Lesson Tuesday

Introduction to Programming (/introduction-to-programming) / JavaScript and Web Browsers (/introduction-to-programming/javascript-and-web-browsers) / Accessing HTML Element Attributes and Properties in the DOM

Text Cheat sheet

In this lesson, we're going to learn how to access and change the attributes of HTML elements in the DOM. We'll work with specific examples for the `id`, `class` and `style` attributes, but all of the examples we cover can be applied to any attribute. We'll also learn about the `innerText` property of DOM elements, which allows us to change the text of an element, like a `<p>` paragraph element, and the read-only `tagName` property that tells us the tag name of the HTML element we have accessed.

Also, **this is the first lesson that we'll start to follow the new convention of omitting `window` when we access `window` properties.** For example, instead of this:

```
> let h1 = window.document.querySelector("h1");
```

We'll code like this:

```
> let h1 = document.querySelector("h1");
```

For this lesson's examples, we'll continue to work with the Cookie Recipe HTML, which we've pasted below. The code snippets are meant to be used in the DevTools console, and you can just read through this lesson or practice in the console while you read. Note that there will be an upcoming practice lesson to put all of these new tools to use.

cookie-recipe.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title> Best Chocolate Chip Cookies </title>
  </head>
  <body>
    <h1 id="specialHeader">Best Chocolate Chip Cookies</h1>
    <img src="https://static01.nyt.com/images/2022/02/12/di
ning/JT-Chocolate-Chip-Cookies/JT-Chocolate-Chip-Cookies-ar
ticleLarge.jpg" alt="An image of a cookie"/>

    <p>This recipe is from my dad and they are a favorite a
mong friends and family. The secret ingredient is the cocon
ut! <em>Be warned</em>, these will fly off of the plate!</p
>

    <h2 class="h2styles">Ingredients</h2>
    <ul class="ul-style">
      <li>Butter</li>
      <li>White sugar</li>
      <li>Brown sugar</li>
      <li>Eggs</li>
      <li>Vanilla</li>
      <li>Flour</li>
      <li>Baking soda</li>
      <li>Salt</li>
      <li>Chocolate chips</li>
      <li>Oatmeal</li>
      <li>Coconut</li>
    </ul>

    <h2 class="h2styles">Directions</h2>
    <ol class="ol-style">
      <li>Preheat the oven to 325.</li>
      <li>Beat the butter, sugar, eggs and vanilla together
until creamy.</li>
      <li>Mix together the flour, baking soda and salt in a
separate bowl.</li>
      <li>Add flour mixture to butter mixture slowly.</li>
      <li>Stir in chocolate chips, oatmeal and coconut.</li
```

```
>
      <li>Bake for <strong>10 minutes</strong> or until gol
den brown.</li>
    </ol>

    <p><a href="http://allrecipes.com">Click here</a> to ch
eck out my other great recipes.</p>
  </body>
</html>
```

Accessing DOM Element Attributes

Once we've accessed an element with `document.querySelector()` or `document.getElementById()`, what exactly are we working with? I mean, what data type are we working with? Let's find out:

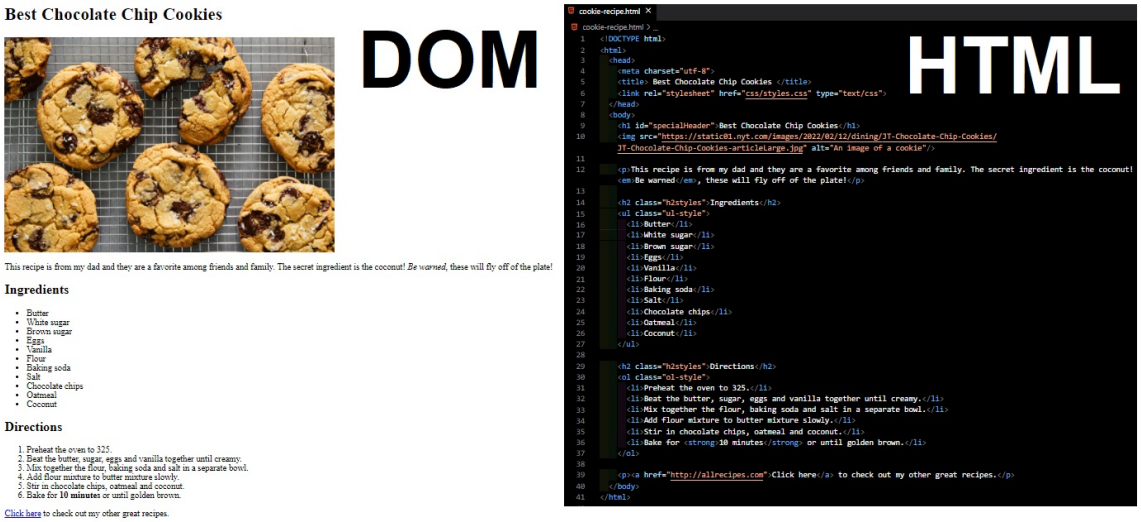
```
> let h1 = document.querySelector("h1");
> h1;
<h1 id="specialHeader">Best Chocolate Chip Cookies</h1>
> typeof h1;
object
```

We're working with an object! In fact, every element we access in the DOM will be an object. Like all objects, the DOM element objects are a collection of properties. This means if we want to access the `id` attribute of our H1 element, or even the text "Best Chocolate Chip Cookies", we need to access the right property or use the right method for the DOM element to get the data we're interested in.

Language Choices

Before we dive into accessing DOM element properties, we should pause to review the terminology we'll be using in this lesson so that there's less confusion.

First, we should remind ourselves that the HTML elements we write in VS Code are just that — HTML. This is in contrast to the DOM elements in the browser, which are all objects that happen to represent HTML elements within the DOM. The Document Object Model (DOM) is the interactive representation of our HTML document that we can use JavaScript to access and modify. As the name suggests, it's an object model of our HTML, meaning that the DOM is a series of nested objects that are arranged in a hierarchy. So, when we reference "**HTML element**", we're referring to an element in our HTML (generally speaking or specifically in our source code). When we reference "**DOM element**" or "**HTML DOM element**", we're referring to an object in the DOM that represents an HTML element.



Accessing DOM Element Properties to Change `id` and `class` Attributes

Let's use our Cookie Recipe's H1 element for our first example. The first step is to get that DOM element. In the following code snippet we've used the `document.querySelector()` method to get the H1 element and save it to a variable called `h1`.

