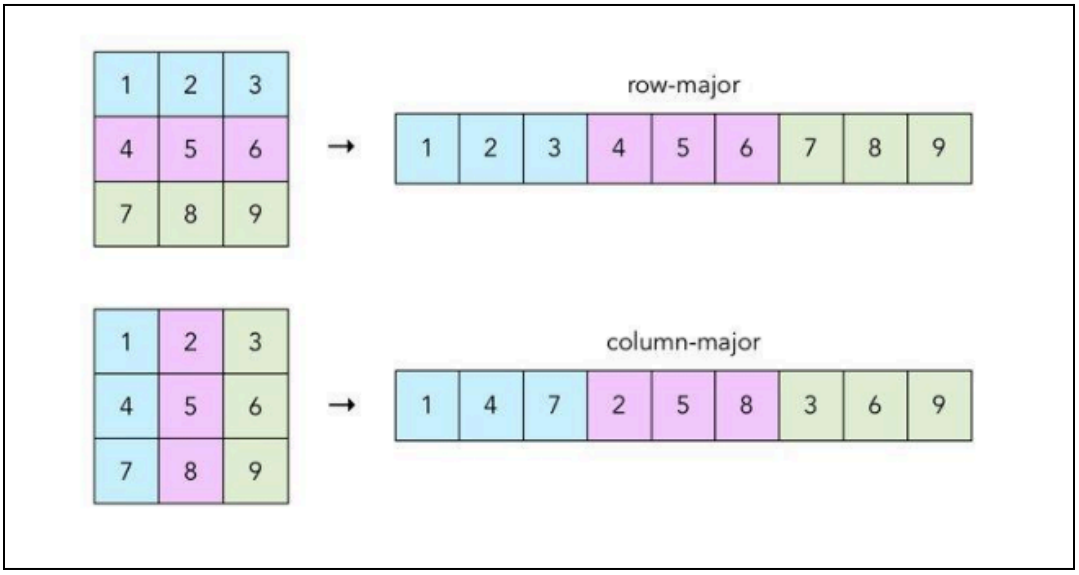
Multidimensional Array

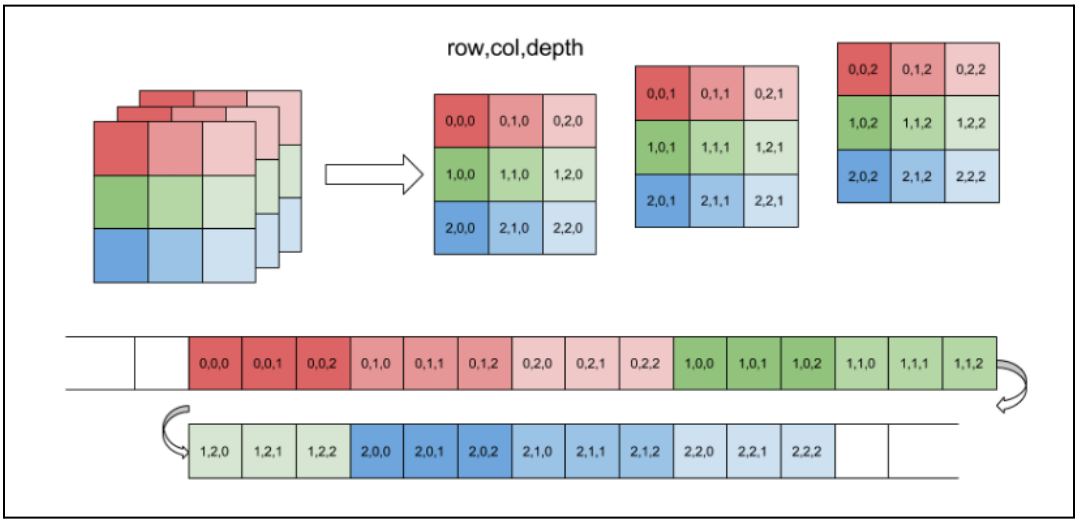
Multidimensional arrays give us a natural way to deal with real-world objects that are multidimensional, and such objects are quite frequent.

