Data Structures

Lecture 2
Multidimensional Array

Credit: Prantik Paul, Lecturer, BRAC University

MultiDimensional Array (Initialize)

1D Array

```
array1D = np.zeros(5)
```

2D Array

```
array2D = np.zeros((5,2))
```

3D Array

```
array3D = np.zeros(?)
```

MultiDimensional Array (Initialize)

1D Array

```
array1D = np.zeros(5, dtype = int)
```

2D Array

```
array2D = np.zeros((5,2), dtype = int)
```

3D Array

```
array3D = np.zeros(?)
```

MultiDimensional Array (Initialize)

1D Array

```
array1D = np.array([1,2])
```

2D Array

```
array2D = np.array([[1,2], [3,4]])
```

3D Array

```
array3D = np.array(?)
```

MultiDimensional Array (Initialize)

1D Array

```
array1D = np.array([1,2])
```

2D Array

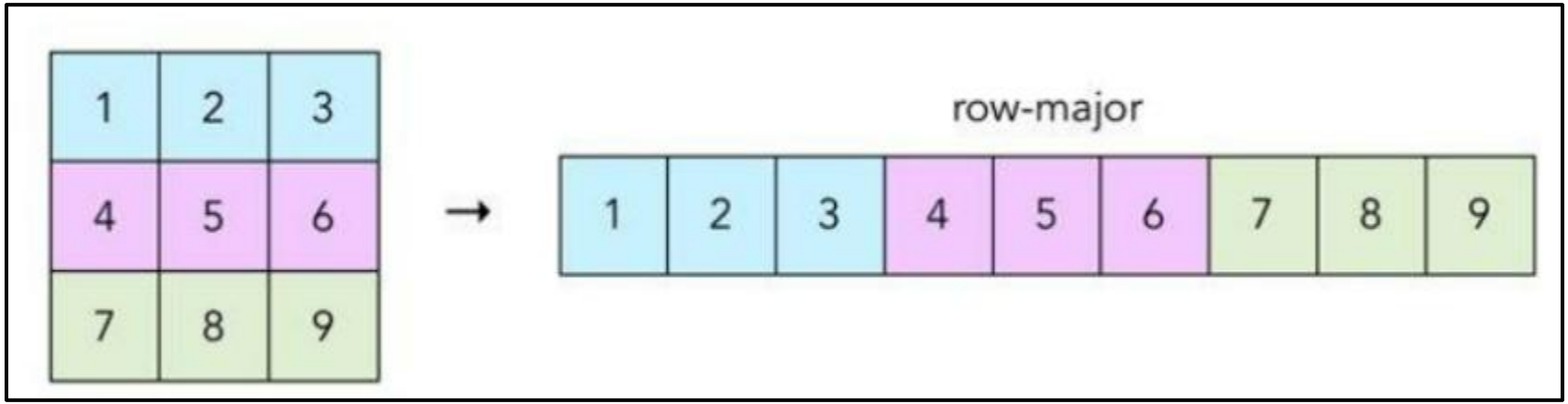
```
array2D = np.array([[1,2], [3,4]])
```

