# Chapter 11
# Lists for Multi-dimensional Data

# Motivations

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]
]
```

# Objectives

- To give examples of representing data using two-dimensional lists (§11.1).
- To access elements in a two-dimensional list using row and column indexes (§11.2).
- To program common operations for two-dimensional lists (displaying lists, summing all elements, finding min and max elements, and random shuffling) (§11.2).
- To pass two-dimensional lists to functions (§11.3).
- To write a program for grading multiple-choice questions using two-dimensional lists (§11.4).
- To solve the closest-pair problem using two-dimensional lists (§§11.5-11.6).
- To check a Sudoku solution using two-dimensional lists (§§11.7-11.8).
- To use multidimensional lists (§11.9).

# Processing Two-Dimensional lists

You can view a two-dimensional list as a list that consists of rows. Each row is a list that contains the values. 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
```

# Processing Two-Dimensional lists

See the examples in the text.

1. (Initializing lists with input values)
2. (Initializing lists with random values)
3. (Printing lists)
4. (Summing all elements)
5. (Summing all elements by column)
6. (Which row has the largest sum)
7. (*Random shuffling*)

# Initializing lists with input values

```
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)
```

# Initializing lists with random values

```
import random
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):
        matrix[row].append(random.randrange(0, 100))

print(matrix)
```

# Printing lists

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]] # Assume a list is given
for row in range(0, len(matrix)):
    for column in range(0, len(matrix[row])):
        print(matrix[row][column], end = " ")
    print() # Print a newline
```

# 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))
```

# Summing elements by column

```
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))
```

# Random shuffling

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

for row in range(0, len(matrix)):
        for column in range(0, len(matrix[row])):
         i = random.randrange(0, len(matrix))
         j = random.randrange(0, len(matrix[row]))
         # Swap matrix[row][column] with matrix[i][j]
         matrix[row][column], matrix[i][j] = \
             matrix[i][j], matrix[row][column]

