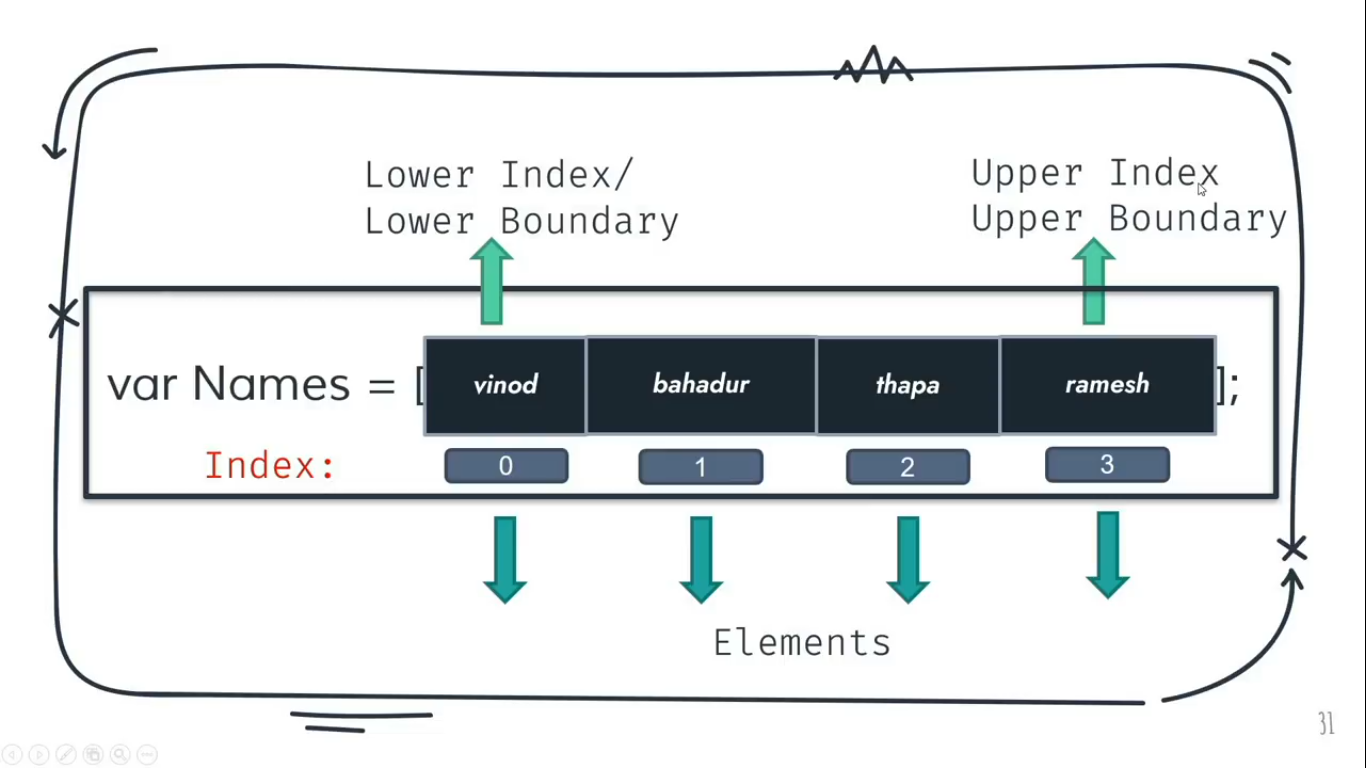
**Array**



When we use only var, we can stored only one value at a time. When we feel like storing multiple values in one variable then instead of using only var, we will use var as an Array.