3D Array

```
array3D = np.array([[[1,2],[3,4]],[[5,6],[7,8]]])
```

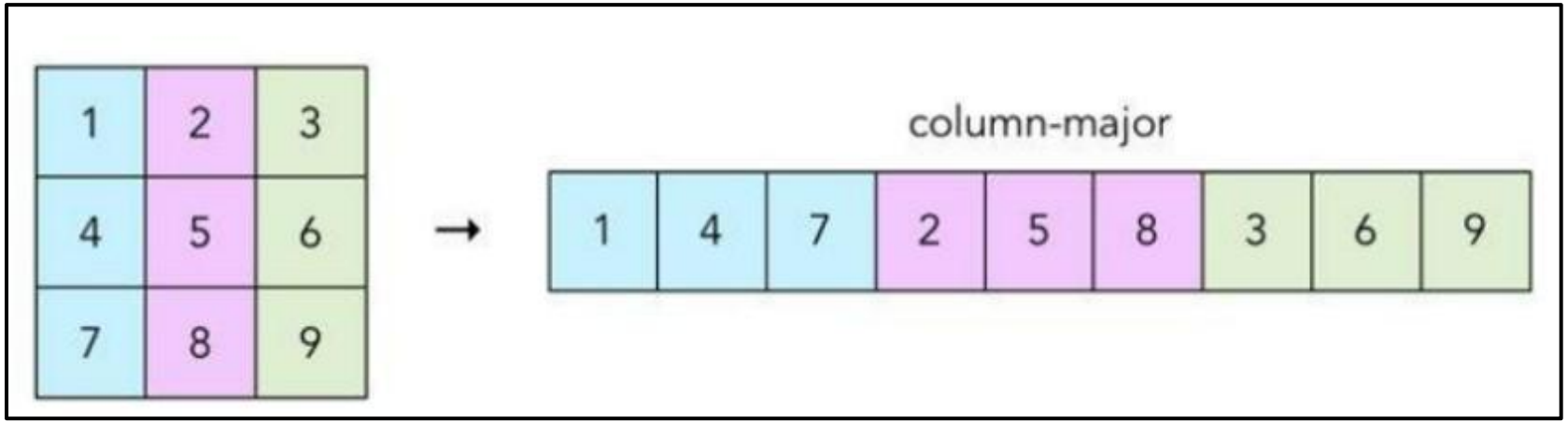
MultiDimensional Array (In Memory)

Row Major Ordering



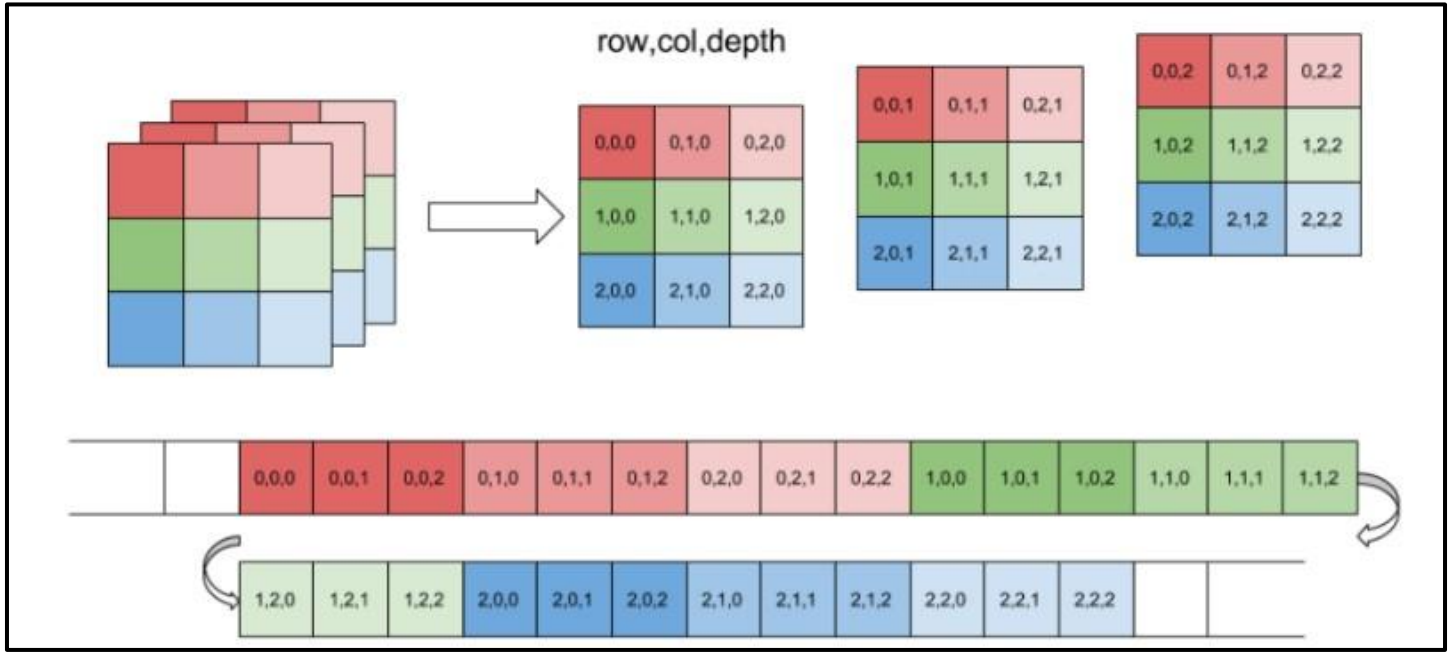
MultiDimensional Array (In Memory)

