Lecture 2: Multi-Dimentional Arrays

[CSE220 | Data Structures | STV/SFT]

A. Answer the Following Questions:

1. Which languages support multi-dimentional arrays and which do not?

> **Answer:** Java, C, C++, JavaScript etc. do not support mult-dimentional arrays to be created dynamically (which means during program runtime). We can however, implement a multi-dimentional array in these languages but this is actually an array of arrays where each of the index of the original array holds the reference to another array.
> On the other hand, Python supports creation of pure multidimentional arrays.

2. What is the difference between array of arrays and multi-dimentional arrays?

> **Answer:** In array of arrays, each of the index of the original array holds the reference to another array.
> In multidimentional arrays, each of the index of the array holds a value and no other refernece to any other lower dimension arrays.

3. How are multi-dimentional arrays stored in memory?

> **Answer:** Multi-dimentional arrays are stored in two ways depending on which programming language is being used:

> - Increasing the final dimension's index first and progressively move to earlier dimensions, this is called the Row-Major Ordering. For example, Java and C take this approach.

> - Increasing the starting dimension's index first and progressively move to later dimensions, this is called the Column-Major Ordering. For example, FORTRAN takes this approach.

4. Write the equation to find linear index of a cell from a 4-dimentional array:

> **Answer:** If the dimensions of the array are M * N * O * P, and we are accessing the cell [w][x][y][z] from the array:
> Linear index= w * (N * O * P) + x * (O * P) + y * P + z

5. Write the equation and find linear index of a cell from a multi-dimentional array if:

> - Dimensions 3 * 4 * 2 and cell index [2][3][1]. **Answer:**
> - Dimensions 4 * 4 * 2 and cell index [3][4][1]. **Answer:**
> - Dimensions 6 * 4 * 2 * 5 and cell index [4][3][0][3]. **Answer:**

6. Suppose a linear array of length 128 actually stores a 3D array of dimensions 4 * 4 * 8. What are the multidimensional indexes of the element stored at location 111? Remember that indexing starts from 0 in each dimension.

> **Answer:** [3][1][7]

7. Suppose a linear array of length 72 actually stores a 4D array of dimensions 3 * 3 * 2 * 4. What are the multidimensional indexes of the element stored at location 69? Remember that indexing starts from 0 in each dimension.

> **Answer:**

8. What is the difference between multi-dimensional array and matrix?

> **Answer:** Multi-dimensional Arrays can have any number of dimensions, but a matrix will have only have 2 dimensions.

9. How to determine whether two matrices can be added/subtracted or multiplied?

> **Answer:**
> - For Addition/Subtraction, both matrices must be identical in terms of dimensions. For example, only two 2 * 2 or two 3 * 4 matrices can be added or subtracted. But one 2 * 3 and one 3 * 3 matrices can not be added or subtracted.
> - For Multiplication, the number of column of the first matrix must be equal to the number of rows of the second matrix. For example a 2 * 3 matrix can be multiplied with a 3 * 5 matrix.

10. What are the ways of iterating 2D arrays or matrix?

> **Answer:** Two ways of iteration: Row-wise and Column-wise
> - Row-Wise: The outer loop will be associated with the number of rows and the inner loop with the number of columns
> - Column-Wise: The outer loop will be associated with the number of columns and the inner loop with the number of rows

B. Initialization of a Multidimensional Array

```
import numpy as np

#first way
arr1 = np.array([[1, 2, 3],[5, 6, 7]])
print(arr1)
print(type(arr1))

#second way
arr2 = np.zeros((4,2), dtype=int)
print(arr2)
print(type(arr2))
```

```
[[1 2 3]
 [5 6 7]]
<class 'numpy.ndarray'>
[[0 0]
 [0 0]
 [0 0]
 [0 0]]
<class 'numpy.ndarray'>
```

C. Introducing New Property of Multi-Dimensional Array: Shape

Using shape we can get the dimensions of any Python object including multi-dimensional arrays. It returns a tuple.

For 2D matrices, the first element of the tuple is the number of rows and the second is the number of columns.

```
arr = np.array([[1, 2, 3], [5, 6, 7]]) #2D
print(len(arr))
print(arr.shape)

arr = np.array([[[1, 2, 3], [5, 6, 7], [3, 4, 1]]]) #3D
print(len(arr))
print(arr.shape)

arr = np.array([[[1, 2, 3], [5, 6, 7]], [[3, 4, 1], [0, -1, -5]]]) #3D
print(len(arr))
print(arr.shape)

arr = np.array([1, 2, 3]) #1D
print(len(arr))
print(arr.shape)
```

```
2
(2, 3)
1
(1, 3, 3)
2
(2, 2, 3)
3
(3,)
```