```
> let h1 = document.querySelector("h1");
> h1;
<h1 id="specialHeader">Best Chocolate Chip Cookies</h1>
```

Every HTML DOM element has properties that let us access its attributes. Here's how we access the value of the `id` attribute:

```
> h1.id;
"specialHeader"
```

As we can see the `id` attribute of the H1 DOM element can be accessed by a property itself called `id`. Remember that all of the DOM elements that are returned from calling `document.querySelector()` or `document.getElementById()` are objects, and like all objects, they are a collection of properties.

If we want to access the `class` attribute of our H1 element, we'll need to access the `className` property of the DOM element. Check it out:

```
> h1.className;
""
```

You may have noticed that we don't actually have a `class` attribute on our H1 element in our Cookie Recipe. Because of this, when we access it, we get an empty string returned.

We can create a `class` attribute for our DOM element, and set a value by using the assignment operator `=`:

```
> h1.className = "cookies";
"cookies"
> h1;
<h1 id="specialHeader" class="cookies">Best Chocolate Chip
Cookies</h1>
```

As we can see, when we access our `h1` variable to see the value of our H1 DOM element, we now have a `class` attribute with a value of "cookies". Pretty cool!

Before we move on, **it's important to note that any changes that we make to the DOM elements do not correspond to permanent changes in our HTML source code. Only the DOM gets updated.** Remember, the DOM is the interactive representation of our HTML document (source code) displayed in our webpages.

If we wanted to remove the class, we can do this:

```
> h1.className = "";
""
> h1;
<h1 id="specialHeader" class>Best Chocolate Chip Cookies</h
1>
```

As we can see here, setting the `className` property of our DOM element to an empty string removes the value from our `class` attribute, but it doesn't remove the attribute itself. Later in this lesson, we'll learn how to use the `Element.removeAttribute()` method to completely remove the attribute.

Everything that we learned about so far with accessing and setting the value of attributes can be applied to any DOM element, like paragraphs, unordered lists, or images. This means that every DOM element has properties that represent its attributes. In other words,

every attribute that can be applied to an element in our HTML source code is represented by a property on the corresponding DOM element. Let's look at one more example, this time of an HTML image element. This HTML element is from our Cookie Recipe:

```
<img src="https://static01.nyt.com/images/2022/02/12/dinin
g/JT-Chocolate-Chip-Cookies/JT-Chocolate-Chip-Cookies-artic
leLarge.jpg" alt="An image of a cookie"/>
```

And it corresponds to this DOM element object:

```
> let img = document.querySelector("img");
> img;
<img src="https://static01.nyt.com/images/2022/02/12/dinin
g/JT-Chocolate-Chip-Cookies/JT-Chocolate-Chip-Cookies-artic
leLarge.jpg" alt="An image of a cookie">
```

With these properties, representing the HTML element's attributes:

