mdn web docs

# Array.prototype.find()

**Baseline** Widely available 

The `find()` method of `Array` instances returns the first element in the provided array that satisfies the provided testing function. If no values satisfy the testing function, `undefined` is returned.

- If you need the **index** of the found element in the array, use `findIndex()`.
- If you need to find the **index of a value**, use `indexOf()`. (It's similar to `findIndex()`, but checks each element for equality with the value instead of using a testing function.)
- If you need to find if a value **exists** in an array, use `includes()`. Again, it checks each element for equality with the value instead of using a testing function.
- If you need to find if any element satisfies the provided testing function, use `some()`.
- If you need to find all elements that satisfy the provided testing function, use `filter()`.

## Try it

JavaScript Demo: Array.prototype.find()

```
1 const array1 = [5, 12, 8, 130, 44];
2
3 const found = array1.find((element) => element > 10);
4
5 console.log(found);
6 // Expected output: 12
7
```

Run

Reset

# Syntax

JS

```
find(callbackFn)
find(callbackFn, thisArg)
```

## Parameters

`callbackFn`

A function to execute for each element in the array. It should return a truthy value to indicate a matching element has been found, 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 `find()` was called upon.

`thisArg` Optional

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

## Return value

The first element in the array that satisfies the provided testing function. Otherwise, `undefined` is returned.

## Description

The `find()` method is an iterative method. It calls a provided `callbackFn` function once for each element in an array in ascending-index order, until `callbackFn` returns a truthy value. `find()` then returns that element and stops iterating through the array. If `callbackFn` never returns a truthy value, `find()` returns `undefined`. Read the iterative methods section for more information about how these methods work in general.

`callbackFn` is invoked for *every* index of the array, not just those with assigned values. Empty slots in sparse arrays behave the same as `undefined`.

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

## Examples

### Find an object in an array by one of its properties

JS

```
const inventory = [
  { name: "apples", quantity: 2 },
  { name: "bananas", quantity: 0 },
  { name: "cherries", quantity: 5 },
];

function isCherries(fruit) {
  return fruit.name === "cherries";
}

console.log(inventory.find(isCherries));
// { name: 'cherries', quantity: 5 }
```

### Using arrow function and destructuring

JS

```
const inventory = [
  { name: "apples", quantity: 2 },
  { name: "bananas", quantity: 0 },
  { name: "cherries", quantity: 5 },
];

const result = inventory.find(({ name }) => name === "cherries");

console.log(result); // { name: 'cherries', quantity: 5 }
```

## Find the first prime number in an array

The following example returns the first element in the array that is a prime number, or `undefined` if there is no prime number.

JS

```
function isPrime(element, index, array) {
  let start = 2;
  while (start <= Math.sqrt(element)) {
    if (element % start++ < 1) {
      return false;
    }
  }
  return element > 1;
}

console.log([4, 6, 8, 12].find(isPrime)); // undefined, not found
console.log([4, 5, 8, 12].find(isPrime)); // 5
```

## 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 `find()` to find the first element that is less than its neighbors.

JS

```
const numbers = [3, -1, 1, 4, 1, 5, 9, 2, 6];
const firstTrough = numbers
  .filter((num) => num > 0)
  .find((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 && num >= arr[idx - 1]) return false;
    if (idx < arr.length - 1 && num >= arr[idx + 1]) return false;
    return true;
  });
console.log(firstTrough); // 1
```

## Using find() on sparse arrays

Empty slots in sparse arrays *are* visited, and are treated the same as `undefined`.

JS

```
// Declare array with no elements at indexes 2, 3, and 4
const array = [0, 1, , , , 5, 6];

// Shows all indexes, not just those with assigned values
array.find((value, index) => {
  console.log("Visited index", index, "with value", value);
});
// Visited index 0 with value 0
// Visited index 1 with value 1
// Visited index 2 with value undefined
// Visited index 3 with value undefined
// Visited index 4 with value undefined
// Visited index 5 with value 5
// Visited index 6 with value 6

// Shows all indexes, including deleted
array.find((value, index) => {
  // Delete element 5 on first iteration
  if (index === 0) {
    console.log("Deleting array[5] with value", array[5]);
    delete array[5];
  }
  // Element 5 is still visited even though deleted
  console.log("Visited index", index, "with value", value);
});
// Deleting array[5] with value 5
// Visited index 0 with value 0
// Visited index 1 with value 1
// Visited index 2 with value undefined
// Visited index 3 with value undefined
```

```
// Visited index 4 with value undefined
// Visited index 5 with value undefined
// Visited index 6 with value 6
```

### Calling find() on non-array objects

The `find()` 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,
  "-1": 0.1, // ignored by find() since -1 < 0
  0: 2,
  1: 7.3,
  2: 4,
};
console.log(Array.prototype.find.call(arrayLike, (x) =>
!Number.isInteger(x)));
// 7.3
```

## Specifications

| Specification |
| --- |
| ECMAScript® 2026 Language Specification<br># sec-array.prototype.find |

## 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 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| `find` | ✓ 45 | ✓ 12 | ✓ 25 | ✓ 32 | ✓ 8 | ✓ 45 | ✓ 4 | ✓ 32 | ✓ 8 | ✓ 5 | ✓ 45 | ✓ 8 | ✓ 1 | ✓ 4 |

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

✓ | Full support

## See also

- Polyfill of `Array.prototype.find` in `core-js`
- es-shims polyfill of `Array.prototype.find`
- Indexed collections guide
- `Array`
- `Array.prototype.findIndex()`
- `Array.prototype.findLast()`
- `Array.prototype.findLastIndex()`
- `Array.prototype.includes()`
- `Array.prototype.filter()`
- `Array.prototype.every()`
- `Array.prototype.some()`
- `TypedArray.prototype.find()`