forEach

Learn the third of the most powerful array functions. We'll cover how `forEach` allows us to quickly process data and how it differs from `map` and `filter`.

We're going to learn the third Array function that can help us work efficiently with arrays and the data inside them.

`forEach`

`Array.forEach` is different from `map` & `filter`. You may find it easier to reason about. `map` & `filter` return us new, altered arrays based off of the original array. `forEach` returns us `undefined`. It's used purely to do something with the items in an array, not to get a new array out of it.

Here's an example. The following two code blocks are equivalent.

```
const arr = ['Hello', 'there!', 'How', 'are', 'you', 'doing?'];
let str = '';

for(let i = 0; i < arr.length; i++) {
    const word = arr[i];
    console.log(word);
    str += word + ' ';
}

console.log(`Final string: "${str}"`);
```

```
const arr = ['Hello', 'there!', 'How', 'are', 'you', 'doing?'];
let str = '';

arr.forEach(word => {
    console.log(word);
    str += word + ' ';
});

console.log(`Final string: "${str}"`);
```

We're taking an array of individual words and logging each to the console. We're also concatenating them by adding them all one by one to our initially empty string `str`.

`forEach` vs `map` & `filter`

We're using `forEach` to implement a side effect. The callback we pass in isn't meant to be a pure function. It's meant to affect something outside its own local scope. This contrasts with `map` and `filter` which should ideally be given pure functions.

We *could* do this same thing with `map` or `filter`.

```
const arr = ['Hello', 'there!', 'How', 'are', 'you', 'doing?'];
let str = '';

arr.map(word => {
    console.log(word);
    str += word + ' ';
});

console.log(`Final string: "${str}"`);
```

```
const arr = ['Hello', 'there!', 'How', 'are', 'you', 'doing?'];
let str = '';

arr.filter(word => {
    console.log(word);
    str += word + ' ';
});

console.log(`Final string: "${str}"`);
```

Both of these blocks above do the same thing for us. We pass in the same exact callback that we gave to `forEach`. Although `map` and `filter` are still providing us a return value, we're not using it.

Why to use `forEach`

The reason to use `forEach` over either of those functions is to make it clear that we want to affect the outer scope. When we see `map` and `filter`, by convention, we should expect the functions to do nothing except return us a new array.

`forEach` is essentially meant to be an impure version of `map`. An engineer will typically use this method to signify the intent of changing an external value by using the array.

We should follow this convention when we write code. The use of `forEach` shows a different intent than any of the other functions.

Like `map` and `filter`, `forEach` passes in two additional arguments as well, the index and original array.

```
const arr = ['a', 'b', 'c', 'd', 'e'];

arr.forEach((letter, index, array) => {
    console.log(letter, index, array);
});
```

Since you know how to write map and filter, you should now be able to write your own version of `forEach`. If you want, go ahead and give it a try!

Writing `forEach` Exercise

```
function forEach() {
}
```

That's it for `forEach`.

We'll cover the last of the main array functions, `reduce`, in the next lesson.