Introduction

Welcome Back!

As you probably know, CSS allows you to create external style sheets, which can be applied to multiple pages in your site. This allows you to use code in many different places, without having to repeat that code each place. For instance, let's say you have a website with 1,000 pages and you decide you want to make a design change:

- **Without an external style sheet,** you would have to go through each page and make the change within each page.
- **With an external style sheet,** you just change the style rule in the external style sheet and that change is automatically reflected in every page of the site.

The same idea works for JavaScript. You can put JavaScript code in a file, and then use it across every page in your site without repeating any code. This way any change you make to that eternal code is automatically applied to every page within the site that uses that JavaScript code. You'll see how it's done in this lesson.

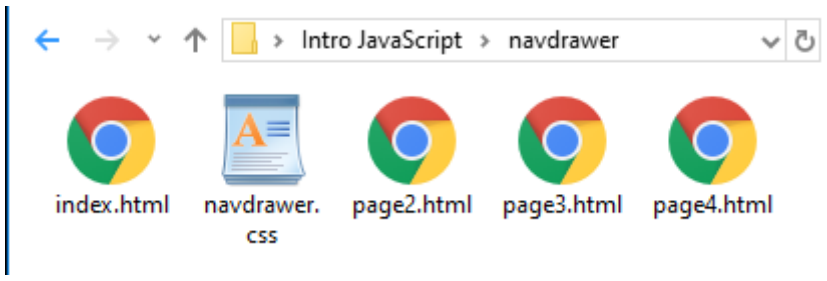
Creating a Navigation Drawer

For our working example, we're going to create a modern fancy navigation menu system that uses a *navigation drawer*. This is a set of links that usually shows only icons for navigating, to minimize the amount of space used by the navigation menu. But users can slide the drawer out for the full menu any time they need a reminder.

Watch the video below to see what we're trying to accomplish.

Download

To really test a navigation system, we'll need a website with multiple pages. To keep things organized for future reference, we've put all the files into a single folder named *navdrawer*.



Below is the link to the ZIP file which contains the resources you'll need for this lesson:

Save the **navdrawer** folder in your Intro JavaScript folder. Inside you'll find four (4) files:

- **navdrawer.css** – This file contains all of the CSS code you need for the CSS style sheet.
- **index.html** – This file contains the HTML for your index page (page1).
- **page2.html** – This file contains the HTML for page 2.
- **page3.html** – This file contains the HTML for page 3.
- **page4.html** – This file contains the HTML for page 4.

Download Lesson-10_ClassProject.zip

3.4 KB (Gigabytes) ZIP

When you open any page, you should see a black header bar across the top and a navigation menu down the left side. However, the drawer won't slide in an out just yet because we need to write some JavaScript code for that. We'll also show you how to brighten the link that represents the page that the user is currently views.

Are you ready? Come on let's get started!

Chapter 1: Externalizing JavaScript

How External JavaScript Files Work

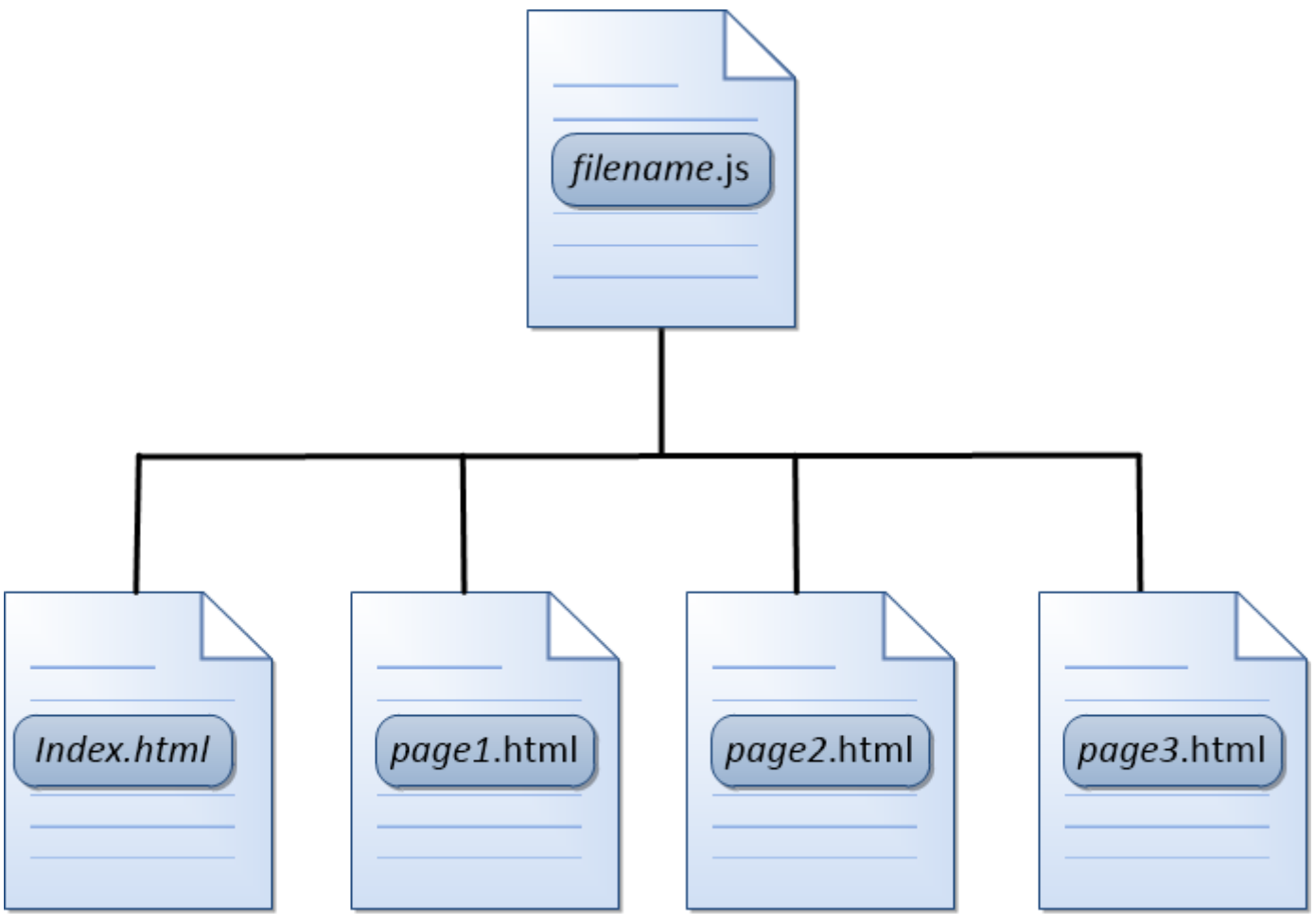
If you've been doing Web development for a while you're probably familiar with external style sheets. An external style sheet contains CSS code that can be applied to multiple pages in your site. It's the same thing for an external JavaScript file.