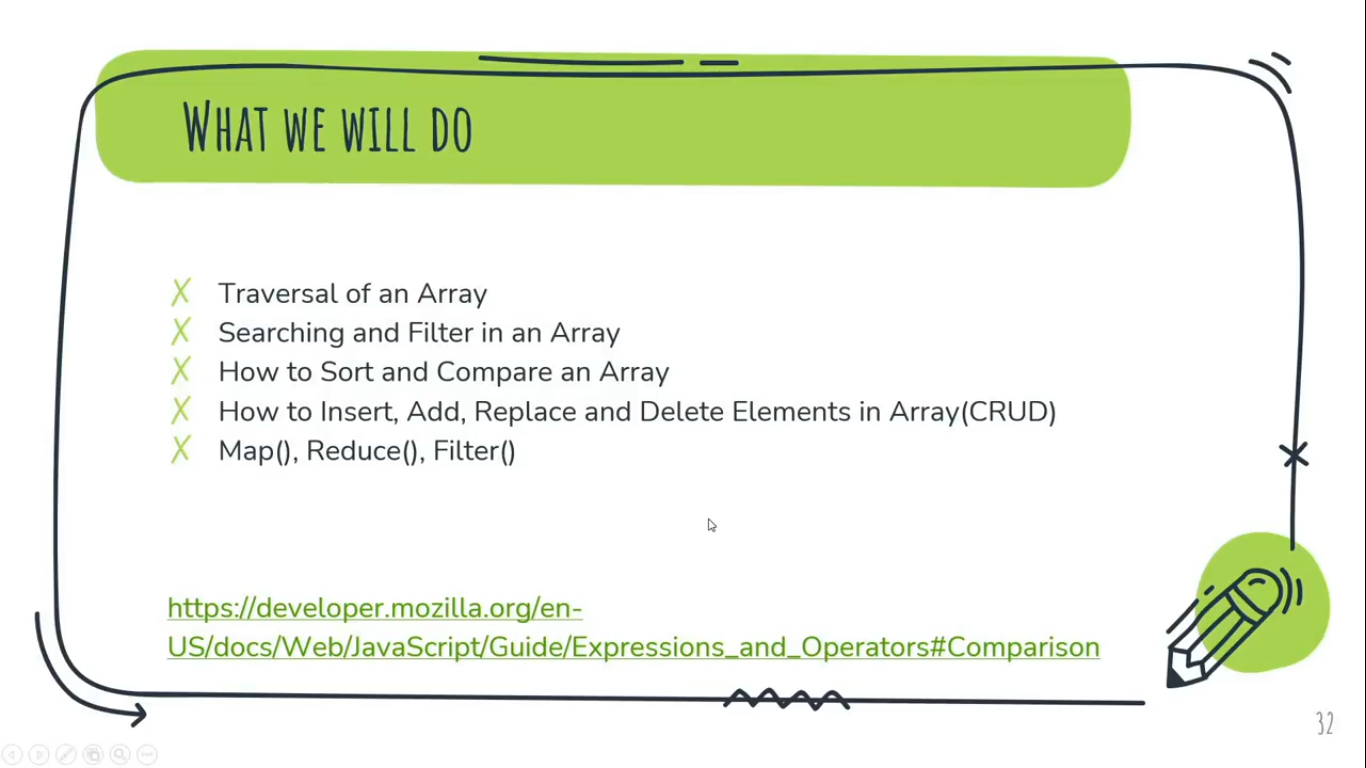
In JavaScript, we have an Array class, and arrays are the prototype of this class.

**Note:**

* JavaScript allows us to store multiple data/elements of different data types in a single variable unlike other programming languages.  
  Ex: var myBio = [‘Chandan’, ‘Kumar’, true, 19];
* We can also create an array using array class but this is optional in JavaScript.  
  Ex:  
  var myBio = new Array;  
  Here, ‘new’ is a keyword which creating a particular instance of that particular class called ‘Array’.

**Characteristics of Array:**

* Every element in an array, represented by index value/number which always start with 0.
* The first element in an array called Upper Index / Upper Boundary and the last element called Lower Index / Lower Boundary.