Column Major Ordering



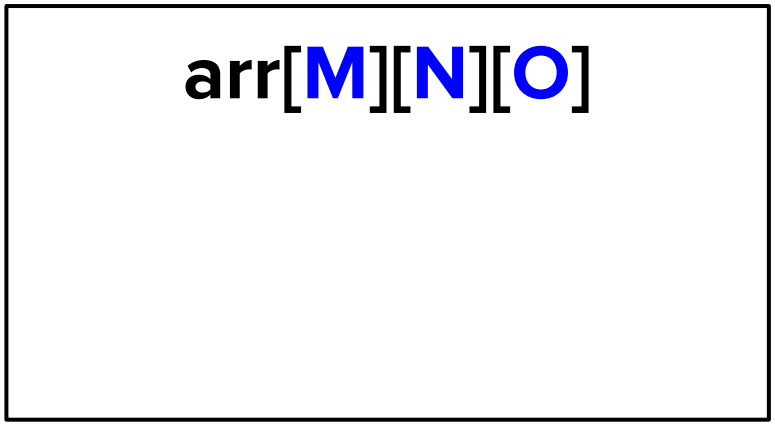
MultiDimensional Array (In Memory)

3D Array



MultiDimensional Array (Index Finding)

3D Array



MultiDimensional Array (Index Finding)

4D Array

arr[M][N][O][P]

MultiDimensional Array (Reverse Indexing)

3D Array

arr[M][N][O]

M = 4

N= 4

O = 8

Location 111

MultiDimensional Array (Reverse Indexing)

3D Array

$$111 = M * (4*8) + N * 8 + O$$

X = 111//(4*8) = 3 and 111 % (4*8) = 15

Y = 15 // 8 = 1 and 15 % 8 = 7

Z = 7

MultiDimensional Array (Reverse Indexing)

3D Array

96, 107, 60

MultiDimensional Array (Iteration - array)

2D Array

MultiDimensional Array (Iteration - array)

```
import numpy as np

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

for x in arr:
  print(x)
```

MultiDimensional Array (Iteration - element)

2D Array

MultiDimensional Array (Iteration - element)

```
import numpy as np

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

for x in arr:
  for y in x:
    print(y)
```

MultiDimensional Array (Iteration - element)

3D Array

MultiDimensional Array (Iteration - element)

```
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]],
                [[7, 8, 9], [10, 11, 12]]])

for x in arr:
  for y in x:
    for z in y:
      print(z)
```

2D Array (Shape)

```
m = np.zeros((2,3), dtype=int)
print(m.shape)
```

```
(2, 3)
```

2D Array (Iteration - Row Wise)

```
FUNCTION print_row(m: ARRAY)
    DECLARE row, col: INTEGER
    row, col = m.DIMENSIONS
    FOR i = 0 TO row-1
        FOR j = 0 TO col-1
            PRINT m[i][j] + " "
        END FOR
        PRINT "\n"
    END FOR
END FUNCTION
```

2D Array (Iteration - Row Wise)

```
def print_row(m):
    row, col = m.shape
    for i in range(row):
        for j in range(col):
            print(m[i][j], end = ' ')
        print()
```

2D Array (Iteration - Column Wise)

```
FUNCTION print_col(m: ARRAY)
    DECLARE row, col: INTEGER
    row, col = m.DIMENSIONS
    FOR i = 0 TO col-1
        FOR j = 0 TO row-1
            PRINT m[j][i] + " "
        END FOR
        PRINT "\n"
    END FOR
END FUNCTION
```

2D Array (Iteration - Column Wise)

```
def print_col(m):
    row, col = m.shape
    for i in range(col):
        for j in range(row):
            print(m[j][i], end = ' ')
        print()
```

