CHAPTER 11
MULTIDIMENSIONAL LISTS

ACKNOWLEDGEMENT: THESE SLIDES ARE ADAPTED FROM SLIDES PROVIDED WITH INTRODUCTION TO PROGRAMMING USING PYTHON, LIANG (PEARSON 2013)

MOTIVATIONS

- Thus far, you have used one-dimensional lists to model linear collections of elements. You can use a two-dimensional lists to represent a matrix or a table. For example, the following table that describes the distances between the cities can be represented using a two-dimensional array.

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 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
```

WHAT IS NEW HERE?

- Really nothing is new. We just learned lists. Now we have a list-of-lists.
- We are trying to gain comfort with working with large amounts of data!

MULTIDIMENSIONAL LIST EXAMPLES

EXAMPLE

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)
```

EXAMPLE
INITIALIZING LISTS WITH RANDOM VALUES

```python
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)
```

EXAMPLE
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
```

Note how you access a single value, by applying the index operator twice.

EXAMPLE
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
```

Important! It is not len(matrix[0]). Why? Because each row could have a different length.

EXAMPLE
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))
```

EXAMPLE
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)
```

EXERCISE AS A TABLE

- Try the following!
  - 1 – Determine if a value exists in a matrix
  - 2 – Copying a matrix
  - 3 – Finding the row with the largest summation
  - 4 – Finding the maximum of each row into a list of maximums

MULTIDIMENSIONAL LIST DETAILS

AGAIN, THINGS THAT ARE NOT NEW

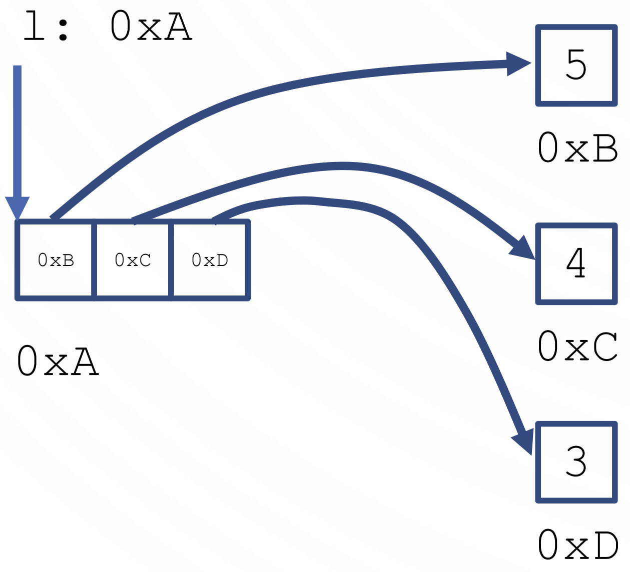
- You can pass a multidimensional list to a function/method
- You can return a multidimensional list from a function/method
- Multidimensional lists are objects, they are passed-by-object-reference
    - Be careful of copying as well!

MEMORY LAYOUT

- A list is a list of objects. So:

  `l = [5, 4, 3]`

  appears like this in memory.

