The for Loop vs. forEach in JavaScript

If you're new to JavaScript, you may be wondering what the difference is between using a classic `for` loop versus using the `forEach()` method on arrays. In this post, I'd like to review how each of these work and discuss why you might want to choose one over the other.

The Mechanics

Let's review the mechanics of the standard `for` loop and the `forEach()` method.

```
for (var i = 0; i < 3; i++) {
  console.log(i);
}
```

The `for` loop takes 3 statements. The first statement `var i = 0;` is executed before the loop starts. The second statement `i < 3` defines the condition for running the block of code. The third statement runs after each loop. The result is that this loop will execute the `console.log()` statement 3 times with the values 0, 1, and 2.

Let's say we have an array of products:

```
var products = [
  { name: 'Running shoes', price: 75 },
  { name: 'Golf shoes',    price: 85 },
  { name: 'Dress shoes',   price: 95 },
  { name: 'Walking shoes', price: 65 },
  { name: 'Sandals',       price: 55 }
];
```

If we wanted to loop over each product, we would change the condition in the `for` loop and use `i` as the numeric index to access the product for the current iteration.

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

The `forEach()` method on arrays can be used to achieve the same thing:

```
products.forEach(function(product, index) {
  console.log(product);
});
```

Because `products` is an array, it inherits all of the methods on `Array.prototype` like `Array.prototype.forEach()` which we can invoke, passing in a function that will execute for each iteration. This function will be passed the product for the current iteration.

Now let's look at why you might want to choose one over the other.

for vs forEach()

1. Improved Readability with forEach()

Both a `for` loop and the `forEach()` method allow you to loop over an array, but let me give you my perspective on why I prefer `forEach()` most of the time. First, I find that `forEach()` has better readability than the `for` loop. In the example above, the product for each iteration is passed to the callback function. I don't have to access the current iteration's product using the temporary `i` variable as such: `products[i]`. Even though it isn't THAT hard to read, when you add more code, it adds a little more cognitive overhead. Imagine if you had a `for` loop within a `for` loop, like this:

```
for (var i = 0; i < products.length; i++) {
  console.log(products[i]);
  for (var j = 0; j < products[i].sizes.length; j++) {
    console.log(products[i].sizes[j]);
  }
}
```

This is even harder to read, and the problem compounds when you do more than a `console.log()`. You could improve this code by using a few variables, as such:

```
for (var i = 0; i < products.length; i++) {
  var product = products[i];
  console.log(product);
  for (var j = 0; j < product.sizes.length; j++) {
    var size = product.sizes[j];
    console.log(size);
  }
}
```

Having variables `product` and `size` helps a little bit, but I still don't like those temporary `i` and `j` variables. Now compare the above code with using `forEach()` below:

```
products.forEach(function(product) {
  product.sizes.forEach(function(size) {
    console.log(size);
  });
});
```

Not only have are there fewer lines of code, we've done away with the temporary `i` and `j` counter variables. I find this much easier to read since there is less noise.

2. Fewer off-by-one errors with forEach()

Wikipedia defines an off-by-one error as:

> *An off-by-one error (OBOE), also commonly known as an OBOB (off-by-one bug) or "that extra inch you didn't really want", is a logic error involving the discrete equivalent of a boundary condition. It often occurs in computer programming when an iterative loop iterates one time too many or too few.*

There are a few ways of producing an off-by-one error, but here is a simple

example.

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

This `for` loop looks pretty similar to the one earlier in this post, right? There is one small difference. Notice that in statement 2 of the `for` loop, the condition statement now contains <= instead of <. As a result, this loop will run one too many times. Even though this is a simple example, these types of errors can easily creep in if you aren't careful. With the `forEach()` method, you don't have to think about the condition statement at all, resulting in fewer, if any, off-by-one bugs.

3. Breaking Out Of Loops Early

One scenario where I choose a `for` loop over the `forEach()` method is when I want to break out of a loop early. Imagine I had a longer list of products and as soon as I found one that matches some criteria, I want to perform some action. If I used `forEach()`, it would iterate over every single product resulting in unnecessary iterations, potentially causing performance issues depending on how long the array is. With a `for` loop, you have the ability to break out early and stop the loop from continuing. For example:

```
for (var i = 0; i < products.length; i++) {
  if (matchesSomeCriteria(products[i])) {
    doSomething();
    break;
  }
}
```

Using the `break` keyword, we can stop the loop from continuing on as soon as soon as we've found what we're looking for. It's worth noting that as of ES6, there is a new method on arrays for finding an element, which is similar to what we're doing above. This method is the `find()` method and I encourage you to check it out. However, not all browsers support this yet at

the time of this writing so you may need to use a transpiler like Babel or add a polyfill.

Summary

This post isn't an exhaustive list of when you'd use a `for` loop over the `forEach()` method on arrays and vice versa, but I covered the more common situations I've run into and when I choose one over the other. One thing I didn't cover was performance. If you look at performance metrics comparing the `for` loop to `forEach()`, the `for` loop is faster. However, a lot of the time this added performance won't make a difference in your application, and using `for` over `forEach()` is premature optimization. Instead, optimize for code readability to ensure easier long term maintenance. When performance becomes an issue with `forEach()`, then reach for the `for` loop and see if it makes a difference.

Disclaimer: Any viewpoints and opinions expressed in this article are those of David Tang and do not reflect those of my employer or any of my colleagues.