  
For more info: [https://developer.mozilla.org/en-us/docs/web/javascript/guide/expressions\_and\_operators#comparison](https://developer.mozilla.org/en-us/docs/web/javascript/guide/expressions_and_operators%23comparison)

**Traversal in Array:**

We use traversal in JavaScript for array, If

* We want to get the single data at a time
* We want to change the data
* We want to check the length of data

Ex:   
var name = [‘Chandan’, ‘Kumar’, ‘Prafful’, ‘Amar’];  
console.log(name[0]);  
Output:- Chandan

If we want to check the length of elements or total number of elements of an array, we are going to use ‘length’ property of array.  
**Note**:

* Length of an array always start with 1. So we can say that, if n is the length of an array then index number of the last element/data of that array would be (n-1).  
  **Ex:**  
  var name = [‘Chandan’, ‘Kumar’, ‘Prafful’, ‘Amar’];  
  console.log(name.length);  
  Output:- 4  
    
  var allNames = [‘Chandan’, ‘Kumar’, ‘Prafful’, ‘Amar’];  
  console.log(allNames[allNames.length - 1]);  
  Output:- Amar
* If we create an array and store it in a variable name as ‘name’ using ‘var’ keyword then index number & the length of that array will count along the alphabet of the string i.e. start with the first alphabet of the first string to the last alphabet of that string.  
  **Ex:**  
  var name =['Chandan','Kumar'];

document.write(name[1]);  
Output:- h  
  
var name =['Chandan','Kumar'];

document.write(name.length);  
Output:- 13  
  
If we use the above example same with **‘let’ or ‘const’ keyword then output would be different**.  
**Ex:**  
let name =['Chandan','Kumar'];

document.write(name[1]);  
Output:- Kumar  
  
let name =['Chandan','Kumar'];

document.write(name.length);  
Output:- 2

**Array using for Loop:**

We are going to use ‘for loop’ to get all data at once inside an array.

Example:

var allNames = ['Chandan', 'Kumar', 'Prafful', 'Amar'];

for(var i=0; i<allNames.length; i++)

{

document.write('<br>'+ allNames[i]);

}

In ES6(Modern JavaScript), For In and For Of loop was introduced. So after ES6 we have for..in and for..of loop.

**For in Loop:**

For in loop provide the index number of all the elements of an array.  
**Example:**  
var allNames = ['Chandan', 'Kumar', 'Prafful', 'Amar'];

**for**(let elements **in** allNames) //’for in loop’

{

document.write('<br> + elements);

}

**For of Loop:**

For of loop directly provides all the elements of an array.  
**Example:**  
var allNames = ['Chandan', 'Kumar', 'Prafful', 'Amar'];

**for**(let elements **of** allNames) //’for of loop’

{

document.write('<br> + elements);

}

**forEach loop:**

ForEach loop calls a function for each element in the array.

To use for each loop, we have to pass a call back function along with passing 3 parameters i.e. element, index, array, this  
  
element: It provide the current value of an array.

index: It provide the index number of the element.

array: It provide the data of the current working array.

We can say that, forEach loop is a combination of for in and for of loop.

Example:

var allNames = ['Chandan', 'Kumar', 'Prafful', 'Amar'];

**//Using Normal Traditional function**

allNames.forEach(function(element, index, array){

document.write('<br>' + element);

})

var allNames = ['Chandan', 'Kumar', 'Prafful', 'Amar'];

**//Using Fat Arrow function**

allNames.forEach( (element, index, array)=>{

document.write('<br>' + element);

})

Note:

* We cannot use break keyword in forEach loop.
* We cannot use ‘this’ parameter with fat arrow function because it doesn’t support ‘this’ parameter.

**Searching and Filter in an Array**

**#1 Method/Property -** Array.prototype.indexOf()

Returns the first (least) index of an element within the array equal to an element, or -1 if none is found. It search the element from the 0th index number (First to Last), if we not provide the default starting index number i.e.(2nd parameter of indexOf() method).

Example:

const arr = ['Chandan', 'Kumar', 'Singh', 'Abhishek', 'Kumar'];

document.write(arr.indexOf('Kumar'));

Output:-

1 //first (least) index of element ‘Kumar’

**Note:**

indexOf() method always search forward in direction. If, we put, count number as ‘2’, the element in the 2nd index number will consider as the first index of the element and after that, indexOf() method will move forward but considered -1 to any of the behind elements.

Example-1:  
const arr = ['Chandan', 'Kumar', 'Singh', 'Abhishek', 'Kumar'];

document.write(arr.indexOf('Kumar', 2));

Output:-  
4

Example-2:  
const arr = ['Chandan', 'Kumar', 'Singh', 'Abhishek', 'Kumar'];

document.write(arr.indexOf('Chandan', 2));

Output:-  
-1

**#2 Method/Property –** Array.prototype.lastIndexOf()

Returns the last (greatest) index of an element within the array equal to an element, or -1 if none is found. It search the element Last to First.

