Multi-Dimensional List

- Two dimensional list
 - For some problems we may have a table or matrix of values that we would like to store in an organized manner
 - Examples
 - Pollution samples across the surface of a lake
 - Temperatures on a thin metal plate
 - We organize the data into rows and columns specifying rows first and then columns
- Consider a box having 9 compartments in a 3 x 3 grid
 - You can view this a grid (or table) with 3 rows and 3 columns
 - Depending on the contents, the box itself can be represented as a two-dimensional list of a certain data type



Multi-Dimensional List

Distance between two cities

Distance Table (in miles)										
	Chicago	Boston	New York	Atlanta	Miami	Dallas	Houston			
Chicago	0	983	787	714	1375	967	1087			
Boston	983	0	214	1102	1763	1723	1842			
New York	787	214	0	888	1549	1548	1627			
Atlanta	714	1102	888	0	661	781	810			
Miami	1375	1763	1549	661	0	1426	1187			
Dallas	967	1723	1548	781	1426	0	239			
Houston	1087	1842	1627	810	1187	239	0			

```
distances = [
    [0, 983, 787, 714, 1375, 967, 1087],
    [983, 0, 214, 1102, 1763, 1723, 1842],
    [787, 214, 0, 888, 1549, 1548, 1627],
    [714, 1102, 888, 0, 661, 781, 810],
    [1375, 1763, 1549, 661, 0, 1426, 1187],
    [967, 1723, 1548, 781, 1426, 0, 239],
    [1087, 1842, 1627, 810, 1187, 239, 0]
]
```

Processing 2-D Lists

- You can view a two-dimensional list as an object that consists of rows, each row being a single-dimensional list
- The rows can be accessed using the index, conveniently called a row index
- The values in each row can be accessed through another index, conveniently called a column index

```
matrix = [
    [1, 2, 3, 4, 5],
    [6, 7, 0, 0, 0],
    [0, 1, 0, 0, 0],
    [1, 0, 0, 0, 8],
    [0, 0, 9, 0, 3],
]
```

	[0]	[1]	[2]	[3]	[4]
[0]	1	2	3	4	5
[1]	6	7	0	0	0
[2]	0	1	0	0	0
[3]	1	0	0	0	8
[4]	0	0	9	0	3

```
matrix[0] is [1, 2, 3, 4, 5]
matrix[1] is [6, 7, 0, 0, 0]
matrix[2] is [0, 1, 0, 0, 0]
matrix[3] is [1, 0, 0, 0, 8]
matrix[4] is [0, 0, 9, 0, 3]

matrix[0][0] is 1
matrix[4][4] is 3
```

Initializing lists

```
matrix = [] # Create an empty list
numberOfRows = eval(input("Enter the number of rows: "))
numberOfColumns = eval(input("Enter the number of columns: "))
for row in range(0, numberOfRows):
  matrix.append([]) # Add an empty new row
  for column in range(0, numberOfColumns):
    value = eval(input("Enter an element and press Enter: "))
    matrix[row].append(value)
print(matrix)
```

Summing All Elements

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]] # Assume a list is given
total = 0

for row in range(0, len(matrix)):
   for column in range(0, len(matrix[row])):
     total += matrix[row][column]

print("Total is " + str(total)) # Print the total
```

Summing Elements by Column

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]] # Assume a list is given
total = 0

for column in range(0, len(matrix[0])):
   for row in range(0, len(matrix)):
     total += matrix[row][column]
   print("Sum for column " + str(column) + " is " + str(total))
```

Finding the Row with the largest sum

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]] # Assume a list is given
maxRow = sum(matrix[0]) # Get sum of the first row in maxRow

indexOfMaxRow = 0
for row in range(1, len(matrix)):
    if sum(matrix[row]) > maxRow:
        maxRow = sum(matrix[row])
        indexOfMaxRow = row

print("Row " + str(indexOfMaxRow)
```

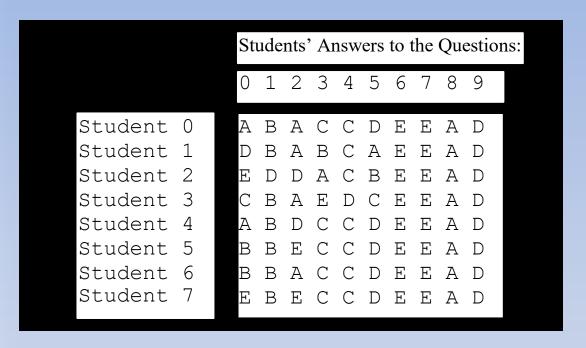
2-D Lists

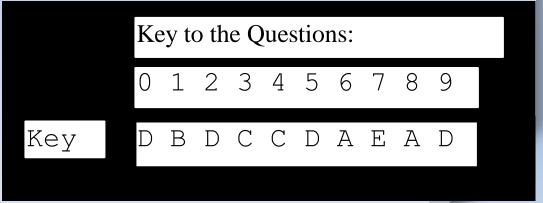
- Sorting
 - Apply the "sort" method to sort a 2-D list
 - It sorts the rows on their first elements
 - For rows with the same first element, they are sorted on the second elements (.. And so on for multidimensional lists)

```
points = [[4, 2], [1,7], [4, 5], [1, 2], [1, 1], [4, 1]]
points.sort()
[[1, 1], [1, 2], [1, 7], [4, 1], [4,2], [4, 5]]
```

- Passing 2-D Lists to Functions or returning values
 - Same as before!

Problem: Grading a Multiple-Choice Test





Grading a Multiple-Choice Test: Solution

```
def main():
    # Students' answers to the questions
    answers = [
      ['A', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A', 'D'],
      ['D', 'B', 'A', 'B', 'C', 'A', 'E', 'E', 'A', 'D'],
      ['E', 'D', 'D', 'A', 'C', 'B', 'E', 'E', 'A', 'D'],
      ['C', 'B', 'A', 'E', 'D', 'C', 'E', 'E', 'A', 'D'],
      ['A', 'B', 'D', 'C', 'C', 'D', 'E', 'E', 'A', 'D'],
      ['B', 'B', 'E', 'C', 'C', 'D', 'E', 'E', 'A', 'D'],
      ['B', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A', 'D'],
      ['E', 'B', 'E', 'C', 'C', 'D', 'E', 'E', 'A', 'D']]
    # Kev to the questions
    kevs = ['D', 'B', 'D', 'C', 'C', 'D', 'A', 'E', 'A', 'D']
    # Grade all answers
    for i in range(len(answers)):
        # Grade one student
        correctCount = 0
        for j in range(len(answers[i])):
            if answers[i][j] == keys[j]:
                correctCount += 1
        print("Student", i, "'s correct count is", correctCount)
main() # Call the main function
```

Multidimensional Lists

Want to extend it to 3 dimensions?
6 students, 5 exams, 2 parts
Scores[6][5][2]

```
scores = [
[[0, 20.5], [1, 22.5], [2, 24.5], [3, 21.0], [4, 22.0]],
[[0, 21.5], [1, 22.0], [2, 24.5], [3, 20.5], [4, 22.5]],
[[0, 20.5], [1, 10.5], [2, 23.5], [3, 23.0], [4, 18.5]],
[[0, 23.5], [1, 20.5], [2, 24.0], [3, 20.5], [4, 19.0]],
[[0, 23.5], [1, 22.0], [2, 21.5], [3, 24.5], [4, 12.0]],
[[0, 20.5], [1, 12.0], [2, 11.5], [3, 20.0], [4, 16.5]]]
```

