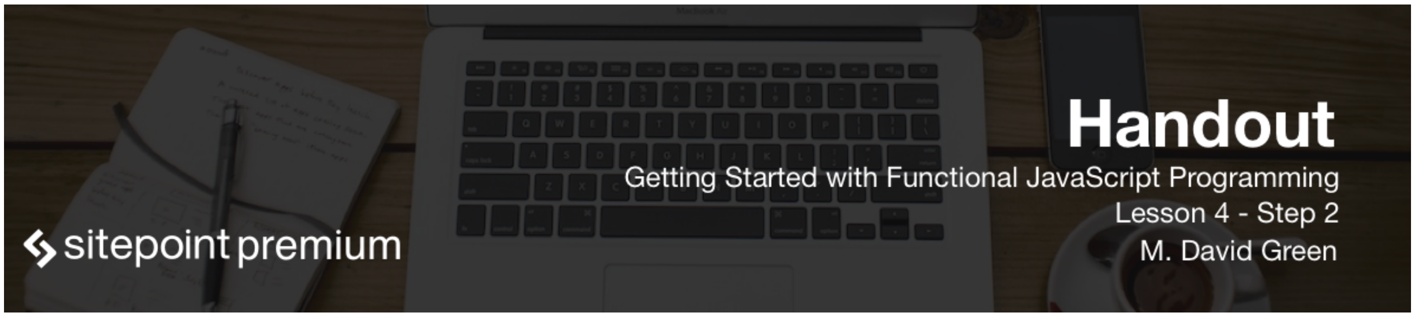

So we were looking at different ways that we could approach looping inside of JavaScript without the need to maintain state and all those extra variables. And the first new method that I'd like to introduce you to is the `map` method on the array. Mapping lets you create a new array by modifying the values in an existing array, and returning another array that has exactly the same number of elements but they've all been modified according to a function that you pass in.

Mapping operates on each element in the array, and then returns a new array with altered values, leaving the original array completely unchanged. And ideally, you don't want to alter the state of the your array or anything outside of the array while you're performing this method. And you also don't want to rely on anything outside of the array that you've passed in while you're processing the elements.

So solving the same problem that we were trying to solve in our first example with the loop, let's create a new array containing the lengths of all of the strings in an existing array using the `map` method.

```
const animals = ["cat","dog","fish"];
let letterCounts = animals.map(animal => animal.length);
console.log(letterCounts); //[3, 3, 4]
```

The `map` method can operate with any function that returns a value and as you can see we're not even returning a value of the same type. Our new array has numbers, our initial array had strings. What's more, using pure functions, it's possible to make our `map` even easier to read.

```
const animals = ["cat","dog","fish"];
const getLength = item => item.length;
let letterCounts = animals.map(getLength);
console.log(letterCounts); //[3, 3, 4]
```

All that we've done here is a tiny refactor on our initial code. We defined a new `getLength` function and then passed it in instead of passing in the function inline. Now that might seem like sleight of hand, but there are some advantages to that. Because that `getLength` function is now a pure function that we can use elsewhere, it's also very easy to test.

In this case it's an incredibly simple function, but you can imagine where you might have a much more complex function and you'd want to be able to pass it in just as easily as that. So good mapping function is a pure function. It takes a single value of the type of the element in the array being mapped, and it operates on that value without creating side effects.

And it always produces the same result for the same input value. Ideally, the mapping function returns a value of the type that's expected in the new array, because every element in the new array is going to be of the type that that function returns. And you don't want it to alter any variables outside of its own scope.

The nice thing about these pure functions, is that they can be used without modification, in other contexts. So you could pull in a function, that you defined elsewhere, and just pass it to your map function and have it work exactly the way that you expect it to. And just as a little bit of trivia, have you ever heard the term "functor" before?

It’s a functional programming term that some people throw around. But the cool thing is, a functor is simply something that holds a set of values and implements a map function to operate on each element. And that map function always returns a functor of the same size. As of ECMAScript 5, the JavaScript array is now officially a functor, pull that one out and you’re sure to be popular at all the parties.