Array

1. **Creating an Array:**

Using an array literal is the easiest way to create a JavaScript Array.

Syntax:

const *array\_name* = [*item1*, *item2*, ...];

It is a common practice to declare arrays with the const keyword.

**or**

const cars = [];  
cars[0]= "Saab";  
cars[1]= "Volvo";  
cars[2]= "BMW";

**or**

const cars = new Array("Saab", "Volvo", "BMW");

Note

There is no need to use new Array().

For simplicity, readability and execution speed, use the array literal method.

## Converting an Array to a String:

The JavaScript method toString() converts an array to a string of (comma separated) array values.

EX: const fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits.toString();

## Arrays are Objects:

Arrays are a special type of objects. The typeof operator in JavaScript returns "object" for arrays.

But, JavaScript arrays are best described as arrays.

Arrays use **numbers** to access its "elements". In this example, person[0] returns John:

## EX: const person = ["John", "Doe", 46];

## Array Elements Can Be Objects:

JavaScript variables can be objects. Arrays are special kinds of objects.

Because of this, you can have variables of different types in the same Array.

## myArray[0] = Date.now; myArray[1] = myFunction; myArray[2] = myCars;

## Array Properties and Methods:

## cars.length   // Returns the number of elements cars.sort()   // Sorts the array

## EX: const fruits = ["Banana", "Orange", "Apple", "Mango"]; let length = fruits.length;

## Adding Array Elements:

## Adding element at last

## const fruits = ["Banana", "Orange", "Apple"]; fruits.push("Lemon");

**Note:**

Adding elements with high indexes can create undefined "holes" in an array:

Ex: const fruits = ["Banana", "Orange", "Apple"];  
fruits[6] = "Lemon";

JavaScript does **not** support arrays with named indexes.

In JavaScript, **arrays** always use **numbered indexes**.

**WARNING !!**  
If you use named indexes, JavaScript will redefine the array to an object.

After that, some array methods and properties will produce **incorrect results**.

**JavaScript new Array():**

JavaScript has a built-in array constructor new Array().

But you can safely use [] instead.

These two different statements both create a new empty array named points:

const points = new Array();  
const points = [];

The new keyword can produce some unexpected results:

// Create an array with three elements:  
const points = new Array(40, 100, 1);

// Create an array with two elements:  
const points = new Array(40, 100);

// Create an array with one element ???

//It create an array with 40 undefined elements:  
const points = new Array(40);

**Methods =>**

## at():

method returns an indexed element from an array.

Get the third element of fruits using at():

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
let fruit = fruits.at(2);

## join():

## The join() method also joins all array elements into a string.

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits.join(" \* ");

## pop():

## The pop() method removes the last element from an array.

## const fruits = ["Banana", "Orange", "Apple", "Mango"]; fruits.pop();

## push():

## The push() method adds a new element to an array.

## const fruits = ["Banana", "Orange", "Apple", "Mango"]; let length = fruits.push("Kiwi");

## Shifting Elements :

Shifting is equivalent to popping, but working on the first element instead of the last.

## Shift()

## The shift() method removes the first array element and "shifts" all other elements to a lower index.

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
let fruit = fruits.shift();

## unshift()

The unshift() method adds a new element to an array (at the beginning), and "unshifts" older elements

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.unshift("Lemon");

## Array delete():

Using delete() leaves undefined holes in the array.

Use pop() or shift() instead.

## const fruits = ["Banana", "Orange", "Apple", "Mango"]; delete fruits[0];

## Merging Arrays:

Concatenation "snow" and "ball" gives "snowball".

Concatenating arrays means joining arrays end-to-end.

**concat():**

## const myGirls = ["Cecilie", "Lone"]; const myBoys = ["Emil", "Tobias", "Linus"]; const myChildren = myGirls.concat(myBoys);

## Array copyWithin():

## The copyWithin() method copies array elements to another position in an array:

1. Copy to index 2, all elements from index 0:

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.copyWithin(2, 0);

## output: Banana,Orange,Banana,Orange

## Copy to index 2, the elements from index 0 to 2:

## const fruits = ["Banana", "Orange", "Apple", "Mango", "Kiwi"]; fruits.copyWithin(2, 0, 2);

## output: Banana,Orange,Banana,Orange,Kiwi,Papaya

## Flattening an Array:

Flattening an array is the process of reducing the dimensionality of an array.

Flattening is useful when you want to convert a multi-dimensional array into a one-dimensional array.

## Flat():

## The flat() method creates a new array with sub-array elements concatenated to a specified depth.

const myArr = [[1,2],[3,4],[5,6]];  
const newArr = myArr.flat();

o/p =[1,2,3,4,5,6]

## Splicing and Slicing Arrays:

The splice() method adds new items to an array.

The slice() method slices out a piece of an array.

## splice():

## The splice() method can be used to add new items to an array:

## const fruits = ["Banana", "Orange", "Apple", "Mango"]; fruits.splice(2, 0, "Lemon", "Kiwi");

## o/p => Banana,Orange,Lemon,Kiwi,Apple,Mango

## splice() to Remove Elements:

## With clever parameter setting, you can use splice() to remove elements without leaving "holes" in the array:

## const fruits = ["Banana", "Orange", "Apple", "Mango"]; fruits.splice(0, 1);

## o/p =>["Orange", "Apple", "Mango"];

## Array toSpliced()

The difference between the new **toSpliced()** method and the old **splice()** method is that the new method creates a new array, keeping the original array unchanged, while the old method altered the original array.

const months = ["Jan", "Feb", "Mar", "Apr"];  
const spliced = months.toSpliced(0, 1);

## Array slice()

1. The slice() method slices out a piece of an array into a new array:

const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];  
const citrus = fruits.slice(1);

o/p => Orange,Lemon,Apple,Mango

## Note

The slice() method creates a new array.

The slice() method does not remove any elements from the source array.

1. The slice() method can take two arguments like slice(1, 3).

The method then selects elements from the start argument, and up to (but not including) the end argument.

const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];

const citrus = fruits.slice(1,3);

O/P => Orange,Lemon

**Set =>**