CECS 229 Programming Assignment #5

Due Date:

Tuesday, 11/7 @ 11:59 PM

Submission Instructions:

Complete the programming problems in the file named `pa5.py` . You may test your implementation on your Repl.it workspace by running main.py . When you are satisfied with your implementation,

1. Submit your Repl.it workspace
2. Download the file `pa5.py` and submit it to the appropriate CodePost auto-grader folder.

Objectives:

1. Define a matrix data structure with relevant matrix operations.
2. Understand the role of matrices in simple image processing applications.

Notes:

Unless otherwise stated in the FIXME comment, you may not change the outline of the algorithm provided by introducing new loops or conditionals, or by calling any built-in functions that perform the entire algorithm or replaces a part of the algorithm.

Directions:

Implement a class `Matrix` that represents $m \times n$ matrix objects with attributes

1. `colsp` -column space of the `Matrix` object, as a list of columns (also lists)
2. `rowsp` -row space of the `Matrix` object, as a list of rows (also lists)

The constructor takes a Python `list` of rows as an argument, and constructs the column space from this rowspace. If a list is not provided, the parameter defaults to an empty list.

You must implement the following methods in the `Matrix` class:

Setters

- `set_row(self, i, new_row)` - changes the i-th row to be the list `new_row`. If `new_row` is not the same length as the existing rows, then method raises a `ValueError` with the message `Incompatible row length.`
- `set_col(self, j, new_col)` - changes the j-th column to be the list `new_col`. If `new_col` is not the same length as the existing columns, then the method raises a `ValueError` with the message `Incompatible column length.`

- `set_entry(self,i, j, val)` - changes the existing a_{ij} entry in the matrix to `val`. Raises `IndexError` if i does not satisfy $1 \leq i \leq m$ or j does not satisfy $1 \leq j \leq n$, where $m =$ number of rows and $n =$ number of columns.

Getters

- `get_row(self, i)` - returns the i-th row as a list. Raises `IndexError` if i does not satisfy $1 \leq i \leq m$.
- `get_col(self, j)` - returns the j-th column as a list. Raises `IndexError` if j does not satisfy $1 \leq j \leq n$.
- `get_entry(self, i, j)` - returns the existing a_{ij} entry in the matrix. Raises `IndexError` if i does not satisfy $1 \leq i \leq m$ or j does not satisfy $1 \leq j \leq n$, where $m =$ number of rows and $n =$ number of columns.
- `col_space(self)` - returns the *list* of vectors that make up the column space of the matrix object
- `row_space(self)` - returns the *list* of vectors that make up the row space of the matrix object
- `get_diag(self, k)` - returns the k-th diagonal of a matrix where $k = 0$ returns the main diagonal, $k > 0$ returns the diagonal beginning at $a_{1(k+1)}$, and $k < 0$ returns the diagonal beginning at $a_{(-k+1)1}$. e.g. `get_diag(1)` for an $n \times n$ matrix returns [$a_{12}, a_{23}, a_{34}, \ldots, a_{(n-1)n}$]

Helper methods

- `_construct_rowsp(self, colsp)` - constructs the row space of this `Matrix` using the given list of lists `colsp` representing the column space of this `Matrix`

- `_construct_colsp(self, rowsp)` - constructs the column space of this `Matrix` using the given list of lists `rowsp` representing the row space of this `Matrix`

Overloaded operators

In addition to the methods above, the `Matrix` class must also overload the `+`, `-`, and `*` operators to support:

1. `Matrix + Matrix` addition; must return `Matrix` result
2. `Matrix - Matrix` subtraction; must return `Matrix` result
3. `Matrix * scalar` multiplication; must return `Matrix` result
4. `Matrix * Matrix` multiplication; must return `Matrix` result
5. `Matrix * Vec` multiplication; must return `Vec` result
6. `scalar * Matrix` multiplication; must return `Matrix` result

In [ ]:
```python
class Matrix:

    def __init__(self, rowsp):
        self.rowsp = rowsp
        self.colsp = self._construct_colsp(rowsp)
```

```
    # todo
    """
    INSERT MISSING SETTERS AND GETTERS HERE
    """

    def _construct_colsp(self, rowsp):
        colsp = []
        # todo: INSERT YOUR IMPLEMENTATION HERE
        return colsp

    def _construct_rowsp(self, colsp):
        rowsp = []
        # todo: INSERT YOUR IMPLEMENTATION HERE
        return rowsp

    def __add__(self, other):
        pass # todo: REPLACE WITH IMPLEMENTATION

    def __sub__(self, other):
        pass # todo: REPLACE WITH IMPLEMENTATION

    def __mul__(self, other):
        if type(other) == float or type(other) == int:
            print("FIXME: Insert implementation of MATRIX-SCALAR multiplication")  # t
        elif type(other) == Matrix:
            print("FIXME: Insert implementation of MATRIX-MATRIX multiplication") # to
        elif type(other) == Vec:
            print("FIXME: Insert implementation for MATRIX-VECTOR multiplication")  #
        else:
            print("ERROR: Unsupported Type.")
        return

    def __rmul__(self, other):
        if type(other) == float or type(other) == int:
            print("FIXME: Insert implementation of SCALAR-MATRIX multiplication")  # t
        else:
            print("ERROR: Unsupported Type.")
        return

    def __str__(self):
        """prints the rows and columns in matrix form """
        mat_str = ""
        for row in self.rowsp:
            mat_str += str(row) + "\n"
        return mat_str

    def __eq__(self, other):
        """overloads the == operator to return True if
        two Matrix objects have the same row space and column space"""
        return self.row_space() == other.row_space() and self.col_space() == other.col

    def __req__(self, other):
        """overloads the == operator to return True if
        two Matrix objects have the same row space and column space"""
        return self.row_space() == other.row_space() and self.col_space() == other.col
```