What is an External JavaScript File?

Just like a web page or style sheet, your external JavaScript file will be a regular text file. The difference is in the filename extension:

- **For web pages,** we typically use the *.html* or *.htm* extension.
- **For style sheets,** we use the *.css* extension.
- **For external JavaScript files,** the *.js* extension is employed.

Like a CSS style sheet (.css file), a JavaScript (.js file) can be applied to multiple pages. There is no minimum or maximum, it can be applied to any number of pages.



Linking to an External JavaScript File

As with external style sheet, a webpage doesn't automatically know when there is an external JavaScript file for it to link to and draw from. You have to manually provide the link yourself.

Like a style sheet link, your JavaScript link has to be added between the <head> and </head> tags of the page. However, unlike a style sheet, you don't use a <link> tag to link to an external style sheet. Instead, you use a script tag with this syntax:

```
<script src="path"></script>
```

Simply replace *path* with a standard HTML path that points to the JavaScript file. For our example, we'll be creating a JavaScript file named *navdrawer.js* so the link would be:

```
<script src="navdrawer.js"></script>
```

Note

If the JavaScript file is in a subfolder, you'll need to precede the file name with the folder name and slash. For example, if the external style sheet is in a subfolder named *scripts*, then the link would be:

```
<script src="scripts/navdrawer.js"></script>
```

Warning!

Unlike a style sheet link, which uses href= as the attribute. the <script> tag uses src= like an img tag.

You can even link to JavaScript files on external sites, including JavaScript files written by other people. You just have to provide the complete URL which starts with http:// or https://. For now, we'll stick to using styles sheets within our own website.

Creating an External JavaScript File

You don't need to add links to the pages you downloaded earlier, because they each already have a link for an external file named *navdrawer.js*. However, since we haven't created the *navdrawer.js* file yet, that code isn't doing anything right now. So let's do that now.

Here are the Steps

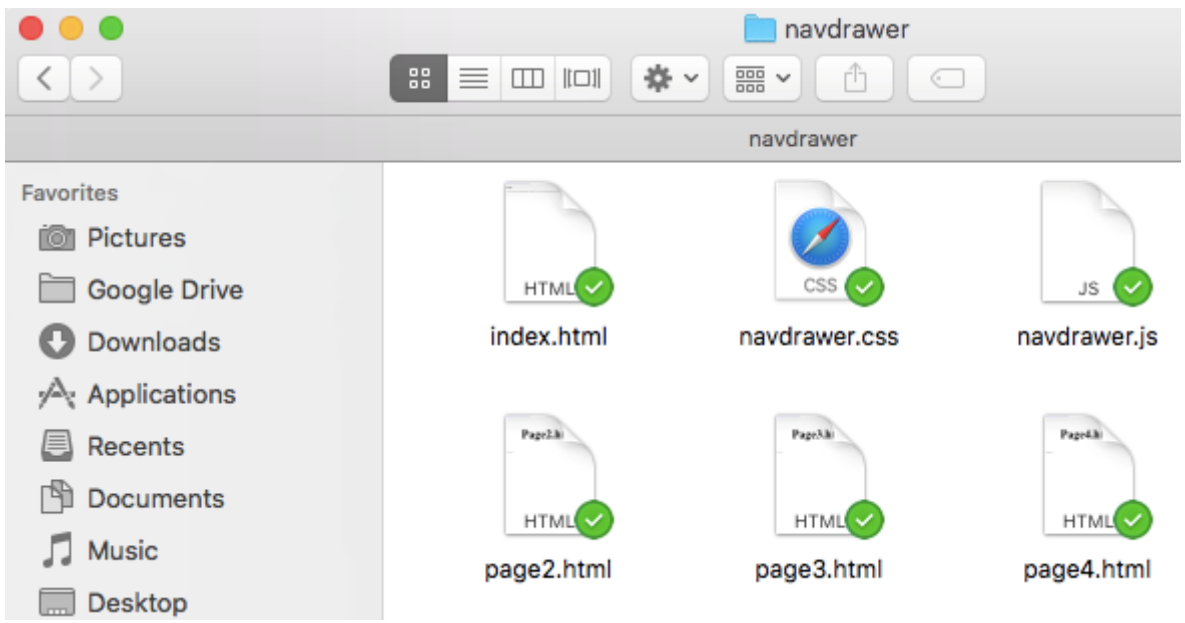
1. Create a new empty text file.

2. Save this file as a JavaScript file in your *navdrawer* folder using the **navdrawer.js** name and extension.

Files in Your Folder

You should now have six (6) files in your *navdrawer* folder:

1. index.html
2. navdrawer.css
3. navdrawer.js
4. page2.html
5. page3.html
6. page4.html



Of course an empty .js file doesn't do anything, so let's add some code to it. Go ahead and open the *navdrawer.js* file in your code editor.

Adding Code to Delay JavaScript Execution

When your JavaScript code refers to an element on the page by its ID, like when you're using *.getElementById()*, that code will fail if it's executed before the entire page has rendered.

If the code is in a function that's not called until after the page is fully loaded, and the user clicks a button or something to run the JavaScript code, then that's not problem. However, if the code must access an element by its ID before the user even has a chance to interact with the page, then you need to manually delay code execution until the page has been fully loaded.

One way to delay JavaScript code execution until the page is loaded it to put the JavaScript code at the end of the page, after the *</body>* tag. That works because code executes left-to-right, top-to-bottom, in the same order that we read. However, many developers find this to be an awkward way to manage things, preferring to keep all the information and code "about" the page up between the *<head>...</head>* tags.