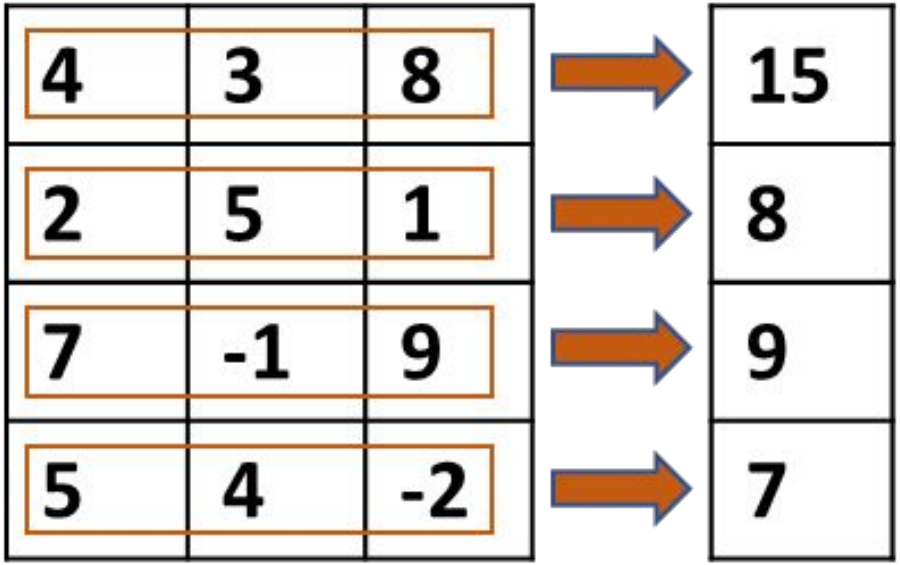
2D Array (Summation)

```
FUNCTION array_sum(m: ARRAY)
    DECLARE sum, row, col: INTEGER
    sum = 0
    row, col = m.DIMENSIONS
    FOR i = 0 TO row-1
        FOR j = 0 TO col-1
            sum = sum + m[i][j]
        END FOR
    END FOR
    RETURN sum
END FUNCTION
```

2D Array (Summation)

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

2D Array (Summation along Rows)



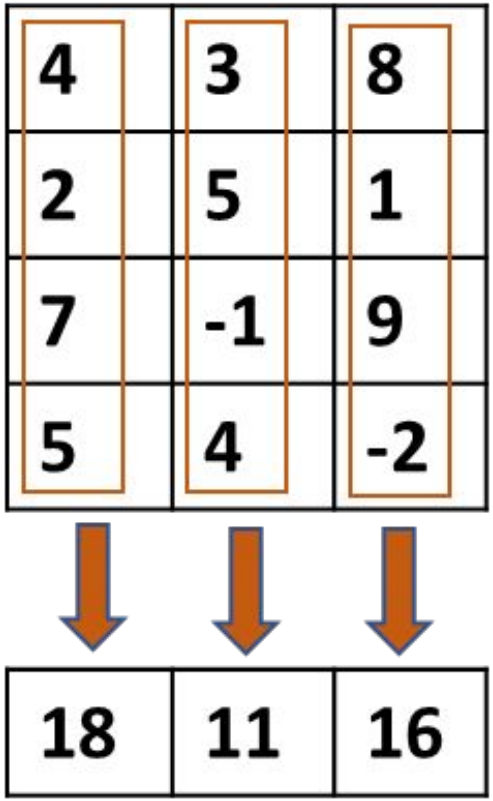
2D Array (Summation along Rows)

```
FUNCTION row_wise_sum(m: ARRAY)
    DECLARE row, col: INTEGER
    DECLARE result: ARRAY [0:row-1, 0:0] OF INTEGER
    row, col = m.DIMENSIONS
    FOR i = 0 TO row-1
        FOR j = 0 TO col-1
            result[i][0] = result[i][0] + m[i][j]
        END FOR
    END FOR
    RETURN result
END FUNCTION
```

2D Array (Summation along Rows)

```
def row_wise_sum(m):
    row, col = m.shape
    result = np.zeros((row,1), dtype = int)
    for i in range(row):
        for j in range(col):
            result[i][0] += m[i][j]
    return result
```

2D Array (Summation along Columns)



