Array.prototype.filter()

Baseline Widely available

The `filter()` method of [`Array`] instances creates a [shallow copy] of a portion of a given array, filtered down to just the elements from the given array that pass the test implemented by the provided function.

Try it

JavaScript Demo: Array.prototype.filter()

```
const words = ["spray", "elite", "exuberant", "destruction", "present"];

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

console.log(result);
// Expected output: Array ["exuberant", "destruction", "present"]

```

Run

Reset

Syntax

JS

```
filter(callbackFn)
filter(callbackFn, thisArg)
```

Parameters

`callbackFn`

A function to execute for each element in the array. It should return a truthy value to keep the element in the resulting array, 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 `filter()` was called upon.

`thisArg` Optional

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

Return value

A shallow copy of the given array containing just the elements that pass the test. If no elements pass the test, an empty array is returned.

Description

The `filter()` method is an iterative method. It calls a provided `callbackFn` function once for each element in an array, and constructs a new array of all the values for which `callbackFn` returns a truthy value. Array elements which do not pass the `callbackFn` test are not included in the new array. Read the iterative methods section for more information about how these methods work in general.

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

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

Examples

Filtering out all small values

The following example uses `filter()` to create a filtered array that has all elements with values less than 10 removed.

JS

```
function isBigEnough(value) {
  return value >= 10;
}

const filtered = [12, 5, 8, 130, 44].filter(isBigEnough);
// filtered is [12, 130, 44]
```

Find all prime numbers in an array

The following example returns all prime numbers in the array:

JS

```
const array = [-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13];

function isPrime(num) {
  for (let i = 2; num > i; i++) {
    if (num % i === 0) {
      return false;
    }
  }
  return num > 1;
}

console.log(array.filter(isPrime)); // [2, 3, 5, 7, 11, 13]
```

Filtering invalid entries from JSON

The following example uses `filter()` to create a filtered JSON of all elements with non-zero, numeric `id`.

JS

```js
const arr = [
  { id: 15 },
  { id: -1 },
  { id: 0 },
  { id: 3 },
  { id: 12.2 },
  {},
  { id: null },
  { id: NaN },
  { id: "undefined" },
];

let invalidEntries = 0;

function filterByID(item) {
  if (Number.isFinite(item.id) && item.id !== 0) {
    return true;
  }
  invalidEntries++;
  return false;
}

const arrByID = arr.filter(filterByID);

console.log("Filtered Array\n", arrByID);
// Filtered Array
// [{ id: 15 }, { id: -1 }, { id: 3 }, { id: 12.2 }]

console.log("Number of Invalid Entries =", invalidEntries);
// Number of Invalid Entries = 5
```

Searching in array

Following example uses `filter()` to filter array content based on search criteria.

JS

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

/**
 * Filter array items based on search criteria (query)
 */
function filterItems(arr, query) {
  return arr.filter((el) => el.toLowerCase().includes(query.toLowerCase()));
}

console.log(filterItems(fruits, "ap")); // ['apple', 'grapes']
console.log(filterItems(fruits, "an")); // ['banana', 'mango', 'orange']
```

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 `map()` to extract the numerical ID from each name and then uses `filter()` to select the ones that are greater than its neighbors.

JS

```
const names = ["JC63", "Bob132", "Ursula89", "Ben96"];
const greatIDs = names
  .map((name) => parseInt(name.match(/[0-9]+/)[0], 10))
  .filter((id, 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 && id <= arr[idx - 1]) return false;
    if (idx < arr.length - 1 && id <= arr[idx + 1]) return false;
    return true;
  });
console.log(greatIDs); // [132, 96]
```

The `array` argument is *not* the array that is being built — there is no way to access the array being built from the callback function.

Using filter() on sparse arrays

`filter()` will skip empty slots.

JS

```
console.log([1, , undefined].filter((x) => x === undefined)); // [undefined]
console.log([1, , undefined].filter((x) => x !== 2)); // [1, undefined]
```

Calling filter() on non-array objects

The `filter()` method reads the `length` property of `this` and then accesses each property whose key is a nonnegative integer less than `length`.

JS

```
const arrayLike = {
  length: 3,
  0: "a",
  1: "b",
  2: "c",
  3: "a", // ignored by filter() since length is 3
};
console.log(Array.prototype.filter.call(arrayLike, (x) => x <= "b"));
// [ 'a', 'b' ]
```

Specifications

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

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
`filter`	✓ 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.filter` in `core-js`
- es-shims polyfill of `Array.prototype.filter`
- Indexed collections guide
- `Array`
- `Array.prototype.forEach()`
- `Array.prototype.every()`
- `Array.prototype.map()`
- `Array.prototype.some()`
- `Array.prototype.reduce()`
- `TypedArray.prototype.filter()`

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.