mdn web docs

Array.prototype.some()

Baseline Widely available

The `some()` method of `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.

Try it

JavaScript Demo: Array.prototype.some()

```
1 const array = [1, 2, 3, 4, 5];
2
3 // Checks whether an element is even
4 const even = (element) => element % 2 === 0;
5
6 console.log(array.some(even));
7 // Expected output: true
8
```

Run

Reset

Syntax

JS

```
some(callbackFn)
some(callbackFn, thisArg)
```

Parameters

`callbackFn`

A function to execute for each element in the array. It should return a truthy value to indicate the element passes the test, and a falsy value otherwise. The function is called with the following arguments:

> `element`
>
> The current element being processed in the array.
>
> `index`
>
> The index of the current element being processed in the array.
>
> `array`
>
> The array `some()` was called upon.

`thisArg` Optional

A value to use as `this` when executing `callbackFn`. See iterative methods.

Return value

`false` unless `callbackFn` returns a truthy value for an array element, in which case `true` is immediately returned.

Description

The `some()` method is an iterative method. It calls a provided `callbackFn` function once for each element in an array, until the `callbackFn` returns a truthy value. If such an element is found, `some()` immediately returns `true` and stops iterating through the array. Otherwise, if `callbackFn` returns a falsy value for all elements,

`some()` returns `false`. Read the iterative methods section for more information about how these methods work in general.

`some()` acts like the "there exists" quantifier in mathematics. In particular, for an empty array, it returns `false` for any condition.

`callbackFn` is invoked only for array indexes which have assigned values. It is not invoked for empty slots in sparse arrays.

`some()` does not mutate the array on which it is called, but the function provided as `callbackFn` can. Note, however, that the length of the array is saved *before* the first invocation of `callbackFn`. Therefore:

- `callbackFn` will not visit any elements added beyond the array's initial length when the call to `some()` began.
- Changes to already-visited indexes do not cause `callbackFn` to be invoked on them again.
- If an existing, yet-unvisited element of the array is changed by `callbackFn`, its value passed to the `callbackFn` will be the value at the time that element gets visited. Deleted elements are not visited.

> **Warning:** Concurrent modifications of the kind described above frequently lead to hard-to-understand code and are generally to be avoided (except in special cases).

The `some()` method is generic. It only expects the `this` value to have a `length` property and integer-keyed properties.

Examples

Testing value of array elements

The following example tests whether any element in the array is bigger than 10.

```js
function isBiggerThan10(element, index, array) {
  return element > 10;
}

[2, 5, 8, 1, 4].some(isBiggerThan10); // false
[12, 5, 8, 1, 4].some(isBiggerThan10); // true
```

Testing array elements using arrow functions

Arrow functions provide a shorter syntax for the same test.

```js
[2, 5, 8, 1, 4].some((x) => x > 10); // false
[12, 5, 8, 1, 4].some((x) => x > 10); // true
```

Checking whether a value exists in an array

To mimic the function of the `includes()` method, this custom function returns `true` if the element exists in the array:

```js
const fruits = ["apple", "banana", "mango", "guava"];

function checkAvailability(arr, val) {
  return arr.some((arrVal) => val === arrVal);
}

checkAvailability(fruits, "grapefruit"); // false
checkAvailability(fruits, "banana"); // true
```

Converting any value to Boolean

```js
const TRUTHY_VALUES = [true, "true", 1];

function getBoolean(value) {
  if (typeof value === "string") {
    value = value.toLowerCase().trim();
  }
```

```
  return TRUTHY_VALUES.some((t) => t === value);
}

getBoolean(false); // false
getBoolean("false"); // false
getBoolean(1); // true
getBoolean("true"); // true
```

Using the third argument of callbackFn

The `array` argument is useful if you want to access another element in the array, especially when you don't have an existing variable that refers to the array. The following example first uses `filter()` to extract the positive values and then uses `some()` to check whether the array is strictly increasing.

JS

```
const numbers = [3, -1, 1, 4, 1, 5];
const isIncreasing = !numbers
  .filter((num) => num > 0)
  .some((num, idx, arr) => {
    // Without the arr argument, there's no way to easily access the
    // intermediate array without saving it to a variable.
    if (idx === 0) return false;
    return num <= arr[idx - 1];
  });
console.log(isIncreasing); // false
```

Using some() on sparse arrays

`some()` will not run its predicate on empty slots.

JS

```
console.log([1, , 3].some((x) => x === undefined)); // false
console.log([1, , 1].some((x) => x !== 1)); // false
console.log([1, undefined, 1].some((x) => x !== 1)); // true
```

Calling some() on non-array objects

The `some()` method reads the `length` property of `this` and then accesses each property whose key is a nonnegative integer less than `length` until they all have been accessed or `callbackFn` returns `true`.

JS

```
const arrayLike = {
  length: 3,
  0: "a",
  1: "b",
  2: "c",
  3: 3, // ignored by some() since length is 3
};
console.log(Array.prototype.some.call(arrayLike, (x) => typeof x === "number"));
// false
```

Specifications

Specification
ECMAScript® 2026 Language Specification # sec-array.prototype.some

Browser compatibility

Report problems with this compatibility data • View data on GitHub

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebView Android	WebView on iOS	Deno	Node.js
`some`	✓ 1	✓ 12	✓ 1.5	✓ 9.5	✓ 3	✓ 18	✓ 4	✓ 10.1	✓ 1	✓ 1	✓ 4.4	✓ 1	✓ 1	✓ 0.10

Tip: you can click/tap on a cell for more information.

✓ Full support

See also

- Polyfill of `Array.prototype.some` in `core-js`
- es-shims polyfill of `Array.prototype.some`

- Indexed collections guide
- `Array`
- `Array.prototype.every()`
- `Array.prototype.forEach()`
- `Array.prototype.find()`
- `Array.prototype.includes()`
- `TypedArray.prototype.some()`

Help improve MDN

Was this page helpful to you?

Yes No

Learn how to contribute.

This page was last modified on Mar 14, 2025 by MDN contributors.