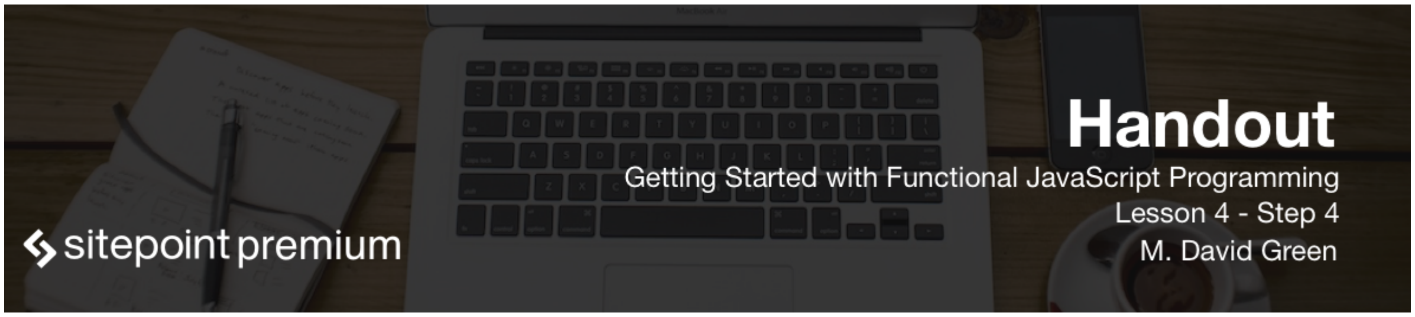

The last of the new functional methods on the array that was introduced in ECMAScript 5 that we want to talk about is filtering, which produces a new array containing a subset of the elements in an existing array based on the condition that you set. Filtering limits the contents of an array.

So, it evaluates each item in the array based on the test condition, that returns true or false, and it returns a new array with only the items that pass the test, leaving the original array completely unchanged, just like functional methods should. It doesn't alter the state outside of the array that's being returned.

And it doesn't rely on the state of the application or anything outside of itself in order to perform its operation properly. In the past, you may have used a loop to filter arrays. And an example of the type of problems that filtering can help with would be finding only the strings inside of an array that contain exactly three letters.

```
const animals = ["cat","dog","fish"];
const animalsLength = animals.length;
let threeLetterAnimals = [];
for (let count = 0; count < animalsLength; count++){
  if (animals[count].length === 3) {
    threeLetterAnimals.push(animals[count]);
  }
}
console.log(threeLetterAnimals); // ["cat", "dog"]
```

Now let's eliminate the loop by using the `filter` method, taking a function that returns a boolean value, as a condition.

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

That `filter` method went through our array, applied that boolean analysis to each of the elements, and if the result was true, it added that element to the array that it was going to return. And as a result, we got back an array that was filtered containing a subset of the elements that matched true for that boolean condition.

Now as we've done for `map` and `reduce`, we can do the same thing for `filter`. We can pull out a named function from that anonymous function to make our code cleaner and easier to read. And by now you can probably do this just as easily as I'm going to.

```
const animals = ["cat","dog","fish"];
const exactlyThree = item => item.length === 3;
let threeLetterAnimals = animals.filter(exactlyThree);
console.log(threeLetterAnimals); // ["cat", "dog"]
```

The type of functions that we want to be able to pass into our `filter` method are pure functions, of course. They take a single value of the type that's in the array that we want to filter, and then apply a test on that value without any side effects, and without relying on the state of the application.

They would always produce the same result for the same input value, and the functions that we want to pass into the filter method are always returning a boolean value. This shouldn't alter any values outside of their scope, and of course, as with all pure functions, they shouldn't alter any variables outside of their scope.

And because they're pure, they could potentially be used in other contexts without any modification.