When you put the JavaScript code (or link to external JavaScript file) up in the *<head>...</head>* tags, you need to place all that code inside a JavaScript function that doesn't execute until after all the HTML and CSS code has been loaded and rendered on the screen. This is the most common method.

Like all computer languages and products, JavaScript evolves over time, and the language is updated based on current technology and lessons learned from previous versions. The code for delaying execution has also evolved.

Move through the slideshow by clicking either the right or left arrow. You can also navigate to a specific slide by selecting the page from the menu located at the bottom of the slideshow.

Slideshow Slide 1 starts here

```
object.onload = function(){
    javascript to execute
    };
```

The cleanest, simplest way to handle it, historically, has been to use a simple block like this. Here you replace object with the word window or document to delay execution until the entire page is loaded (usually the word window as that's the last thing to fully load).

Slideshow Slide 1 ends here

Slideshow Slide 2 starts here

```
document.addEventListener("DOMContentLoaded  ", function (event) {
    javascript to execute
    })
```

Over the years people started noticing subtle differences among different brands and version of browser with those, especially with extremely complex pages with mountains of JavaScript code and events. People have since gravitated to a more complex syntax that looks like this.

Slideshow Slide 2 ends here

Slideshow Slide 3 starts here

```
document.addEventListener('DOMContentLoaded',  (event) => {
    javascript to execute
    })
```

This slightly shorter version using an arrow function => (an equal sign character followed by a greater-than sign character) is also common.

Slideshow Slide 3 ends here

Either *document.addEventListener* syntax will work, so just use whichever you prefer. The important part is the word DOMContentLoaded which means "all the HTML and CSS code has been executed and rendered to the page".

The bottom line is that we want the JavaScript to wait for the DOM content to finish loading, so there is no ambiguity as there might be across different brands and versions of web browsers with the older, more general *onload* syntax. In real life, you may find they all work the same with modern browsers. But since the *DOMContentLoaded* syntax is the most common these days, we'll stick with that in this lesson.

When using an external JavaScript file, it's very common to see all that code within that file encased is such a block of code. There is no downside to using it, and in solves the common problem of trying to execute JavaScript code too early as a page is loading. So we will use the more common, modern syntax to encase all the JavaScript code we write in this lesson.

Here are the Steps

1. Open *navdrawer.js* in your code editor.

2. Let's start by adding the following comment:

```
//Delay JavaScript execution until after all HTML and CSS is rendered
```

3. Beneath that type or copy/paste the following code to delay JavaScript execution until the page loads:

```
document.addEventListener("DOMContentLoaded", function(event){
```

4. Then press **ENTER** a few times to create some space, as a reminder to put *all* the rest of the code you need for this file within that one large block.

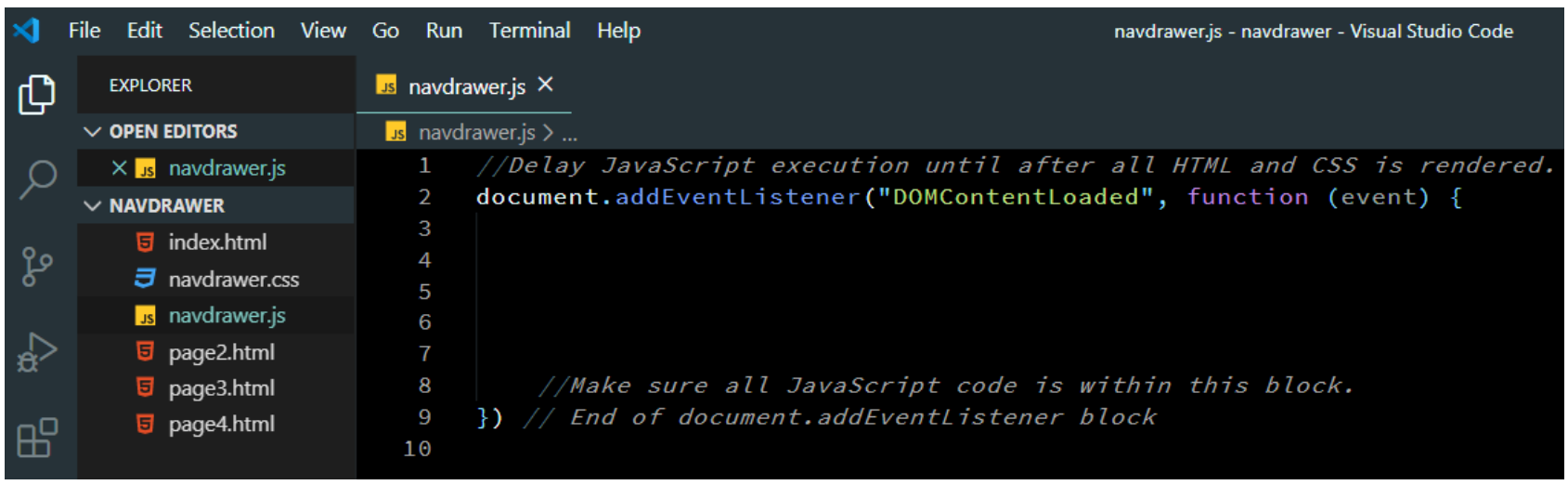
5. Next, add the following comments and closing braces to encapsulate your code block:

```
//Make sure all JavaScript code is within this block
```

6. Save your progress.

Sneak Peek

Your external JavaScript file should look something like this:



We're using VS Code as the editor, but you can use whatever code or text editor you normally use.

Important

Notice that the .js file contains only JavaScript code and comments. You cannot put any HTML tags in there (not even <script> and </script>, nor CSS style rules. The JavaScript file is for JavaScript code *only*.

So now we have a block of code that waits for the all the HTML and CSS code to load first, and we can start adding in the code we need to make our navigation drawer more interesting. Up next, well look at a new way to deal with event handlers in JavaScript.

Chapter 2: Handling Events Outside HTML

Working with Anonymous Functions

Throughout this course we've used many JavaScript event handlers *inside* HTML tags as a way to execute JavaScript code.