```
> img.src;
"https://static01.nyt.com/images/2022/02/12/dining/JT-Choco
late-Chip-Cookies/JT-Chocolate-Chip-Cookies-articleLarge.jp
g"
> img.alt;
"An image of a cookie"
```

Accessing and Setting Inline Styles with DOM Element `style` Property

Let's now explore adjusting the `style` attribute on our HTML elements. We'll do this by accessing the DOM element `style` property. Keep in mind that the `style` attribute is only for inline

styles, and does not access or update any CSS file. We'll continue to use our H1 element in these examples:

```
> let h1 = document.querySelector("h1");
> h1;
<h1 id="specialHeader">Best Chocolate Chip Cookies</h1>
```

Just like with the `id` and `className` properties, we can access inline styles like this:

```
> h1.style;
CSSStyleDeclaration {accentColor: '', additiveSymbols: '',
alignContent: '', alignItems: '', alignSelf: '', …}
```

When we access `h1.style`, we get a really big object called `CSSStyleDeclaration` returned to us. Unlike the DOM element `id` and `className` properties that have a string as their value, the `style` property of a DOM element has an object as its value, the `CSSStyleDeclaration` object.

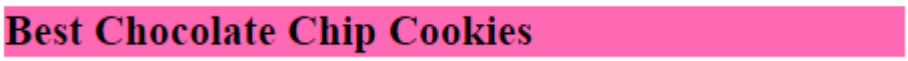
The `CSSStyleDeclaration` object is actually a part of the CSSOM — the CSS Object Model (https://developer.mozilla.org/en-US/docs/Web/API/CSS_Object_Model). The CSSOM is a bit beyond the scope of this lesson and this section's material, so we won't be spending too much time on it. However, a short introduction to the CSSOM is helpful context: just like our HTML is turned into the DOM, our CSS is also turned into an object model by our browsers, and this allows developers to write code to manipulate CSS. We'll learn about the CSSOM in a later course section, so it's not important understand this information now. We just need to understand how to set inline styles, so let's continue!

You can expand the `CSSStyleDeclaration` object in the console by clicking the triangle icon to the left of the object's name. This object shows all of the different styles that we can add inline to our DOM element.

This is a big object with lots of unfamiliar properties. However, there's one I'm certain that you'll recognize: `backgroundColor`. See if you can find that property in the image above or in your DevTools console. We can use the `backgroundColor` property to set the background color of a DOM element. Let's do that now:

```
> h1.style.backgroundColor = "hotpink";
"hotpink"
```

In this example, we've accessed the variable `h1` representing our HTML DOM element, then we've accessed the DOM element's `style` property (which itself is an object), and then we've accessed the `style` object's `backgroundColor` property to assign it a value of `"hotpink"`. And what do we see in the browser? We've changed the background color of our H1 element to hot pink. Neat!



With the code `h1.style.backgroundColor = "hotpink";`, we've set the inline style for the CSS background color. If we did this in our CSS stylesheet `styles.css` instead, the code would look like this:

```
/* styles.css */

h1 {
  background-color: hotpink;
}
```

Notice that the `background-color` CSS property is hyphenated in our CSS stylesheet, but in the DOM element's `style` object that same property is not hyphenated, but instead follows lower camelCase naming: `backgroundColor`. Also, notice how the values of every `style` object property are set to strings when we're setting inline styles for a DOM element. Let's look at a few more examples. Let's change the text color and size of our H1 element:

```
> h1.style.color = "olive";
"olive"
> h1.style.fontSize = "72px";
```

When we view the H1 element, we'll see that all of the inline styles we've added are listed in the `style` attribute of the DOM element:

```
> h1;
<h1 id="specialHeader" style="background-color: hotpink; co
lor: olive; font-size: 72px;">Best Chocolate Chip Cookies</
h1>
```

To remove an inline style, we have two options: setting the CSS property to `""` or to `null`

```
> h1.style.color = "";
> h1.style.fontSize = null;
> h1;
<h1 id="specialHeader" style="background-color: hotpink;">B
est Chocolate Chip Cookies</h1>
```

Before we move on, it's important to reiterate that we can't access any styles for any DOM element listed in an external CSS stylesheet (like `styles.css`) through the DOM element `style` property. Instead we can only access CSS properties that are listed as inline styles either in your HTML source code or ones that you set via the DOM.

`innerText` Property and Read-Only `tagName` Property

Next, let's look at two more properties that belong to (most) DOM elements: `innerText` and `tagName`. Can you guess what information we can get from these properties? We'll continue to use the H1 element for the next examples.

innerText

Check out the following example:

```
> let h1 = document.querySelector("h1");
> h1.innerText;
"Best Chocolate Chip Cookies"
> h1.innerText = "The Very Best Chocolate Chip Cookies";
> h1;
<h1 id="specialHeader">The Very Best Chocolate Chip Cookies
</h1>
```

As we can see in the code snippet above, with the `innerText` property we can retrieve and set the value of an HTML DOM element's text.

tagName

Check out the following example:

