JavaScript Array.prototype.forEach

The JavaScript Array prototype forEach method allows you to iterate over an array without the need to set a variable.

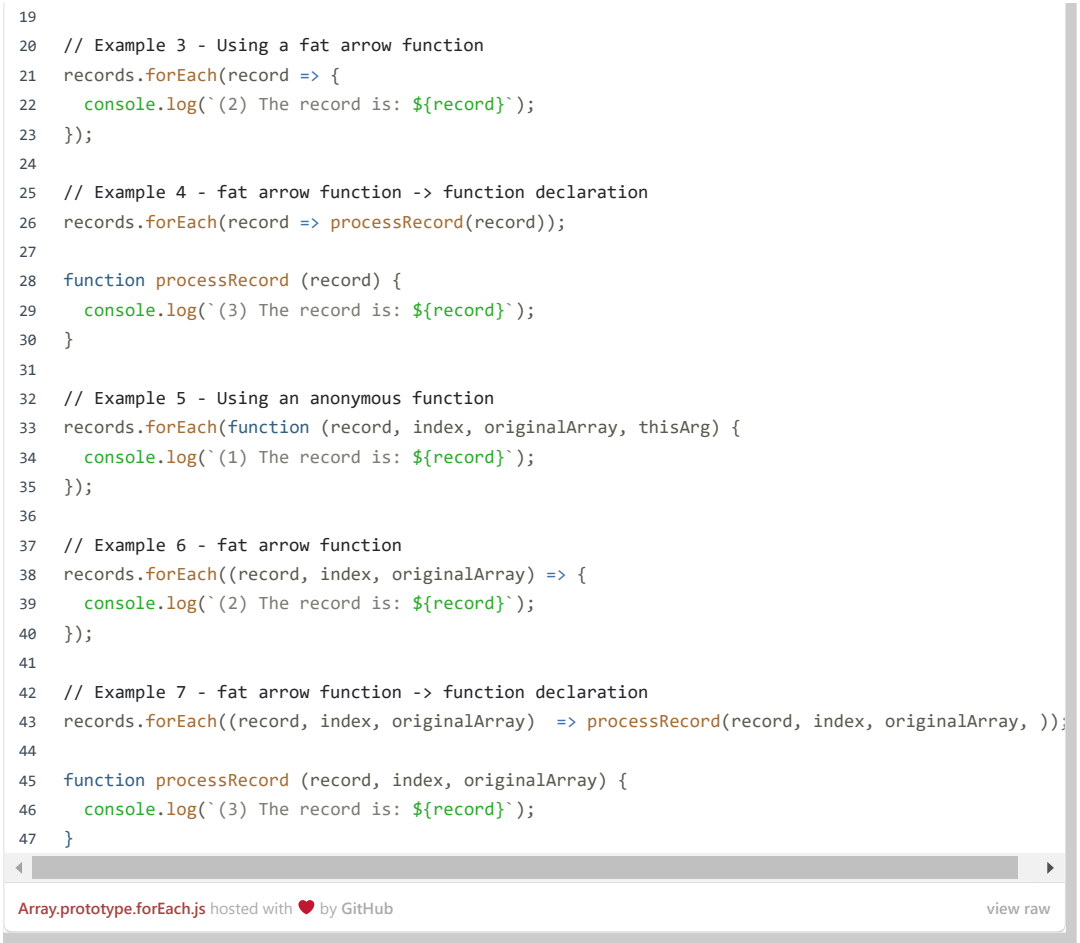
A forEach loop is a powerful tool that can simplify your code and help reduce the number of lines required to complete a task. The JavaScript Array prototype forEach method is like a for loop but is designed especially for use with arrays. A for loop is not tied to an array and is more general purpose.

Example 1

To cycle through every element of an array without the JavaScript Array.prototype.forEach method, we would use a for loop, like Example 1. We need a variable to hold the current iteration number and we usually use the lowercase letter i to store the iteration number. We also use the Length property on the array to get its size and store that value in a variable called **arrayLength**. Then we can use a for loop to iterate through the array as long as the value of i is less than the value of arrayLength.