print(matrix)
```

# Passing Tow-Dimensional lists to Functions

PassTwoDimensionalList

Run

# Problem: Grading Multiple-Choice Test

Students' Answers to the Questions:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| Student 0 | A | B | A | C | C | D | E | E | A | D |
| Student 1 | D | B | A | B | C | A | E | E | A | D |
| Student 2 | E | D | D | A | C | B | E | E | A | D |
| Student 3 | C | B | A | E | D | C | E | E | A | D |
| Student 4 | A | B | D | C | C | D | E | E | A | D |
| Student 5 | B | B | E | C | C | D | E | E | A | D |
| Student 6 | B | B | A | C | C | D | E | E | A | D |
| Student 7 | E | B | E | C | C | D | E | E | A | D |

- Objective: write a program that grades multiple-choice test.

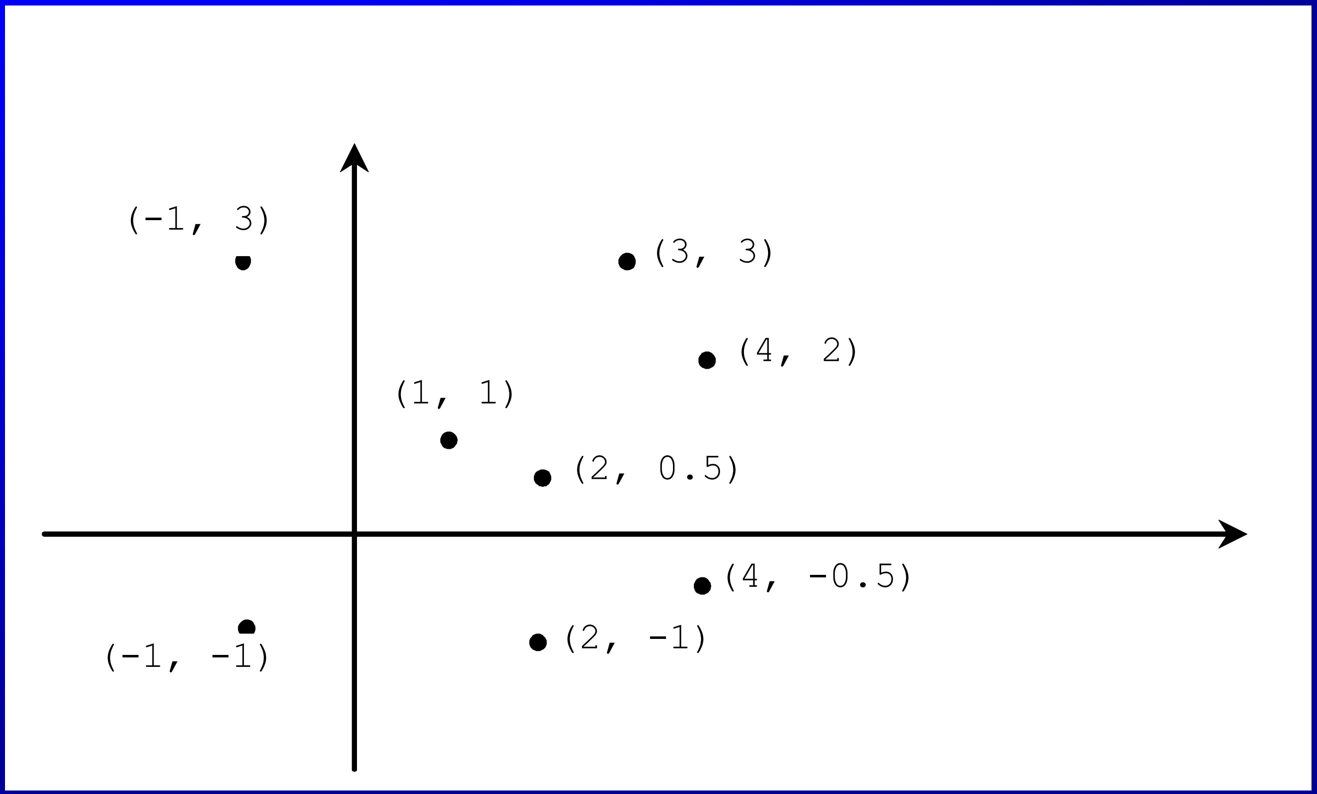
Key to the Questions:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| Key | D | B | D | C | C | D | A | E | A | D |

GradeExam

Run

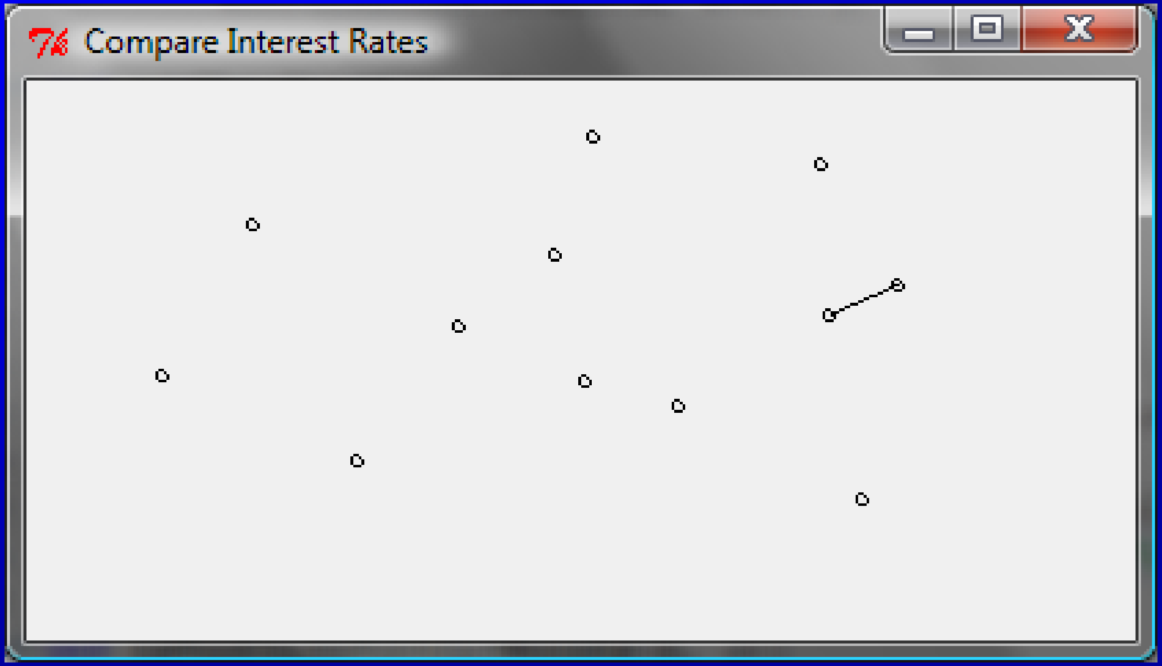
# Problem: Finding Two Points Nearest to Each Other




| | x | y |
|---|---|---|
| 0 | -1 | 3 |
| 1 | -1 | -1 |
| 2 | 1 | 1 |
| 3 | 2 | 0.5 |
| 4 | 2 | -1 |
| 5 | 3 | 3 |
| 6 | 4 | 2 |
| 7 | 4 | -0.5 |

NearestPoints FindNearestPoints Run

# GUI: Finding Two Points Nearest to Each Other




NearestPoints

Run

# What is Sudoku?

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | | | 7 | | | | |
| 6 | | | 1 | 9 | 5 | | | |
| | 9 | 8 | | | | | 6 | |
| 8 | | | | 6 | | | | 3 |
| 4 | | | 8 | | 3 | | | 1 |
| 7 | | | | 2 | | | | 6 |
| | 6 | | | | | | | |
| | | | 4 | 1 | 9 | | | 5 |
| | | | | 8 | | | 7 | 9 |

# Every row contains the numbers 1 to 9

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | | | 7 | | | | |
| 6 | | | 1 | 9 | 5 | | | |
