JavaScript Query Selector Methods

(https://github.com/learn-co-curriculum/phase-0-the-dom-query-selector-methods) (https://github.com/learn-co-curriculum/phase-0-the-dom-query-selector-methods/issues/new)

Learning Goals

- Use `querySelector()` and `querySelectorAll()` to find nested nodes
- Modify attributes of DOM nodes

Introduction

One of the most essential skills in our web development toolbox is finding elements in the DOM.

While `document.getElementById()` and `document.getElementsByClassName()` are good, we can improve our search when we use document structure (tag, `id`, `class`) **along with** the tree structure of the DOM. It turns out CSS is a *great* language for expressing those relationships! With the `querySelector()` and `querySelectorAll()` methods, we provide one or more CSS selectors as an argument and we get back the matching element or elements. Because they can take a string containing multiple selectors, they allow us to create very specific, complex queries.

Finding Nested Nodes

If you would like to follow along in the console, fork and clone this lesson, open the files in your text editor, and open `index.html` in Google Chrome. As you go, copy each HTML example into `index.html`.

`querySelector()`

The `querySelector()` method takes one argument, a string of one or more CSS-compatible **selectors** (https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Getting_Started/Selectors), and returns the *first* element that matches.

Given a document like:

```
<body>
  <div>Hello!</div>
  <div>Goodbye!</div>
</body>
```

If we called `document.querySelector('div')`, the method would return the first `div`. If we check its `innerHTML`, we should see `Hello!`.

Selectors aren't limited to one tag name, though. Otherwise, why not just use `document.getElementsByTagName('div')[0]`? We can get very specific.

```
<body>
  <div>
    <ul class="ranked-list">
      <li>1</li>
      <li>
        <div>
          <ul>
            <li>2</li>
          </ul>
        </div>
      </li>
      <li>3</li>
    </ul>
  </div>

  <div>
    <ul class="unranked-list">
      <li>6</li>
      <li>2</li>
      <li>
        <div>4</div>
      </li>
    </ul>
```

```
  </div>
</body>

const li2 = document.querySelector("ul.ranked-list li ul li");
li2;
//=> <li>2</li>

const div4 = document.querySelector("ul.unranked-list li div");
div4;
//=> <div>4</div>
```

In the above example, the first query says, "Starting from `document` (the object we've called `querySelector()` on), find a `ul` with a `className` of `ranked-list` (recall from CSS that the `.` indicates that `ranked-list` is a `className`). Then find an `li` that is a descendant of that `ul`. Next find a `ul` that is a descendant (but not necessarily a direct child) of that `li`. Finally, find an `li` that is a descendant of that (second) `ul`."

> **Note**: The HTML property `class` is referred to as `className` in JavaScript.

What does the second call to `querySelector()` say? Think about it for a minute, and then read on.

Wait for it...

The second call says, "Starting from `document`, find a `ul` with a `className` of `unranked-list`. Then find an `li` descended from `ul.unranked-list` and a `div` descended from that `li`."

CSS Selectors

If using CSS to target elements isn't feeling natural, now might be a good time to brush up on **selectors (https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Getting_Started/Selectors)**. Play around on the MDN page, then come back when you're ready.

querySelectorAll()

`querySelectorAll()` works a lot like `querySelector()` — it accepts a string containing one or more selectors as its argument, and it searches starting from the object that it's called on (either `document` or an element). However, instead of returning the first match, it returns a `NodeList` collection of all matching elements. A `NodeList` is similar to an `HTMLCollection`: it is an array-like structure containing, in this case, a list of DOM nodes.

Given a document like

```
<body>
  <main id="app">
    <ul class="ranked-list">
      <li>1</li>
      <li>2</li>
    </ul>

    <ul class="ranked-list">
      <li>10</li>
      <li>11</li>
    </ul>
  </main>
</body>
```

If we called:

```
document.getElementById("app").querySelectorAll("ul.ranked-list li");
```

We'd get back a list of nodes corresponding to:

```
<li>1</li>, <li>2</li>, <li>10</li>, <li>11</li>
```

Conclusion

The DOM selection methods `document.querySelector()` and `document.querySelectorAll()` are powerful tools for finding the elements we need to update and change. They use the familiar CSS selector syntax and allow us to create very specific queries that give us access to elements in complex DOM trees.

Resources

- **document.querySelector() (https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector)**
- **document.querySelectorAll() (https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelectorAll)**