Lesson | Monday

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / Introduction to Looping

Text | Cheat sheet

Over the next several lessons, we are going to dive into a concept that is often very challenging for beginners: **looping**.

A loop is just what it sounds like — a piece of code that loops over and over until a condition is met.

In this lesson, we'll start by covering the basics of `Array.prototype.forEach()`, which we can use to loop through each element in an array. In the next lesson, we'll move on to a more in-depth discussion of `Array.prototype.forEach()`, including ways we can use this kind of loop.

Later, we will explore `for` loops and `Array.prototype.map()`. `for` loops have been around since the beginning of JavaScript — and they are a more traditional way of looping. In fact, `Array.prototype.forEach()` is **syntactic sugar** that's been added to a traditional `for` loop. Syntactic sugar just means syntax that makes a piece of code easier to write. A loop that uses `Array.prototype.forEach()` is really using a `for` loop under the hood. Meanwhile, `Array.prototype.map()`, like `Array.prototype.forEach()`, provides syntactic sugar as well as other functionality that can make looping easier in many situations.

Finally, for the sake of thoroughness, we'll cover `while` and `for...of` loops, which you probably won't use often (and aren't necessary for this section's independent project). However, it's still important to be aware of them.

We'll cover `Array.prototype.forEach()` first because this kind of loop is easier to use — and will likely be one of your main go-tos for looping.

`Array.prototype.forEach()`

`Array.prototype.forEach()` loops over *every* element in the array it is called on. Let's take a look at a simple example. Try running the following in the DevTools console:

```
const languages = ['HTML', 'CSS', 'JavaScript'];
languages.forEach(function(language) {
  alert('I love ' + language + '!');
});
```

The first time through the loop, we'll get an alert that says "I love HTML!". The second time, we'll get an alert that says "I love CSS!". The third and final time through the loop, we'll get an alert that says "I love JavaScript!".

So what exactly is happening? The `Array.prototype.forEach()` calls the callback function we pass into it on every element in our array. Let's break this down.

An `Array.prototype.forEach()` loop takes a function as an argument. Here's the function that is passed into the loop above:

```
function(language) {
  alert('I love ' + language + '!');
}
```

When a function is passed into another function as an argument to be executed later, it is known as a **callback**. Callbacks are a huge part of programming — and you'll often see complex and even confusing examples of callbacks in JavaScript.

We first learned about callbacks when we learned how to use event listeners. Even though we are familiar with event listeners, callbacks can still be confusing and take a while to wrap your head around, so let's review now. Check out this code, which includes two callback functions:

```
function handleClick() {
  console.log("The button has been clicked!");
}

window.addEventListener("load", function() {
  const btn = querySelector("button");
  btn.addEventListener("click", handleClick);
});
```

The `EventTarget.addEventListener()` method takes two arguments: the first is the name of the event, and the second is the callback function that is called on to run when the event happens. Identifying a function as a callback is all about how a function is being used in the code: if it is used as an argument in another function or method call, then that function is a callback.

Notably, whether or not a function is a callback function has nothing to do with how its been defined. For example, in the `window.addEventListener(...)` method call, we use an anonymous function expression as the callback function. In the

`btn.addEventListener(...)` method call, we use a function declaration as the callback function, which we define at the start of the scripts, and then reference by name in the `addEventListener()` method call.

The reason we pass a function into `addEventListener()` is so that the event listener can call the function when the corresponding event happens. On the other hand, we pass a function into `Array.prototype.forEach()` so that this method can call the function for *every* element in the array. Let's look back to our first example to understand this:

```
const languages = ['HTML', 'CSS', 'JavaScript'];
languages.forEach(function(language) {
  window.alert('I love ' + language + '!');
});
```

The `language` parameter in our callback function represents an array element. Because our `languages` array has 3 elements in it, the forEach loop will loop 3 times. For each loop, the `Array.prototype.forEach()` method calls the callback function, passing the value of the current array element into the `language` parameter. Depending on which loop iteration we're on, the value of the current array element will be either `'HTML'`, `'CSS'`, or `'JavaScript'`. That's how our loop creates a separate alert for each language in the array! If this is confusing — don't worry, we'll get a lot of practice with this.

The cool thing here is that the same code that we describe in the callback function is called on every element of the array. This means that our `languages` array could have a *thousand* items, popping up *one thousand* alerts as a result. The users of our website will definitely love that. (All joking aside, you should avoid alerts in your code... they are extremely unpopular for obvious reasons — mainly their obtrusiveness.)

While the above example is not a very useful one, it demonstrates the power of looping. We can execute code on all of the elements of an array instead of just dealing with a single item. As a result, looping is a foundational cornerstone of computer programming and it's one of the most important programming concepts you'll ever learn.

In the next lesson, we'll do a deeper dive into using `Array.prototype.forEach()`. We'll learn how we can use it to modify arrays, add numbers and more.

Previous (/introduction-to-programming/arrays-and-looping/practice-javascript-arrays)
Next (/introduction-to-programming/arrays-and-looping/foreach-loops)

Lesson 17 of 50
Last updated March 24, 2023

disable dark mode

(http://www.epicodus.com)