Unit Four 2D Lists

removeRowAndCol(A,row,col) (5 points)
Write the function removeRowAndCol(A, row, col) that takes a 2d list A, a row index and a col index, and non-destructively returns a new 2d list with the given row and column removed. Also, A remains unchanged. So, if:

```
A = [ [ 2, 3, 4, 5],
      [ 8, 7, 6, 5],
      [ 0, 1, 2, 3]]
```

Then removeRowAndCol(A, 1, 2) returns:

```
    [ [ 2, 3, 5],
      [ 0, 1, 3]]
```

isMagicSquare(a) (10 points)
Write the function isMagicSquare(a) that takes an arbitrary list (that is, a possibly-empty, possibly-ragged, possibly-2d list of arbitrary values) and returns True if it is a magic square and False otherwise, where a magic square has these properties:

- The list is 2d, non-empty, square, and contains only integers, where no integer occurs more than once in the square.
- Each row, each column, and each of the 2 diagonals each sum to the same total.

If you are curious, go here for more details, including this sample magic square:

2	7	6	→15
9	5	1	→15
4	3	8	→15
↓15	↓15	↓15	↘15

makeMagicSquare(n) (15 points)
Write the function makeMagicSquare(n) that takes a positive odd integer n and returns an nxn magic square. One way to do this is by following De La Loubere's Method (sometimes known as the Siamese method). If n is not a positive odd integer, return [].

findPrimitives(p) (10, 15 or 20 points)
Write the function findPrimitives(p) that finds all primitive Pythagorean triples with perimeters less than or equal to p. Return is a list of tuples that is sorted based on perimeter. The tuples should be formatted in ascending order, like this (3,4,5). So your program should do this:
findPrimitives(10)==[]
findPrimitives(12)==[(3,4,5)]
findPrimitives(84)== [(3, 4, 5), (5, 12, 13), (8, 15, 17), (7, 24, 25), (20, 21, 29), (12, 35, 37)]
findPrimitives(144)== [(3, 4, 5), (5, 12, 13), (8, 15, 17), (7, 24, 25), (20, 21, 29), (12, 35, 37), (9, 40, 41), (28, 45, 53), (11, 60, 61), (16, 63, 65)]
To get 20 points, your algorithm has to find all primitives with perimeters below one million in less than 1 second. 15 points for finding all primitives with perimeters below 100,000 in less than one second.

isLatinSquare(a) (5 points)
Write the function isLatinSquare(a) that takes a 2d list and returns True if it is a Latin square and False otherwise. An example of a Latin Square is shown to the right. A Latin Square has these properties:

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \\ 3 & 4 & 1 & 2 \\ 4 & 3 & 2 & 1 \end{bmatrix}$$

- The list is 2d, non-empty, square, and contains only integers, where no integer occurs more than once in any row or column.
- Each row and each column contain the same set of integers

makeLatinSquare(n) (15 points)
Write the function makeLatinSquare(n) that takes in a positive even integer(n) and returns a valid Latin Square of dimension nxn. There are many ways to do this. There are many methods to generate Latin Squares. Find one and get some easy points.

matrixMultiply(m1,m2) (15 points or 25 points)
Write the function matrixMultiply(m1, m2) that takes two 2d lists (think of them as matrices) and returns a new 2d list that is the result of multiplying the two matrices. For 15 points, simply perform m1 x m2. For 25 points, you have to keep in mind that matrix multiplication is not commutative-- m1 x m2 might be possible, m2 x m1 might be possible or both might be possible. Your program will have to decide what works and produce appropriate results. Return is one list (in the event there are two possible answers, return a list containing both lists)
Return [] if the two matrices cannot be multiplied for any reason.