Example

Take a look at the code below:

```
<!DOCTYPE html>
<html>
<head>
<title>Sample</title>
<script>
function showalert() {
alert("You clicked the btton");
}
</script>
</head>
<body>
<button onclick="showalert()">
Click
</button>
</body>
</html>
```

This code contains a JavaScript function named *showalert()*. On the body of the page there's a button displaying the words *Click Me*. When a user clicks that button, the button calls the JavaScript function. This function generates an alert box saying "You clicked the button".

This is certainly a perfectly valid way to do things. But is isn't the only way.

As an alternative, you can give the object that calls the function a unique ID name. Then, you write a function that gets the object by name, followed by the event handler (e.g. onclick) followed by an equal sign and an *anonymous function*.

What is an Anonymous Function?

An anonymous function is one that doesn't have a name. It doesn't need a name because the object and event that trigger the function are already there, before the word *function* in the code. Here is the syntax:

```
object.onevent = function(){
 code to execute
};
```

The *object* can be any predefined object (like *document*), or it can be a an HTML element on the page that has a unique ID and can be accessed with .getElementById(). However, it's important that all the HTML and CSS code already be rendered before the JavaScript code that depends on those IDs is executed. This means the code must go inside a block that delays JavaScript execution until the page is fully rendered.

Sliding the Navigate Drawer

Detecting a Click on the Bars

We already have a block delaying JavaScript execution in our external JavaScript file, so we're good to go there. Our working example has a *hamburger menu* (a nickname for the menu icon that displays three stacked horizontal bars) above the navigation menu on the left. We want to use it to allow the user slide the menu in and out of view. The code for this icon can be found in the page header, using the Font Awesome (https://fontawesome.com/v4.7.0/icons/) bars icon. It's id is *burger*, and you can see that code below:

```
<header>
 <!-- Three bars at left of header -->
 <span id="burger" class="fa fa-bars fa-2x"></span>
</header>
```

Notice that there is no onclick event handler in the span, which is why clicking the hamburger menu in the page currently does nothing. Let's fix that!

Here are the Steps

1. Open *navdrawer.js* in your editor.

2. Place the cursor just below the line that starts with *document.addEventListener*.

3. Type or copy/paste in this comment and block of code to detect a click on the bars:

```
//Event listener for clicking on hamburger menu
 document.getElementById('burger').addEventListener('click', function () {
 })
```

4. Save your progress.

The order and nesting of code here is very important.

Get any part of it wrong, and it might not work. Notice that the entire block is contained within the larger DOMContentLoaded block, to ensure that the page is fully rendered before JavaScript goes looking for the element named *burger* in the code that reads *document.getElementById('burger').*

```
//Delay JavaScript execution until after all HTML and CSS is rendered
document.addEventListener("DOMContentLoaded", function (event){

        //Event listener for clicking on hamburger menu
        document.getElementById('burger').addEventListener('click', function () {
        })

        //Make sure all Javascript code is within this block.
})      //End of document.addEventListener block
```

So far your code does nothing other than to tell the web browser that when the user clicks on the element with the id name *burger*, we want it to execute some JavaScript code. Now we have to write the JavaScript code to be executed.

Hiding and Showing the Menu

If you look in *navdrawer.css*, you will notice three CSS styles have been created for displaying the navigation menu:

Text equivalent start.

Topic	Information

`nav`	The first style rule, nav, applies to the entire nav bar. It's fixed-positioned to the left side of the page.
	
	`/* The entire nav drawer */`
	`nav {`
	`font-family: 'Roboto', sans-serif;`
	`background: #000;`
	`width: 200px;`
	`position: fixed;`
	`left: 0;`
	`top: 72px;`
	`z-index: 10000;`
	`border-bottom-right-radius: 3px;`
	`box-shadow: 2px 2px 4px rgba(0, 0, 0, 0, .5);`
	`}`

`.iconsonly`	Below that style rule is a CSS class named .iconsonly that moves the left edge of the browser 145 pixels to the left, so it's mostly out of view on the page, but there's enough room to show just the icons at the right side of each option.
	
	`/* Nav hidden from view (Just slid over 145 pixels to the left) */`
	`#nav.iconsonly {`
	`left: -145px;`
	`transition: 500ms;`
	`}`

Topic	Information
`.fullmenu`	An alternative class named .fullmenu displays the menu so its left side is flush with the left side of the browser window, making it completely visible.

```
/* Nav drawer fully visible (not slid
over to the left) */
  #nav.fullmenu {
  left: 0;
  transition: 500ms;
  }
```

Read the topic in the first column. Then read the second column for the information.

Text equivalent stop.

What we need now is some Javascript code that applies the *.iconsonly* class to the menu if it's visible (e.g. current class applied is *.fullmenu*) and applies the *.fullmenu* class if it's hidden (e.g. current class applied is *.iconsonly*). So in other words, it switches between the two CSS style classes. Here is how we can do that with two lines of JavaScript code:

```
var menu = document.getElementById('nav')
 menu.className = menu.className == 'iconsonly' ? 'fullmenu' : 'iconsonly';
```

In the first line, you're simply assigning the short name *menu* to that nav menu. Then the next line uses a ternary operator to say "If the menu currently has the *iconsonly* CSS class applied, then apply the *fullmenu* style, otherwise, apply the *iconsonly* style."

Anyway, let's complete the function.

Here are the Steps

1. Go back to *navdrawer.js* in your code editor.

2. Place the cursor inside the block of code you just wrote that starts with *document.getElementByID*.

3. Type or copy/paste this comment and JavaScript code:

```
//Swap full menu / icons only styles
var menu = document.getElementById('nav')
menu.className = menu.className == 'iconsonly' ? 'fullmenu' : 'iconsonly';
```

4. Save your progress.

Sneak Peek

Once again, it's important that the code be nested properly, so make sure that last bit of code you added is inside the *document.getElementById()* block.

```
//Delay JavaScript execution until after all HTML and CSS is rendered.
document.addEventListener("DOMContentLoaded", function (event) {

        //Event listener for clicking on hamburger menu.
    document.getElementById('burger').addEventListener('click', function() {
                //Swap full menu / icon only styles
                var menu = document.getElementById('nav')
                menu.className = menu.className == 'iconsonly' ? 'fullmenu' : 'iconsonly';
            })


            //Make sure all JavaScript code is within this block.
})      //End of document.addEventListener block
```

Setting the Style for Initial Page Load

You can have the page open with the nav drawer already slid out, or already slid in. Since none of this code executes until the page is fully rendered, all you need to do is write a little JavaScript code inside this block to start with one position or the other. The syntax is:

```
document.getElementById('nav').className = "class";
```

- **If you want the page to open with the menu out,** the class would be *iconsonly*.
- **If you want the page to open with the menu in,** the class would be *fullmenu*.

