Lesson Weekend

Intermediate JavaScript (/intermediate-javascript) / Object-Oriented JavaScript (/intermediate-javascript/object-oriented-javascript) / Objects Within Objects

Text Cheat sheet

Let's take a brief break from the address book project we just started to learn how to create an object that contains another object. Then we'll use this to build out our address book in the next lesson.

Note that we've worked with nested objects a lot already by using Web APIs. Remember that a **nested object** is simply an object that is a property of another object. Or, an object that contains another object. Here are some examples that should be familiar to you:

```
> window.document;
#document
> window.location;
Location {ancestorOrigins: DOMStringList, href: 'https://ww
w.learnhowtoprogram.com/introduction-to-…/getting-started-a
t-epicodus/learn-how-to-program', origin: 'https://www.lear
nhowtoprogram.com', protocol: 'https:', host: 'www.learnhow
toprogram.com', …}
> document.body;
<body>...</body>
```

At their most basic, Web APIs are a bunch of different object types, and Web API structures like the DOM and the browser window are a series of nested objects.

With this in mind, let's create our own object with a nested object inside. You are welcome to just read through this lesson, or code along. All of the following examples can be tested in the DevTools console.

Objects within Objects

Very often, we'll want to create associations between objects. For example, we may want to know a list of cities in a particular country. Here's how we can do this. First, let's create a series of cities:

```
> let pdx = { name: "Portland" };
> let sfo = { name: "San Francisco" };
> let sea = { name: "Seattle" };
> let cso = { name: "Aktau" };
> let dzn = { name: "Zhezkazgan" };
```

Then, let's create some countries. Note that each country below has a `cities` property that contains an array of city objects, the ones we created in the last code snippet.

```
> let usa = { name: "United States of America", cities: [pd
x, sfo, sea] };
> let kazakhstan = { name: "Kazakhstan", cities: [cso, dzn]
};
> let uruguay = { name: "Uruguay", cities: [] };
```

Let's take a look at the properties of `usa` — and also how we can get to the `name` property of the city `pdx` within the country `usa`:

```
> usa.name;
"United States of America"
> usa.cities;
(3) [{...}, {...}, {...}]
  0: {name: "Portland"}
  1: {name: "San Francisco"}
  2: {name: "Seattle"}
  length: 3
  [[Prototype]]: Array(0)
> usa.cities[0];
{name: "Portland"}
> usa.cities[0].name;
"Portland"
```

As we can see, all of the pieces of information stored inside an object's properties are accessible. However, accessing more deeply nested properties is often a huge point of confusion for beginners. In the example above, we can see that `usa.cities` is the array of cities we created. We can use bracket notation to get the first city `pdx` from that array — after all, it works just like any other array. We can also loop through that array:

```
> usa.cities.forEach(function(city) {
  console.log("Let's go to " + city.name + "!");
});
Let's go to Portland!
Let's go to San Francisco!
Let's go to Seattle!
```

We can also add cities to a country after the object is initially created:

```
> let mlz = { name: "Melo" };
> uruguay.cities.push(mlz);
```

`uruguay.cities` returns an array, and then we chain `.push(mlz)` to add the new city object onto the array.

When retrieving more deeply nested properties, it's important to note the *type* of object you are working with.

Accessing A More Complicated Object and Indentation

Let's look at a slightly more confusing object that holds a variety of objects, arrays, and strings:

```
> let boxOfStuff = {
    book: "Object-Oriented JavaScript",
    smallerBox: {
      stuff: ["pencils", "pens"],
      smallestBox: {
        stuff: ["paper clips", "thumbtacks"]
      }
    }
  };
```

While we could write out this entire nested object in a single line, it's much more readable using indentation. Each layer of nesting results in an extra level of indentation with the property `stuff` of the most nested object `smallestBox` being the most indented.

See if you can access `"pens"` and `"paper clips"` in the console before moving on.

Now let's take a look at how we can access different things inside our jumbled `boxOfStuff`.

Getting the book is easy:

```
> boxOfStuff.book;
"Object-Oriented JavaScript"
```

We can also use bracket notation to retrieve this property, though you won't see this as often:

```
> boxOfStuff["book"];
"Object-Oriented JavaScript"
```

Note that the property has to be in string form to do this. `boxOfStuff[book]` will not work, unless you are referencing a variable called `book`, like so:

```
> const book = "book";
> boxOfStuff[book];
"Object-Oriented JavaScript"
```

We recommend using dot notation, not bracket notation, for grabbing object properties, but there are cases when you'll have to use bracket notation, like when you are working with a variable or a string with special characters in it (this second case is less likely to come up, but does).