largestProductInAGrid(grid,z) (15 or25 points)
Write a function largestProductInAGrid(grid,z) that takes a rectangular 2D list and finds the largest product of z adjacent elements in any row, column or diagonal. This problem is modeled after Project Euler Problem #11. Click here for the Euler Problem Statement. The Euler problem is for a 20x20 grid with a product of 4 numbers. That will earn partial credit. Generalizing so that any rectangular grid with any length product (up to the smaller of the list dimensions) earns the 25 points.

powerSet(L) (10 points or 20 points)

Write the function powerSet(L) that takes in a list and finds its power set as a list of lists. If you ask me, I'll give you a method to create a power set by using binary numbers. You are allowed to use other methods if you choose, but you cannot use itertools from the python library. Also, do not use any recursive techniques you find on the Internet that use generators. Converting decimal to binary numbers is discussed here. The subsets do not have to be in any particular order. So for example:

powerSet([1,2,3])== [[], [3], [2], [2, 3], [1], [1, 3], [1, 2], [1, 2, 3]]

powerSet([4,5,6,7])== [[], [7], [6], [6, 7], [5], [5, 7], [5, 6], [5, 6, 7], [4], [4, 7], [4, 6], [4, 6, 7], [4, 5], [4, 5, 7], [4, 5, 6], [4, 5, 6, 7]]

Note that a power set does not have any duplicate subsets of different order. For example:

powerSet([1,2,3])!= [[], [3], [2], [2, 3], [3,2],[1], [1, 3], [1, 2], [1, 2, 3],[3,2,1]]

However, this is not to say that duplicate subsets can't occur. They can. But only if there are duplicates in the data. For example:

powerSet([1,2,3,2])== [[], [1], [2], [2], [3], [1, 2], [1, 2], [1, 3], [2, 2], [2, 3], [3, 2], [1, 2, 2], [1, 2, 3], [1, 3, 2], [2, 3, 2], [1, 2, 3, 2]]

For 20 points, you have to order the power set.

The goal here is to get the power set into a format like the first example from above. So the empty set is first, followed by all single element sets in ascending order, then two element sets in ascending order etc. You get 20 points if your program outputs like this:

powerSet ([1,7,3]) [[], [1], [3], [7], [1, 3], [1, 7], [3, 7], [1, 3, 7]]

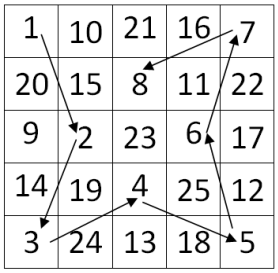
powerSet ([2,8,6,3])== [[], [2], [3], [6], [8], [2, 3], [2, 6], [2, 8], [3, 6], [3, 8], [6, 8], [2, 3, 6], [2, 3, 8], [2, 6, 8], [3, 6, 8], [2, 3, 6, 8]]

subsetSum(L,s) (10 or 20 points)

Write the function subsetSum(a,s) that takes a list of int values (L) and returns a list of list values $[L_x, L_y,\ldots,L_z]$ such that sum($[L_x,L_y,\ldots L_z]$)=s, in increasing order. If no such list exists, return []. For example, subsetSum([2,5,-13,-6,4,3,-7],0) could return [-6,2,4] since -6+2+4== 0 (or it could return [-7,2,5] since -7+2,5==0. There may be several valid answers for any given list and subset sum, in which case your function may return any one of them. Return them all as a list of lists for the extra 10 points. If you return them all your return must be an ordered list and your list cannot contain duplicates.

isKnightsTour(board) (15 points)

Background: A "knight's tour" in chess is a sequence of legal knight moves such that the knight visits every square exactly once. We can represent a (supposed) knight's tour as an NxN list of the integers from 1 to N^2 listing the positions in order that the knight occupied on the tour. If it is a legal knight's tour, then all the numbers from 1 to N^2 will be included and each move will be a legal knight's move. The example to the right shows a legal knights tour on a 5x5 board demonstrating the first 7 moves. With this in mind, write the function isKnightsTour(board) that takes such a 2d list of integers and returns True if it represents a legal knight's tour and False otherwise.



