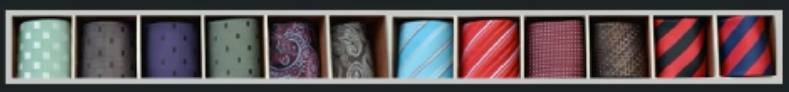
What is an Array ?



Observations from above Picture:

- ✓ It's a box of 'Tie'.
- ✓ All the compartments are contiguous.
- ✓ Each compartment can be identified uniquely.
- ✓ Size of the box is fixed and cannot be modified (as they come from standard manufacturers).

Properties of Array:

- ✓ Array can store data of specified data type'.
- ✓ It has contiguous memory location.
- ✓ Every 'cell' of an Array has an unique 'Index'.
- ✓ 'Index' starts with 0.
- ✓ 'Size of Array' needs to be specified mandatorily and can not be modified.

10	20	30	40	50	60
0	1	2			

Definition of Array:

- ✓ Array is a data structure consisting of a collection of elements, each identified by array index. An array is stored such that the position of each element can be computed from its index cell by a mathematical formula.

✓ **Problem Statement:** *We want to store 1 million similar data types in memory.*

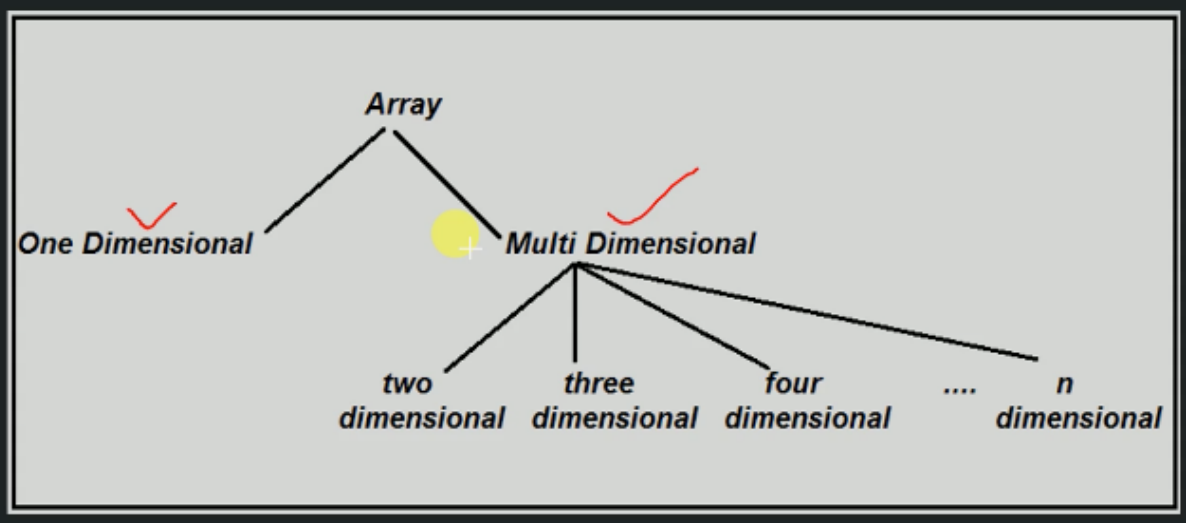
✓ **Solution#1:**

- ✓ *We declare 1 million Primitive Data Structure like Integer, Float, Character, Boolean.* ✗
- ✓ **Problem:** *How will we maintain such a huge list of variables ?*

✓ **Solution#2:**

- ✓ *We declare an array of size 1 million.* ✓
- ✓ **Advantage:** *We just need to reference the cell number of the array and we can access that cell.*

Types of Array ?



Types of Array ?

✓ ***One Dimensional Array:*** *In it each element is represented by a single subscript. The elements are stored in consecutive memory locations. Ex: Arr [7], Arr[col]*

10	20	30	40	50	60	70

arr[2]

✓ ***Multi Dimensional Array:***

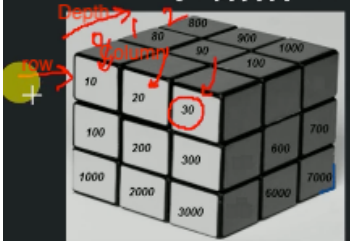
✓ ***Two dimensional array:*** *In it each element is represented by two subscripts. Thus a two dimensional m x n array A has m rows and n columns and contains m*n elements. Ex: Arr [3] [7] has 3 rows and 7 columns and 2*3 = 6 elements. Arr[row][col]*

10	20	30	40	50	60	70
100	200	300	400	500	600	700
1000	2000	3000	4000	5000	6000	7000

Text

arr[0][5]