Programming languages store multidimensional arrays as linear arrays in the RAM as RAM is a linear array of memory cells.

2D to 1D Array:



3D to 1D Array:



We will stick to row-major ordering, as that is more common in application programming.

Conversion of Linear and Multidimensional Index:

If we have a 4D array with dimensions M×N×O×P, and we want to access an element at arr[w][x][y][z]. Then the index of that element in the linear array is:

$$\text{Linear Index} = w \times (N \times O \times P) + x\,(O \times P) + y \times P + z$$

Let A be a 4D array with dimensions 5x2x4x3
What is the linear index of A[2][1][3][2]?

Ans: Linear Index = $2 \times (2 \times 4 \times 3) + 1 \times (4 \times 3) + 3 \times (3) + 2$
= 71

Follow-Up Question:

Q) How to calculate a Multidimensional Index from a Linear Index? Let us consider an example but in reverse.

Let A be a 4D array with dimensions 5x2x4x3,
What is the multidimensional index of 71?

Ans:

$$71 = w \times (2 \times 4 \times 3) + x \times (4 \times 3) + y \times (3) + z$$

$w = 71 \,//\, (2 \times 4 \times 3) = 2,$
$\quad 71 \,\%\, (2 \times 4 \times 3) = 23$

$x = 23 \,//\, (4 \times 3) = 1$
$\quad 23 \,\%\, (4 \times 3) = 11$

$y = 11 \,//\, (3) = 3$
$\quad 11 \,\%\, (3) = 2$

$z = 2$

Therefore, Multidimensional Index = [2][1][3][2]

Multidimensional Array Declaration/Initialization:

- Array name
- Type of data to be stored in Array
- Size of Array

```
import numpy as np

# Creating a 2D array with zeros of size 4x5
arr = np.zeros((4,5), dtype=int)

# Creating a 2D array with integers
arr = np.array([[1, 2, 3], [4, 5, 6]], dtype=int)
```

Multidimensional Array Operations:

- Read/Write:

```
# Reading the third row and fourth column of a 2D array
arr_2D[2][3]

# Writing the third row and fourth column of a 2D array
arr_2D[2][3] = ‘hello’
```

- Iteration:

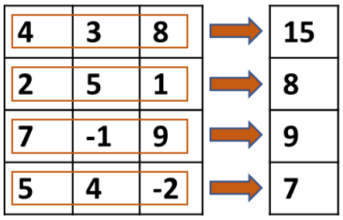
```
#Printing row-wise
def print_row(arr):
        row, col = arr.shape
        for i in range(row):
                for j in range(col):
                        print(arr[i][j])
                print()
```

```
#Printing column-wise
def print_col(arr):
        row, col = arr.shape
        for i in range(col):
                for j in range(row):
                        print(arr[j][i])
                print()
```

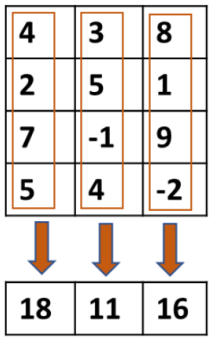
- Summation:

```
#Summing all elements
def array_sum(arr):
        sum = 0
        row, col = arr.shape
        for i in range(row):
                for j in range(col):
                        sum += arr[i][j]
        return sum
```

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

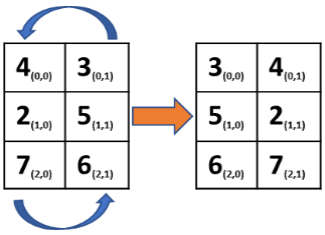


```
#Summing every column
def col_wise_sum(arr):
        row, col = arr.shape
        result = np.zeros((1, col), dtype=int)
        for i in range(col):
                for j in range(row):
                        result[0][i] += arr[j][i]
        return result
```

