# Destructuring with JavaScript

Learn a new syntax for handling objects and arrays in JavaScript.

What is Destructuring?

Destructuring, or destructuring assignment, is a JavaScript feature that makes it easier to extract data from arrays and objects, introduced in the ES6 version of JavaScript.

At this point, we assume you already know how to extract data from arrays and objects. That means that you don’t NEED destructuring: destructuring is a form of syntactic sugar, which means that it is an easier way to write certain expressions, usually by being shorter and clearer than other forms. Even if you don’t use it yourself, you’ll probably see it in someone else’s code, so it’s valuable to understand.

Destructuring Arrays

Data structures like arrays and objects can be very useful for storing large amounts of data. The process of converting individual elements of an array into individual variables can be tedious:

let cars = ['ferrari', 'tesla', 'cadillac'];

If we wanted to access these cars individually and assign them to variables we could do this:

let cars = ['ferrari', 'tesla', 'cadillac'];

let car1 = cars[0];

let car2 = cars[1];

let car3 = cars[2];

console.log(car1, car2, car3); // Prints: ferrari tesla cadillac

We can use destructuring to shorten our code and make it more concise:

let cars = ['ferrari', 'tesla', 'cadillac'];

let [car1, car2, car3] = cars;

console.log(car1, car2, car3); // Prints: ferrari tesla cadillac

In the code above, we created three variables (car1, car2, car3) that correspond to the three elements in the cars array. Each variable name in the new array will be assigned the value of the corresponding element. As you can see, we are able to achieve what we did previously with three lines of code, in one! Imagine how many lines of code we would save working with an array with 10 elements!

# Destructuring Objects

We can also use destructuring assignment with objects. Let’s look at a basic use case in which we capture an object’s properties with new variables:

let destinations = { x: 'LA', y: 'NYC', z: 'MIA' };

let x = destinations.x;

let y = destinations.y;

let z = destinations.z;

console.log(x, y, z); // Prints LA NYC MIA

With the simplified destructuring syntax, we access the values by matching the variable names to the property names.

let destinations = { x: 'LA', y: 'NYC', z: 'MIA' };

let { x, y, z } = destinations;

console.log(x, y, z); // Prints LA NYC MIA

Using destructuring syntax, we’re able to create new variables directly from an object’s properties. In this case, we took the values of destination.x, destination.y, and destination.z and stored them in new variables x, y, z, respectively.

# Destructuring Function Parameters

Function arguments are another place where destructuring is useful. Instead of accepting a complete object as an argument, a function can use destructuring to capture specific properties as named parameters.

let truck = {

model: '1977 Mustang convertible',

maker: 'Ford',

city: 'Detroit',

year: '1977',

convertible: true

};

const printCarInfo = ({model, maker, city}) => {

console.log(`The ${model}, or ${maker}, is in the city ${city}.`);

};

printCarInfo(truck);

// Prints: The 1977 Mustang convertible, or Ford, is in the city Detroit.

The printCarInfo function uses object destructuring to create three parameter variables: model, maker, and city. When the function is invoked with the truck object, these parameters are assigned the corresponding values from that object.

Note that we don’t have to use every property from the truck object: we only create parameter variables for the values that we need.

Conclusion

Destructuring is a way to unpack values from arrays and objects and assign them to variables or parameters. There are even more cool things you can do with destructuring: renaming variables, assigning default values, ignoring values, etc. We encourage you to check out the MDN documentation and experiment. Happy Coding!

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring\_assignment