smallestSumInARectangularSubGrid(grid,m,n) (15 points)

Write the function smallestSumInARectangularSubGrid that takes in a rectangular grid and finds the mxn block of elements (with m being rows and n being columns of the subGrid) within the grid having the smallest sum. For example, given the grid to the right:

31	11	40	32	11	45
31	40	9	11	46	25
12	43	40	28	38	5
9	45	33	24	32	38
14	40	8	39	15	19

smallestSumInARectangularSubGrid(grid,1,1)==5 (Shown in Yellow)

smallestSumInARectangularSubGrid(grid,2,1)==21 (Shown in Red)

smallestSumInARectangularSubGrid(grid,2,3)==143 (Shown in Blue)

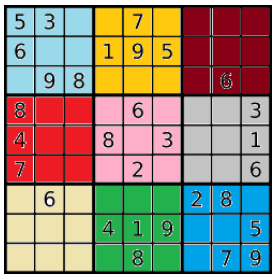
smallestSumInARectangularSubGrid(grid,1,6)==135 (Shown in Green)

smallestSumInARectangularSubGrid(grid,5,6)==814 (Every number in the grid)

multiplyLinearFactors(F) (7 points):
Write the function multiplyLinearFactors(F) that takes in a 2D list (representing a sequence of linear factor binomials) and outputs a 1D list (representing a polynomial function). For example:
multiplyLinearFactors[([1,2],[1,3]])==[1,5,6]
multiplyLinearFactors([[1,2],[1,2],[2,5],[-3,7]]) == [-6, -25, 7, 136, 140]
You should make use of multiplyPolynomials from the last unit. For a really elegant solution, consider doing this one recursively.

volumeOfTetrahedron(coordinates) (7 points)
Write the function volumeOfTetrahedron that calculates the volume of a tetrahedron formed by 4 points in space. Input is a list of 4 tuples (each of which contains three coordinates). Return is a float representing the volume. Go here to see a method of calculating the volume through vector methods.

isLegalSudoku(board)
This problem involves the game Sudoku, though we will generalize it to the N^2xN^2 case, where N is a positive integer (and not just the 3^2x3^2 case which is most commonly played). First, read the top part (up to History) of the Wikipedia page on Sudoku so we can agree on the rules. As for terminology, we will refer to each of the N^2 different N-by-N sub-regions as "blocks". The following figure shows each of the 9 blocks in a 3^2x3^2 puzzle highlighted in a different color:



A Sudoku is in a legal state if all N^4 cells are either blank (in our data contain a zero) or contain a single integer from 1 to N^2 (inclusive), and if each integer from 1 to N^2 occurs at most once in each row, each column, and each block. A Sudoku is solved if it is in a legal state and contains no blanks.

With this description in mind, your task is to write some functions to indicate if a given Sudoku board is legal. To make this problem more approachable, we (the great Professor Kosbie) are providing a specific design for you to follow. In the order given:

areLegalValues(values) (5 points)
This function takes a 1d list of values, which you should verify is of length N^2 for some positive integer N and contains only integers in the range 0 to N^2 (inclusive). These values may be extracted from any given row, column, or block in a Sudoku board (and, in fact, that is exactly what the next few functions will do -- they will each call this helper function). The function returns True if the values are legal: that is, if every value is an integer between 0 and N^2, inclusive, and if each integer from 1 to N^2 occurs at most once in the given list (0 may be repeated, of course). Note that this function does not take a 2d Sudoku board, but only a 1d list of values that presumably have been extracted from some Sudoku board.

isLegalRow(board, row) (2 points)
This function takes a Sudoku board and a row number. The function returns True if the given row in the given board is legal (where row 0 is the top row and row (N^2-1) is the bottom row), and False otherwise. To do this, your function must create a 1d list of length N^2 holding the values from the given row, and then provide these values to the areLegalValues function you previously wrote. (Actually, because areLegalValues is non-destructive, you do not have to copy the row; you may use an alias.)