✓ ***Three dimensional array:*** *In it each element is represented by three subscripts. Thus a three dimensional m x n x l array A contains m*n*l elements. E.g. A [3] [3] [3] has 3*3*3 = 27 elements. Arr[depth][row][col]*

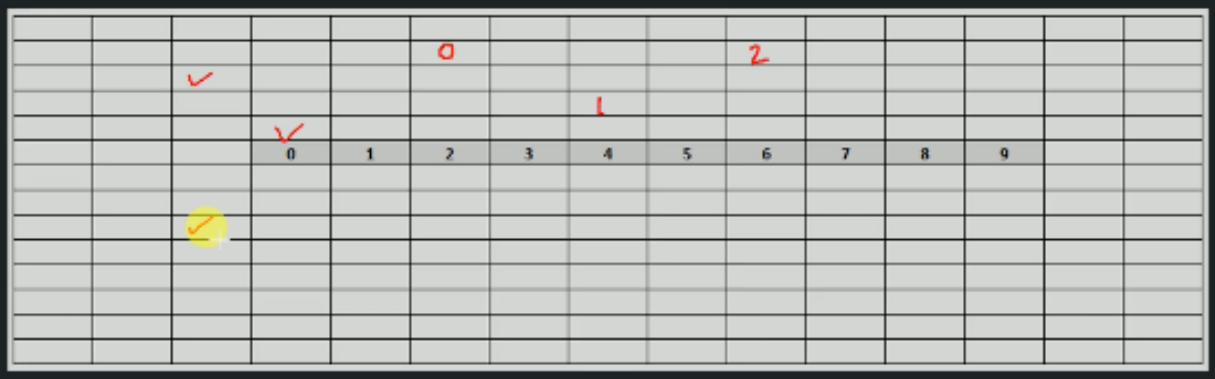


arr[Depth][row][Column]

arr[0][0][2]

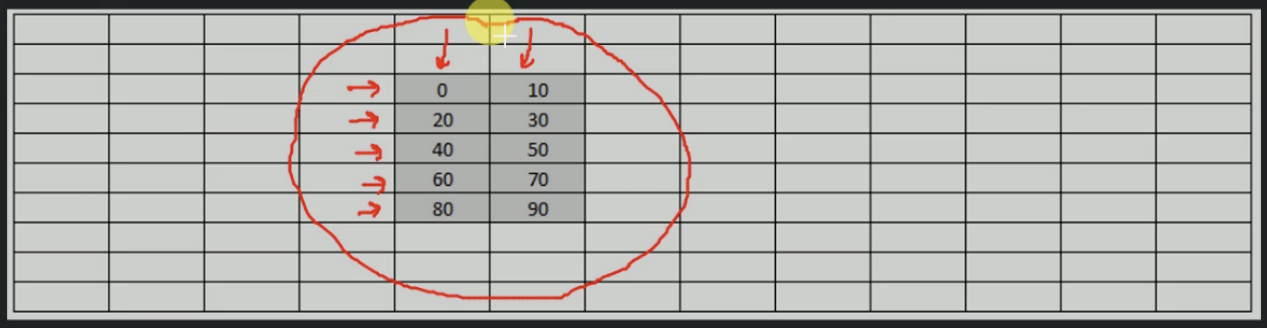
How is an Array represented in Memory ?

- **1D Array:**
  - Arr[col]
  - Arr[10] = {0,1,2,3,4,5,6,7,8,9};
  - Storage in RAM: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

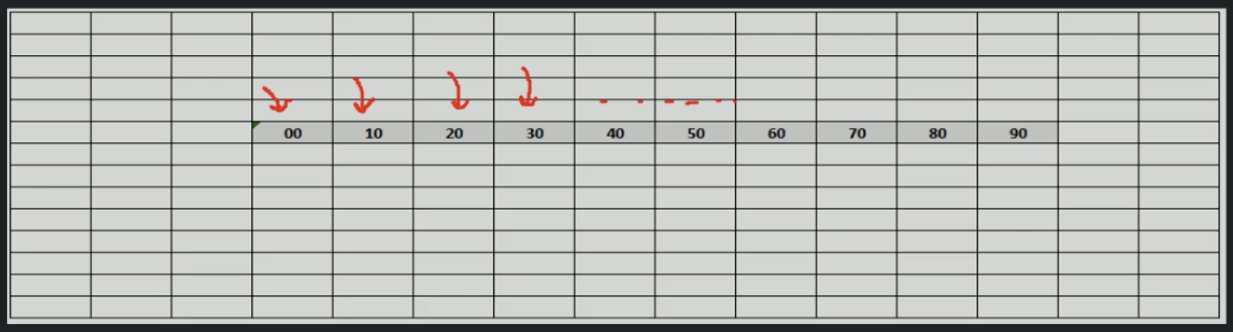


How is an Array represented in Memory ?