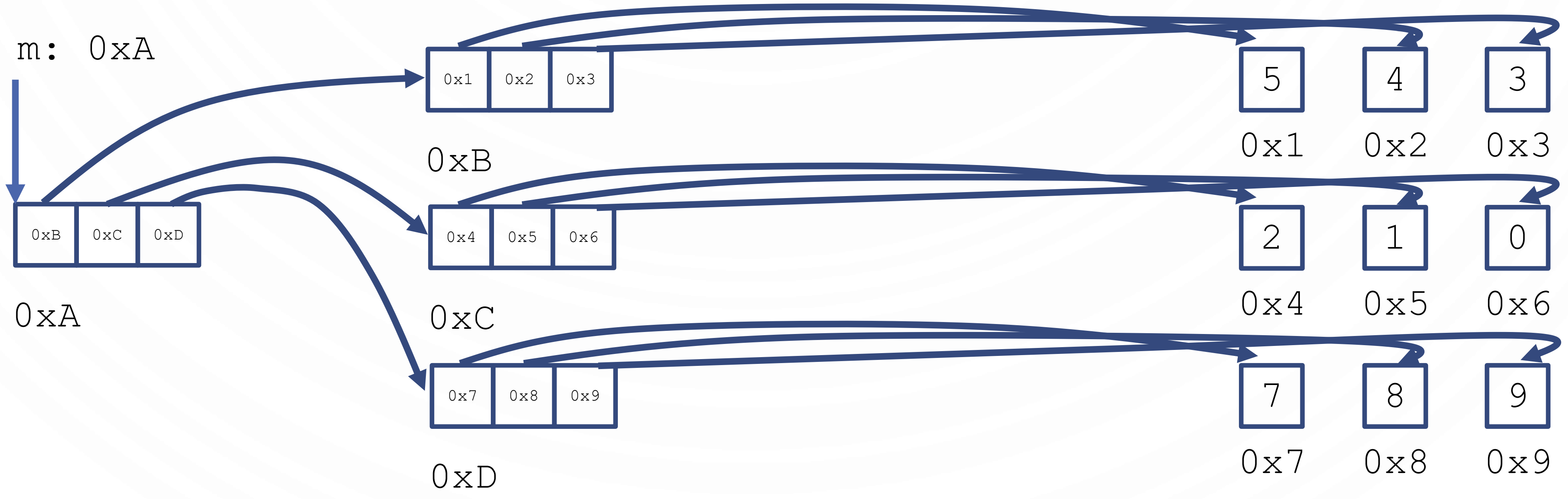


MEMORY LAYOUT

- A multi-dimensional list is a list of list of objects. So:

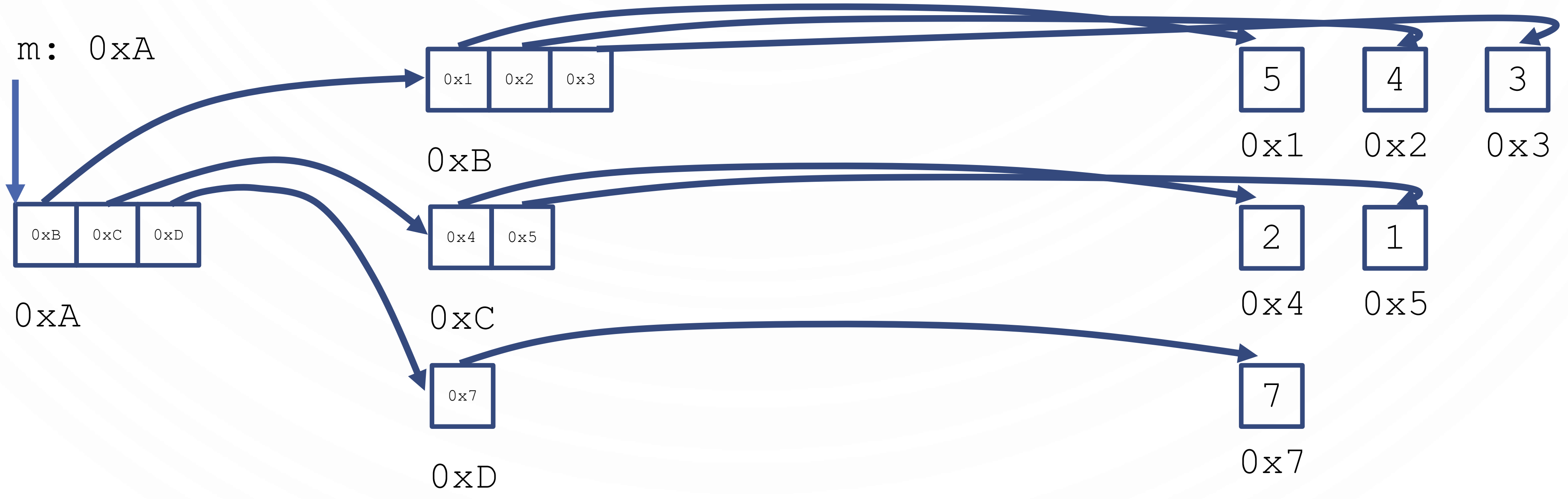
  `m = [[5, 4, 3], [2, 1, 0], [7, 8, 9]]`

  appears like this in memory.



MEMORY LAYOUT

- A multi-dimensional list can also be **ragged** meaning it contains lists of different lengths. So: `m = [[5, 4, 3], [2, 1], [7]]` appears like this in memory.



MULTIDIMENSIONAL LISTS

- Multidimensional lists can be 3, 4, 5, and higher dimensions.

```
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 ]
```

SUMMARY

- Multidimensional Lists.
  - Organized way to store huge quantities of data.
  - Remember, they are lists-of-lists.
  - Can directly access elements at their row/column.