```
> let h1 = document.querySelector("h1");
> h1.tagName;
"H1"
> let body = document.querySelector("body");
> body.tagName;
"BODY"
> let ul = document.querySelector("ul");
> ul.tagName;
"UL"
> let firstLi = document.querySelector("ul>li");
> firstLi.tagName.toLowerCase();
"li"
```

As we can see in the code snippet above, the `tagName` property of a DOM element returns the name of the HTML element, in all caps. The very last line of code `firstLi.tagName.toLowerCase();` reminds us that we can call JavaScript methods on DOM elements. Any JavaScript method for a data type (like strings) can be applied to the DOM element property of the same data type (like `tagName`, whose value is a string).

Take note that `tagName` is a read-only property, which means we can't use the assignment operator `=` to give the `tagName` property a new value.

Chaining Methods and Properties Accessors

Let's look at a more complex example of chaining methods and property accessors. By the way, a **property accessor** is just a way to access an object property, and the one we know about is called **dot notation**. Let's look again at the final example in the last code snippet:

```
> let firstLi = document.querySelector("ul>li");
> firstLi.tagName.toLowerCase();
"li"
```

If our goal is to get the lower cased tag name, but not the list item DOM element object, we can rewrite the above code on one line:

```
> let firstLiTagName = document.querySelector("ul>li").tagN
ame.toLowerCase();
> firstLiTagName;
"li"
```

Code gets executed from the left to the right, which means for this line of code `let firstLiTagName = document.querySelector("ul>li").tagName.toLowerCase();`:

- First, I am creating a new variable called `firstLiTagName` with `let`.
- Then, I am assigning the variable a value.
- For the value, I am:
  - Accessing the `document` object.
  - Calling a `document` method to access an element in the DOM.
  - Accessing the `tagName` property on the HTML Element that's returned from the `document.querySelector("ul>li")` call.
  - And finally calling the `String.prototype.toLowerCase()` method on the value of the `tagName` property, which I know is a string.

You don't have to write code in one line like in the above example. Do whatever you are comfortable with, and whatever is easiest for you to understand. Using one line of code does not always make your code easier to read and understand!

Accessing DOM Element Attributes with Property Methods

So far, we've used dot notation to access DOM element properties to get and set our DOM element attributes. However, we can also use methods to get the value of, set the value of, check the existence of, and remove attributes:

- `Element.getAttribute()` — gets the value of an attribute of a DOM element by the name of the attribute.
- `Element.setAttribute()` — sets the value of an attribute on a DOM element by the name of the attribute and the specified new value.
- `Element.hasAttribute()` — returns a boolean based on whether or not a DOM element has an attribute by the name specified.
- `Element.removeAttribute()` — removes an attribute from a DOM element by the name of the attribute.

Take note that `Element` is the name of the object that the above methods belong to. We'll learn more about the `Element` object in the next lesson. For now know that `Element` represents the HTML DOM element that we happen to be working with, like our H1 element from the Cookie Recipe that we've been using as an example.

Let's check out some examples using our H1 element now:

```
> let h1 = document.querySelector("h1");
> h1;
<h1 id="specialHeader">Best Chocolate Chip Cookies</h1>
> h1.getAttribute("id");
"specialHeader"
> h1.setAttribute("class", "coolStyles");
> h1;
<h1 id="specialHeader" class="coolStyles">Best Chocolate Ch
ip Cookies</h1>
> h1.hasAttribute("id");
true
> h1.hasAttribute("style");
false
> h1.removeAttribute("class");
> h1;
<h1 id="specialHeader">Best Chocolate Chip Cookies</h1>
```

Since we are familiar with calling methods, these examples should be easy to understand. There's a couple things to point out:

- Notice that all of the arguments that we pass into these methods are strings.
- Notice that the `setAttribute()` method takes two arguments, one for the attribute name, and the other for the value we want to assign to that attribute.
- Also notice that when we want to target the `class` attribute with these methods, we use the spelling `class` and not `className`. In fact, `className` is the spelling used only for the HTML DOM element property representing the `class` attribute.

In an upcoming lesson, we'll put what we've learned into practice!

Summary

In this lesson we learned how to access HTML DOM elements to get and set the values of their attributes:

- In the DOM, the HTML elements from our source code are transformed into HTML DOM element objects. Like all objects, an HTML DOM element is a collection of properties. These properties represent things like:
  - The HTML tag name.
  - The inner text of an element, if there is any. For example, a heading element will have inner text, but an image element won't.
  - Any attributes associated with the HTML element, including inline styles with the `style` attribute.
- We can access the attributes of a DOM element by accessing a property or by calling a method on it.

Previous (/introduction-to-programming/javascript-and-web-browsers/javascript-s-global-object)
Next (/introduction-to-programming/javascript-and-web-browsers/understanding-web-apis-interfaces-object-types-and-inheritance)