2D Array

Find The Largest Number in a 2D Array

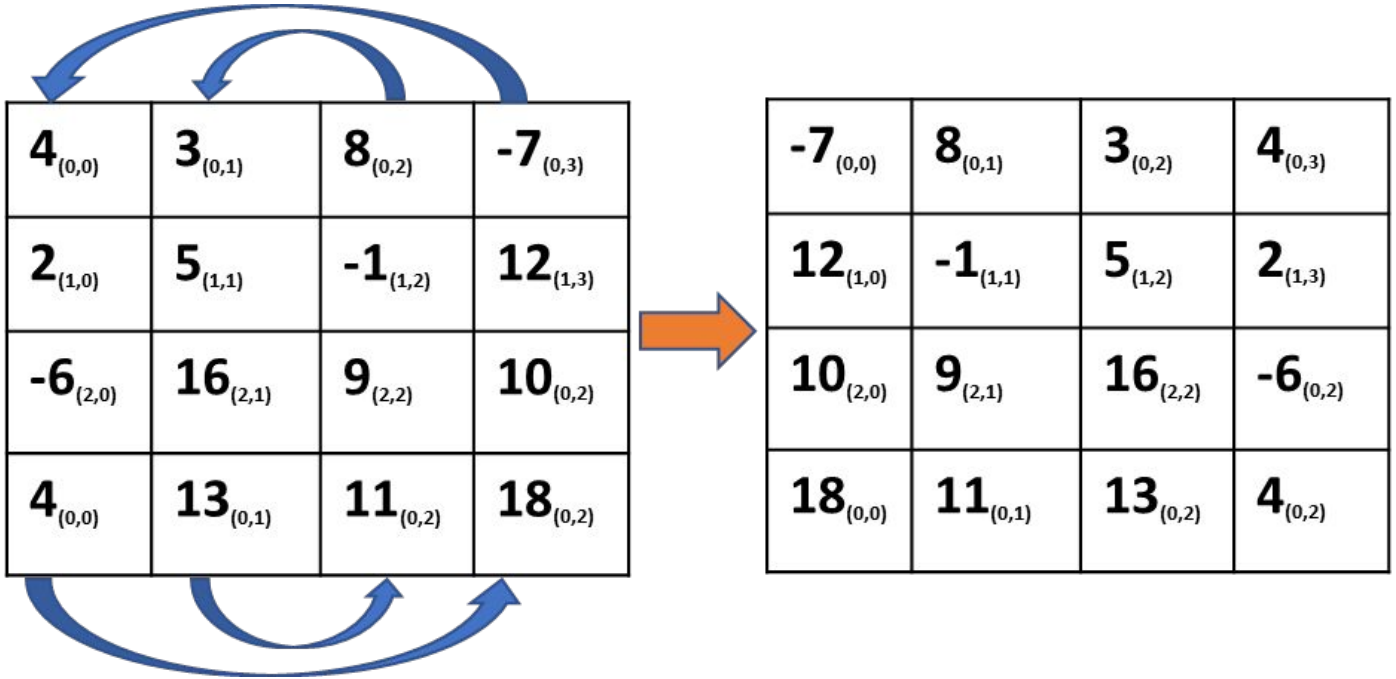
2D Array

Find The Largest Number in Each Row in a 2D Array. Then Find The Smallest Among Them