The JavaScript Array prototype forEach method removes the need for the extra variables, and you can see by comparing Example 1 with Example 2, that it helps make it easier to read. Let's look at some examples of how the JavaScript Array prototype forEach method can be helpful, and how to use it.

```
// Example 1 - Using a for loop

var i = 0,
        records = ['foo', 'bar', 'baz'],
        arrayLength = records.length;

for (; i < arrayLength; i++) {
        console.log(`(${i}) The record is: ${records[i]}`);
}

// Example 2 - Using the forEach method

var records = ['foo', 'bar', 'baz'];

// anonymous function
records.forEach(function (record) {
  console.log(`(1) The record is: ${record}`);
});
```



```
19
20  // Example 3 - Using a fat arrow function
21  records.forEach(record => {
22    console.log(`(2) The record is: ${record}`);
23  });
24
25  // Example 4 - fat arrow function -> function declaration
26  records.forEach(record => processRecord(record));
27
28  function processRecord (record) {
29    console.log(`(3) The record is: ${record}`);
30  }
31
32  // Example 5 - Using an anonymous function
33  records.forEach(function (record, index, originalArray, thisArg) {
34    console.log(`(1) The record is: ${record}`);
35  });
36
37  // Example 6 - fat arrow function
38  records.forEach((record, index, originalArray) => {
39    console.log(`(2) The record is: ${record}`);
40  });
41
42  // Example 7 - fat arrow function -> function declaration
43  records.forEach((record, index, originalArray)  => processRecord(record, index, originalArray, ));
44
45  function processRecord (record, index, originalArray) {
46    console.log(`(3) The record is: ${record}`);
47  }
```

Array.prototype.forEach.js hosted with ♥ by GitHub view raw

Example 2

In Example 2, we use the JavaScript Array prototype forEach method and an anonymous function to print the value of every element in an array.

The code **records.forEach();** tells the computer to start at the first element in the array named records, and for each iteration, do what it says in parentheses. In this case, it will update the variable **(record)** and pass that value to an anonymous function.

We call it an anonymous function because it doesn't have a name. It has only the function itself between the brackets **{}**. We could name it processRecords or something, but since we aren't going to use this function elsewhere, it's unnecessary. The function does the work of printing the value to the screen with the **console.log** command. Console.log prints the value of **record** to the screen.

Example 3

In Example 3, we use an arrow function as a shorthand way to write the anonymous function from Example 2. It's more concise and more comfortable to read, but we still update the value of **record** and pass it to the anonymous function where **console.log** prints it on the screen as we do in Example 1.

Example 4

In Example 4, we use a traditional standalone function called processRecord that also passes a variable called **record** to **console.log**. It cannot set the value of record, nor iterate through the array. To iterate through the array and set the value of record, we use the arrow function with the JavaScript Array prototype forEach method.

For each iteration in this example, we set the value of record and send it to the processRecords function where it will pass the value to console.log.

We would use this example when we want to access the processRecords function from other places in our code.

Syntax

- Now that we have taken a look at a few different ways to use the Array.prototype.forEach method, let's dig a little deeper and look at the syntax to see what else we can do.
  arr.forEach(callback(currentValue [, index [, array]])[, thisArg]);
- arr = Name of the array that we are iterating. We have been using **records**.
- callback = The function we are calling each iteration. We have used **anonymous** and **processRecords**.
- currentValue = The value of the current array element. We have been setting this to **record**.
- index = The current element id or number. We will be using **index** as well in upcoming examples.
- array = The array we are using the forEach method to iterate through. This variable refers to the entire contents of the array, not the name. We will be setting this to **originalArray** in upcoming examples.
- thisArg = The value to use as ***this*** when executing a callback. This value is for updating values in the array, among other things.

Now that we have looked over the syntax of a JavaScript Array prototype forEach method let's look at a few more examples. So far, in our first three examples, we have only passed the current value to the callback function in the form of a variable called record, but we can also pass the index, array, and thisArg values as well.

Optional Values

There is one thing to remember about the optional values of index, array, and thisArg. You must assign the variables in order. To assign a variable to the array value, you must also assign one to index. If you only want to know the value of index, there is no need to assign a variable to array and thisArg.

Example 5

Example 5 is the same as Example 1, but in this example, we have assigned variables to each of the optional values.

The variable **record** = current value
The variable **index** = index
The variable **originalArray** = array
The variable **thisArg** = thisArg

We send each of these values to the anonymous function on each iteration. In our example, we still have **console.log** print the value of the record variable to the screen, but you can change it to index, originalArray, or thisArg to see those values.

Example 6

Example 6 is the same as Example 2, but with two of the three additional values included. Notice that like Example 5, we could change **console.log** to print the values of index or originalArray to the screen, but to have access to the thisArg value, you would need to assign it before the arrow function.

Example 7

Example 7 is the same as Example 4. Once again, we split the function and the forEach method into two parts, and we also add two of the three optional variables to the list of information sent to the function on each iteration. If you want to use the **thisArrg** value in this situation, you will need to declare it before the arrow function in the forEach method and in the parameters of the processRecord function.

Conclusion

Hopefully, after reading over these examples, you have a better handle on the JavaScript Array prototype forEach method. It can be a helpful tool that makes life easier. There are some downsides, though, and one of the biggest is that you can't stop it once It starts. So, for instance, if you had an array with several hundred elements, you could not print out ten at a time; you would have to print them all.

If you have learned something new, and this explanation of the JavaScript Array prototype forEach method has been helpful, please share it on Facebook and Twitter.