| | 9 | 8 | | | | | 6 | |
| 8 | | | | 6 | | | | 3 |
| 4 | | | 8 | | 3 | | | 1 |
| 7 | | | | 2 | | | | 6 |
| | 6 | | | | | | | |
| | | | 4 | 1 | 9 | | | 5 |
| | | | | 8 | | | 7 | 9 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | *4* | *6* | 7 | *8* | *9* | *1* | *2* |
| 6 | *7* | *2* | 1 | 9 | 5 | *3* | *4* | *8* |
| *1* | 9 | 8 | *3* | *4* | *2* | *5* | 6 | *7* |
| 8 | *5* | *9* | *7* | 6 | *1* | *4* | *2* | 3 |
| 4 | *2* | *6* | 8 | *5* | 3 | *7* | *9* | 1 |
| 7 | *1* | *3* | *9* | 2 | *4* | *8* | *5* | 6 |
| *9* | 6 | *1* | *5* | *3* | *7* | *2* | *8* | *4* |
| *2* | *8* | *7* | 4 | 1 | 9 | *6* | *3* | 5 |
| *3* | *4* | *5* | *2* | 8 | *6* | *1* | 7 | 9 |

# Every column contains the numbers 1 to 9

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | | | 7 | | | | |
| 6 | | | 1 | 9 | 5 | | | |
| | 9 | 8 | | | | | 6 | |
| 8 | | | | 6 | | | | 3 |
| 4 | | | 8 | | 3 | | | 1 |
| 7 | | | | 2 | | | | 6 |
| | 6 | | | | | | | |
| | | | 4 | 1 | 9 | | | 5 |
| | | | | 8 | | | 7 | 9 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | *4* | *6* | 7 | *8* | *9* | *1* | *2* |
| 6 | *7* | *2* | 1 | 9 | 5 | *3* | *4* | *8* |
| *1* | 9 | 8 | *3* | *4* | *2* | *5* | 6 | *7* |
| 8 | *5* | *9* | *7* | 6 | *1* | *4* | *2* | 3 |
| 4 | *2* | *6* | 8 | *5* | 3 | *7* | *9* | 1 |
| 7 | *1* | *3* | *9* | 2 | *4* | *8* | *5* | 6 |
| *9* | 6 | *1* | *5* | *3* | *7* | *2* | *8* | *4* |
| *2* | *8* | *7* | 4 | 1 | 9 | *6* | *3* | 5 |
| *3* | *4* | *5* | *2* | 8 | *6* | *1* | 7 | 9 |

# Every 3×3 box contains the numbers 1 to 9

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | | | 7 | | | | |
| 6 | | | 1 | 9 | 5 | | | |
| | 9 | 8 | | | | | 6 | |
| 8 | | | | 6 | | | | 3 |
| 4 | | | 8 | | 3 | | | 1 |
