Algorithms for Data Science
Lab 1
Matrix Multiply

For this lab, you will be writing a Python version of matrix multiply. In the video/PowerPoint slides, the concept of matrix multiplication was discussed, complete with an example and even pseudocode for multiplying two NxN square matrices.

Below you will find the starting code you are to use. The given code contains a helper function called printMatrix (which simply prints a given matrix in a nicer format) as well as the function you will create called matrixMult. It also contains the testing code you are to use. That is, you are not to change the given code. The matrices must be defined, matrixMult called, and return values used in the way given. You are free to add extra test cases if you wish.

The first test case is the one from the video/PowerPoint slides. A and B are square 3x3 matrices and should result in a square 3x3 matrix being returned from matrixMult. I would start by commenting out the other test cases and getting this first one to work. The pseudocode given in the video/PowerPoint slides gives the blueprint of what you are to write.

However, there are a few things you need to add to go from pseudocode to Python implementation. The first is that you are not given the dimensions of the matrices passed to the function, but you can find this out by taking the length of the matrix and the length of the first element of the matrix:

```
len(A)
len(A[0])
```

The second is that in the pseudocode the Matrix C was just used and never created first. Thus, you will need to create an empty matrix of the correct dimensions. There are a number of ways you could do this, but one of the easiest in Python is to do something like the following:

```
C = [[None for i in range(3)] for j in range(3)]
```

After you have the first test case working, uncomment the second test case. The only difference in this test case is the dimensions of Matrix A have been changed. Note that you can no longer multiply AxB because the inner dimensions of the matrices much match to multiply them together. Add to your code to provide this dimension check. If you can't multiply the matrices, then you should print out something that says so and return None. Otherwise, you can proceed as usual.

The third case introduces nonsquare matrices. Modify your code so that it works in these cases as well. Note that the result for this particular test case is

```
[12, 22, 23]
[15, 34, 57]
```

```
def printMatrix(m):
    for row in m:
        print(row)

def matrixMult(A, B):

     # Your code goes here



# Testing code
# Test1
A = [[ 2, -3,  3],
     [-2,  6,  5],
     [ 4,  7,  8]]
B = [[-1,  9,  1],
     [ 0,  6,  5],
     [ 3,  4,  7]]

C = matrixMult(A, B)
if not C == None:
    printMatrix(C)

# Test2
A = [[ 2, -3,  3,  0],
     [-2,  6,  5,  1],
     [ 4,  7,  8,  2]]
B = [[-1,  9,  1],
     [ 0,  6,  5],
     [ 3,  4,  7]]

C = matrixMult(A, B)
if not C == None:
    printMatrix(C)

# Test3
A = [[ 2, -3,  3,  5],
     [-2,  6,  5, -2]]
B = [[-1,  9,  1],
     [ 0,  6,  5],
     [ 3,  4,  7],
     [ 1,  2,  3]]

C = matrixMult(A, B)
if not C == None:
    printMatrix(C)
```