**ARRAYS**

1. The Array object, as with arrays in other programming languages, enables [storing a collection of multiple items under a single variable name](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Arrays)
2. **JavaScript arrays are resizable** and **can contain a mix of different**[data types](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures)
3. **JavaScript arrays are not associative arrays** and so, array elements cannot be accessed using arbitrary strings as indexes, but must be accessed using nonnegative integers
4. **JavaScript arrays are**[zero-indexed](https://en.wikipedia.org/wiki/Zero-based_numbering" \t "https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/_blank): the first element of an array is at index 0, the second is at index 1, and so on — and the last element is at the value of the array's [length](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/length) property minus 1
5. **JavaScript**[array-copy operations](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array" \l "copy_an_array)**create**[shallow copies](https://developer.mozilla.org/en-US/docs/Glossary/Shallow_copy).
6. JavaScript syntax requires properties beginning with a digit to be accessed using [bracket notation](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_objects" \l "objects_and_properties) instead of [dot notation](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Property_accessors). It's also possible to quote the array indices (e.g., years['2'] instead of years[2]), although usually not necessary.

The 2 in years[2] is coerced into a string by the JavaScript engine through an implicit toString conversion. As a result, '2' and '02' would refer to two different slots on the years object, and the following example could be true:

console.log(years["2"] !== years["02"]); //true

1. When setting a property on a JavaScript array when the property is a valid array index and that index is outside the current bounds of the array, the engine will update the array's [length](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/length) property accordingly:

const fruits = [];

fruits.push("banana", "apple", "peach");

console.log(fruits.length); // 3

fruits[5] = "mango";

console.log(fruits[5]); // 'mango'

console.log(Object.keys(fruits)); // ['0', '1', '2', '5']

console.log(fruits.length); // 6

1. Increasing the [length](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/length) extends the array by adding empty slots without creating any new elements — not even undefined.

fruits.length = 10;

console.log(fruits); // ['banana', 'apple', 'peach', empty x 2, 'mango', empty x 4]

console.log(Object.keys(fruits)); // ['0', '1', '2', '5']

console.log(fruits.length); // 10

console.log(fruits[8]); // undefined

1. Decreasing the [length](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/length) property does, however, delete elements

**fruits.length = 2;**

**console.log(Object.keys(fruits)); // ['0', '1']**

**console.log(fruits.length); // 2**

1. **Copying methods and mutating methods**

Some methods do not mutate the existing array that the method was called on, but instead return a new array. They do so by first constructing a new array and then populating it with elements. The copy always happens [shallowly](https://developer.mozilla.org/en-US/docs/Glossary/Shallow_copy) — the method never copies anything beyond the initially created array. Elements of the original array(s) are copied into the new array as follows:

* Objects: the object reference is copied into the new array. Both the original and new array refer to the same object. That is, if a referenced object is modified, the changes are visible to both the new and original arrays.
* Primitive types such as strings, numbers and booleans (not [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String), [Number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number), and [Boolean](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Boolean) objects): their values are copied into the new array.