Example:

const arr = ['Chandan', 'Kumar', 'Singh', 'Abhishek', 'Kumar'];

document.write(arr.lastIndexOf('Kumar'));

Output:-

4 //last (greatest) index of element ‘Kumar’

**Note:**

lastIndexOf() method always search backward in direction. If, we put, count number as ‘2’, the element in the 2nd index number will consider as the last index of the element and before that, lastIndexOf() method will move backward but considered -1 to any of the after elements from the last index number.

Example-1:  
const arr = ['Chandan', 'Kumar', 'Singh', 'Abhishek', 'Kumar'];

document.write(arr.indexOf('Kumar', 2));

Output:-  
1

Example-2:  
const arr = ['Chandan', 'Kumar', 'Singh', 'Abhishek', 'Kumar'];

document.write(arr.indexOf('Abhishek', 2));

Output:-  
-1

**#3 Method/Property –** Array.prototype.includes()

Includes() method determines whether the array contains a value, returning true or false (Boolean value) as appropriate.

Example:

const arr = ['Chandan', 'Kumar', 'Singh', 'Abhishek', 'Kumar'];

document.write(arr.includes('Abhishek'));

Output:-

true

**Note:** includes() method always search elements forward in direction across an array.  
Example-1:  
const arr = ['Chandan', 'Kumar', 'Singh', 'Abhishek', 'Kumar'];

document.write(arr.includes('Abhishek', 2));

Output:-

true

Example-2:  
const arr = ['Chandan', 'Kumar', 'Singh', 'Abhishek', 'Kumar'];

document.write(arr.includes('Abhishek', 4));

Output:-

false

**#4 Method/Property –** Array.prototype.find()

Find() method returns the found element in the array, if some element in the array satisfies the testing functions, or undefined if not found. Only problem is that it return only one element.

Syntax:   
arr.find(callback(element, index, array)=>)

Example: **// Let I wanna buy clothes having prices less than < 400.**

const prices = [200, 300, 350, 400, 450, 500, 600];

const new\_prices = prices.find((curr\_elment, index, array)=>{

return curr\_elment<400 ;

})

Output:-

200 **// It return only one element instantly after satisfying the returning condition.**

**#5 Method/Property –** Array.prototype.findIndex()

findIndex() method returns the index of the founded element in the array, if an element in the array satisfies the testing function, or -1 if not found.

Example: **// Let I wanna buy clothes having prices less than < 400.**

const prices = [200, 300, 350, 400, 450, 500, 600];

const new\_prices = prices.findIndex((curr\_elment, index, array)=>{

return curr\_elment<400 ;

})

Output:-

0 **//Because the index no. Of 200 is 0.**

**Note:**

The difference between find() and findIndex() method is that, if condition will not satisfied, find() method would return ‘undefined’ whereas findIndex() method would return ‘-1’.  
Example-1:  
const prices = [200, 300, 350, 400, 450, 500, 600];

const new\_prices = prices.find ((curr\_elment, index, array)=>{

return curr\_elment<100 ;

})

Output:-

undefined **// find() method , under condition unsatisfied**

Example-2:  
const prices = [200, 300, 350, 400, 450, 500, 600];

const new\_prices = prices.findIndex((curr\_elment, index, array)=>{

return curr\_elment<100 ;

})

Output:-  
-1 **// find() method , under condition unsatisfied**

**#6 Method/Property –** Array.prototype.filter()

Filter() method returns a new array containing all elements of the calling array for which the provided filtering function returns true. This method doesn’t muted/disturb the current or original array.  
If the condition will not satisfied, it will return an empty array.

Example:

const prices = [200, 300, 350, 400, 450, 500, 600];

const new\_prices = prices.filter((curr\_elment, index, array)=> curr\_elment<400); **// condition**

console.log(new\_prices);

Output:-   
[200, 300, 350] **//prices less than 400**

**How to Sort and Compare an Array**

**#1 Method/Property** array.prototype.sort()

The sort() method sorts the elements of an array in place and returns the sorted array. The default sort order is ascending, built upon converting the elements into strings, then comparing their sequences of UTF-16 code units values.

