JavaScript Array Methods

❮ Previous

Next ❯

Basic Array Methods

Array length
Array toString()
Array at()
Array join()
Array pop()
Array push()
Array shift()
Array unshift()
Array delete()
Array concat()
Array copyWithin()
Array flat()
Array splice()
Array toSpliced()
Array slice()

See Also:

Search Methods
Sort Methods
Iteration Methods

JavaScript Array length

The `length` property returns the length (size) of an array:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let size = fruits.length;
```

Try it Yourself »

JavaScript Array toString()

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

Example

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

Result:

Banana,Orange,Apple,Mango

Try it Yourself »

JavaScript Array at()

ES2022 intoduced the array method `at()` :

Examples

Get the third element of fruits using at():

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

Try it Yourself »

Get the third element of fruits using []:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits[2];
```

Try it Yourself »

The `at()` method returns an indexed element from an array.

The `at()` method returns the same as `[]`.

The `at()` method is supported in all modern browsers since March 2022:

Chrome 92	Edge 92	Firefox 90	Safari 15.4	Opera 78
Apr 2021	Jul 2021	Jul 2021	Mar 2022	Aug 2021

Note

Many languages allow `negative bracket indexing` like [-1] to access elements from the end of an object / array / string.

This is not possible in JavaScript, because [] is used for accessing both arrays and objects. obj[-1] refers to the value of key -1, not to the last property of the object.

The `at()` method was introduced in ES2022 to solve this problem.

JavaScript Array join()

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

It behaves just like `toString()`, but in addition you can specify the separator:

Example

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

Result:

Banana * Orange * Apple * Mango

Try it Yourself »

Popping and Pushing

When you work with arrays, it is easy to remove elements and add new elements.

This is what popping and pushing is:

Popping items **out** of an array, or pushing items **into** an array.

JavaScript Array pop()

The `pop()` method removes the last element from an array:

Example

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

Try it Yourself »

The `pop()` method returns the value that was "popped out":

Example

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

Try it Yourself »

JavaScript Array push()

The `push()` method adds a new element to an array (at the end):

Example

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

Try it Yourself »

The `push()` method returns the new array length:

Example

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

Try it Yourself »

Shifting Elements

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

JavaScript Array shift()

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

Example

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

Try it Yourself »

The `shift()` method returns the value that was "shifted out":

Example

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

Try it Yourself »

JavaScript Array unshift()

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

Example

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

Try it Yourself »

The `unshift()` method returns the new array length:

Example

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

Try it Yourself »

Changing Elements

Array elements are accessed using their **index number**:

Array **indexes** start with 0:

[0] is the first array element
[1] is the second
[2] is the third ...

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[0] = "Kiwi";
```

Try it Yourself »

JavaScript Array length

The `length` property provides an easy way to append a new element to an array:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[fruits.length] = "Kiwi";
```

Try it Yourself »

JavaScript Array delete()

Warning !

Using `delete()` leaves `undefined` holes in the array.

Use pop() or shift() instead.

Example

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

Try it Yourself »

Merging Arrays (Concatenating)

In programming languages, concatenation means joining strings end-to-end.

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

Concatenating arrays means joining arrays end-to-end.

JavaScript Array concat()

The `concat()` method creates a new array by merging (concatenating) existing arrays:

Example (Merging Two Arrays)

```
const myGirls = ["Cecilie", "Lone"];
const myBoys = ["Emil", "Tobias", "Linus"];

const myChildren = myGirls.concat(myBoys);
```

Try it Yourself »

Note

The `concat()` method does not change the existing arrays. It always

returns a new array.

The `concat()` method can take any number of array arguments.

Example (Merging Three Arrays)

```
const arr1 = ["Cecilie", "Lone"];
const arr2 = ["Emil", "Tobias", "Linus"];
const arr3 = ["Robin", "Morgan"];
const myChildren = arr1.concat(arr2, arr3);
```

Try it Yourself »

The `concat()` method can also take strings as arguments:

Example (Merging an Array with Values)

```
const arr1 = ["Emil", "Tobias", "Linus"];
const myChildren = arr1.concat("Peter");
```

Try it Yourself »

Array copyWithin()

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

Examples

Copy to index 2, all elements from index 0:

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

Try it Yourself »

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

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

Try it Yourself »

Note

The `copyWithin()` method overwrites the existing values.

The `copyWithin()` method does not add items to the array.

The `copyWithin()` method does not change the length of the array.

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.

JavaScript Array flat()

ES2019 Introduced the Array flat() method.

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

Example

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

Try it Yourself »

Browser Support

JavaScript Array `flat()` is supported in all modern browsers since January 2020:

Chrome 69	Edge 79	Firefox 62	Safari 12	Opera 56
Sep 2018	Jan 2020	Sep 2018	Sep 2018	Sep 2018

JavaScript Array flatMap()

ES2019 added the Array `flatMap()` method to JavaScript.

The `flatMap()` method first maps all elements of an array and then creates a new array by flattening the array.

Example

```
const myArr = [1, 2, 3, 4, 5, 6];
const newArr = myArr.flatMap(x => [x, x * 10]);
```

Try it Yourself »

Browser Support

JavaScript Array `flatMap()` is supported in all modern browsers since January 2020:

Chrome 69	Edge 79	Firefox 62	Safari 12	Opera 56
Sep 2018	Jan 2020	Sep 2018	Sep 2018	Sep 2018

Splicing and Slicing Arrays

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

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

JavaScript Array splice()

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

Example

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

Try it Yourself »

The first parameter (2) defines the position **where** new elements should be **added** (spliced in).

The second parameter (0) defines **how many** elements should be **removed**.

The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be **added**.

The `splice()` method returns an array with the deleted items:

Example

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

Try it Yourself »

Using splice() to Remove Elements

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

Example

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

Try it Yourself »

The first parameter (0) defines the position where new elements should be **added** (spliced in).

The second parameter (1) defines **how many** elements should be **removed**.

The rest of the parameters are omitted. No new elements will be added.

JavaScript Array toSpliced()

ES2023 added the Array toSpliced() method as a safe way to splice an array without altering the original array.

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.

Example

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

Try it Yourself »

JavaScript Array slice()

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

Example

Slice out a part of an array starting from array element 1 ("Orange"):

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

Try it Yourself »

Note

The `slice()` method creates a new array.

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

Example

Slice out a part of an array starting from array element 3 ("Apple"):

```
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(3);
```

Try it Yourself »

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.

Example

```
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(1, 3);
```

Try it Yourself »

If the end argument is omitted, like in the first examples, the `slice()` method slices out the rest of the array.

Example

```
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(2);
```

Try it Yourself »

Automatic toString()

JavaScript automatically converts an array to a comma separated

string when a primitive value is expected.

This is always the case when you try to output an array.

These two examples will produce the same result:

Example

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

Try it Yourself »

Example

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

Try it Yourself »

Note

All JavaScript objects have a toString() method.

Searching Arrays

Searching arrays are covered in the next chapter of this tutorial.

Sorting Arrays

Sorting arrays covers the methods used to sort arraysg.

Iterating Arrays

Iterating arrays covers methods that operate on all array elements.

Complete Array Reference

For a complete Array reference, go to our:

Complete JavaScript Array Reference.

The reference contains descriptions and examples of all Array properties and methods.

Exercise?

After executing the following code:

```
const fruits = ['Banana', 'Orange', 'Apple'];
fruits.pop();
```

What will the `fruits` array look like?

- `['', 'Banana', 'Orange', 'Apple']`
- `['Banana', 'Orange']`
- `['Orange', 'Apple']`

Submit Answer »

‹ Previous Next ›