Lesson Weekend

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / Document Query Methods that Return Collections

Text Cheat sheet

Since we're now familiar with arrays, let's review `document` query methods that return collections. An array is a type of collection, but in this case we're using the term collection to describe two array-like objects: `NodeList` and `HTMLCollection`. Both of these objects are Web APIs that make up the functionality of the Document Object Model (DOM). As we'll learn later, objects can be structured so they look and act like arrays, though they are not arrays!

In this lesson, we'll briefly cover all of these methods:

- `document.querySelectorAll()`
- `document.getElementsByClassName()`
- `document.getElementsByTagName()`
- `document.getElementsByName()`

And at the end of the lesson, we'll learn how to turn these collections into arrays so we can call array methods on them.

You are welcome to code along with this lesson, or simply read through it. We recommend using this lesson as a reference when building projects in the coming weeks.

`document` Query Methods

For demonstration purposes we'll use the HTML from the Mad Libs project. We last updated that project in the lesson "Removing Event Listeners", and it contains both an advertisement (solely created in our scripts) and a reset button (which is in our HTML). Here's the HTML from that project:

mad-libs.html

```
<!DOCTYPE html>
<html lang="en-US">
<head>
  <script src="js/scripts.js"></script>
  <title>A fantastical adventure</title>
</head>
<body>
  <h1>Fill in the blanks to write your story!</h1>
  <form>
    <label for="person1Input">A name</label>
    <input id="person1Input" type="text" name="person1Inpu
t">
    <label for="person2Input">Another name</label>
    <input id="person2Input" type="text" name="person2Inpu
t">
    <label for="animalInput">An animal</label>
    <input id="animalInput" type="text" name="animalInput">
    <label for="exclamationInput">An exclamation</label>
    <input id="exclamationInput" type="text" name="exclamat
ionInput">
    <label for="verbInput">A past tense verb</label>
    <input id="verbInput" type="text" name="verbInput">
    <label for="nounInput">A noun</label>
    <input id="nounInput" type="text" name="nounInput">
    <button type="submit">Show me the story!</button>
  </form>
  <button type="button" id="reset">Reset Form</button>
  <div id="story">
    <h1>A fantastical adventure</h1>
    <p>
      One day, <span id="person1a">_________</span> and <sp
an id="person2a">_________</span> were walking through the
woods, when suddenly a giant <span id="animal">_________</s
pan> appeared. "<span id="exclamation">_________</span>", <
span id="person1b">_________</span> cried. The two of them
<span id="verb">_________</span> as quickly possible, and w
hen they were safe, <span id="person1c">_________</span> an
d <span id="person2b">_________</span> gave each other a gi
ant <span id="noun">_________</span>.
    </p>
  </div>
```

```
</body>
</html>
```

`document.querySelectorAll()`

Just like `document.querySelector()`, we can input any valid CSS selector into `document.querySelectorAll()` (https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelectorAll) to get HTML element objects returned to us. The only difference is that `document.querySelectorAll()` always returns a collection of elements, even if there is only one or no element that matches the query. The collection that's returned is actually an object (a Web API) that's set up to look and act like an array, but we'll discuss that after we learn how to use the new methods.

In this first example, we're passing in the argument `"span"`. This should be familiar, since we've done this a lot with `document.querySelector()`. We can pass in the name of any HTML tag in quotes as an argument:

```
> document.querySelectorAll("span");
NodeList(9) [span#person1a, span#person2a, span#animal, spa
n#exclamation, span#person1b, span#verb, span#person1c, spa
n#person2b, span#noun]
```

A `NodeList` object is returned to us with every element that matches the query. We'll revisit the `NodeList` object later in this lesson.

