JavaScript Array Iteration Methods

❮ Previous Next ❯

Array iteration methods operate on every array item.

Array.forEach()

The `forEach()` method calls a function (a callback function) once for each array element.

Example

```
var txt = "";
var numbers = [45, 4, 9, 16, 25];
numbers.forEach(myFunction);

function myFunction(value, index, array) {
  txt = txt + value + "<br>";
}
```

Try it Yourself »

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

The example above uses only the value parameter. The example can be rewritten to:

Example

```
var txt = "";
var numbers = [45, 4, 9, 16, 25];
numbers.forEach(myFunction);
```

```
function myFunction(value) {
  txt = txt + value + "<br>";
}
```

Try it Yourself »

`Array.forEach()` is supported in all browsers except Internet Explorer 8 or earlier:

Yes	9.0	Yes	Yes	Yes

Array.map()

The `map()` method creates a new array by performing a function on each array element.

The `map()` method does not execute the function for array elements without values.

The `map()` method does not change the original array.

This example multiplies each array value by 2:

Example

```
var numbers1 = [45, 4, 9, 16, 25];
var numbers2 = numbers1.map(myFunction);

function myFunction(value, index, array) {
  return value * 2;
}
```

Try it Yourself »

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

When a callback function uses only the value parameter, the index and array parameters can be omitted:

Example

```
var numbers1 = [45, 4, 9, 16, 25];
var numbers2 = numbers1.map(myFunction);

function myFunction(value) {
  return value * 2;
}
```

Try it Yourself »

`Array.map()` is supported in all browsers except Internet Explorer 8 or earlier.

				
Yes	9.0	Yes	Yes	Yes

Array.filter()

The `filter()` method creates a new array with array elements that passes a test.

This example creates a new array from elements with a value larger than 18:

Example

```
var numbers = [45, 4, 9, 16, 25];
var over18 = numbers.filter(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

Try it Yourself »

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

In the example above, the callback function does not use the index and array parameters, so they can be omitted:

Example

```
var numbers = [45, 4, 9, 16, 25];
var over18 = numbers.filter(myFunction);

function myFunction(value) {
  return value > 18;
}
```

Try it Yourself »

`Array.filter()` is supported in all browsers except Internet Explorer 8 or earlier.

Yes	9.0	Yes	Yes	Yes

Array.reduce()

The `reduce()` method runs a function on each array element to produce (reduce it to) a single value.

The `reduce()` method works from left-to-right in the array. See also `reduceRight()`.

> The `reduce()` method does not reduce the original array.

This example finds the sum of all numbers in an array:

Example

```
var numbers1 = [45, 4, 9, 16, 25];
var sum = numbers1.reduce(myFunction);

function myFunction(total, value, index, array) {
  return total + value;
}
```

Try it Yourself »

Note that the function takes 4 arguments:

- The total (the initial value / previously returned value)
- The item value
- The item index
- The array itself

The example above does not use the index and array parameters. It can be rewritten to:

Example

```
var numbers1 = [45, 4, 9, 16, 25];
var sum = numbers1.reduce(myFunction);

function myFunction(total, value) {
  return total + value;
}
```

Try it Yourself »

The `reduce()` method can accept an initial value:

Example

```
var numbers1 = [45, 4, 9, 16, 25];
var sum = numbers1.reduce(myFunction, 100);

function myFunction(total, value) {
  return total + value;
}
```

Try it Yourself »

`Array.reduce()` is supported in all browsers except Internet Explorer 8 or earlier.

				
Yes	9.0	Yes	Yes	Yes

Array.reduceRight()

The `reduceRight()` method runs a function on each array element to produce (reduce it to) a single value.

The `reduceRight()` works from right-to-left in the array. See also `reduce()`.

> The `reduceRight()` method does not reduce the original array.

This example finds the sum of all numbers in an array:

Example

```
var numbers1 = [45, 4, 9, 16, 25];
var sum = numbers1.reduceRight(myFunction);
```

