CSE 20
Beginning Programming in Python
Programming Assignment 7

In this assignment you will write a Python class called `Matrix` that represents an $n \times m$ rectangular matrix as a dictionary. The source code file containing this class will be `matrix.py`, making the module name `matrix`. This follows the naming convention used in Python library modules. (Module name in all lower case and class name capitalized, like the `turtle` module containing class `Turtle`, and the `random` module containing class `Random`.) Begin by studying the examples `vector.py` in /Examples/Vector on the class webpage. In that example, we represent 3-dimensional vectors as dictionaries, and implement some common vector operations.

The files `matrix-stub.py`, `MatrixTest.py` and `MatrixTestOut` are located in /Examples/pa7, and will be of use in designing and testing your `Matrix` class. In particular, `matrix-stub.py` contains the definition of the `Matrix` class, along with headings for 11 required functions (3 built-in functions, 5 instance methods, and 3 class methods).

Most importantly, the `__init__()` function definition is provided for you. It establishes that a `Matrix` object consists of the 3 attributes: `numRows` (int), `numCols` (int), and `elements` (dictionary). The first two give the number of rows and columns in the matrix, respectively. The `elements` dictionary contains nm key-value pairs, where $n = \text{numRows}$ and $m = \text{numCols}$. Each pair is of the form $(i, j){:}\, x$. The tuple (i, j) is the key ($1 \le i \le n$, $1 \le j \le m$) and x is the value, which specifies the element in the i^{th} row and j^{th} column of the matrix. Function `__init__()` takes as its input, the list-of-lists representation of a matrix, something with which you are by now familiar. If the input list-of-lists has inner lists of differing lengths, say for instance `[[1,2,3],[4,5]]`, the function raises a `ValueError` exception with the message

```
could not create Matrix from ragged list:
[[1, 2, 3], [4, 5]]
```

Observe that the offending list-of-lists is quoted in the message. If the input list-of-lists is empty `[]` (which is the default), an empty matrix is created, i.e. one having no rows, no columns and an empty dictionary.

The specifications of the remaining 10 methods can be inferred from the program `MatrixTest.py` and its expected output `MatrixTestOut`. Note that the last line in this program is a call to `help(Matrix)`, which prints the required doc strings for all methods in the class. A good place to begin the project would be to change the name of the file `matrix-stub.py` to `matrix.py`, include your standard comment block at the top, and then type each of the required doc strings below the headings of the 10 remaining functions.

Once you have completed the body of each function, you can type

```
$ python3 MatrixTest.py > myMatrixTestOut
```

at the Unix command prompt `$`. The Unix redirect operator > sends the standard output of a program to a file, in this case `myMatrixTestOut`. The Unix command

```
$ diff myMatrixTestOut MatrixTestOut
```

prints the differences between your output and the model output in `MatrixTestOut`. If this difference is nothing (i.e. no output printed), then your `Matrix` class has passed this test. You should of course perform your own, more stringent tests.

Pay special attention to the exceptions raised by functions `add()`, `sub()`, `mult()` and `from_string()` when passed 'bad' input, and the resulting `ValueError` messages that result. What constitutes 'bad' input for these functions, and how these messages should read, can again be inferred from the files `MatrixTest.py` and `MatrixTestOut`.

The class function `from_string()` is very similar to the initialization function `__init__()`, except that it takes a string `s` as input, instead of a list. This string consists of a space separated list of elements for each row, and separates rows by a newline '`\n`' character. Extra spaces in this string are not significant. It returns a new Matrix object represented by the string `s`. For instance, the string

```
s = '1 2 3\n4 5 6'
```

as input to function `from_string()`, and the list

```
L = [[1,2,3],[4,5,6]]
```

as input to the constructor `Matrix()` (which implicitly calls `__init__()`), would create the same underlying dictionary representation

```
{(1,1):1, (1,2):2, (1,3):3, (2,1):4, (2,2):5, (2,3):6}
```

which itself represents the 2×3 matrix

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}.$$

Function `from_string()` returns an empty `Matrix` (`numRows=0`, `numCols=0` and `elements={}`) when given an empty string `''` (or no argument) as input.

The string returned by `__str__()`, can also be inferred from the output of `MatrixTest.py`. Its form is basically the same as the above string `s`, but with added formatting so that it looks nice when printed. In particular, each numerical value is formatted as a float, right justified in a field of some width (which you must determine), with 2 digits to the right of the decimal. Thus, using `s` from above, if we assign the variable `t = str(Matrix.from_string(s))`, we get

```
t = '    1.00     2.00     3.00\n    4.00     5.00     6.00'
```

Function `__eq__()` is the third built in function you will define. It overloads the == operator in such a way that `A==B` returns `True` if and only if matrices `A` and `B` have the same values in each row and column. A good way to test all three functions `__eq__()`, `__str__()` and `from_string()` is to evaluate the expression

```
A==Matrix.from_string(str(A))
```

which should be `True` for any matrix object `A`.