isLegalCol(board, col) (5 points)
This function works just like the isLegalRow function, only for columns, where column 0 is the leftmost column and column (N^2-1) is the rightmost column. Similarly to isLegalRow, this function must create a 1d list of length N^2 holding the values from the given column, and then provide these values to the areLegalValues function you previously wrote.

isLegalBlock(board, block) (10 points)
This function works just like the isLegalRow function, only for blocks, where block 0 is the left-top block, and block numbers proceed across and then down, as described earlier. Similarly to isLegalRow and isLegalCol, this function must create a 1d list of length N^2 holding the values from the given block, and then provide these values to the areLegalValues function you previously wrote.

isLegalSudoku(board) (2 points)
This function takes a Sudoku board (which you may assume is a N^2xN^2 2d list of integers), and returns True if the board is legal, as described above. To do this, your function must call isLegalRow over every row, isLegalCol over every column, and isLegalBlock over every block. See how helpful those helper functions are? Seriously, this exercise is a very clear demonstration of the principle of top-down design and function decomposition. By the way, isLegalSudoku can be easily turned into isSolvedSudoku, which can be of use if you attempt the Sudoku problems in the next unit.

closestDiamondToZero(s,G) (25 points)

Given a 2D list (G), find the diamond of side length (s) whose sum is the closest to zero. Return the diamond as a 2D list and the sum as members of a larger list. If there is a tie, return the diamond that occurs closest to the top and left of the grid. If a diamond cannot be formed given the input size, return []. Consider the minimum edge length of a diamond to be one. Some examples are shown in the 15x15 grid below. You can assume that the grid is rectangular, but not necessarily square.

closestDiamondToZero(1,G)== [[[1]], 1] (shown in blue—wins tie)

closestDiamondToZero(2,G)== [[[-81], [-24, 96, -31], [40]], 0]
(shown in green)

closestDiamondToZero(3,G)== [[[38], [-61, 36, 11], [32, -35, 44, 8, 51],
[-41, -40, -14], [-29]], 0]
(shown in yellow)

closestDiamondToZero(4,G)== [[[-93], [6, 27, -89], [-72, 73, -49, 66, 26],
[96, -75, 25, -72, 71, 92, -32],
[-23, -54, 42, 32, 47], [51, 22, -62],
[-49]], 6]
(shown in red)

closestDiamondToZero(8,G)== [[[65],…,[-96]],-342]
(only diamond of s=8 that can be formed)

closestDiamondToZero(9,G)==[]

-39	85	75	-20	-86	-43	-50	65	-19	-63	-34	-63	78	-86	-2
-89	96	27	-93	-70	-8	1	-66	31	33	70	89	34	-48	94
-67	-35	6	27	-89	6	-14	-86	-63	-95	-73	38	-95	-16	68
39	-72	73	-49	66	26	-87	3	3	61	-61	36	11	39	-44
96	-75	25	-72	71	92	-32	46	-22	32	-35	44	8	51	-2
-36	-23	-54	42	32	47	18	-38	100	14	-41	-40	-14	2	38
10	74	51	22	-62	-12	-5	-65	91	3	-74	-29	-8	98	48
49	-35	-81	-49	98	75	-26	-40	69	54	-47	39	-89	34	-92
97	-24	96	-31	50	82	79	19	-80	-50	-8	-97	-73	77	-93
-88	66	40	-14	-61	39	-43	-57	-72	-79	-86	-4	21	12	-87
-33	-25	81	86	-54	-96	58	-82	7	81	67	86	-20	-8	63
-95	57	85	-65	-99	-46	5	68	80	-98	-79	40	-93	-76	-86
40	41	-87	-61	-37	-46	11	-93	-2	84	100	-76	34	-83	35
-87	-15	85	50	-78	-51	10	87	70	55	55	12	-95	-1	-78
58	-55	-6	67	71	67	23	-96	50	-15	1	61	85	25	96