Lesson | Wednesday

Intermediate JavaScript (/intermediate-javascript) / Test-Driven Development and Environments with JavaScript (/intermediate-javascript/test-driven-development-and-environments-with-javascript) / ES6 Array and Object Destructuring

Text

One of the most popular ES6 features is object and array **destructuring**. Destructuring is the process of taking specific elements from an array or properties from an object and then turning them into variables.

While it's not essential to incorporate this feature in our projects right now, it's a useful convenience. Object destructuring is very common in React, so even if you don't use it much now, it will come up again once we start working with React.

Note, that you will not be expected to incorporate destructuring for this section's independent project. However, you should recognize destructuring out in the wild, and give it a whirl in your projects.

Array Destructuring

Let's say we have an array of numbers that looks like this:

```
const numArray = [1, 2, 3];
```

To get elements from the array and assign them to variables, we might do something like this:

```
const firstElement = numArray[0];
const secondElement = numArray[1];
const thirdElement = numArray[2];
```

We can make this process much simpler with array destructuring:

```
const [firstElement, secondElement, thirdElement] = numArra
y;
```

In this example, we are destructuring the elements of the array into variables. `firstElement` corresponds to the first element of the array, `secondElement` corresponds to the second element, and so on. We could name these variables anything, but we decided to name them something that would help you make more sense of the example. For example, this would also work:

```
const [numberOfDogs, numberOfCats, numberOfChicks] = numArr
ay;
```

Try this out in the DevTools console and check the values of each of the variables.

And with array destructuring, we've reduced three lines of code to one!

Note that we don't have to destructure the entire array. For instance, if we just need the first element, we could do this:

```
const [firstEl] = numArray;
```

The `firstElement` variable is now set to the first element in `numArray`, but there is no `secondElement` or `thirdElement`. If we wanted to get the third element instead, we'd have to provide a comma for every element we don't want to destructure:

```
const [ , , thirdEl] = numArray;
```

Object Destructuring

Object destructuring is both more useful and more common than array destructuring. We can use it to assign specific properties of an object to variables. Here's an example:

```
const obj = {
  color1: "red",
  color2: "blue",
  description: "Information we don't need",
  color3: "yellow",
  anotherProp: "We don't need this info, either"
};
```

Let's say we just want to get the colors from this object. We can destructure them like this:

```
const { color1, color2, color3 } = obj;
```

In this example, we are pulling out the properties from the object and saving them as constants with the same names. Unlike with array destructuring, with objects, we need to create variables that match the name of the property we are destructuring from the object.

Try it in the DevTools console:

```
color1;
> "red"
color2;
> "blue"
color3;
> "yellow"
```

Since we've only destructured the colors, we don't have equivalent variables for `description` or `anotherProp`.

Let's say we want to take this one step further and create a variable for a property, but the variable should have a *different* name from the original property. We can do this:

```
const { color1: red, color3: yellow } = obj;
```

Now we have variables for `red` (which corresponds to the value of the `color1` property) and `yellow` (which corresponds to the value of the `color3` property). This syntax is a little tricky, so let's look at another example.

```
const hero = {
  name: "Spider-Man",
  realName: "Peter Parker"
};

const { realName: secretName } = hero;

secretName; // => "Peter Parker"
```

In this example, we created a new variable called `secretName` and we set its value to the value of the `realName` property. So, `secretName` is equal to `"Peter Parker"`. The syntax that we're following in pseudocode looks like this:

```
// this is pseudocode!
const { objectProperty: newIdentifier } = object;
```

There's a few things to note about this pseudocode syntax:

- `object` represents the object we're destructuring.
- `objectProperty` is the value we want to destructure from the object.
- `newIdentifier` is the new variable we're creating to hold the value of the `objectProperty`.

If this syntax is confusing, or you are worried about being able to remember it, don't worry about either of those. Practice with it a bit, and reference MDN when you find yourself in need of object destructuring to review usage options and syntax.

By the way, the syntax for object destructuring should already look familiar from the import statements we've used with named exports. For instance, as we discussed in the ES6 Imports and

Exports (https://www.learnhowtoprogram.com/intermediate-javascript/test-driven-development-and-environments-with-javascript/es6-imports-and-exports) lesson, we can do the following:

```
import { Triangle, Rectangle, Circle } from './shapes.js';
```

As we can see, import statements that are used with named exports use object destructuring syntax as well.

This lesson covers the basics of array and object destructuring, but there are plenty of other things we can do with destructuring, too! Check out the Mozilla documentation on Destructuring Assignment (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment) to learn more.

Previous (/intermediate-javascript/test-driven-development-and-environments-with-javascript/managing-images-with-webpack)
Next (/intermediate-javascript/test-driven-development-and-environments-with-javascript/es6-maps-and-sets)

Lesson 43 of 49

Last updated more than 3 months ago.

disable dark mode

(http://www.epicodus.com)