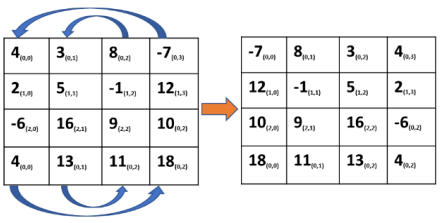


- Swapping:

```
#Swapping the two columns of a m × 2 matrix
def swap_two_col(arr):
        row, col = arr.shape
        for i in range(row):
                arr[i][0], arr[i][1] = arr[i][1], arr[i][0]
        return arr
```



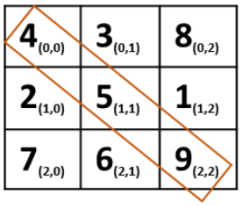

```
#Swapping the columns of a m × n matrix
def swap_col(arr):
        row, col = arr.shape
        for i in range(row):
                for j in range(col//2):
                        arr[i][j], arr[i][col-1-j] = arr[i][col-1-j], arr[i][j]
        return arr
```



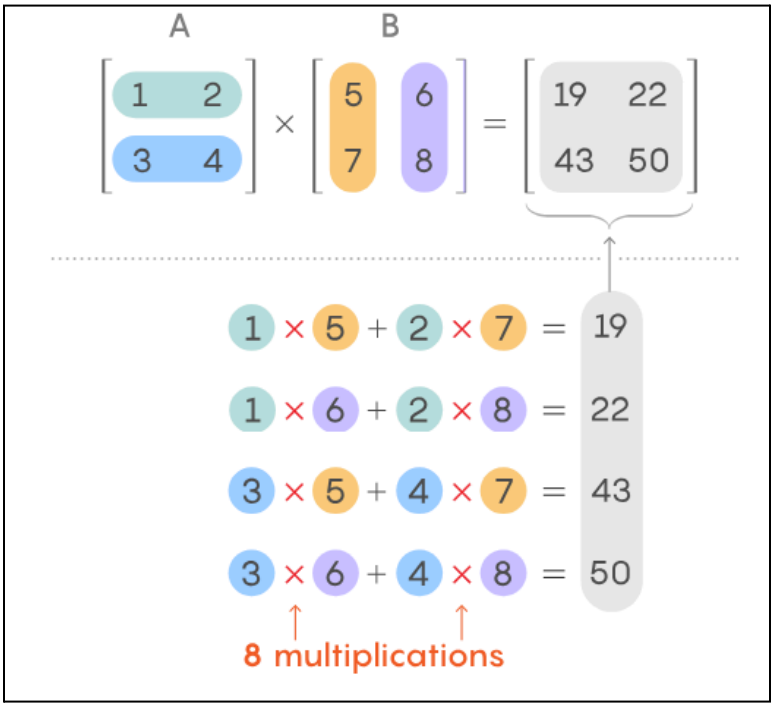
- Addition:

```
#Adding two matrices of the same dimension
def add_matrices(arr1,arr2):
        row1, col1 = arr1.shape
        row2, col2 = arr2.shape
        if row1 != row2 or col1 != col2:
                return "Dimension mismatch"
        result = np.zeros((row1,col1), dtype=int)
        for i in range(row1):
                for j in range(col1):
                        result[i][j] = arr1[i][j] + arr2[i][j]
        return result
```

```
#Adding elements of the primary diagonal in a square matrix
def sum_primary_diagonal(arr):
        sum = 0
        row, col = arr.shape
        if row != col:
                return "Not a square matrix"
        for i in range(row):
                sum += arr[i][i]
        return sum
```



- Multiplication




```
#Multiplying a m × n matrix and a n × p matrix
def multiply_matrices(arr1,arr2):
        row1, col1 = arr1.shape
        row2, col2 = arr2.shape
        if col1 != row2:
                return "Cannot Multiply"
        result = np.zeros((row1,col2), dtype=int)
        for i in range(row1): #For each row in arr1
                for j in range(col2): #For each column in arr2
                        for k in range(col1): #For multiplying and adding
                                result[i][j] += arr1[i][k] * arr2[k][j]
        return result
```