1. **Create an Array**

**// 'fruits' array created using array literal notation.**

**const fruits = ["Apple", "Banana"];**

**console.log(fruits.length);**

**// 2**

**// 'fruits2' array created using the Array() constructor.**

**const fruits2 = new Array("Apple", "Banana");**

**console.log(fruits2.length);**

**// 2**

**// 'fruits3' array created using String.prototype.split().**

**const fruits3 = "Apple, Banana".split(", ");**

**console.log(fruits3.length);**

**// 2**

1. **Create string from array**

**const fruits = ["Apple", "Banana"];**

**const fruitsString = fruits.join(", ");**

**console.log(fruitsString);**

**// "Apple, Banana"**

1. **Accessing array item using index**

**const fruits = ["Apple", "Banana"];**

**// The index of an array's first element is always 0.**

**fruits[0]; // Apple**

**// The index of an array's second element is always 1.**

**fruits[1]; // Banana**

**// The index of an array's last element is always one**

**// less than the length of the array.**

**fruits[fruits.length - 1]; // Banana**

**// Using an index number larger than the array's length**

**// returns 'undefined'.**

**fruits[99]; // undefined**

1. **Find index of an item**

**const fruits = ["Apple", "Banana"];**

**console.log(fruits.indexOf("Banana"));**

**// 1**

1. **Check if an array contains certain item**

**const fruits = ["Apple", "Banana"];**

**fruits.includes("Banana"); // true**

**fruits.includes("Cherry"); // false**

**// If indexOf() doesn't return -1, the array contains the given item.**

**fruits.indexOf("Banana") !== -1; // true**

**fruits.indexOf("Cherry") !== -1; // false**

1. **Append item**

**const fruits = ["Apple", "Banana"];**

**const newLength = fruits.push("Orange");**

**console.log(fruits);**

**// ["Apple", "Banana", "Orange"]**

**console.log(newLength);**

**// 3**

1. **Remove last item**

**const fruits = ["Apple", "Banana", "Orange"];**

**const removedItem = fruits.pop();**

**console.log(fruits);**

**// ["Apple", "Banana"]**

**console.log(removedItem);**

**// Orange**

1. **Removing multiple items from end**

**const fruits = ["Apple", "Banana", "Strawberry", "Mango", "Cherry"];**

**const start = -3;**

**const removedItems = fruits.splice(start);**

**console.log(fruits);**

**// ["Apple", "Banana"]**

**console.log(removedItems);**

**// ["Strawberry", "Mango", "Cherry"]**

1. **Truncate an array down to just its first N items**

**const fruits = ["Apple", "Banana", "Strawberry", "Mango", "Cherry"];**

**const start = 2;**

**const removedItems = fruits.splice(start);**

**console.log(fruits);**

**// ["Apple", "Banana"]**

**console.log(removedItems);**

**// ["Strawberry", "Mango", "Cherry"]**

1. **Remove first item**

**const fruits = ["Apple", "Banana"];**

**const removedItem = fruits.shift();**

**console.log(fruits);**

**// ["Banana"]**

**console.log(removedItem);**

**// Apple**

1. **Removing multiple items from begining**

**const fruits = ["Apple", "Strawberry", "Cherry", "Banana", "Mango"];**

**const start = 0;**

**const deleteCount = 3;**

**const removedItems = fruits.splice(start, deleteCount);**

**console.log(fruits);**

**// ["Banana", "Mango"]**

**console.log(removedItems);**

**// ["Apple", "Strawberry", "Cherry"]**

1. **Add new first item**

**const fruits = ["Banana", "Mango"];**

**const newLength = fruits.unshift("Strawberry");**

**console.log(fruits);**

**// ["Strawberry", "Banana", "Mango"]**

**console.log(newLength);**

**// 3**

1. **Remove single item by Index**

**const fruits = ["Strawberry", "Banana", "Mango"];**

**const start = fruits.indexOf("Banana");**

**const deleteCount = 1;**

**const removedItems = fruits.splice(start, deleteCount);**

**console.log(fruits);**

**// ["Strawberry", "Mango"]**

**console.log(removedItems);**

**// ["Banana"]**

1. **Remove multiple items by Index**

**const fruits = ["Apple", "Banana", "Strawberry", "Mango"];**

**const start = 1;**

**const deleteCount = 2;**

**const removedItems = fruits.splice(start, deleteCount);**

**console.log(fruits);**

**// ["Apple", "Mango"]**

**console.log(removedItems);**

**// ["Banana", "Strawberry"]**

1. **Replace multiple items**

**const fruits = ["Apple", "Banana", "Strawberry"];**

**const start = -2;**

**const deleteCount = 2;**

**const removedItems = fruits.splice(start, deleteCount, "Mango", "Cherry");**

**console.log(fruits);**

**// ["Apple", "Mango", "Cherry"]**

**console.log(removedItems);**

**// ["Banana", "Strawberry"]**

1. **Iterate over an array**

**const fruits = ["Apple", "Mango", "Cherry"];**

**for (const fruit of fruits) {**

**console.log(fruit);**

**}**

**// Apple**

**// Mango**

**// Cherry**

1. **Call a function on each element**

**const fruits = ["Apple", "Mango", "Cherry"];**

**fruits.forEach((item, index, array) => {**

**console.log(item, index);**

**});**

**// Apple 0**

**// Mango 1**

**// Cherry 2**

1. **Merge multiple arrays**

**const fruits = ["Apple", "Banana", "Strawberry"];**

**const moreFruits = ["Mango", "Cherry"];**

**const combinedFruits = fruits.concat(moreFruits);**

**console.log(combinedFruits);**

**// ["Apple", "Banana", "Strawberry", "Mango", "Cherry"]**

**// The 'fruits' array remains unchanged.**

**console.log(fruits);**

**// ["Apple", "Banana", "Strawberry"]**

**// The 'moreFruits' array also remains unchanged.**

**console.log(moreFruits);**

**// ["Mango", "Cherry"]**

1. **Copy am array**

**const fruits = ["Strawberry", "Mango"];**

**// Create a copy using spread syntax.**

**const fruitsCopy = [...fruits];**

**// ["Strawberry", "Mango"]**

**// Create a copy using the from() method.**

**const fruitsCopy2 = Array.from(fruits);**

**// ["Strawberry", "Mango"]**

**// Create a copy using the slice() method.**

**const fruitsCopy3 = fruits.slice();**

**// ["Strawberry", "Mango"]**

1. **Methods**
   1. **At()**
      1. The at() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances takes an integer value and returns the item at that index, allowing for positive and negative integers. Negative integers count back from the last item in the array.
      2. The element in the array matching the given index. Always returns [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined) if index < -array.length or index >= array.length without attempting to access the corresponding property.
   2. **Concat()**
      1. The concat() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances is used to merge two or more arrays. This method does not change the existing arrays, but instead returns a new array
      2. **Takes an** Arrays and/or values to concatenate into a new array
      3. **Syntax**