For our example, we'll set it so that the menu is in. This should happen right after the page loads. We already have an event handler for that in the *DOMContentLoaded* block, so we just need to add the appropriate code (and comment) near the bottom of that block.

Here are the Steps

1. Go back to *navdrawer.js* in your code editor.
2. Place the cursor above the line that starts with *//Make sure all JavaScript code....*
3. Type or copy/paste the following code:

```
 //Start with menu showing icons only
 document.getElementById('nav').className = "iconsonly";
```

```
//Delay JavaScript execution until after all HTML and CSS is rendered.
document.addEventListener("DOMContentLoaded", function (event) {

    //Event listener for clicking on hamburger menu.
document.getElementById('burger').addEventListener('click', function() {
            //Swap full menu / icon only styles
            var menu = document.getElementById('nav')
            menu.className = menu.className == 'iconsonly' ? 'fullmenu' : 'iconsonly';
    })

    //Start with menu showing icons only.
    document.getElementById('nav').className = "iconsonly";

    //Make sure all JavaScript code is within this block.
})      //End of document.addEventListener block
```

4. Save your progress.

5. Now open *index.html* (or any other .html page in the site) in a browser. Initially the navigation menu should only show icons. But you can slide it in and out just by clicking the three bars above the menu.

   Click the hamburger menu below to see how it's supposed to work.

   Well, that's pretty cool, but there's another thing we can do to help visitors to our site. We can highlight the page they're currently viewing, so they know where they current are within the site. We'll write that JavaScript code next.

Chapter 3: CSS Styling with Event Handlers

Highlighting the Nav Links

Establishing the CSS Styles

Professional websites will often highlight whichever link represents the page that the user is currently viewing. This is just a courtesy to help users with general navigation and knowing where they are at the moment. In *navdrawer.css*, each link in the navigation drawer has an opacity of .6, which is a bit dimmed.

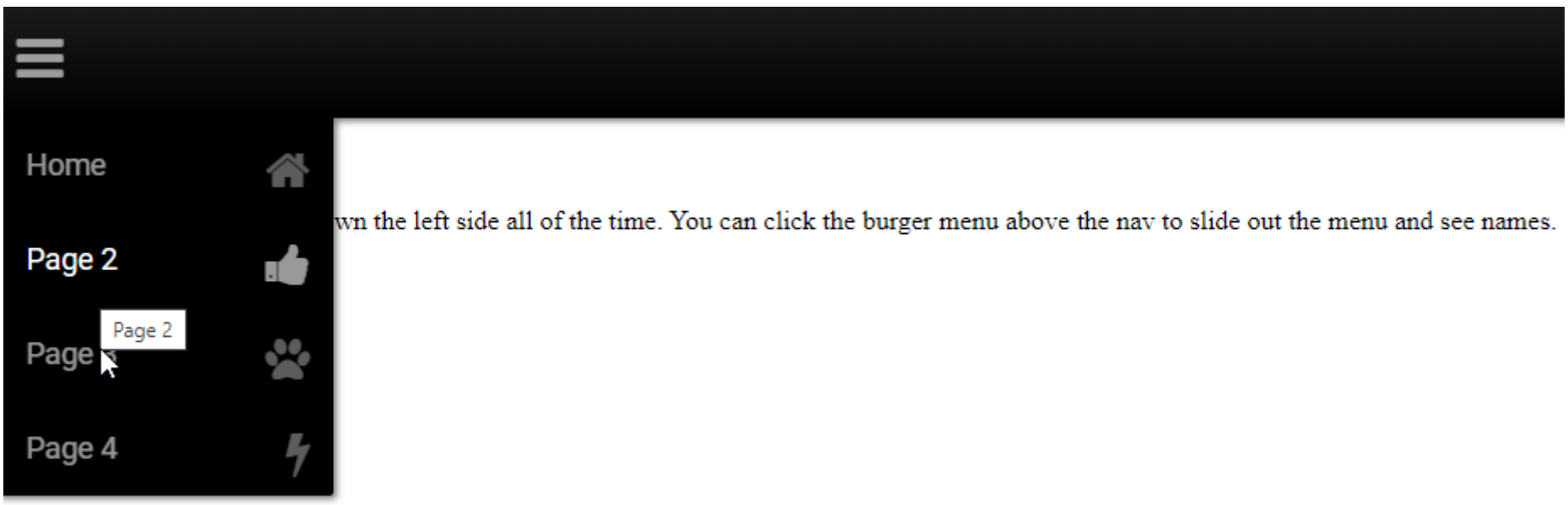
```
/* Each nav link */
 nav a {
 box-sizing: border-box;
 display: block;
 text-align: left;
 color: white;
 text-decoration: none;
 width: 98%;
 padding: 1em;
 opacity: .6;
 position: relative;
 z-index: 11000;
 font-size: 110%;
 }
```

You can see that in the style rules they have the selector *nav a {}* which applies to all links (*<a>...* tags) in between the *<nav>* and *</nav>* tags on the page. Each of those links also has an associated *nav a::after* style that defines the Font Awesome icon shown to the right of each link. All of those, too, have an initial opacity of .6.

```
/* The icon at the right of each nav item */
 nav a::after {
 font-family: FontAwesome;
 font-size: 160%;
 opacity: .6;
 position: absolute;
 right: 10px;
 }
```

The style sheet already has a *nav a:hover* style that changes the opacity of the link (both name and icon) when you hover your mouse over it. It changes the opacity from .6 to 1.

```
/* Brighten the link a little when under the mouse pointer */
 nav a:hover {
 opacity: 1;
 transition: 500ms;
 }
```



Additionally, there is already a style rule in the style sheet that changes the opacity of the link from .6 to 1 if it has the *currentpage* CSS class:

```
/* We use JavaScript to apply this style to the link that */
/* represents the current page so that link is highlighted  */
nav a.currentpage,
nav a.currentpage::after {
opacity: 1;
}
```

However, while this style exists, we need to add JavaScript to actually make it work. That's what we'll do next.

