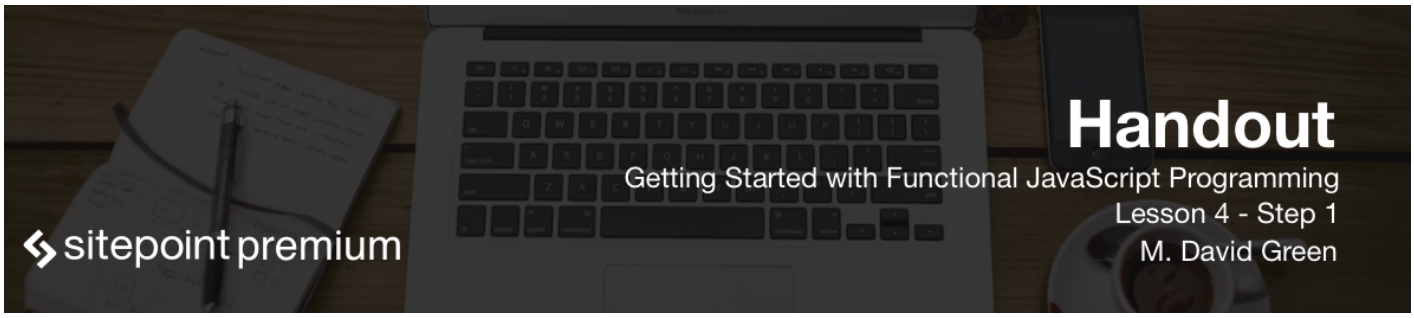

Welcome to Lesson 4 of Functional Programming in JavaScript from SitePoint. In this lesson, we're going to be talking about `map`, `reduce` and `filter`, three of the new array methods that we've gotten in ECMAScript 5. We're also going to show how we can use these together with chaining to take better advantage of the functional potential of JavaScript.

These methods allow you to iterate through an array, much the way that you might currently be doing with familiar loops. Condition-controlled loops are something that vanilla JavaScript has had ever since it was first created. These loops can take an array of items, and then move through the array sequentially.

Usually you perform an operation on each element of the array. The problem is that, unlike a functional paradigm, condition-controlled loops, unlike the filter, map, and reduce methods, often alter variables outside of the loop, and certainly require variables inside of the loop in order to perform their functions properly.

These loops function by creating some state, defining a condition that that state should be in when they terminate, and terminating when that state is achieved. With ECMAScript 5, we've been presented with some alternatives to looping that are more functional and cleaner. One of them is mapping which allows us to produce a new array with modified values from an existing array.

Another is reducing, which produces a summarized result across all of the values from an array. And filtering allows us to limit the set of elements in an array, and return new subset of those elements in a new array. Just to get us all on the same page, let's start by looking at how a traditional for loop is created.

```
const animals = ["cat","dog","fish"];
const animalsLength = animals.length;
let lengths = [];
for (let count = 0; count < animalsLength; count++){
  lengths.push(animals[count].length);
}
console.log(lengths);
```

There shouldn't have been anything in that code that looked unusual to you. We simply created an empty array, pushed the new values into that array inside of a for loop, and then logged the results. So what's wrong with this approach to iterating through an array? Well, it gets the job done, certainly.

And it's very familiar, you can see examples of this kind of looping all the way back to the beginning of JavaScript. The problem is that extra variables need to be declared and managed. For example we had to create our count variable and that had to increment every time that we went through the loop.

We're also making changes outside of our loop by modifying the lengths array, every single time that we ran through the loop. So the behavior inside of the for loop was not contained to that for loop. The syntax honestly is fiddly when you look at how for loops are constructed.

You've got your initial state, you've got your conditional, and you've got your incrementer all stuck together inside of those parentheses at the beginning. Which, yeah, is understandable, you've seen it before, but it's not as clean as the

syntax could be. So let's start looking at some of the ways that these new methods that have been added to the array object can clean up our code and make things a little bit more state free and less messy.