JavaScript for App Development

By Mark Lassoff, Founder Framework Television

Destructuring

After completing this section you will:

- ☐ Create simple and complex JavaScript objects
- ☐ Destructure Simple Objects
- ☐ Identify and Use the Destructuring Operator
- ☐ Destructure Multi-level, Multi-node Complex Objects

Introduction

Object destructuring allows you to easily access the properties of an object without using the awkward traditional dot notation. Object destructuring also allows you to access multiple properties at the same time allowing developers to be less verbose when working with objects.

Destructuring Simple Objects

Simple objects generally have a list of properties stored in key/value pairs. In this section, we'll create and then destructure a simple object.

Creating The Object

Obviously, to demonstrate destructuring we'll need an object to work with. Let's create a simple object representing a person.

```
<script>
let person = {
   first: "Mark",
   last: "Lassoff",
   age: 44,
   city: "Norwalk",
   state: "Connecticut"
}
</script>
```

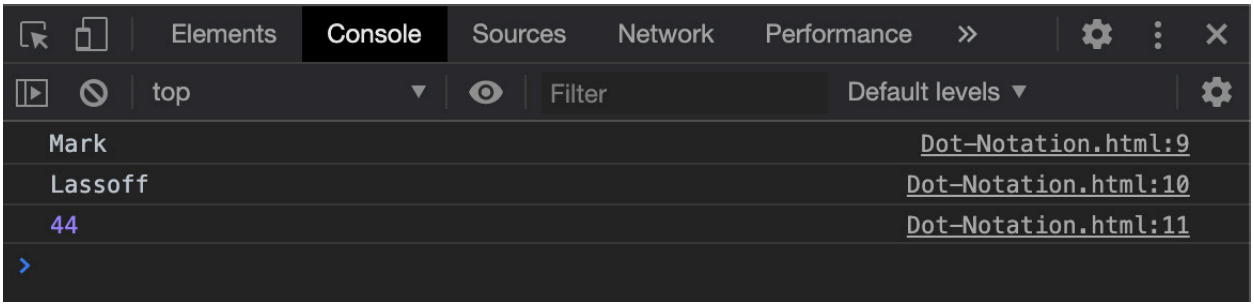
We now have an object with five properties describing a person, represented in JavaScript.

Dot Notation

Traditionally if we wanted to access properties of an object, we'd use the dot notation to access each property. For example, using the person object above, we could use the following to access individual properties and output them to the console:

```
console.log(person.first);
console.log(person.last);
console.log(person.age);
```

Executing the code would cause the value of the `first`, `last`, and age properties of the `person` object to appear in the console log.



Destructuring an Object

Destructuring will allow us to access object properties differently. Let's keep the object we created above and destructure it with the following code:

```
<script>
let person = {
   first: "Mark",
   last: "Lassoff",
   age: 44,
   city: "Norwalk",
   state: "Connecticut"
}

let { first, last, age } = person;
console.log(first);
console.log(last);
console.log(age);

</script>
```

This code would yield the same result in the console as the code above, however, this time, access to object properties was completed by destructuring the object.

Here is the line that does the magic:

```
let { first, last, age } = person;
```

On the right side of the assignment operator is the object we're accessing. In this case, it's the `person` object that we've been working with all along. On the right side of the operator, is the `let` command, our assignment statement and then the names of the properties we're accessing.

Note: The name of the properties must be consistent in the object and destructuring statement.

The names of the properties are converted into variables holding the value of the property from the object. These variables can be accessed in the same way as any other variable in JavaScript.

For example, if we wanted to access the `state` property from the `person` object we could use the following code:

```
let { state } = person;
console.log(state);
```

When the code above executes `state` is now a variable holding the value `Connecticut`.

Destructuring Operator

In this context, the curly brackets act as the destructuring operator. Keep in mind that the curly brackets (sometimes called French brackets) are an overloaded operator and have many different roles in JavaScript. You can only determine how their being used by the context created by the code in which they are being used.

Destructuring Complex Objects

Complex objects have multiple levels of objects and child objects encapsulated within the object. Imagine a weather forecast represented by an object. The forecast might have several child nodes representing each day included in the forecast.

Creating the Complex Object

Let's create a complex object that we can work within JavaScript:

```
let company = {
   name: "Framework Television",
   social: {
      twitter: "https://twitter.com/fw_social",
      facebook: "https://facebook.com/
facebooktechnicallyspeaking"
   }
}
```

This object can be considered complex because to access the `twitter` and `facebook` properties you must go through both the `company` and `social` nodes in the object.