Example-1:

const months = ['March', 'Jan', 'Feb', 'April', 'Dec', 'Nov'];

console.log(months.sort());

Output:-

['April', 'Dec', 'Feb', 'Jan', 'March', 'Nov']

Here, sort() method sorted the months array in terms of Alphabet in ascending order(default sorting order).

**Note:**

Sort() method only works with strings. So, if the given elements is a numbers then it will first convert them into strings as per the sequences of UTF-16 code before sorting the array.

However, if numbers are sorted as strings, “25” is bigger than ‘’100”, because “2” is bigger than “1”.

*Because of this, the sort() method will produce an incorrect result when sorting numbers.*

Example-2:

const months = [1, 30, 4, 21, 100000, 99];

console.log(months.sort());

Output:-  
[1, 100000, 21, 30, 4, 99]

Example- 3:

const months = [1, 2, 3, 4, 5, 6, 7, 8 , 9];

console.log(months.sort());

Output:-  
[1, 2, 3, 4, 5, 6, 7, 8, 9]

**How to Insert, Add, Replace and Delete Elements in Array(CRUD i.e. Create Read Update Delete)**

**#1 Method/Property –** Array.prototype.push()  
The push() method adds one or more elements to the end of an array and returns the new length of the array.

**Example:**

const allNames = ['Chandan', 'Kumar', 'Prafful', 'Amar'];

const count = allNames.push('WWE');  
//document.write(allNames); Output:   
Chandan,Kumar,Prafful,Amar,WWE

document.write(count);

Output: 5

**#2 Method/Property –** Array.prototype.unshift()  
The push() method adds one or more elements to the beginning of an array and returns the new length of the array.

**Example:**

const allNames = ['Chandan', 'Kumar', 'Prafful', 'Amar'];

const count = allNames.unshift('WWE');  
//document.write(allNames); Output: WWE,Chandan,Kumar,Prafful,Amar

document.write(count);

Output: 5

**#3 Method/Property –** Array.prototype.pop()  
The pop() method removes the last element from an array and returns that element. This method changes the length of the array.

**Example:**

const allNames = ['Chandan', 'Kumar', 'Prafful', 'Amar'];

document.write('<br>' + allNames);

document.write('<br>' + allNames.shift());

document.write('<br>' + allNames);

Output:

Chandan,Kumar,Prafful,Amar  
Amar  
Chandan,Kumar,Prafful

**#4 Method/Property –** Array.prototype.shift()  
The shift() method removes the first element from an array and returns that element. This method changes the length of the array.

**Example:**

const allNames = ['Chandan', 'Kumar', 'Prafful', 'Amar'];

document.write('<br>' + allNames);

document.write('<br>' + allNames.shift());

document.write('<br>' + allNames);

Output:

Chandan,Kumar,Prafful,Amar  
Chandan  
Kumar,Prafful,Amar

**#5 Method/Property –** Array.prototype.splice()

The splice method, Adds and/or Removes elements from an Array. It passes parameters in it (index, delete count Number, ‘value’)

index: Get position using index number from where action start from that index number.

**Note:**

* Usually, splice() method used for deletion of element from the array. If we are using splice method for performing other operation like (update, or add element) inspite of deletion of element, then return value of splice() method would be an empty array[].  
  Ex:  
  const months = ['Jan', 'march', 'April', 'June', 'July'];

const rslt = months.splice(months.length, 0, 'DEC' );  
//Here, we are deleting 0 element.

console.log(rslt); Output: []

\* *ADD* *DEC AT THE END OF THE ARRAY*

const months = ['Jan', 'march', 'April', 'June', 'July'];

months.push('DEC');

document.write(months);

Output:   
Jan,march,April,June,July,DEC

\* *What is the return value of splice method?*splice() method returns the deleted value/element from the array. Because it usually, help in deleting value from the array.