Now that we've gotten the book out of the box, let's see how we can get out the `"pens"` and `"paper clips"` so we can make notes in the book. Here's how we can grab the pens:

```
> boxOfStuff.smallerBox.stuff[1];
"pens"
```

It's a bit tricky because `pens` is inside an array saved to the `stuff` property, which is a property of `smallerBox`, which is itself an object inside of `boxOfStuff`. We have to deal with both objects and arrays to get our pens.

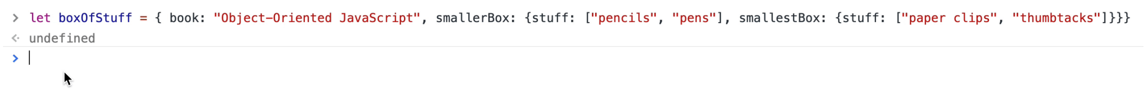
Now let's get the paper clips:

```
> boxOfStuff.smallerBox.smallestBox.stuff[0];
"paper clips"
```

We have to go another level deeper to retrieve the `smallestBox` object, which is a property of the `smallerBox` object.

Note that the `stuff` inside `smallestBox` is a completely different property than the `stuff` inside `smallerBox`. We have two different objects that both have a property named `stuff`. They are different properties, though, just like two people named Jane are different people.

If getting properties from this object was a bit confusing, that's okay. It was meant to be a confusing object. You'll find plenty of other confusing objects out in the wild, though! Don't forget that you can always explore an object one level at a time in the DevTools console:



```
> let boxOfStuff = { book: "Object-Oriented JavaScript", smallerBox: {stuff: ["pencils", "pens"], smallestBox: {stuff: ["paper clips", "thumbtacks"]}}}
< undefined
> |
```

In the GIF above, we use the console to navigate through `boxOfStuff`, gradually getting closer to the array that contains `"paper clips"`.

An Even More Complex Object

Take note that objects can have many nested objects, as many as the design of the data calls for. Let's look at one more example of objects within objects. This is another tricky one, which is by design.

First, let's create a grocery store with veggies for sale:

```
> let tomatoes = { name: "Tomatoes", price: 2.99 };
> let cucumbers = { name: "Cucumbers", price: 0.99 };
> let onions = { name: "Onions", price: 0.79 };
> let groceryStore = { name: "Michael's corner market", pro
ducts: [tomatoes, cucumbers, onions] };
```

Then, a phone store with phones for sale:

```
> let iPhone = { name: "iPhone", price: 699 };
> let android = { name: "Android", price: 499 };
> let windowsPhone = { name: "Windows Phone", price: 399 };
> let phoneStore = { name: "RadioShack", products: [iPhone,
android, windowsPhone] };
```

Then, we'll track the grocery store and phone store together in a `stores` variable, and use a loop to print what each store sells in the console:

```
> let stores = [groceryStore, phoneStore];
> stores.forEach(function(store) {
  console.log(store.name + " sells:");
  store.products.forEach(function(product) {
    console.log(product.name);
  });
  console.log("\n");
});
```

We have several layers of nesting here. We have `stores`, which contains an array with both `groceryStore` and `phoneStore`. In turn, each of those stores has its own `products` property which contains an array of objects — vegetables or phones.

In the first loop, we display the store's name, and then for each store, we loop *again* through the products it contains, displaying their names. Note that `"\n"` creates a new line. (A new line is the equivalent of hitting `Enter` start a new line).

As you can see, we can do a lot with objects contained inside other objects. Working with properties and deeply-nested objects can be challenging at first. However, it's an important part of JavaScript, and Web APIs for that matter!

In fact, working with JavaScript objects like this is so common in the industry that **JSON (JavaScript Object Notation)** is a standard form of sharing data across *many* programming languages, not just JavaScript. We'll work more with JSON later — but for now, just be aware that being comfortable with navigating through JavaScript objects is an absolutely essential skill — and not too hard to master with a little practice.

Previous (/intermediate-javascript/object-oriented-javascript/accessing-code-from-different-branches)
Next (/intermediate-javascript/object-oriented-javascript/address-book-objects-within-objects)
Lesson 9 of 33
Last updated March 23, 2023

disable dark mode

Epicodus (http://www.epicodus.com)