In this second example, we're using a more interesting CSS selector: `nth-child`. This is a pseudo class in CSS and it is really helpful in getting the nth element in a list or a series of elements. As we can see below, when we pass in `span:nth-child(4)`, we're getting the 4th span element in the document.

```
> document.querySelectorAll("span:nth-child(4)");
NodeList [span#exclamation]
```

We can also include an `n` to get every nth element. With the argument `"span:nth-child(4n)"`, we're getting every 4th span element in the document.

```
> document.querySelectorAll("span:nth-child(4n)");
NodeList(2) [span#exclamation, span#person2b]
```

We can also use `even` or `odd` with the `nth-child` CSS pseudo class. As you might expect, using `even` returns every even element and using `odd` returns every odd element.

```
> document.querySelectorAll("span:nth-child(odd)");
NodeList(5) [span#person1a, span#animal, span#person1b, spa
n#person1c, span#noun]
```

If we want to get one item from the `NodeList` collection, we'll use bracket notation, passing in the index of the element we want (starting from 0). Just like with arrays, if we pass in 0, we'll get the first element returned:

```
> document.querySelectorAll("span:nth-child(odd)")[0];
<span id="person1a">_________</span>
```

If I pass the returned span element into my handy method that checks exact types, I'll find that I get a `HTMLSpanElement`:

```
> const personSpan = document.querySelectorAll("span:nth-ch
ild(odd)")[0];
> personSpan;
<span id="person1a">_________</span>
> Object.prototype.toString.call(personSpan);
"[object HTMLSpanElement]"
```

Why is this important? It shows us that even though a `NodeList` object is holding all of the span elements in one collection, each span element is of the type `HTMLSpanElement`, and we've worked with that category of Web API objects before!

By the way, we definitely recommend revisiting CSS Selectors when you have the time! There is *a lot* to explore! Check out this great reference on CSS Selectors on MDN. (https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors)

`document.getElementsByClassName()`

The `document.getElementsByClassName()` (https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementsByClassName) method gets all elements that have the same value for their `class` attribute. Since the Mad Libs project doesn't use any classes, the return is an empty collection. However, we can see the return type: `HTMLCollection`, another Web API object that is array-like. Don't worry, we'll demystify this array-like behavior down below.

```
> document.getElementsByClassName('x');
HTMLCollection []length: 0[[Prototype]]: HTMLCollection
```

If we want to get an element from the `HTMLCollection`, we'll also use bracket notation and pass in the index (starting at 0) of the element that we want to get.

```
> document.getElementsByClassName('x')[0];
undefined
```

Since our collection is empty, we get `undefined` returned to us.

`document.getElementsByTagName()`

The `document.getElementsByTagName()` (https://developer.mozilla.org/en-US/docs/Web/API/Element/getElementsByTagName) method gets all elements by their tag name. The same tag name that's returned from the `Element.tagName` (https://developer.mozilla.org/en-US/docs/Web/API/Element/tagName) property.

For example, we could check the tag name of our heading. In fact, we've done this before!

```
> const h1 = document.querySelector("h1");
> h1;
<h1>Fill in the blanks to write your story!</h1>
> h1.tagName;
"H1"
```

Remember if there are two H1 elements, which there are in the Mad Libs HTML, `document.querySelector("h1")` will only return the first one it finds.

When we use `document.getElementsByTagName()`, we don't have to capitalize the tag name, even though that's how tag names are returned to us from accessing the `Element.tagName` property.

```
> document.getElementsByTagName("h1");
HTMLCollection(2) [h1, h1]
```

This makes the `document.getElementsByTagName()` very similar to the `document.querySelector()` method, only we can't use other CSS selectors in it, like `"h1:nth-child(2n)"`.

To get a single element from the collection, we also use bracket notation. Here, we're getting the second element:

```
> const secondH1 = document.getElementsByTagName("h1")[1];
> secondH1;
<h1>A fantastical adventure</h1>
> Object.prototype.toString.call(secondH1);
"[object HTMLHeadingElement]"
```

As we see in the above code snippet, when we check the type of the individual H1 element, it's the familiar `HTMLHeadingElement` that we worked with in the last course section.

`document.getElementsByName()`

This last method `document.getElementsByName()` (https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementsByName) will get all elements that have the same value for their `name` attribute. Here's an example:

```
> document.getElementsByName("person1Input");
NodeList [input#person1Input]
```

This method would likely be more useful for radio buttons or checkboxes (as we'll learn), since all inputs of those types must share the same `name` attribute for them to function properly.