Example:  
const months = ['Jan', 'march', 'April', 'June', 'July'];

const gtIndx = months.indexOf('June');

const updRslt = months.splice(gtIndx, 2 );

document.write(updRslt);

Output:

June,July

\* *update march to October (update)?*

const months = ['Jan', 'march', 'April', 'June', 'July'];

const gtIndx = months.indexOf('march');   
//get index number of any element by searching that element of the array. It will return -1 if entered element doesn’t found in the given array.

document.write(gtIndx); Output: 1

const updRslt = months.splice(gtIndx, 1, 'October' );

// document.write(updRslt); Output: march

If(gtIndx != -1)  
document.write(months);

Output:   
Jan,October,April,June,July

else  
Element Doesn’t found.

Here, we are updating element ‘march’ to ‘October’ by getting indexOf ‘march’ (i.e. 1) then updating ‘march’ with ‘October’ by deleting ‘march’ using counter number (1 i.e. deleting one value) and replacing with ‘October’

\* *Delete June from an Array?*

const months = ['Jan', 'march', 'April', 'June', 'July'];

const gtIndx = months.indexOf('June');

const updRslt = months.splice(gtIndx, 1 );

if(gtIndx !=-1)

document.write(months);

else  
 document.write("Element doesn't found");

Output:  
Jan,march,April,July

Here delete action taken from the index number June and delete only 1st element.   
If count number taken as ‘2’ like *(gtIndx, 2 )*; then delete action will take from the index number of June and delete upto 2nd element i.e.(July).  
If count number taken as ‘Infinity’ like *(gtIndx, Infinity)*; then delete action will take from the index number of June and delete upto infinite i.e. (All Elements) from the index number of June.

**Filter, Map and Reduce Method**

**#1 Method/Property – Array.prototype.filter()**

Filter() method loop over an array and returns the result with a new array by perform the given condition in the function that is passing through it. It’s operation is only execute conditional statement.

const original\_array = [1, 4, 9, 14, 15];

const newArray = original\_array.filter((curr\_elment, index, array)=>{

return curr\_elment>9;

});

console.log(newArray );

Output:  
[14,15]

**\*Remove Specific Element from the array:**

**Ex:   
1st Solution:**function myFunction ( a, b ) {  
  
let arr = a.filter((item)=> {

return item !== b ***//return all those element except b()***

})

return arr;

}  
myFunction([1,2,'2'], '2');

**Output:**[1, 2] ***//remaining array (a)***['2'] ***//return value of splice method***

**2nd Solution:**function myFunction ( a, b ) {  
  
return a.filter((item) => item !== b)

}  
myFunction([1,2,'2'], '2');

**Output:**[1, 2] ***//remaining array (a)***

**#2 Method/Property – Array.prototype.map()**

Map() method, loop over an array and *returns element with a new array* containing the results of calling a function on every element of the older/previous array.

We can pass an anonymous function inside this method.

#1 Example  
const original\_array = [1, 4, 9, 14, 15];

const newArray = original\_array.map((curr\_elment, index, array)=>{

return curr\_elment\*2;

});

console.log(newArray );

Output:  
 [2, 8, 18, 28, 30]

#2 Example

**Note:**

* Map() method return new array without mutating/disturbing the original array.
* We have to use call back ‘()’ with fat arrow ‘=>’ function by passing three parameters (curr\_element, index, array).

**Difference between forEach() & map() method:**

1. **Returning value**: the first difference between map() and forEach() is the **returning value**. The forEach () method returns undefined and map() returns a new array with the transformed elements. Even if they do the same job, the returning value remains different.  
   Example:  
   const original\_array = [1, 4, 9, 14, 15];

const newArray = original\_array.forEach((curr\_elment, index, array)=>{

return `Index No: ${index} and the value is ${curr\_elment} of this ${array} <br> `;

});

document.write(newArray );  
Output:  
undefined

1. **Ability to chain other methods:** Thesecond difference between these array methods is the fact that map() is chainable. This means that you can attach reduce(), sort(), filter() and so on after performing a map() method on an array.  
   That’s something we can’t do with forEach() because, as we know it returns undefined.