Highlighting the Current Page Link

Identifying the Current Page

If you use JavaScript to apply the CSS *currentpage* class name to a link, its opacity is increased to 1, which makes that link and its icon brighter. To highlight the link in the navbar that represents the current page, we need to figure out what page the user is on, and then apply that style class to the appropriate link. Let's do it.

Here are the Steps

1. Open *navdrawer.js* in your code editor.

2. Place your cursor just above the comment that reads *//Make sure all JavaScript code is within this block.*

3. Type or copy/paste in the following comment and code:

```
//This runs every time the page loads, doesn't need to be called as a function
 var filename = location.pathname.split('/').pop();
 alert(filename);
```

4. Save your changes.

5. Now, open *index.html* in a browser. You should see an alert box that says *index.html*, which is the name of the page you're viewing.

6. Click **OK**, then click the link to another page in the navigation bar. This time the file name of that page will show in the alert box.

Let's take a moment to discuss how JavaScript "knows" what page is opening. The first line we added creates a variable named *filename*:

```
var filename = location.pathname.split('/').pop();
```

The second line creates an alert that displays the variable *filename*:

```
alert(filename);
```

Let's break this down:

Text equivalent start.

Topic	Information
`location.pathname`	The *location.pathname* is the complete URL that's currently in the browser's address bar. **When the file is on your computer,** that location.pathname will look something like this: *file:///C:/Users/Username/Desktop/Intro%20JavaScript/navdrawer/index.html* **When the file has been published it out to your website,** that location.pathname will look more like this: *https://www.yourwebsite.com/index.html*
`.split("/")`	All we care about is what comes after the last slash, because that's where the current page file name is located. To isolate that, you use the JavaScript *.split("/")* to split that long URL into smaller pieces at each slash.
`pop()`	Then the *pop()* at the end says "after you split it up at the slashes, give me just what comes after the last slash". So no matter what the complete URL is in the web browser, what ends up in the filename variable is the filename at the end.

Read the topic in the first column. Then read the second column for the information.

Text equivalent stop.

Defining the Action to Take

In real life, we don't need that to pop up on the screen. We just put in the alert() to test the code and make sure it's getting just the page name. Now that we've identified the filename, we need to define what we want the code to do with it.

Here are the Steps

1. Go back to *navdrawer.js* in your code editor.

2. Replace the *alert(filename);* line with is code:

```
//If it's nothing, or index.html, or index.htm, it's the home page.
 if (filename == '' || filename == 'index.html' || filename == 'index.htm') {
 document.getElementById('home').className = "currentpage";
 }
```

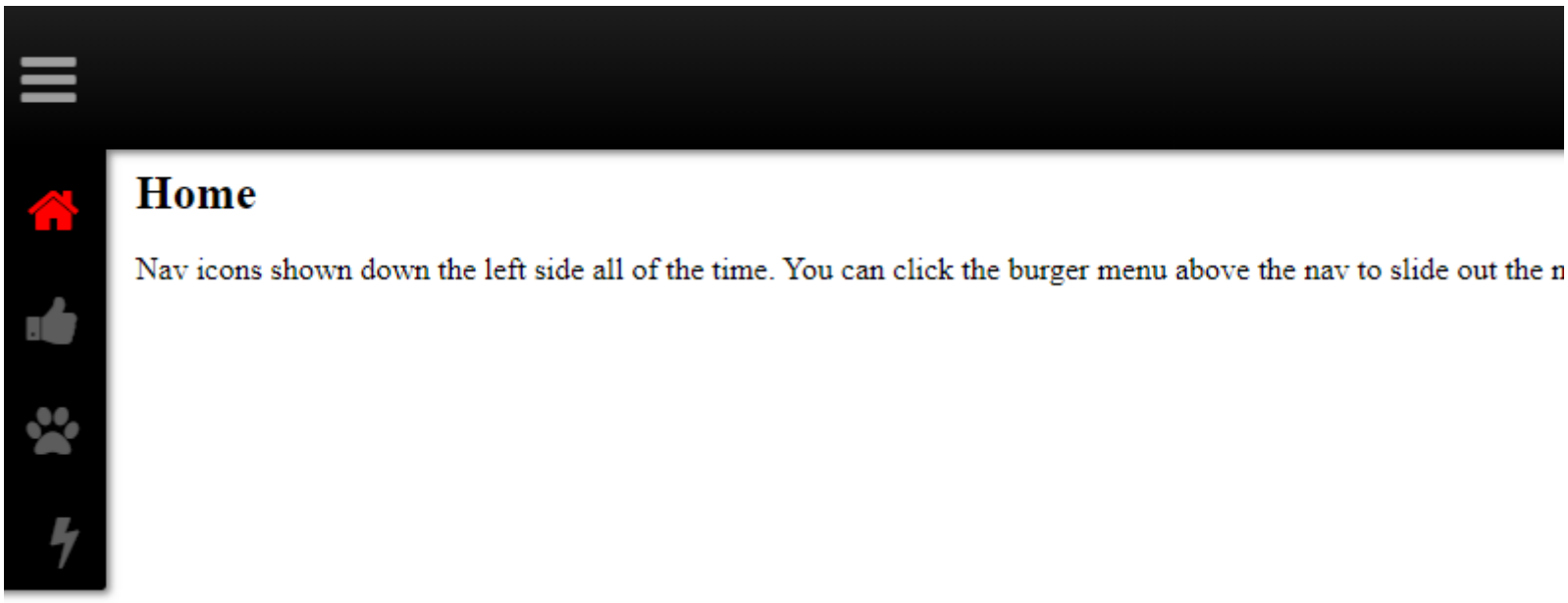
What does it say?