| 7 | | | | 2 | | | | 6 |
| | 6 | | | | | | | |
| | | | 4 | 1 | 9 | | | 5 |
| | | | | 8 | | | 7 | 9 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | *4* | *6* | 7 | *8* | *9* | *1* | *2* |
| 6 | *7* | *2* | 1 | 9 | 5 | *3* | *4* | *8* |
| *1* | 9 | 8 | *3* | *4* | *2* | *5* | 6 | *7* |
| 8 | *5* | *9* | *7* | 6 | *1* | *4* | *2* | 3 |
| 4 | *2* | *6* | 8 | *5* | 3 | *7* | *9* | 1 |
| 7 | *1* | *3* | *9* | 2 | *4* | *8* | *5* | 6 |
| *9* | 6 | *1* | *5* | *3* | *7* | *2* | *8* | *4* |
| *2* | *8* | *7* | 4 | 1 | 9 | *6* | *3* | 5 |
| *3* | *4* | *5* | *2* | 8 | *6* | *1* | 7 | 9 |

# Checking Whether a Solution Is Correct

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | | | 7 | | | | |
| 6 | | | 1 | 9 | 5 | | | |
| | 9 | 8 | | | | | 6 | |
| 8 | | | | 6 | | | | 3 |
| 4 | | | 8 | | 3 | | | 1 |
| 7 | | | | 2 | | | | 6 |
| | 6 | | | | | | | |
| | | | 4 | 1 | 9 | | | 5 |
| | | | | 8 | | | 7 | 9 |

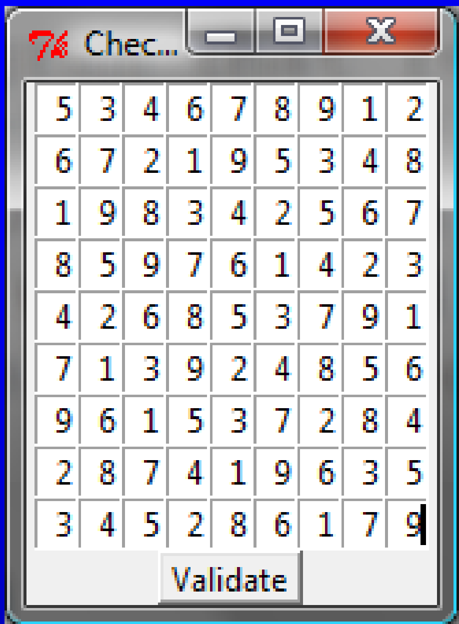
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | *4* | *6* | 7 | *8* | *9* | *1* | *2* |
| 6 | *7* | *2* | 1 | 9 | 5 | *3* | *4* | *8* |
| *1* | 9 | 8 | *3* | *4* | *2* | *5* | 6 | *7* |
| 8 | *5* | *9* | *7* | 6 | *1* | *4* | *2* | 3 |
| 4 | *2* | *6* | 8 | *5* | 3 | *7* | *9* | 1 |
| 7 | *1* | *3* | *9* | 2 | *4* | *8* | *5* | 6 |
| *9* | 6 | *1* | *5* | *3* | *7* | *2* | *8* | *4* |
| *2* | *8* | *7* | 4 | 1 | 9 | *6* | *3* | 5 |
| *3* | *4* | *5* | *2* | 8 | *6* | *1* | 7 | 9 |

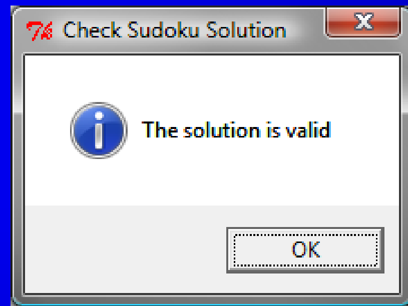
CheckSudokuSolution

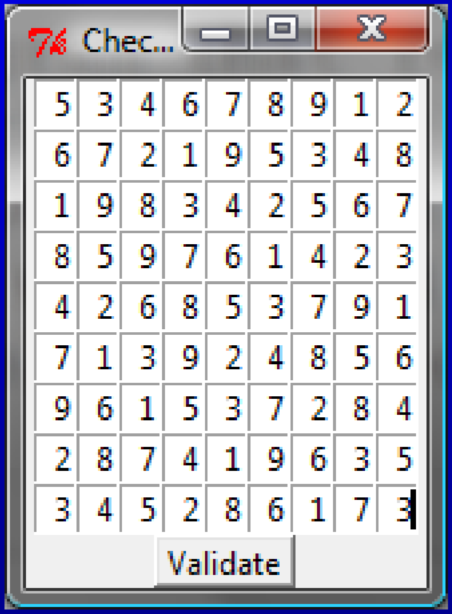
TestheckSudokuSolution

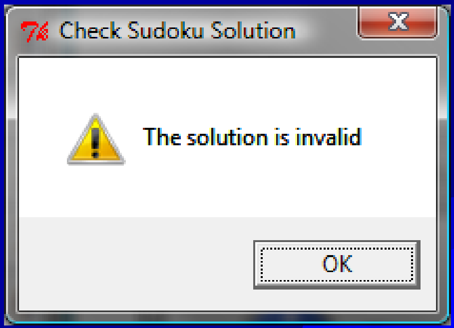
Run

# Sudoku GUI




SudokuGUI

Run

# Multidimensional lists