concat(value1)

concat(value1, value2)

concat(value1, value2, /\* …, \*/ valueN)

* 1. **Fill()**
     1. The fill() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances changes all elements within a range of indices in an array to a static value. It returns the modified array.
     2. Syntax

**fill(value)**

**fill(value, start)**

**fill(value, start, end)**

* + 1. **Returns t**he modified array, filled with value
    2. **End index is exclusive**
    3. **Note:** Using Array.prototype.fill() on an empty array (length = 0) would not modify it as the array has nothing to be modified. To use Array.prototype.fill() when declaring an array, make sure the array has non-zero length..And if the value passed in start and end paramter should be within the array index else does;nt modify
  1. **Filter()**

**const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];**

**const result = words.filter((word) => word.length > 6);**

**console.log(result);**

**// Expected output: Array ["exuberant", "destruction", "present"]**

* 1. **Find()**

The find() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances returns the first element in the provided array that satisfies the provided testing function. If no values satisfy the testing function, [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined) is returned.

const array1 = [5, 12, 8, 130, 44];

const found = array1.find((element) => element > 10);

console.log(found);

// Expected output: 12

* 1. **FindIndex()**

The findIndex() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances returns the index of the first element in an array that satisfies the provided testing function. If no elements satisfy the testing function, -1 is returned.

const array1 = [5, 12, 8, 130, 44];

const isLargeNumber = (element) => element > 13;

console.log(array1.findIndex(isLargeNumber));

// Expected output: 3

* 1. **FindLast()**

The findLast() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances iterates the array in reverse order and returns the value of the first element that satisfies the provided testing function. If no elements satisfy the testing function, [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined) is returned.

* 1. **FindLastIndex()**

The findLastIndex() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances iterates the array in reverse order and returns the index of the first element that satisfies the provided testing function. If no elements satisfy the testing function, -1 is returned.

* 1. **forEach()**

The forEach() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances executes a provided function once for each array element.

* 1. **Includes()**
     1. The includes() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances determines whether an array includes a certain value among its entries, returning true or false as appropriate.
     2. **Syntax**

includes(searchElement)

includes(searchElement, fromIndex)

* 1. **indexOf()**
     1. The indexOf() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances returns the first index at which a given element can be found in the array, or -1 if it is not present.
     2. Syntax

indexOf(searchElement)

indexOf(searchElement, fromIndex)

* 1. **lastIndexOf()**
     1. The lastIndexOf() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances returns the last index at which a given element can be found in the array, or -1 if it is not present. The array is searched backwards, starting at fromIndex.
     2. **Syntax**

lastIndexOf(searchElement)

lastIndexOf(searchElement, fromIndex)

* 1. **isArray()**
     1. The Array.isArray() static method determines whether the passed value is an [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array).
  2. **Join()**
     1. The join() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances creates and returns a new string by concatenating all of the elements in this array, separated by commas or a specified separator string. If the array has only one item, then that item will be returned without using the separator.

**const elements = ['Fire', 'Air', 'Water'];**

**console.log(elements.join());**

**// Expected output: "Fire,Air,Water"**

**console.log(elements.join(''));**

**// Expected output: "FireAirWater"**

**console.log(elements.join('-'));**

**// Expected output: "Fire-Air-Water"**

* 1. **Keys()**
     1. The keys() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances returns a new [array iterator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Iterator) object that contains the keys for each index in the array.
  2. **Map()**
     1. The map() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances creates a new array populated with the results of calling a provided function on every element in the calling array

**const array1 = [1, 4, 9, 16];**

**// Pass a function to map**

**const map1 = array1.map((x) => x \* 2);**

**console.log(map1);**

**// Expected output: Array [2, 8, 18, 32]**

* 1. **Of()**
     1. The Array.of() static method creates a new Array instance from a variable number of arguments, regardless of number or type of the arguments.
     2. Syntax

Array.of()

Array.of(element1)

Array.of(element1, element2)

Array.of(element1, element2, /\* …, \*/ elementN)