When you first browse to a site, you automatically see only the URL without the file name of the page. That's because the web server knows that when someone browses to the site without specifying a page name, it should show the home page (which is usually named index.htm or index.html).

The if statement says "if the file is nothing ("") or if it's index.html or index.htm then execute this line of code":

```
document.getElementById('home').className = "currentpage";
```

3. Save your changes and then open *index.html* in your browser.

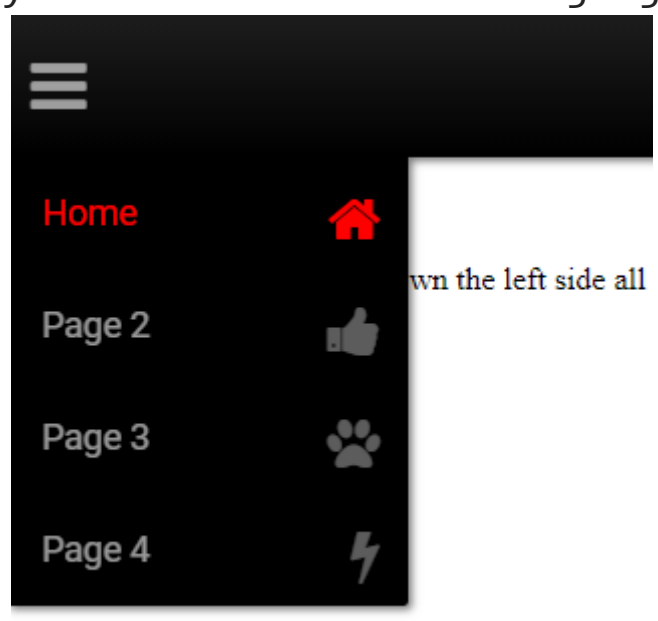


Highlighting the Page

As you can see, that line of code then applies the *currentpage* style class name to the link that has the id of "home". If you look at the tags inside the navbar, you'll see that's the link that represents the home page:

```
<nav id="nav">
<a href="index.html" id="home" title="Home">Home</a>
<!-- The id of each page below here MUST exactly match the page filename -->
<!-- for the JavaScript code to work. Titles are optional -->
<a href="page2.html" id="page2" title="Page 2">Page 2</a>
<a href="page3.html" id="page3" title="Page 3">Page 3</a>
<a href="page4.html" id="page4" title="Page 4">Page 4</a>
</nav>
```

If you slide the nav drawer out, you'll see the whole link is highlighted.



4. Click another link to visit another page. You'll see that the Home link is no longer highlight. However, it won't highlight the new page's link yet because we've only told it what to do for the Home page.

5. So, go back to *navdrawer.js* in your code editor.

6. Type or copy/paste in the code below just after the closing curly brace of the if statement we just added:

```
else {
 //Otherwise loop through the rest of the links and apply highligh CSS style to
 //the link whose id matches the current page filemame (without the extension)
 var nav = document.getElementById('nav');
 var links = nav.getElementsByTagName('a');
 for (i = 1; i < links.length; i++) {
 if (links[i].getAttribute('href').indexOf(filename) > -1) {
 links[i].className = "currentpage";
 }
 }
 }
```

7. Save your changes.

8. Go ahead and open *index.html* in a browser again and click the other links. As you navigate from page-to-page, the link that represents the current page should be highlighted. That's true even if the nav drawer it slid in, or slid out.

Now, let's talk about how it works.

Text equivalent start.

Topic	Information
`else {`	First of all the "else" means they are not on the home page, because the current page's file name isn't blank or index.html or index.html.
`var nav = document.getElementById('nav');`	So, if they are not on the home page, this line of code gets executed. That line identifies the entire navigation bar, which is contained in <nav id="nav">...</nav> links, and gives that navigation bar the short name *nav* in the code.

Topic

```
var links = nav.getElementsByTagName('a');
```

Information

Then this line of code creates a variable named *links* (again, just a made up name) that ends up containing a *collection* of all the links (defined by <a>... tags) inside that nav div. This is new because typically we store only one value, like a number or word or something like that. However *getElementsByTagName* is different in that is stores a collection of all the links inside the nav div. So, the variable links actually end up containing all of the a tags, which look like this in the page:

```
<a href="index.html"
id="home"
title="Home">Home</a>
 <a href="page2.html"
id="page2" title="Page
2">Page 2</a>
 <a href="page3.html"
id="page3" title="Page
3">Page 3</a>
 <a href="page4.html"
id="page4" title="Page
4">Page 4</a>
```

Those tags are what JavaScript "Sees" in the collection named *links*.

Topic	Information

`for (i = 1; i < links.length; i++) {`
`if (links[i].getAttribute('href').indexOf(filename) > -1) {`
	Then this line sets up a loop to look at each link, one at a time, within that collection. The if statement tests for the following conditions:
	
	The *links[i].getAttribute('href')* looks at what href= is in each link. So it "sees" something like this as it goes through the loop:
	
	`index.html`
	`page2.html`
	`page3.html`
	`page4.html`

`links[i].className = "currentpage";`	The part that reads *.indexOf(filename) > -1* says "if the filename is contained within the href attribute of this tag is greater than -1". We use >-1 because if the filename does not exist within the href of this link, then JavaScript returns -1 (meaning *nowhere to be found*). If the file name is found to be greater than -1, then this line is executed. It says "add the currentpage class name to whatever link you're looking at right now". This means that only the link whose href= filename matches the filename of the page the user is currently viewing gets that style class added to it. Hence, it's the only link that ends up being highlighted in the menu.

Read the topic in the first column. Then read the second column for the information.

Text equivalent stop.

And that is how that bit of magic works. It's not the easiest thing to wrap your brain around, but once you do you can see how truly powerful and magical JavaScript can be at times. Let's wrap things up with a quick review.

Review

Lesson 10 Review

Whew, quite a bit to take in from the previous chapters. But you can see how amazingly precise the instructions you give to a web browser can be when you write JavaScript code.

- **Externalizing JavaScript:** Here you learned how you can write JavaScript into an external file, link all your pages to it, and then re-use that JavaScript code in each page of your website without repeating the code inside any one page. Good stuff to know!

