain any loops, and may use only the index and splice operators for strings. No other built-in functions may be used.

2. [7 points] Write a recursive function called num_double_letters with a single string parameter astr that returns the number of occurrences of double letters in astr. A double letter is simply a consecutive pair of the same character. For example,
num_double_letters("mississippi") should return 3 because double letters occur 3 times in the string ("ss", "ss", and "pp"). Note that if the same letter occurs consecutively three or more times (this doesn’t happen too often in English!), we count only distinct pairs. Hence, for example, num_double_letters("hmmmm") should return 1 because it has only one distinct pair of ms and num_double_letters("hmmm") or num_double_letters("mmhmmm") should return 2 because it has two distinct pairs of ms.

3. [7 points] Write a recursive function called has_repeats with a single list parameter L, that returns True if L has repeating elements (i.e., an element occurring more than once) and False otherwise. For example, has_repeats([3, 2, 1, 5, 12]) should return False and has_repeats([3, 5, 2, 1, 5, 12]) should return True. Your implementation should not contain any loops. The only built-in function you should use is the len function, along with the index and splice operators.

Problem 2 [40 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 \times 3 matrix A and a 3 \times 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

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

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

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

\[
\det(B) = 6\det(B_{11}) - 9\det(B_{12}) + 12\det(B_{13})
\]

\[
= 6\det\left(\begin{bmatrix} 6 & -4 \\ 11 & 13 \end{bmatrix}\right) - 9\det\left(\begin{bmatrix} -8 & -4 \\ 7 & 13 \end{bmatrix}\right) + 12\det\left(\begin{bmatrix} -8 & 6 \\ 7 & 11 \end{bmatrix}\right)
\]

\[
= 6(78 + 44) - 9(-104 + 28) + 12(-88 - 42)
\]

\[
= 732 + 684 - 1560
\]

\[
= -144
\]

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 $A = \begin{bmatrix} [5, 3, -2], [9, 4, 12] \end{bmatrix}$. 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. **(2 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. **(2 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. **(4 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. **(4 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. **(3 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. **(8 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 product compatible, the function should print an error message. Use functions `row` and `column` to implement this function.

7. **(5 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. **(8 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.

---

**Problem 3 [40 points] A Matrix Class.** In Problem 2, you wrote several functions to manipulate matrices. Indeed, a matrix is a good example of a new type of object for which it would make more sense to use object-oriented design. In this problem, you will create a class called `Matrix` in a module called `problem3.py`. The methods you are asked to implement are exactly similar to the functions you were asked to implement in Problem 2. In fact, you should adapt the code in the previous problem to implement this class. You will also need to rewrite the test code (the code that appears in the `if __name__ == "__main__"` part) to use the `Matrix` class. In other words, you are asked to create `Matrix` objects and make appropriate calls to `Matrix` methods to test out your `Matrix` class implementation.
Implement the following methods for your Matrix class. Recall that all class methods must have self as the first parameter.

- __init__: The constructor initializes the matrix by storing its elements as a list of lists. Hence, the constructor will have two parameters: self and a list of equal-length lists.
- dimension(self): Returns the dimension of the matrix.
- row(self, i): Returns the i-th row of the matrix.
- column(self, j): Returns the j-th column of the matrix.
- __add__: The overloaded operator that returns the matrix sum of its two matrix parameters. Note that a Matrix object is returned.
- __sub__: The overloaded operator that returns the matrix difference of its two matrix parameters. Note that a Matrix object is returned.
- __mul__: The overloaded operator that returns the matrix product of its two matrix parameters. Note that a Matrix object is returned.
- reduce_matrix(self, i, j): Returns the matrix obtained by deleting row i and column j from the matrix. Note that a Matrix object is returned.
- determinant: Returns the determinant of the matrix.
- __str__: Returns a printable string representation of the matrix. The string should be neatly formatted, in a manner similar to the pretty_print function of the previous problem.
- Finally, rewrite the test code provided to you (in problem2.py) to create Matrix objects and make appropriate calls to Matrix methods.

Point distribution will be similar to Problem 2, but reduced by about 1 point per method. The remaining points will be assigned to the test code portion of the problem.

**Submission Guidelines**

Implement the first problem in a file called problem1.py, the second problem in a file called problem2.py (download this file from Sakai, as it contains test code for your functions), and the third one in a file called problem3.py. 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 sakai.rutgers.edu.
2. Log in by using your Rutgers login id and password, and click on the 'OBJECT-ORIENTED PROG S17' tab.
3. Click on the 'Assignments' link on the left and go to 'Homework Assignment #3' to find the homework file (hw3.pdf) and the module stub problem2.py for Problem 2.
4. Use this same link to upload your two homework files (problem1.py, problem2.py, and problem3.py) when you are ready to submit.

You must submit your assignment at or before 11:55PM on March 19, 2017.