If you are familiar with the matrix operations: addition, subtraction, scalar multiplication, matrix multiplication, and transpose, then you are well situated to proceed with this project. If not, here is a brief review of these topics, usually studied in an algebra class.

Suppose we are given two $n \times m$ matrices $A = (a_{ij})$ and $B = (b_{ij})$, where a_{ij} and b_{ij} denote the elements in row i, column j, respectively ($1 \le i \le n$, $1 \le j \le m$). Their *sum* and *difference* are the $n \times m$ matrices

$$A + B = (a_{ij} + b_{ij})$$

and

$$A - B = (a_{ij} - b_{ij}).$$

In other words, we simply add (respectively subtract) corresponding elements in the two matrices to find their sum (respectively difference). For instance, if

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 4 & 0 & -1 \\ 2 & -7 & 3 \end{pmatrix}$$

then

$$A + B = \begin{pmatrix} 5 & 2 & 2 \\ 6 & -2 & 9 \end{pmatrix} \quad \text{and} \quad A - B = \begin{pmatrix} -3 & 2 & 4 \\ 2 & 12 & 3 \end{pmatrix}$$

Matrix addition and subtraction are only defined if A and B have the same number of rows and columns. Thus 'bad' input for functions `add()` and `sub()` means that the two arguments are not of compatible sizes, and `ValueError` should be raised in such a case.

If c is a real number (also called a *scalar*), then the *scalar product* of c with A is the matrix

$$cA = (c \cdot a_{ij})$$

for $1 \le i \le n$ and $1 \le j \le m$. In other words, multiply each element of A by the number c to obtain the scalar product.

If $A = (a_{ik})$ is an $n \times p$ matrix ($1 \le i \le n$, $1 \le k \le p$), and $B = (b_{kj})$ is a $p \times m$ matrix ($1 \le k \le p$, $1 \le j \le m$), then the *matrix product* of A with B is the $n \times m$ matrix

$$C = A \cdot B$$

Where $C = (c_{ij})$ has elements defined by

$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}$$

for $1 \le i \le n$ and $1 \le j \le m$. For instance, let

$$A = \begin{pmatrix} 2 & 0 & 1 \\ 1 & 3 & 0 \end{pmatrix} \qquad\qquad 2 \times 3$$

and

$$B = \begin{pmatrix} 0 & 1 \\ 1 & -1 \\ 4 & 2 \end{pmatrix} \qquad 3 \times 2.$$

Then

$$A \cdot B = \begin{pmatrix} 2 & 0 & 1 \\ 1 & 3 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & -1 \\ 4 & 2 \end{pmatrix} = \begin{pmatrix} 4 & 4 \\ 3 & -2 \end{pmatrix} \qquad 2 \times 2,$$

and

$$B \cdot A = \begin{pmatrix} 0 & 1 \\ 1 & -1 \\ 4 & 2 \end{pmatrix} \cdot \begin{pmatrix} 2 & 0 & 1 \\ 1 & 3 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 3 & 0 \\ 1 & -3 & 1 \\ 10 & 6 & 4 \end{pmatrix} \qquad 3 \times 3.$$

Note that $A \cdot B$ is defined only when the number of columns of A equals the number of rows of B. This identifies 'bad' input for function `mult()`. If A is of size $n \times p$ and B is of size $q \times m$, and if $p \neq q$, then the product `A.mult(B)` is undefined, and a `ValueError` should be raised.

Finally, if $A = (a_{ij})$ is an $n \times m$ matrix, then its transpose is the $m \times n$ matrix $A^T = (a_{ji})$, where $1 \leq i \leq n$ and $1 \leq j \leq m$. In other words, A^T is obtained form A by interchanging rows with columns. For instance, given

$$A = \begin{pmatrix} 2 & 0 & 1 \\ 1 & 3 & 0 \end{pmatrix} \qquad 2 \times 3,$$

then

$$A^T = \begin{pmatrix} 2 & 1 \\ 0 & 3 \\ 1 & 0 \end{pmatrix} \qquad 3 \times 2.$$

For this assignment, submit both files `matrix.py` (created by you) and `MatrixTest.py` (unchanged) to assignment pa7 before the due date. This project may be the most difficult of the quarter, if only because there is more to do. For that reason, do not delay in getting started, and ask questions and get help as soon as possible.