D. Iteration of a 2D Matrix

```
#Row Wise Iteration
def row_wise_iter(arr):
  row, col= arr.shape
  for i in range(row):
    for j in range(col):
      print(arr[i][j], end=" ")
    print()

arr = np.array([[1, 2, 3], [5, 6, 7], [2, 4, 9]])
row_wise_iter(arr)
```

```
1 2 3
5 6 7
2 4 9
```

```
#Column Wise Iteration
def col_wise_iter(arr):
  row, col= arr.shape

  for i in range(col):
    for j in range(row):
      print(arr[j][i], end=" ")
    print()

arr = np.array([[1, 2, 3], [5, 6, 7], [2, 4, 9]])
col_wise_iter(arr)
```

```
1 5 2
2 6 4
3 7 9
```

E. Problem Solving

1. Sum of all elements in a 2D matrix
2. Sum of every row in a 2D matrix
3. Sum of every column in a 2D matrix
4. Swap the two columns of a m x 2 matrix
5. Reversing the columns of a m x n matrix
6. Add the elements of the primary diagonal in a square matrix. The primary diagonal of a square matrix are the elements who's row number and column number are equal
7. Add the elements of the secondary diagonal in a square matrix. The secondary diagonal is a diagonal of a square matrix that goes from the lower left corner to the upper right corner.
8. Add two matrices of same dimension
9. Multiply two matrices
10. Given a matrix, rotate elements column-wise to the left.

> For 4*4 matrix
> Input:
> 1 2 3 4
> 5 6 7 8
> 9 1 5 2
> 3 4 5 6
> Output:
> 2 3 4 1
> 6 7 8 5
> 1 5 2 9
> 4 5 6 3

E1. Sum of all elements in a 2D matrix

```
def matrix_sum(arr):
  row, col= arr.shape
  sum= 0
  for i in range(row):
    for j in range(col):
      sum+= arr[i][j]
  return sum

arr = np.array([[1, 2, 3], [5, 6, 7], [2, 4, 9]])
print(matrix_sum(arr))
```

```
39
```

E2. Sum of every row in a 2D matrix

```python
def row_sum(arr):
  #Do by yourself


arr = np.array([[1, 2, 3], [5, 6, 7], [2, 4, 9]])
row_sum(arr)
```

```
6
18
15
```

E3. Sum of every column in a 2D matrix

```python
def col_sum(arr):
  row, col= arr.shape
  for i in range(col):
    sum= 0
    for j in range(row):
      sum+= arr[j][i]
    print(sum)

arr = np.array([[1, 2, 3], [5, 6, 7], [2, 4, 9]])
col_sum(arr)
```

```
8
12
19
```

E4. Swap the two columns of a m x 2 matrix

```python
def swap_columns(arr):
  row, col= arr.shape
  for i in range(row):
    arr[i][0], arr[i][1] = arr[i][1], arr[i][0]
  return arr

arr = np.array([[1, 2, 3], [5, 6, 7], [2, 4, 9]])
row_wise_iter(arr)
print()
row_wise_iter(swap_columns(arr))
```

```
1 2 3
5 6 7
2 4 9

2 1 3
6 5 7
4 2 9
```

E5. Reversing the columns of a m x n matrix

> 0th column ⇌ (n-1)th column
> 1st column ⇌ (n-2)th column
> 2nd column ⇌ (n-3)th column
> and so on and so forth

```python
def rev_columns(arr):
  row, col= arr.shape
  for i in range(row):
    for j in range(col//2):
      arr[i][j], arr[i][col-1-j] = arr[i][col-1-j], arr[i][j]
  return arr

import numpy as np
arr = np.array([[1, 2, 3], [5, 6, 7], [2, 4, 9]])
row_wise_iter(arr)
print()
row_wise_iter(rev_columns(arr))
```

```
1 2 3
5 6 7
2 4 9

3 2 1
7 6 5
9 4 2
```

E6. Addition of Primary Diagonal

Add the elements of the primary diagonal in a square matrix. The primary diagonal of a square matrix are the elements who's row number and column number are equal

```python
def add_primary_diagonal(arr):
  row, col= arr.shape
  if row!=col:
    return "Not a square matrix"
  sum= 0
  for i in range(row):
    for j in range(col):
      if i==j:
        sum+= arr[i][j]
  return sum

import numpy as np
arr = np.array([[1, 2, 3], [5, 6, 7], [2, 4, 9]])
add_primary_diagonal(arr)
```

```
16
```

E8. Add two matrices of same dimension

```python
def add_matrices(a, b):
  row1, col1= a.shape
  row2, col2= b.shape

  if (row1!=row2 or col1!=col2):
    return "Dimensions do not match"

  arr= np.zeros((row1,col1), dtype=int)
  for i in range(row1):
    for j in range(col1):
      arr[i][j]= a[i][j]+b[i][j]
  return arr

import numpy as np
arr1 = np.array([[1, 2, 3], [5, 6, 7], [2, 4, 9]])
arr2 = np.array([[3, 2, 1], [7, 6, 8], [9, 4, 2]])
row_wise_iter(add_matrices(arr1, arr2))
arr3 = np.array([[1, 2], [5, 6], [9, 4]])
print(add_matrices(arr1, arr3))
```

```
4 4 4
12 12 15
11 8 11
Dimensions do not match
```

Complete E2, E7, E9 and E10 by yourself