CSCI 1133
Exercise Set 5

You should attempt as many of these problems as you can. Practice is *essential* to learning and reinforcing computational problem solving skills. We encourage you to do these without any outside assistance..

Create a directory named `exercise5` and save each of your problem solutions in this directory. You will need to save your source files in this directory and push it to your repository in order to get help/feedback from the TAs. This requires that you follow a rigid set of naming requirements. Name your individual Python source files using "ex5" followed by the letter of the exercise, e.g., "`ex5a.py`", "`ex5b.py`", "`ex5c.py`", etc.

Automatic testing of your solutions is performed using a GitHub agent, so it is necessary to name your source files precisely as shown and push them to your repository as you work.

A. **Matrix Data**

Mathematical matrices are generally represented in computer programs using 2-dimensional lists of scalar values. For example, the matrix:

```
1  2  3
4  5  6
7  8  9
```

would be represented by a `list` of 3 "rows", in which each row is a list containing 3 scalar values. e.g:

```
[[1,2,3],[4,5,6],[7,8,9]]
```

The Python language does not provide a built-in type to represent matrices, so there are no built-in methods for entering matrix data and constructing matrices. It has to be done using a loop that constructs the multidimensional list one element or row at a time.

Write a Python program that will solicit matrix data from a user and print out the constructed matrix in "standard" matrix form after it has been entered. Your program needs to include two non-pure functions: `getMatrix()` and `printMatrix(m)` that will input matrix data from a user, and print out a matrix in standard form, respectively.

The `getMatrix()` function takes no arguments, inputs data from the user a *row* at a time, dynamically constructs the matrix and returns it as a 2-dimensional list. The input should be terminated when the user enters a null-string as input. See the note below for an explanation of entering rows of data using the built-in Python `eval()` function.

The `printMatrix(m)` function takes a single 2-dimensional list argument representing a matrix as described above and prints it out in "standard" form: one row per line, elements aligned vertically.

Your `main()` program function simply needs to contain the following line:

```
printMatrix(getMatrix())
```

Using the built-in Python `eval()` function to input lists and tuples:

The very powerful built-in function, `eval()` takes any string and "evaluates" it as a Python expression, returning the value of the expression!

Recall that the Python `input()` function returns a string object. So if several values (separated by commas) are entered, `eval()` can be used to convert the entered values to a tuple:

```
instring = input("enter row of data separated by commas")
rowdata = eval(instring)
```

How the user enters the data affects the resulting object. For example, if the user enters:

1, 2, 3, 4, 5

`eval()` will convert it and return a tuple. However if the user enters:

[1, 2, 3, 4, 5]

`eval()` will convert it and return a list!

You can easily deal with this by using the `list()` constructor and converting the evaluated object to a list. e.g.:

```
row = list(rowdata)
```

Example:

```
enter values separated by commas: 1,5,7,9
enter values separated by commas: 14,3.14,2.3,10
enter values separated by commas: 2,2,2,2
enter values separated by commas:

 1      5      7      9
14   3.14    2.3     10
 2      2      2      2
```

B. **Matrix Addition**

The sum of two matrices, *A* and *B,* is a new matrix, *R* whose values consist of the sum of the corresponding elements in *A* and *B,* i.e.:

$$R_{ij} = A_{ij} + B_{ij}$$

Note that the *size* of the two matrices must match!

Write a Python program that will input two matrices from the user and output their sum in standard matrix form. Use the two functions you constructed in problem (A) and construct two new functions:

1. A pure function, `matrixSum(A,B)`, that takes a pair of matrices and returns their sum. All matrices are represented using 2-dimensional lists of rows of columns and must be the same size. Do this with a nested loop, do not import any modules.

2. A pure Boolean function, `sizeMatch(A,B)` that will take a pair of matrices and return `True` if they both have the same number of rows and columns, `False` otherwise.

Your main program function should input two matrices and compare their sizes. The sizes do not match, output an appropriate error message, otherwise compute and print the resulting matrix sum.

Examples:

```
Matrix A:
enter values separated by commas: 1,2,3
enter values separated by commas: 4,5,6
enter values separated by commas: 7,8,9
enter values separated by commas:
Matrix B:
enter values separated by commas: 2,2,2
enter values separated by commas: 3,3,3
enter values separated by commas: 4,4,4
enter values separated by commas:

 3    4    5
 7    8    9
11   12   13
```

```
Matrix A:
enter values separated by commas: 1,1
enter values separated by commas: 2,2
enter values separated by commas:
Matrix B:
enter values separated by commas: 2,2,2
enter values separated by commas: 3,3,3
enter values separated by commas:

error: cannot sum matrices of different sizes
```

C. **List Insertion**

Write a Python program that constructs and prints a lexicographically ordered `list` of words by inserting each word into its correct positions in the list as it is entered (i.e., do not use the built-in sort() method!)

Include a function named `insertString(alist,newword)` that constructs and returns a *new* list by inserting the new word into the list in the proper location. The function takes two arguments: 1) a list of words that is already in (increasing) lexicographic order and 2) the word to be inserted It returns a *new* list with the inserted string element in the correct lexicographic position in the list. Do not modify the caller's list argument and use only `while` loop(s) with basic indexing operations (i.e., no slicing, sorting, etc.), The only list method you may use is `.append()`:

Example:

```
enter word: aardvark
enter word: zebra
enter word: hawk
enter word: lion
enter word: anteater
enter word:

aardvark
anteater
hawk
lion
zebra
```

D. **Shortest Distance**

The shortest distance between 2 points (x_1,y_1), (x_2,y_2) on a Cartesian plane is given by Pythagoras:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Write a Python program that will input a 2-dimensional list of points and print the shortest distance between any two points in the list. (Use the matrix input function you created in problem (A) to input the points)

Include a pure function named `shortestDist` that will take a *list* of points as its only argument and return the shortest distance between any two points in the list. Each point in the list is represented by a list of two elements:

$$[\ [x_1,y_1], [x_2,y_2], \ldots , [x_n,y_n]\ ]$$

Note that this will require an "all-pairs" comparison. Avoid comparing a point with itself!

Example:

```
enter values separated by commas: 1,1
enter values separated by commas: 2,2
enter values separated by commas: 4,4
enter values separated by commas:

shortest distance is 1.414213
```