**ITERATORS**

Introduction to Iterators

One of the most common tasks a developer translates into code is looping over the contents of an array. Programmers often accomplish this with a for loop. However, as is often the case when a task occurs frequently, JavaScript now provides methods that simplify this task.

These methods, called *iterators*, are called on arrays and complete such tasks as altering each element and selecting elements that fit certain criteria. In this lesson, you will learn the syntax for these methods, their return values, how to use the documentation to understand them, and how to choose the right iterator method for a given task.

# .forEach()

The first iterator method that we're going to learn is .forEach(). Aptly named, .forEach() will execute the same code on each element of an array.

let groceries = ['whole wheat flour', 'brown sugar', 'salt', 'cranberries', 'walnuts']; groceries.forEach(function(groceryItem)

{ console.log(' - ' + groceryItem); });

The code above will log a nicely formatted list of the groceries to the console. Let's explore each bit of syntax.

1. The first line is an array of grocery items.
2. groceries.forEach calls the .forEach() method on the groceries array.
3. (function(groceryItem) { creates a function that takes a single parameter, groceryItem and opens the block of code for that function. Because .forEach() is an iterator method, every element in the groceries array will be passed to this function as an argument in place of groceryItem. Syntactically, the name of the parameter does not matter. However, it is a best practice to give parameters descriptive names so that other developers who read your code can easily understand what it does.
4. console.log(' - ' + groceryItem); is the code we wish to execute upon each element in the array. Logging the item to the console with a - in front of it makes the elements look like a list as they're printed out.
5. }); closes the function code block and .forEach() method in that order.

We can simplify this code using arrow function syntax.

groceries.forEach(groceryItem => console.log(' - ' + groceryItem));

Both of these examples execute precisely the same process.

There are three important things to know about the .forEach() method.

1. It is an array method. It must be called upon an array.
2. Any changes to the iterated array value won't be updated in the original array.
3. The return value is undefined.

.map()

In the previous exercise, we called the .forEach() method and learned that it returns undefined. It also does not change the array it is called upon. What if we do want to change the contents of the array? We can use .map()!

let numbers = [1, 2, 3, 4, 5];

let bigNumbers = numbers.map(function(number) { return number \* 10; });

1. The first line is an array of numbers.
2. let bigNumbers = numbers.map creates a new array, bigNumbers, in which the returned values of the .map() method will be saved and calls the .map() method on the numbers array.
3. (function(number) { creates a function that takes a single parameter, number, and opens the block of code for that function.
4. return number \* 10; is the code we wish to execute upon each element in the array. This will save each value from the numbersarray, multiplied by 10, to the bigNumbers array.
5. }); closes the function code block and .map() method in that order.

The syntax for .map() is almost the same as the syntax for .forEach(), with one important change. Notice that directly before the function call, the code reads, let bigNumbers =. This is because .map() returns a new array with elements that have been modified by the code in its block. bigNumbers is the new array in which the method will save the values.

.map() can also be written with arrow function syntax.

let numbers = [1, 2, 3, 4, 5]; let bigNumbers = numbers.map(numbers => numbers \* 10);

.filter()

Another useful iterator method is .filter(). Like .map(), .filter() returns a new array. However, .filter() returns certain elements from the original array that evaluate to truthy based on conditions written in the block of the method.

let words = ['chair', 'music', 'pillow', 'brick', 'pen', 'door']; let shortWords = words.filter(function(word) { return word.length < 6; });

1. On the first line, an array is created.
2. let shortWords = declares a new array that will contain the returned elements of the .filter() method.
3. words.filter(function(word) { calls the .filter() method on the wordsarray and creates a function that will take a single parameter, word. Each element in the words array will be passed to this function as an argument.
4. return word.length < 6; is the condition to filter for all elements in the words array that have fewer than 6characters will be added to the shortWords array.
5. }); closes the code block and .filter() method in that order.

.filter() can also be written using arrow function syntax.

let shortWords = words.filter(word => word.length < 6);

Iterator Documentation

There are many additional built-in array methods, a complete list of which is on the [Mozilla Developer Network](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array).

The documentation for each method contains several sections:

1. A short definition
2. A block with the correct syntax for using the method
3. A list of parameters the method accepts or requires
4. The return value of the function
5. An extended description
6. Examples of the method's use
7. Polyfill, Specifications, Browser Compatibility

A list of iterator methods can be found [here](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array#Iteration_methods).

In the instructions below, there are some errors in the code. Use the documentation for a given method to determine the error or fill in a blank to make the code run correctly.

Review: Iterators

You have learned a number of useful methods in this lesson as well as how to use the JavaScript documentation from the Mozilla Developer Network to discover and understand additional methods. Let's review!

* .forEach() is used to execute the same code on every element in an array but does not change the array and returns undefined.
* .map() executes the same code on every element in an array and returns a new array with the updated elements.
* .filter() checks every element in an array to see if it meets certain criteria and returns a new array with the elements that return truthy for the criteria.
* All iterator methods can be written using arrow function syntax. In fact, given the succinctness and the implicit return of arrow function syntax, this is quickly becoming the preferred way to write these types of method calls.
* You can visit the [Mozilla Developer Network](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array" \t "_blank)to learn more about iterator methods (and all other parts of JavaScript!).
* Additional iterator methods such as .some(), .every(), .reduce() perform different functions.