Just like in previous examples, if we want to get a single element from the list, we'll use bracket notation. And if we check the exact type of the returned element, we'll see we're working with

`HTMLInputElement`, a category of Web API objects that we're familiar with.

```
> document.getElementsByName("person1Input")[0];
<input id="person1Input" type="text" name="person1Input">
```

```
> Object.prototype.toString.call(document.getElementsByName
("person1Input")[0]);
"[object HTMLInputElement]"
```

Objects that Look and Act like Arrays

So what is this funny business of an object that looks and acts like an array? Well, objects can be structured to look and act like an array. An array-like object will have a `length` property and properties indexed from zero, but they do not have access to JavaScript Array methods. Also, since these are objects, they can be given a name. As we've learned, we're working with two array-like objects called `NodeList` and `HTMLCollection`, both of which are Web APIs. Their job is to hold multiple HTML elements.

Let's look at an example of a `NodeList` object:

```
> document.querySelectorAll("span:nth-child(4n)");
NodeList(2) [span#exclamation, span#person2b]
```

And then let's expand it to look inside:

```
NodeList(2) [span#exclamation, span#person2b]
  0: span#exclamation
  1: span#person2b
  length: 2
  [[Prototype]]: NodeList
```

The `NodeList` object has three properties: `0`, `1`, and `length`. The `0` and `1` properties are both set to two different `HTMLSpanElement` object. The `length` property is set to a number that corresponds to how many `HTMLSpanElement` objects are inside of the `NodeList` object. Right now, there's two, so `length` is set to `2`.

The properties `0` and `1` may seem odd since so far we've only seen object property names use strings and not numbers. However, we can in fact use numbers as property names! However, if we wanted to access an object property that's set to a number, we need to use **bracket notation** instead of dot notation. That's right, just like with arrays, we can use bracket notation to access object properties. They are most commonly used with properties that are not typical strings. So, if we want to get an element from our object, we use bracket notation:

```
> document.querySelectorAll("span:nth-child(4n)")[0];
<span id="exclamation">_________</span>
```

And since this `NodeList` object sets numbers as its property names and starts them at 0, this object looks and acts very much like an array. Hopefully this discussion is helpful in demystifying array-like objects. These exist in Web APIs as well as JS proper, and we'll encounter more of these. At this time, we don't need to understand why these exist, or anything deeper about how they are set up. We just need to know how to use them!

Turning `NodeList` and `HTMLCollection` Objects into Arrays

The biggest deal with the `NodeList` and `HTMLCollection` objects is that we can't call array methods on them. Sometimes we also can't iterate over them (we'll revisit this once we learn about looping). Well, we have a solution for that, and that's turning these into arrays with the `Array.from()` method. Let's look at an example:

```
> const headingCollection = document.getElementsByTagName
("h1");
> headingCollection;
HTMLCollection(2) [h1, h1]
> const headingArray = Array.from(headingCollection);
> headingArray;
(2) [h1, h1]
> Object.prototype.toString.call(headingArray);
'[object Array]'
```

Notably the `Array.from()` method is called on the array object type. We know it's the array object type, because `Array` is capitalized and we don't use `prototype` in the method's name. We'll revisit this type of method in an upcoming lesson.

Take note that there are some limitations for using `Array.from()` in older browsers, but that's true for a lot of JavaScript! As always, if you run into any issues, visit documentation like MDN.

MDN Documentation Links

The `NodeList` and `HTMLCollection` object belong to the many objects that make up the DOM. We won't explore them in depth like we did in the last course section with `Element`, `HTMLElement`, and other objects that also are a part of the DOM.

Here are direct links to the objects and methods we learned about in this lesson:

- `NodeList` (https://developer.mozilla.org/en-US/docs/Web/API/NodeList)
- `HTMLCollection` (https://developer.mozilla.org/en-US/docs/Web/API/HTMLCollection)
- `document` object methods (https://developer.mozilla.org/en-US/docs/Web/API/document)
- `Array.from()` (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/from)
- Reference for CSS Selectors (https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors)
- A complete list of all HTML element objects (https://developer.mozilla.org/en-US/docs/Web/API/HTML_DOM_API#html_element_interfaces_2).

Previous (/introduction-to-programming/arrays-and-looping/comparing-and-cloning-arrays)
Next (/introduction-to-programming/arrays-and-looping/adding-and-removing-html-elements)

Lesson 9 of 50
Last updated February 28, 2023

disable dark mode

(http://www.epicodus.com)