```
function myFunction(total, value, index, array) {
  return total + value;
}
```

Try it Yourself »

Note that the function takes 4 arguments:

- The total (the initial value / previously returned value)
- The item value
- The item index
- The array itself

The example above does not use the index and array parameters. It can be rewritten to:

Example

```
var numbers1 = [45, 4, 9, 16, 25];
var sum = numbers1.reduceRight(myFunction);

function myFunction(total, value) {
  return total + value;
}
```

Try it Yourself »

`Array.reduceRight()` is supported in all browsers except Internet Explorer 8 or earlier.

Yes	9.0	Yes	Yes	Yes

Array.every()

The `every()` method check if all array values pass a test.

This example check if all array values are larger than 18:

Example

```
var numbers = [45, 4, 9, 16, 25];
var allOver18 = numbers.every(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

Try it Yourself »

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

When a callback function uses the first parameter only (value), the other parameters can be omitted:

Example

```
var numbers = [45, 4, 9, 16, 25];
var allOver18 = numbers.every(myFunction);

function myFunction(value) {
  return value > 18;
}
```

Try it Yourself »

`Array.every()` is supported in all browsers except Internet Explorer 8 or earlier.

				
Yes	9.0	Yes	Yes	Yes

Array.some()

The `some()` method check if some array values pass a test.

This example check if some array values are larger than 18:

Example

```
var numbers = [45, 4, 9, 16, 25];
var someOver18 = numbers.some(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

Try it Yourself »

Note that the function takes 3 arguments:

- The item value
- The item index

- The array itself

`Array.some()` is supported in all browsers except Internet Explorer 8 or earlier.

				
Yes	9.0	Yes	Yes	Yes

Array.indexOf()

The `indexOf()` method searches an array for an element value and returns its position.

Note: The first item has position 0, the second item has position 1, and so on.

Example

Search an array for the item "Apple":

```
var fruits = ["Apple", "Orange", "Apple", "Mango"];
var a = fruits.indexOf("Apple");
```

Try it Yourself »

`Array.indexOf()` is supported in all browsers except Internet Explorer 8 or earlier.

				
Yes	9.0	Yes	Yes	Yes

Syntax

```
array.indexOf(item, start)
```

item	Required. The item to search for.
start	Optional. Where to start the search. Negative values will start at the given position counting from the end, and search to the end.

`Array.indexOf()` returns -1 if the item is not found.

If the item is present more than once, it returns the position of the first occurrence.

Array.lastIndexOf()

`Array.lastIndexOf()` is the same as `Array.indexOf()`, but searches from the end of the array.

Example

Search an array for the item "Apple":

```
var fruits = ["Apple", "Orange", "Apple", "Mango"];
var a = fruits.lastIndexOf("Apple");
```

Try it Yourself »

`Array.lastIndexOf()` is supported in all browsers except Internet Explorer 8 or earlier.

Yes	9.0	Yes	Yes	Yes

Syntax

```
array.lastIndexOf(item, start)
```

item	Required. The item to search for
start	Optional. Where to start the search. Negative values will start at the given position counting from the end, and search to the beginning

Array.find()

The `find()` method returns the value of the first array element that passes a test function.

This example finds (returns the value of) the first element that is larger than 18:

Example

```
var numbers = [4, 9, 16, 25, 29];
var first = numbers.find(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

Try it Yourself »

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

`Array.find()` is not supported in older browsers. The first browser versions with full support is listed below.

45	12	25	8	32

Array.findIndex()

The `findIndex()` method returns the index of the first array element that passes a test function.

This example finds the index of the first element that is larger than 18:

Example

```
var numbers = [4, 9, 16, 25, 29];
var first = numbers.findIndex(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

Try it Yourself »

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

`Array.findIndex()` is not supported in older browsers. The first browser versions with full support is listed below.

				
45	12	25	8	32