**Find the square root of each element in an array?**

Given: let arr = [25, 36, 49, 64, 81];

Sol:- There is a method called sqrt() method in Math property to the sqare root of any number.

let arr = [25, 36, 49, 64, 81];

let arrSqr = arr.map((curr\_elment)=>{

return Math.sqrt(curr\_elment);

});

console.log(arrSqr);

Output:  
[5,6,7,8,9]

**Multiply each Element by 2 and return only those elements which are greater than 10?**

Ans:- As we know map() method is chainable, so we’re free to use filter() method to filter nos. > 10.

Given: let arr = [2, 3, 4, 6, 8];

let arr = [2, 3, 4, 6, 8];

let new\_arr = arr.map((curr\_elment) =>{

return curr\_elment \*2; // [4,6,8,12,16]

}).filter((curr\_elment)=>{

return curr\_elment>10;

});

console.log(new\_arr);

Output:  
[12,16]

**OR**

let arr = [2, 3, 4, 6, 8];

let new\_arr = arr.map((curr\_elment) =>curr\_elment \*2).filter((curr\_elment)=> curr\_elment>10);

document.write(new\_arr);

Output:  
12,16

Here, we have reduced the curly braces, return keyword and the termination semicolon ; of block statement because we have used only single statement.

**#3 Method/Property – Array.prototype.reduce()**

Reduce() method flatten an array, means to convert the 3d or 2d array into a single dimensional array.

The reduce() method executes a reducer function (that we provide) on each element of the array, resulting in single output value.

**Note:** Ifwe need a single value like (Question.…. Now, find the total value of…) or say need to show final result in single output. In such cases we uses reduce method.

The reducer function takes four parameters:

1. Accumulator : It accumulates all the final/last/remaining elements of the array, after performing various operations. This parameter, only used in reduce method.
2. Current Value : same definition as used for the last method like foreach(), map(), filter().
3. Current Index : same definition as used for the last method like foreach(), map(), filter().
4. Source Array : same definition as used for the last method like foreach(), map(), filter().

**#1 Example:**

let arr = [3,5,6];

let new\_arr = arr.reduce((Acumulator, curr\_elment, index, array)=>{

return Acumulator+= curr\_elment;

});

document.write(new\_arr);

Output:-  
14

**#2 Example:  
Add the sum of all elements of array by adding another number from outside.**

let arr = [3,5,6];

let new\_arr = arr.reduce((Acumulator, curr\_elment, index, array)=>{

return Acumulator+= curr\_elment;

},10); //Initialize default value of Acumulator as ‘6’

document.write(new\_arr);

Output:-

24

**#3 Example:  
Multiply the given element in the array with 2 then return the sum of the Element that should be Greater than 10.**

let arr = [2, 3, 4, 6, 8];

let new\_arr = arr.map((curr\_elment) =>curr\_elment \*2).filter((curr\_elment)=> curr\_elment>10).reduce((accumulate, curr\_elment)=> accumulate+= curr\_elment);

document.write(new\_arr);

Output:-  
28

\***How to fatten an array(converting 2d and 3d array into one dimensional array)?**

**Sol:-->** 2d array also known as ‘Array of an Array’. Here, we are going to use reduce() method of array, to convert 2d array to 1d or single dimensional array using ‘**concat()**’ method.

const arr = [ //arrays of an array - 2D array

['zone1', 'zone2'],

['zone3', 'zone4'],

['zone5', 'zone6'],

['zone7', 'zone8']

]

const new\_arr = arr.reduce((accumulate, curr\_elment)=>{

return accumulate.concat(curr\_elment);

})

console.log(new\_arr);

Output:  
['zone1', 'zone2', 'zone3', 'zone4', 'zone5', 'zone6', 'zone7', 'zone8']

**Note:**There is a drawback in reduce() method only for Fatten an Array i.e. reduce() unable to fatten nested Arrays.   
Example:  
const arr = [

['zone1', 'zone2'],

['zone3', 'zone4'],

['zone5', 'zone6'],

['zone7', ['zone8', 'zone9']]

]

const new\_arr = arr.reduce((accumulate, curr\_elment)=>{

return accumulate.concat(curr\_elment);

})

console.log(new\_arr);  
Output:-

['zone1', 'zone2', 'zone3', 'zone4', 'zone5', 'zone6', 'zone7', Array(2)]

So there is introduced of another method in ‘ES20’ that will easily fatten any level of nester Arrays.