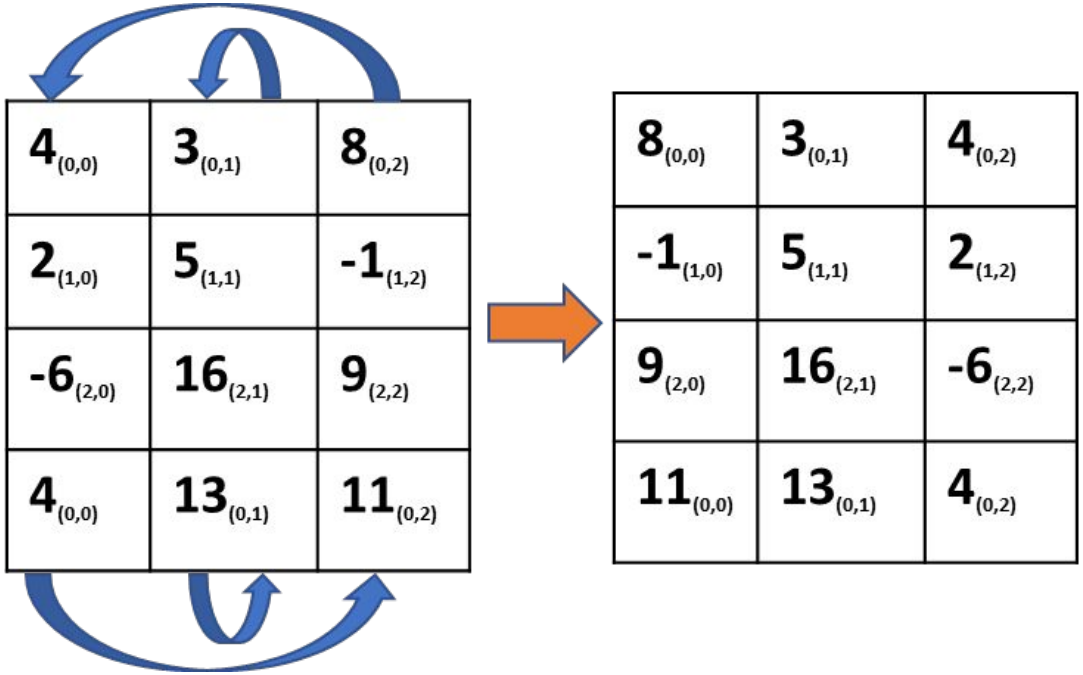
2D Array

Find The Smallest Number in Each Column in a 2D Array. Then Find The Largest Among Them

2D Array (Swap Columns of m*n matrix)



2D Array (Swap Columns of m*n matrix)



2D Array (Swap Columns of m*n matrix)

```
FUNCTION swap_columns(m: ARRAY)
    DECLARE row, col: INTEGER
    row, col = m.DIMENSIONS
    FOR i = 0 TO row-1
        FOR j = 0 TO col/2
            temp = m[i][j]
            m[i][j] = m[i][col-1-j]
            m[i][col-1-j] = temp
        END FOR
    END FOR
    RETURN m
END FUNCTION
```

2D Array (Swap Columns of m*n matrix)

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

2D Array

Swap Rows of m*n matrix

2D Array

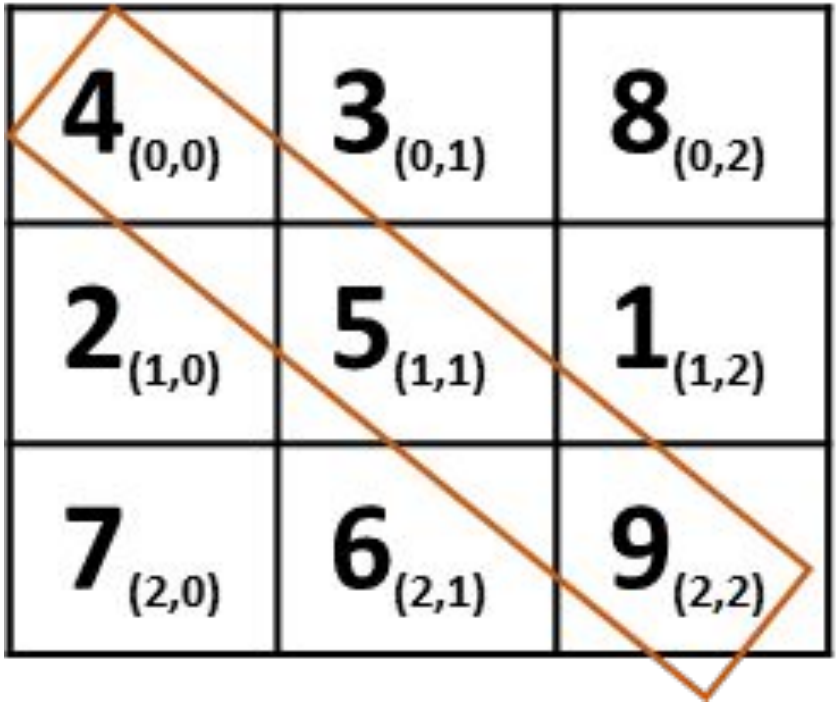
Row Wise Shift

Row Wise Rotation

Column Wise Shift

Column Wise Rotation

2D Array (Add elements of primary diagonal)



