Jeff Lombard Follow
JavaScript Developer
Apr 9 · 3 min read

Why and when to use forEach, map, filter, reduce, and find in JavaScript.

Many posts discuss *how* to use `.forEach()`, `.map()`, `.filter()`, `.reduce()` and `.find()` on arrays in JavaScript. I thought it would be useful to provide an explanation of *when* to use the common array methods.

`.map()`, `.filter()`, `.reduce()` and `.find()` all behave very similarly to `.forEach()` so for now lets just focus on the latter.

What is forEach? A way that to work with items in an array.

If you're not familiar with the array method `.forEach()`, whenever you go to iterate over an array you probably immediately think of a `for` loop.

The `.forEach()` method is just a different way of doing this. The following two code examples effectively accomplish the same thing:

`for` loop

```
var array = [1,2,3];

for (var i = 0; i < array.length; i++){
  console.log(i);
}
```

`.forEach()`

```
var array = [1,2,3];

array.forEach(function(item){
  console.log(i);
});
```

There are however some subtle differences that can have a big impact on your code.

Why forEach? Ease of Use and Readability.

To me, the most compelling case for using `.forEach()` in favor of a `for` loop is that it's easier. Even though it's the same number of lines, there's less setup. With a regular `for` loop you have three steps:

1. Define an iterator value: var i = 0;
2. Define an end point: i < array.length;
3. Tell how the loop how should iterate: i++;

With `.forEach()` you simply pass a function that is executed on each element in the array.

Why forEach? Scope.

If the fact that it's easier, isn't enough for you… there is also a technical reason to use `.forEach()` in favor of `for` loops. It has to do with the scoping of the code.

When using the `.forEach()` you pass an individual function with it's own scope. In a `for` loop you're polluting whatever scope you place the loop in. Most, if not all, of the time, this is a bad thing.

. . .

When to use forEach?

`.forEach()` is great you need to execute a function for each individual element in an array. Good practice is that you should use `.forEach()` when you can't use other array methods to accomplish your goal. I know this may sound vague, but `.forEach()` is a generic tool… only use it when you can't use a more specialized tool.

When to use map?

`.map()` when you want to **transform** elements in an array.

When to use filter?

`.filter()` when you want to **select** a subset of *multiple* elements from an array.

When to use find?

`.find()` When you want to **select** a *single* element from an array.

When to use reduce?

`.reduce()` when you want **derive** a *single* value from *multiple* elements in an array.

Quirks and Criticisms

forEach returns undefined.

If you're *transforming* the entirety of the array, you'll probably want to use `.map()` instead. `.map()` actually returns an array which is useful, because you can chain other array methods.

Example:

```
const arr = [1,2,3];


const transformedArr = arr.map(function()
{}).filter(function(){});
```

A word about speed.

One of the biggest criticisms of `.forEach()` is the speed of the operation.

In reality, you shouldn't be using `.forEach()` unless other methods such as `.map()` can't do the trick. `.map()` is actually slightly faster than `.forEach()` .

Admittedly, `.forEach()` and `.map()` are still slower than a vanilla `for` loop. But judging a method solely based on execution speed is tunnel-visioned. This argument completely ignores readability and scope.