- **Handling Events Outside HTML:** In this segment you learned about using anonymous functions in your external JavaScript file to handle events in a more global way.

- **CSS Styling with Event Handlers:** You discovered that you can tap into the CSS style sheet through your external JavaScript file to identify and highlight links in your nav menu. There are so many ways you can apply this concept!

Our code, however, is getting a bit lengthy and complex. When you are out googling for tips on how to accomplish things in JavaScript, you will occasionally come across jQuery code which is not quite as long or complicated to look at. However, this can open up a whole new can of worms of confusion. So, in the next lesson you'll learn what all the jQuery business is about, so that when you see it in other peoples' code, it's not quite so bewildering. See you there!

Lesson 10 Assignment

Separate the Code

Download

Below is the link to the ZIP which contains following files you'll need:

- *jssplitpage.html*
- *jssplitpage.txt*

These files contain the HTML and CSS, and most of the JavaScript code.

Download L10-Assignment.zip

1.7 KB (Gigabytes) ZIP

You can retype or copy and paste if you'd like to run through the code for the page, or you can simply download and save the files to your *Intro to JavaScript* folder.

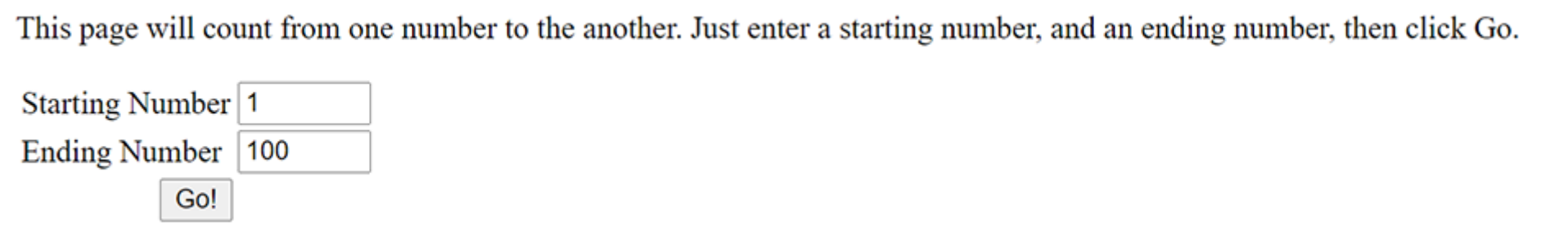
For this assignment, you're going to start with a webpage that contains some JavaScript code. Your goal is to split the code in this page into two files:

- **jsPage.html** – This page will hold your HTML and CSS.
- **jsCode.js** – This page will be your external JavaScript file.

Here are a few things to remember as you work through this exercise:

- **Make sure you link the page to the external JavaScript file.** This will ensure that the page still works with the same, even after the JavaScript has be moved to a separate file. Use the <script>... </script> tags to load the external JavaScript file with a src= attribute.
- **The JavaScript file cannot contain any HTML code.** You'll need to leave the <script>...</script> tags in the *jsPage* file (these are html tags), but remove them from the code you put in the *jsCode* file.

Go ahead and use the *jssplitpage.html* file as your starting point, it contains all the JavaScript you'll need to add to your *jsCode.js* file. Take it for a test drive in your browser to see how it works.



Then externalize the JavaScript and make sure it works the same after you've done that. See if you can do it on your own. Then you can look at the code below to see the answer.

Download

To check your code, you can download the correct code for both jsPage.html and jsCode.js here:

Download L10-AssignmentAnswers.zip

1.0 KB (Gigabytes) ZIP

Lesson 10 Resources for Further Learning

HTML < script > src Attribute (https://www.w3schools.com/tags/att_script_src.asp)

https://www.w3schools.com/tags/att_script_src.asp

Here is a short little reference to the <script src="..."> tag for lining to an external JavaScript file.

JavaScript HTML DOM EventListener (https://www.w3schools.com/js/js_htmldom_eventlistener.asp)

https://www.w3schools.com/js/js_htmldom_eventlistener.asp

Here is some documentation on the addEventListener method we used in this lesson.

JavaScript addEventListener() with Examples (https://www.geeksforgeeks.org/javascript-addeventlistener-with-examples/)

https://www.geeksforgeeks.org/javascript-addeventlistener-with-examples/

Click this link for some more info on addEventListener.

Document: DOMContentLoaded event (https://developer.mozilla.org/en-US/docs/Web/API/Document/DOMContentLoaded_event)

https://developer.mozilla.org/en-US/docs/Web/API/Document/DOMContentLoaded_event

Here is some good info on the DOMContentLoaded event that we used to delay JavaScript code execution in this lesson.

Make Your Code Cleaner with JavaScript Ternary Operator (https://www.javascripttutorial.net/javascript-ternary-operator/)

https://www.javascripttutorial.net/javascript-ternary-operator/

Here is another tutorial on that tricky ternary operator that lets you reduce if...then...else logit to a single line of code.

JavaScript String split() Method (https://www.w3schools.com/jsref/jsref_split.asp)

https://www.w3schools.com/jsref/jsref_split.asp

Click this link for info on the .split() method that we used to break up a URL into smaller pieces.

JavaScript Array pop() Method (https://www.w3schools.com/jsref/jsref_pop.asp)

https://www.w3schools.com/jsref/jsref_pop.asp

Visit this page for information on the .pop() method, which we used in this lesson to pop the file name of the current page off the end of the URL in the browser's Address bar.

Element.getElementsByTagName() (https://developer.mozilla.org/en-US/docs/Web/API/Element/getElementsByTagName)

https://developer.mozilla.org/en-US/docs/Web/API/Element/getElementsByTagName

Here's a good reference on the .getElementsByTagName() method that lets you get all the elements on the page, or within some block on the page, for looping through with JavaScript.

HTML DOM querySelectorAll() Method (https://www.w3schools.com/jsref/met_element_queryselectorall.asp)

https://www.w3schools.com/jsref/met_element_queryselectorall.asp

I didn't have room to talk about this one in the lesson. But it's similar to .getElementsByTagName, but it lets you find elements by other criteria, such as all the elements on the page that contain class= followed by some CSS class name.