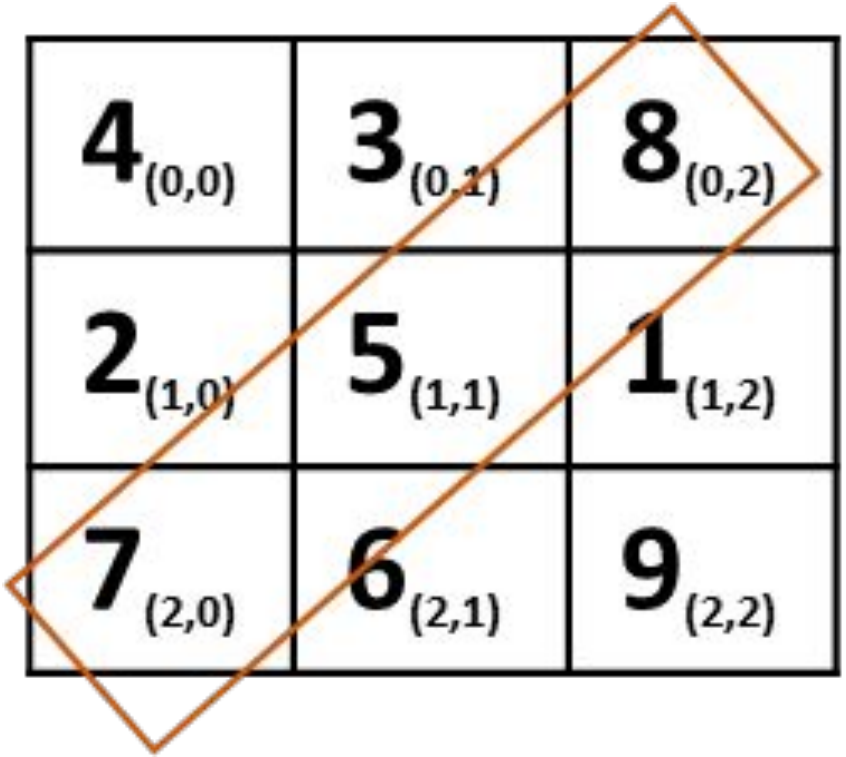
2D Array (Add elements of primary diagonal)

```
FUNCTION sum_primary_diagonal(m: ARRAY)
    DECLARE row, col: INTEGER
    row, col = m.DIMENSIONS
    ASSERT row = col, "Not a square matrix"
    DECLARE sum: INTEGER
    sum = 0
    FOR i = 0 TO row-1
        sum = sum + m[i][i]
    END FOR
    RETURN sum
END FUNCTION
```

