1. A useful application of two dimensional arrays in programming is to represent grid-like structures. Consider the following grid of values:

0	4	8
1	5	9
2	6	10
3	7	11

   (a) If we wanted to store this data in a two dimensional array, valueGrid, how would we initialize it so that we accessed elements in row-major order? (Row major order means that we access the row first, then the column in the row: valueGrid[row][col] is the element at (row, col) in our grid.
   Ex: valueGrid[0][1] would be 4)

   valueGrid =

   (b) While we normally think of grids in row major order, sometimes it is helpful to use column-major order (Column major order means that we access the column first, and then the row; valueGrid[col][row] is the element at (row, col) in our grid. Ex: valueGrid[1][0] would be 4). How would we initialize valueGrid so that we used column-major order for the grid above?

   valueGrid =

2. Consider the following program, which iterates through a two dimensional array of ints, and at each cell, increments the values next to it. The program prints the two dimensional array before and after it is altered. Feel free to run the code through Python Tutor to see how it works.

```
1  def changeGridVals(grid):
2    '''Iterates through cells of the grid of values, and increments all adjacent values'''
3    # iterate through the rows of the grid
4    for row in range( len(grid) ):
5      # iterate through the columns of the grid
6      for col in range( len(grid[row]) ):
7        #increment top
8        if (row - 1) >= 0:
9          grid[row - 1][col] += 1
10       #increment bottom
11       if (row + 1) < len(grid):
12         grid[row + 1][col] += 1
13       #increment left
14       if (col - 1) >= 0:
15         grid[row][col - 1] += 1
16       #increment right
17       if (col + 1) < len(grid[row]):
18         grid[row][col + 1] += 1
19
20 if __name__ == "__main__":
21     grid = [ [0, 0, 0, 0],
22              [0, 0, 0, 0],
23              [0, 0, 0, 0] ]
24
25     print(f"Before: {grid}")
26     changeGridVals(grid)
27     print(f"After: {grid}")
28
```

Observe the if statements within the nested for loops. Explain in your own words why they are necessary.

The if statements are neceesary to ensure the program doesn't access cells outside the valid range of the 2-D array. This prevents index out of bounds errors.

3. Consider the following code:

```
grid = [["zxy","abc", "lmnop"],
        ["an apple a day", "keeps doctors away", "at least, as they say"],
        ["lalala", "loolooloo", "doopdedoop"]]

for row in range(len(grid)):
    grid[row].sort()
    for col in range(len(grid[row])):
        grid[row][col] = grid[row][col].replace("a","@")

print(grid)

```

(a) Explain in your own words what this code does.

The code uses a 2D list "grid", it sorts each sub-list, then it iterates through each cell of the grid and replaces all occurences of "a" with "@"

(b) What are the values in grid by the end of the code?

"abc", "lmnop", "zxy"

"@n apple @ d@y", "doctors @w@y", "keeps doctors @w@y"

"lol@l@", "doopdedoop", "loolooloo"

4. Imagine you are writing a program that reads in a long string of data, edits some of the data at certain positions, and then returns the data in the same format. The data is given to you as a single string, where each piece of data is separated by commas. Explain how the split() and join() functions could be used to help you write this program.

Using split() to break the original input string into a list, then use join() to reassemble the modified list into a string withe the format.

5. Consider a 2D grid of m rows by n columns. You start at position (0, 0). There is a treasure chest at every position on the grid except for (0, 0). At each position (i, j) on the grid, you can do one of the following:

   - Move left, right, up, or down one unit. This takes one second of time.
   - Loot the chest at (i, j) for a value of $T_{i,j}$. Each treasure, once looted, is consumed and teleports you back to (0, 0). This takes zero seconds of time.

   You have at most s seconds to loot this grid, and you are given the value of the treasure at each grid location as a 2D array. What is the most treasure you can obtain in s seconds?

   (a) After giving some thought to the problem and working through several examples, what is the key idea of the solution? Write a 1-2 sentence explanation of the key idea to solve this problem. This should **not** include any implementation details (no references to Python, data structures, etc.).

   (b) Once you are certain your solution is correct, implement it below in pseudocode.