- ✓ 2D Array:
  - ✓ Arr[row][col]
  - ✓ Array[5][2]={{00,10},{20,30},{40,50},{60,70},{80,90}};
  - ✓ It will not store like this in RAM:



  - ✓ It will store like this:

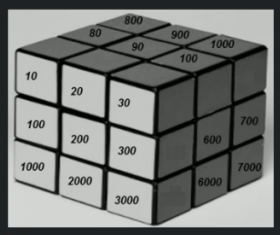


How is an Array represented in Memory ?

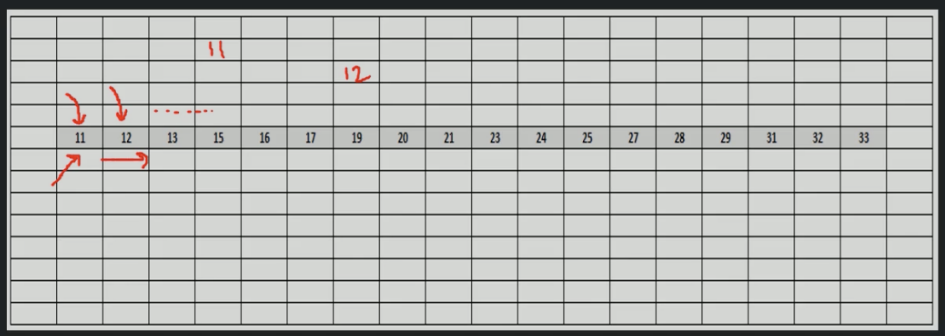
- ✓ 3D Array
  - ✓ Arr[width][row][col]
  - ✓ int arr[2][3][3] = {

```
{ {11, 12, 13}, {15, 16, 17}, {19, 20, 21} },
{ {23, 24, 25}, {27, 28, 29}, {31, 32, 33} }
};
```

  - ✓ It will not store like this in RAM:.



logical

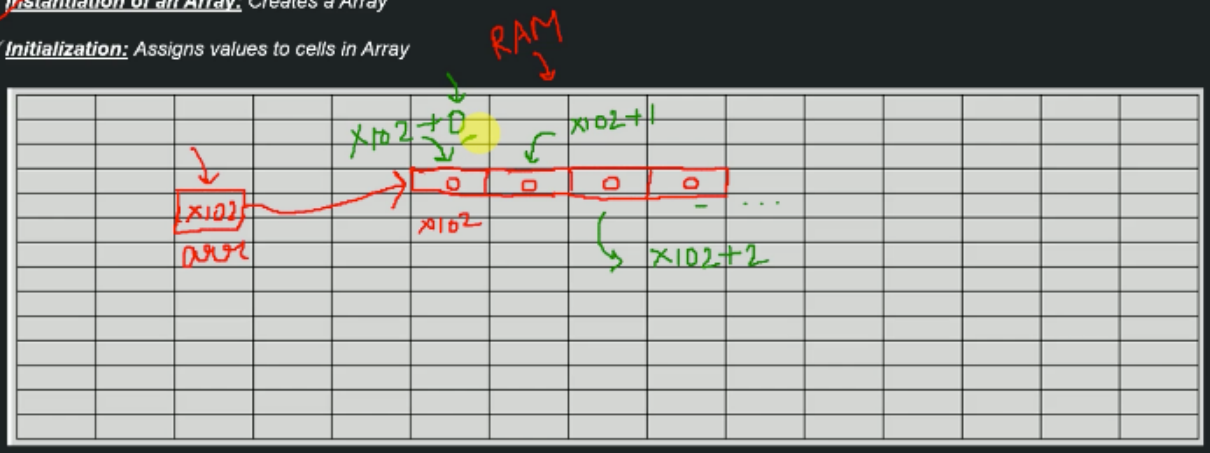


physical

- ✓ It will store like this:

Declaring, Instantiating, Initializing a 1D Array:

- ✓ ***Declare:*** *Creates a reference to Array*
- ✓ ***Instantiation of an Array:*** *Creates a Array*
- ✓ ***Initialization:*** *Assigns values to cells in Array*



Time Complexity - Declaring, Instantiating, Initializing a 1D Array:

✓ **Declare:**
- ✓ dataType[ ] arr ---------- O(1)
- ✓ Example: int[ ] arr

✓ **Instantiation of an Array:**
- ✓ arrayRefVar=new datatype[size]; ---------- O(1)
- ✓ Example: arr = new int[5]