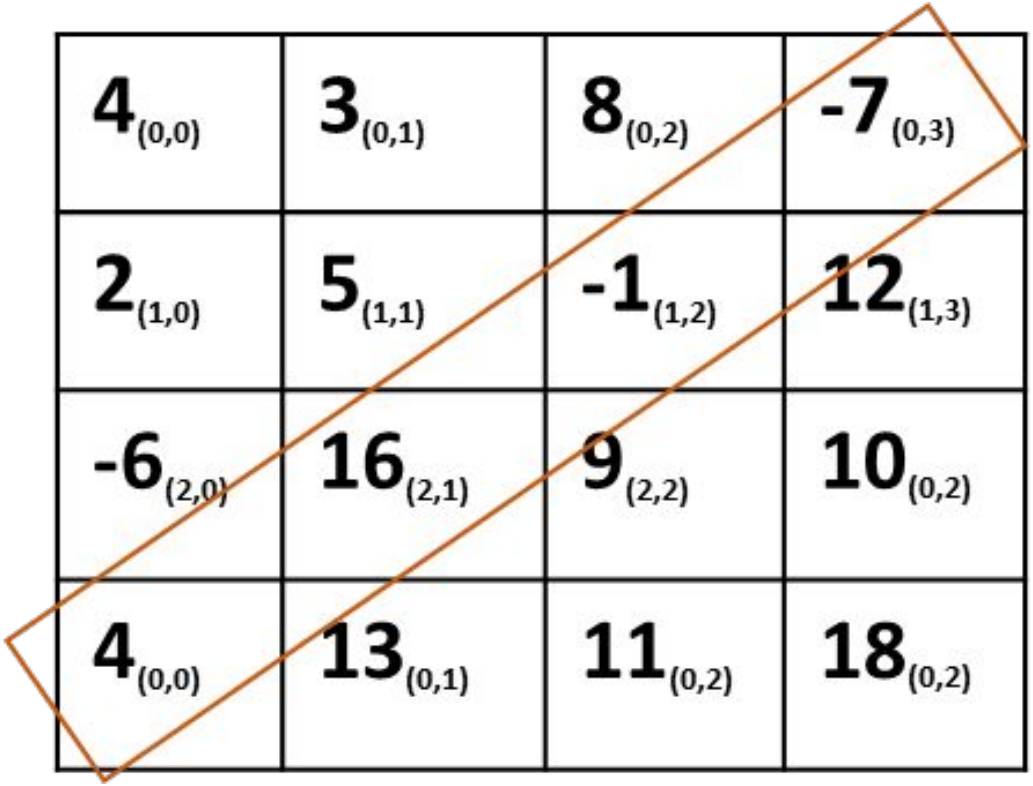
2D Array (Add elements of primary diagonal)

```
def sum_primary_diagonal(m):
  row,col = m.shape
  assert (row==col), 'Not a square matrix'
  sum = 0
  for i in range(row):
    sum += m[i][i]
  return sum
```

2D Array (Add elements of secondary diagonal)



2D Array (Add elements of secondary diagonal)



2D Array (Add 2 Matrices)

```
FUNCTION add_matrix(m: ARRAY, n: ARRAY)
    DECLARE r_m, c_m, r_n, c_n: INTEGER
    r_m, c_m = m.DIMENSIONS
    r_n, c_n = n.DIMENSIONS
    ASSERT r_m = r_n AND c_m = c_n, "Dimension mismatch"
    DECLARE result: ARRAY [0:r_m-1, 0:c_m-1] OF INTEGER
    FOR i = 0 TO r_m-1
        FOR j = 0 TO c_m-1
            result[i][j] = m[i][j] + n[i][j]
        END FOR
    END FOR
    RETURN result
END FUNCTION
```

2D Array (Add 2 Matrices)

```
def add_matrix(m,n):
  r_m, c_m = m.shape
  r_n, c_n = n.shape
  assert (r_m == r_n and c_m == c_n), 'Dimension mismatch'
  result = np.zeros((r_m,c_m), dtype=int)
  for i in range(r_m):
    for j in range(c_m):
      result[i][j] = m[i][j]+n[i][j]
  return result
```

2D Array (Multiply Matrices)

```
matrix1 = [[12,7,3],
          [4 ,5,6],
          [7 ,8,9]]
matrix2 = [[5,8,1],
          [6,7,3],
          [4,5,9]]
```

2D Array (Multiply Matrices)

```
FUNCTION multiply(m: ARRAY, n: ARRAY)
    DECLARE r_m, c_m, r_n, c_n: INTEGER
    r_m, c_m = m.DIMENSIONS
    r_n, c_n = n.DIMENSIONS
    ASSERT c_m = r_n, "Cannot multiply"
    DECLARE result: ARRAY [0:r_m-1, 0:c_n-1] OF INTEGER
    FOR i = 0 TO r_m-1
        FOR j = 0 TO c_n-1
            FOR k = 0 TO c_m-1
                result[i][j] = result[i][j] + m[i][k] * n[k][j]
            END FOR
        END FOR
    END FOR
    RETURN result
END FUNCTION
```

2D Array (Multiply Matrices)

```
def mulitply(m,n):
  r_m, c_m = m.shape
  r_n, c_n = n.shape
  assert c_m == r_n, 'Cannot multiply'
  result = np.zeros((r_m,c_n), dtype=int)
  for i in range(r_m):
    for j in range(c_n):
      for k in range(c_m):
        result[i][j] += m[i][k]*n[k][j]
  return result
```

MultiDimensional Array (Multiply Matrices)

Multiply Matrices (Geeks for Geeks)