Lesson Weekend

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

Text

Until now, we've always dealt with one piece of information at a time: one number, one string, one element on a page. However, we often need to group things together when we are coding. In programming, a group of things is often referred to as a **collection**.

In this lesson, we will cover a very common form of collecting items: the **array**. As you read through this lesson, open up your DevTools console and try out the code examples we work through. The angle bracket `>` indicates that we're putting the code into the DevTools console.

Arrays

Arrays are enclosed in square brackets like this:

```
> let array = [];
```

In the example above, we've created an empty array with square brackets and stored it in a variable called `array`. Note that we use `let` here, not `const`. If we start with an empty array, we are almost

certainly going to want to change it. After all, an array isn't very useful if we aren't using it to hold things. Also, we'll find that `const` doesn't stop arrays from being mutated. We'll cover that soon.

We can also create an array that already has items in it. Here's an array that holds the months of the year:

```
> const months = ["january", "february", "march", "april",
"may", "june", "july", "august", "september", "october", "n
ovember", "december"];
```

Why did we use `const` here? Well, presumably the months of the year aren't going to change, and we use `const` to communicate our intentions to other developers. We'll revisit this array in just a moment.

Arrays can hold strings, numbers, variables, and other JavaScript primitives and objects. They can even hold other arrays — or a mix of things. The items in an array are commonly referred to as **elements**. Here are some examples of arrays that hold strings and numbers:

- `[2, 5, 7, 3423, 85, 65]`
- `["e", "p", "i", "c", "o", "d", "u", "s"]`
- `["word", 45, "word2", 123]`

Here's an example of an array that holds a variable:

```
> const variable = "I'm a variable!";
> const things = [variable, "I'm not a variable!"];
> things;
["I'm a variable!", "I'm not a variable!"]
```

Here's an example of an array that holds an expression:

```
> const numbers = [62, 62 / 2];
> numbers;
[62, 31]
```

Here's an array that holds not just strings and numbers, but other arrays:

```
> const mixOfArraysAndPrimitives = ["string", 123, ["anothe
r string", 456], 321, "yet another string", true];
```

The examples above show some of the elements we can put in an array. As we can see, arrays are pretty flexible — we even slipped a boolean into the array above. Arrays are really just containers for holding collections of things — and we can put just about anything into an array. As developers, we can do things to organize arrays as we see fit, but from the perspective of JavaScript, arrays are just a bucket that can hold different kinds of elements.

Let's look at our original empty array again:

```
> let array = [];
```

Using a Method to Add to an Array

What if we want to add things to this array? Arrays are a type of object, so it has properties and methods. There are many different methods that can be called on arrays. One of the most common is `Array.prototype.push()`, which allows you to add new objects to the end of an array.

Let's take a look:

```
> let array = [];
> array.push("hi");
1
> array;
["hi"]
> array.push("there");
2
> array;
["hi", "there"]
```

When we use the `Array.prototype.push()` method on an array, the return value of the method is the new length of the array. In this case, it's `1`. To see how the array has been modified, we can just check the variable `array`.

Using `const` Doesn't Prevent Modifying an Array

What happens if we try to push a new value into our `months` array?

```
> const months = ["january", "february", "march", "april",
"may", "june", "july", "august", "september", "october", "n
ovember", "december"];
> months.push("I'm not a month mwa ha ha");
13
> months;
["january", "february", "march", "april", "may", "june", "j
uly", "august", "september", "october", "november", "decemb
er", "I'm not a month mwa ha ha"];
```

We can see that `const` doesn't stop an array from being modified. `const` just stops variables from being reassigned or redeclared. That works great with primitives. If we change a variable holding the number `8` for instance (such as by trying to add another number to it), we are reassigning the variable.

That's not the case with arrays, though. When we modify an array, we aren't reassigning or redeclaring the variable. It's still the same array even though we've made changes to it. In the case of using `const` with an array, it's kind of like a permit that protects a house from being knocked down or altered (as you'll often see for historic homes). However, you can still change the furniture in the house even though the permit won't allow the house to be replaced. In this analogy, the array is the house and the elements are the furniture.

In JavaScript, you can't do this:

```
> const array = [];
> array = ["hi"];
Uncaught TypeError: Assignment to constant variable.
```

This is like building a house and then knocking it down and replacing it with another one. We assign an empty array to the variable `array` and then we try to reassign the value of `array`. As the error shows, we can't do that with a `const`.

Note the difference here:

```
> const array = [];
> array.push("hi");
1
> array;
["hi"]
```

We didn't knock down the house here. Our array still exists — we just added an item to it. This is like adding a new piece of furniture to the house.

This is an important gotcha about using `const` with arrays. We can still modify the array (whether that means adding or removing items from it) so it's not truly constant. Why use `const` with arrays at all then?

Well, this is when we can use our code to communicate our intentions. **When we use `const` with an array, we're telling other developers that the array *shouldn't* be modified. It also makes clear that you won't be modifying the array yourself.**

When refactoring your code, you should always check to see if any of your variables need to be changed from `let` to `const` or vice versa. This makes your code clearer and easier to read.

Summary

In this lesson, we covered a few of the basics of arrays. By this point, you should know that an array is just a collection that can contain just about anything. We've also covered the `Array.prototype.push()` method, one of the most ubiquitous array methods, and discussed why `const` doesn't work as you might expect with arrays — and when you should use `const` with arrays anyway.

In the next lesson, we'll learn about the `Array.prototype.length` property of arrays. We'll also cover bracket notation, which we can use to look at specific values inside an array. Then, in the lesson after that, we'll cover commonly used array methods.

Previous (/introduction-to-programming/arrays-and-looping/static-versus-instance-with-built-in-js-objects)
Next (/introduction-to-programming/arrays-and-looping/bracket-notation)

disable dark mode

(http://www.epicodus.com)