```
scores = [
  [[9.5, 20.5], [9.0, 22.5], [15, 33.5], [13, 21.5], [15, 2.5]],
  [[4.5, 21.5], [9.0, 22.5], [15, 34.5], [12, 20.5], [14, 9.5]],
  [[6.5, 30.5], [9.4, 10.5], [11, 33.5], [11, 23.5], [10, 2.5]],
  [[6.5, 23.5], [9.4, 32.5], [13, 34.5], [11, 20.5], [16, 9.5]],
  [[8.5, 26.5], [9.4, 52.5], [13, 36.5], [13, 24.5], [16, 2.5]],
  [[9.5, 20.5], [9.4, 42.5], [13, 31.5], [12, 20.5], [16, 6.5]]]
```

```
Which student          Which exam        Multiple-choice or essay

               scores[ i ] [ j ] [ k ]
```

# Problem: Weather Information

- Suppose a meteorology station records the temperature and humidity at each hour of every day and stores the data for the past ten days in a text file named weather.txt. Each line of the file consists of four numbers that indicate the day, hour, temperature, and humidity. Your task is to write a program that calculates the average daily temperature and humidity for the 10 days.

```
1 1 76.4 0.92
1 2 77.7 0.93
...
10 23 97.7 0.71
10 24 98.7 0.74
```

(a)

```
10 24 98.7 0.74
1 2 77.7 0.93
...
10 23 97.7 0.71
1 1 76.4 0.92
```

(b)

Weather

Run

# Problem: Guessing Birthday

- Listing gives a program that guesses a birthday. The program can be simplified by storing the numbers in five sets in a three-dimensional list, and it prompts the user for the answers using a loop, as shown in Listing 7.6. The sample run of the program can be the same as shown in Listing 3.8.

GuessBirthdayUsingList

Run