✓ **Initialization:**
- ✓ arr[0]=10; ---------- O(1)
- ✓ arr[1]=20; ---------- O(1)
- ✓ arr[2]=30; ---------- O(1)
- ✓ arr[3]=40; ---------- O(1)
- ✓ arr[4]=50; ---------- O(1)

n x 5 ---------- O(n)

✓ **Declaration, instantiation and initialization:**
- ✓ int arr[ ]={10,20,30,40,50}; ---------- O(1)

Time/Space Complexity of 1D Array:

Particulars	Time Complexity	Space Complexity
Creating an empty Array	O(1)	O(n)
Inserting a value in an array	O(1)	O(1)
Traversing a given Array	O(n)	O(1)
Accessing given cell#	O(1)	O(1)
Searching a given value	O(n)	O(1)
Deleting a given cell's value	O(1)	O(1)

Declaring, Instantiating, Initializing a 2D Array:

✓ **Declare:**
- ✓ dataType[ ][ ] arr ---------- O(1)
- ✓ Example: int[ ][ ] arr

✓ **Instantiation of an Array:**
- ✓ arrayRefVar = new datatype[row][col]; ---------- O(1)
- ✓ Example: arr = new int[2][3]

✓ **Initialization:**
- ✓ a[0][0]=10; ---------- O(1)
- ✓ a[0][1]=20; ---------- O(1)
- ✓ a[0][2]=30; ---------- O(1)
- ✓ a[1][0]=40; ---------- O(1)
- ✓ a[1][1]=50; ---------- O(1)
- ✓ a[1][2]=60; ---------- O(1)

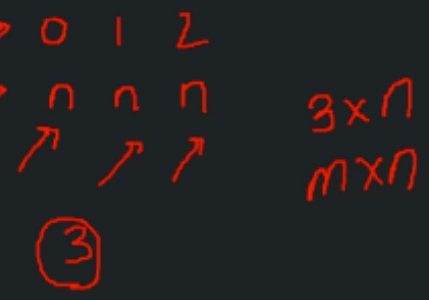
---------- O(mn)

m x n

✓ **Declaration, instantiation and initialization:**
- ✓ int [ ][ ]arr = {{10,20,30}, {40,50,60}}; ---------- O(1)

Time Complexity - Traversing a given 2D Array:

```
TraverseArray(arr):
    loop: row = 0 to rows ---------- O(m)
        loop: col = 0 to col ---------- O(n)
            print arr[row][col] ---------- O(n)
```



Time Complexity = O(mn)

Space Complexity = O(1)

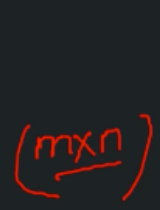
Searching a given value in 2D Array:

10	20	30	40	50	60
70	80	90			

```
SearchInAnArray(arr, valueToSearch):
    loop: row = o to rows
        loop: col = o to col
            if (arr[row][col] equals valueToSearch)
                print (row,col);return;
    print (value not found)
```

Time Complexity - Searching a given value in 2D Array:

```
SearchInAnArray(arr, valueToSearch):
    loop: row = o to rows ---------- O(m)
        loop: col = o to col ---------- O(n)
            if (arr[row][col] equals valueToSearch) ---------- O(1)
                print (row,col);return; ---------- O(1)
    print (value not found) ---------- O(1)
    return ---------- O(1)
```



Time Complexity = O(mn)

Space Complexity = O(1)

Time Complexity - Deleting a given cell's value from 2D Array:

```
DeletingValueFromArray(arr, rowNumber, colNumber):
    arr[rowNumber][colNumber]  = Integer.Mlin_Value      ---------- O(1)
```

Time Complexity = O(1)

Space Complexity = O(1)

Time/Space Complexity of 2D Array:

Particulars	Time Complexity	Space Complexity
Creating an Array	O(1)	O(mn)
Inserting a value	O(1)	O(1)
Traversing given Array	O(mn)	O(1)
Accessing given cell#	O(1)	O(1)
Searching a given value	O(mn)	O(1)
Deleting a given cell's value	O(1)	O(1)

10	20	30	40	50	60
70	80	90			

When to Use/Avoid Array:

- ✓ **When to Use:**
  - ✓ When there is a need to store multiple similar type of data.
  - ✓ When random access is regular affair.

- ✓ **When to Avoid :**
  - ✓ Data to be stored are non-homogenous.
  - ✓ When number of data to be stored is not known in advance.

Practical use of 'Array':

✓ Dynamic Programming

	B	A	N	A	N	A
B	T	F	F	F		
A		T	F	T	F	
N			T	F	T	
A				T	F	T
N					T	F
A						T

Time complexity → $O(n^3)$ $O(n^2)$

Space Complexity → $O(n)$ $O(n^2)$

HASH TABLE

✓ HashTables