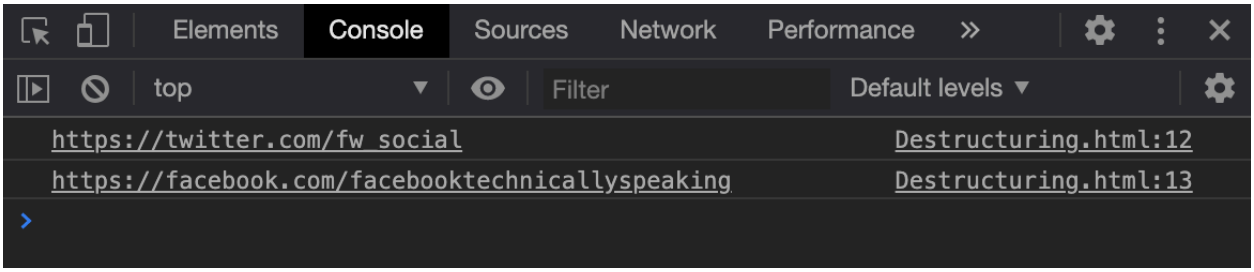
Destructuring the Complex Object

Destructuring a complex object is similar to destructuring a simple object. Here's the code to destructure our `company` object:

```
let {twitter, facebook} = company.social;

console.log(twitter);
console.log(facebook);
```

The result of this code displayed in the console:



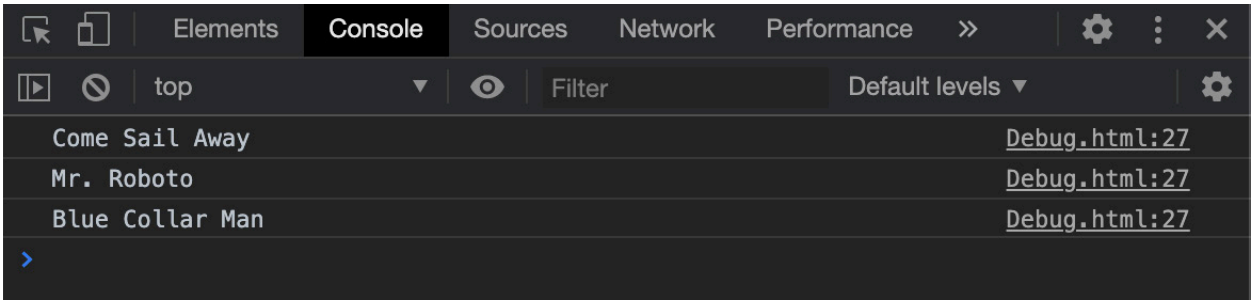
Notice that on the right side of the assignment operator, we used the dot notation to access the `social` node of the object and then destructured its properties.

Debug This:

```
<script>
let catalog =  {
  "artists": [
    { "name":"Journey"
     "hits":[ "Faithfully",
              "Only the Young",
              "Dont Stop Believing" ] },
    { "name":"REO Speedwagon"
     "hits":[ "Keep On Loving You",
              "Time for Me to Fly",
              "Cant Fight This Feeling" ] },
    { "name":"Styx"
     "hits":[ "Come Sail Away",
              "Mr. Roboto",
              "Blue Collar Man"] }
    ]
 };

let { hits } = catalog.artists[x];
for(x in hits){
   console.log(hits[x]);
}
</script>
```

When the code above is correctly debugged the console will appear as below after code is executed.



Submit This

Consider the following JavaScript object:

```
var userList = {
  "people": [

    {  firstName: "Fred",
       lastName: "Smith",
       dateOfBirth: 1980,
       spokenLanguages: {
          native: "English",
          fluent: "Spanish",
          intermediate: "Chinese" }
    },
    {  firstName: "Monica",
       lastName: "Taylor",
       dateOfBirth: 1975,
       spokenLanguages: {
          native: "Spanish",
          fluent: "English",
          intermediate: "French" }
    },
    {  firstName: "Maurice",
       lastName: "Edelson",
       dateOfBirth: 1992,
       spokenLanguages: {
          native: "English",
          fluent: "Spanish",
          }
```

```
    },
    {   firstName: “Kelly”,
        lastName: “Lang”,
        dateOfBirth: 1982,
        spokenLanguages: {
            native: “English”,
            fluent: “German”,
            intermediate: “Dutch” }
    }
]
};
```

Using object destructuring, output an HTML table of each employees name and date of birth, along with the languages that they speak. Make sure your table is well constructed and easy to consume for the user.