* 1. **Pop()**
     1. The pop() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances removes the **last** element from an array and returns that element. This method changes the length of the array.
     2. Return-:The removed element from the array; [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined) if the array is empty.
  2. **Push()**
     1. The push() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances adds the specified elements to the end of an array and returns the new length of the array.
     2. Syntax

push()

push(element1)

push(element1, element2)

push(element1, element2, /\* …, \*/ elementN)

* + 1. .Return-:The new [length](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/length) property of the object upon which the method was called.
  1. **Reverse()**
     1. The reverse() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances reverses an array [in place](https://en.wikipedia.org/wiki/In-place_algorithm" \t "https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/_blank) and returns the reference to the same array,
     2. **Careful: reverse is destructive -- it changes the original array.**
  2. **Shift()**
     1. The shift() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances removes the **first** element from an array and returns that removed element. This method changes the length of the array.
  3. **Unshift()**
     1. The unshift() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances adds the specified elements to the beginning of an array and returns the new length of the array.
     2. Syntax

unshift()

unshift(element1, element2, /\* …, \*/ elementN)

* 1. **Slice()**
     1. The slice() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances returns a [shallow copy](https://developer.mozilla.org/en-US/docs/Glossary/Shallow_copy) of a portion of an array into a new array object selected from start to end (end not included) where start and end represent the index of items in that array. The original array will not be modified.
     2. Syntax:

slice()

slice(start)

slice(start, end)

* 1. **Some()**
     1. The some() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances tests whether at least one element in the array passes the test implemented by the provided function. It returns true if, in the array, it finds an element for which the provided function returns true; otherwise it returns false. It doesn't modify the array.

**const array = [1, 2, 3, 4, 5];**

**// Checks whether an element is even**

**const even = (element) => element % 2 === 0;**

**console.log(array.some(even));**

**// Expected output: true**

* 1. **Sort()**
     1. The sort() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances sorts the elements of an array [in place](https://en.wikipedia.org/wiki/In-place_algorithm" \t "https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/_blank) and returns the reference to the same array, now sorted. The default sort order is ascending, built upon converting the elements into strings, then comparing their sequences of UTF-16 code units values.
     2. Syntax

sort()

sort(compareFn)

* 1. **Splice()**
     1. The splice() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances changes the contents of an array by removing or replacing existing elements and/or adding new elements [in place](https://en.wikipedia.org/wiki/In-place_algorithm" \t "https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/_blank).
     2. **Return:**An array containing the deleted elements.If only one element is removed, an array of one element is returned.If no elements are removed, an empty array is returned.
     3. Syntax

splice(start)

splice(start, deleteCount)

splice(start, deleteCount, item1)

splice(start, deleteCount, item1, item2)

splice(start, deleteCount, item1, item2, /\* …, \*/ itemN)

* 1. **toReversed()**
     1. he toReversed() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances is the [copying](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array" \l "copying_methods_and_mutating_methods) counterpart of the [reverse()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/reverse) method. It returns a new array with the elements in reversed order.
     2. Return:A new array containing the elements in reversed order.
  2. **toSorted()**
     1. The toSorted() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances is the [copying](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array" \l "copying_methods_and_mutating_methods) version of the [sort()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort) method. It returns a new array with the elements sorted in ascending order.
  3. **toSpliced()**
     1. The toSpliced() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances is the [copying](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array" \l "copying_methods_and_mutating_methods) version of the [splice()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/splice) method. It returns a new array with some elements removed and/or replaced at a given index.
  4. **toString()**
     1. The toString() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances returns a string(‘,’ seperated) representing the specified array and its elements.
  5. **with()**
     1. The with() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances is the [copying](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array" \l "copying_methods_and_mutating_methods) version of using the [bracket notation](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Property_accessors" \l "bracket_notation) to change the value of a given index. It returns a new array with the element at the given index replaced with the given value.The original array is not modified
  6. Delete
     1. delete keyword is used to delete an element from array, and it does not update length of the array I.e. now we will have empty value at the deleted index
     2. If we pass index value great then length of array then there will be no change
     3. Syntax: delete arr[2]
     4. **delete arr will not delete all the elements of array,it will keep the array arr as it is**
  7. **Values()**
     1. The values() method of [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) instances returns a new [array iterator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Iterator) object that iterates the value of each item in the array.
     2. Return-:A new [iterable iterator object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Iterator).

**const array1 = ['a', 'b', 'c'];**

**const iterator = array1.values();**

**for (const value of iterator) {**

**console.log(value);**

**}**

**// Expected output: "a"**

**// Expected output: "b"**

**// Expected output: "c"**