DOM Editing Lab

- Due No Due Date
- Points 1
- Submitting a website url

 (https://github.com/learn-co-curriculum/phase-0-the-dom-editing-lab)  (https://github.com/learn-co-curriculum/phase-0-the-dom-editing-lab/issues/new)

Learning Goals

- Identify that DOM nodes are written as HTML

Introduction

We've started looking at the DOM and how it's created. Now it's time to see its structure.

If you haven't already, **fork and clone** this lab into your local environment. Navigate into its directory in the terminal, then run `code .` to open the files in Visual Studio Code.

Identify That DOM Nodes Are Written As HTML

In the previous lesson, we learned that, when we load a web page in a browser, the content we see in the DOM is a representation of the underlying HTML, CSS and JavaScript. If we were to view the DOM in Chrome Dev Tools (we'll learn how to do that shortly), we would see HTML that is a clone of the HTML found in the source HTML file. As we learned earlier in the course, that HTML consists of *elements* that in turn consist of HTML *tags* and their content.

When we're working in the DOM, the structure is the same. We can access objects in the DOM (called *nodes*) that consist of tags, just like the HTML elements that make up the base HTML. Nodes and elements are not the same thing — all elements in the DOM are nodes but not all nodes are HTML elements. However, when we're working in the DOM, the nodes we access and modify are virtually always HTML elements.

The Structure of DOM Content

We'll start by going over how content in the DOM is structured using nodes. The information below should be familiar from what you've learned about HTML elements.

DOM nodes most often have a starting tag and an ending tag. Examples include a paragraph:

```
<p>I am a paragraph.</p>
```

or a `main` section:

```
<main></main>
```

Because they have both starting and ending tags, we can nest other nodes inside them. The inner node is called a child node, and the outer node is called a parent node. To nest items, we simply add the child node and its content between its parent's starting and ending tags:

```
<body>
  <main>
    <p>I am a nested paragraph, inside the main element, inside the body!</p>
  </main>
</body>
```

Some nodes only have a starting tag. Those are called *self-closing elements* or *void elements*. Void elements do not have any content nested inside of them and cannot be parent nodes.

An example of a self-closing tag is an image:

```
<img
  src="https://media.giphy.com/media/3o6MbkZSYy4mI3gLYc/giphy.gif"
  alt="A cartoon gif"
/>
```

In self-closing tags, the trailing `/` is optional. This is valid too:

```
<img
  src="https://media.giphy.com/media/3o6MbkZSYy4mI3gLYc/giphy.gif"
  alt="A cartoon gif"
>
```

Enough review, let's write some HTML!

Instructions

Start by running the tests and taking a look through the errors. You'll see that the tests are looking for certain content to be present in the HTML file.

Next, open the `index.html` file in the browser using the instructions in the previous lesson for your development environment.

Just to speed things up a bit, paste the following code into `index.html`:

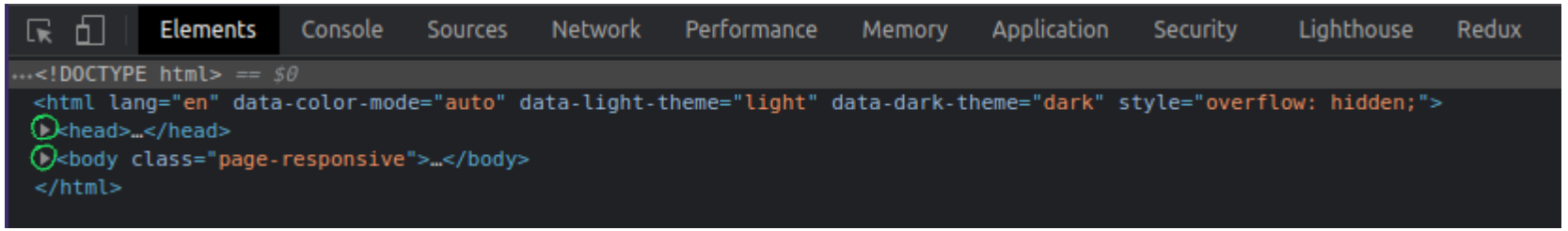
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>Introduction to the DOM Lab</title>
  </head>
  <body>
    <!--All our work for this lesson will go here-->
  </body>
</html>
```

Refresh the browser page to see the changes.

Open the Google Developer Tools by clicking on the "View" menu and selecting Developer -> Developer Tools. The Elements tab should be selected but, if it isn't, click on it. Here we have the DOM representation of the HTML source loaded by the browser. You should see the `head` and `body`

elements nested inside the `html` element. If the `body` element is collapsed, use the disclosure triangle to expand it. You should see that the `body` element is, temporarily, child-less. Let's go ahead and start adding some content in `index.html`.

Note: the **disclosure triangle** (https://en.wikipedia.org/wiki/Disclosure_widget) is the triangle to the left of the `<body>` tag. When you first open the Elements tab, the nodes are generally collapsed, hiding their contents. You can click the triangle to expand the node and see its contents. Disclosure triangles are standard for hiding information throughout Chrome DevTools. If you want to see more, feel free to click on the triangle! You're not going to break anything.



First, let's add a title to our page:

```
<h1>My HTML adventure</h1>
```

Refresh the page to see the changes displayed in the browser. If you view the Elements tab again, you should see that a new child node is nested inside the `body`. Finally, run the tests again; the first test should now be passing.

Next, we'll add a paragraph below the title. We'll also add some highlighted bits of text to the paragraph to make it stand out a little.

```
<p>
  We're writing HTML markup to display in our <strong>browser</strong>. We're
  basically telling computers what to do. <em>Neat!</em>
</p>
```

Save the file and check out the page in the 'Elements' tab. What's happening above is that we added some inline elements, `<strong>` and `<em>` to our paragraph to style things a little. The `<strong>` tag makes any text within look **important**. It's usually bold in browsers by default. The `<em>` tag allows us to *emphasize* certain text. This text is typically rendered as italic in browsers.

Run the tests again; you should now have all but two of the tests passing.

In our paragraph, let's make "HTML" a hyperlink and link to the MDN definition. We'll use the `<a>` tag for this. Add this to our existing `<p>` :

```
We're writing
<a href="https://developer.mozilla.org/en-US/docs/Web/HTML">HTML</a> markup to
display in our <strong>browser</strong>.
```

Notice that HTML attributes (in this case, the `href` attribute) are shown alongside their opening tag.

Lastly, we'll add a table below the paragraph to recap some of the stuff in this lesson:

```
<table>
  <thead>
    <tr>
      <th>Element name</th>
      <th>Display value</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>h1</td>
      <td>block</td>
    </tr>
    <tr>
      <td>p</td>
      <td>block</td>
    </tr>
    <tr>
      <td>strong</td>
      <td>inline</td>
    </tr>
    <tr>
      <td>em</td>
      <td>inline</td>
```

```
      </tr>
    </tbody>
  </table>
```

Woah. That's a **lot** of markup! If you take a look at the result, though, you'll see that it's a fairly complex visual — it's a table! Our table consists of a header and a body. The header allows us to give the columns a name, and the table body contains the rows of content. Both `<thead>` and `<tbody>` tags contains rows, which are represented as `<tr>` (table row). These rows then contain cells which form the table's columns. In the `<thead>` row, cells are represented as `<th>`, while cells in `<tbody>` have their content in `<td>` tags.

That's a *lot* of nesting.

Look again at the Elements tab. Expand out all the children of the `table`. This is the DOM tree!

When you're done, go ahead and run the tests. They should now all be passing.