ARCH A4988

Coding for Spatial Practices

<DOM Manipulation>

Learning Objectives

Learning Objectives

Dynamically modifying your website

1. Explain what the DOM is and how it is structured
2. Target DOM elements using JavaScript selectors
3. Change the attributes or content of a DOM element
4. Explore JavaScript methods for DOM manipulation and traversal

Agenda

Lab Time

DOM Manipulation

Project 03 – Catalog

Review

Class Exercise

Part I

Using your new knowledge of some simple JavaScript functionality, to solve the following:

Imagine you work the information booth at a theme park and help recommend rides to guests.

1. Declare a variable age. Assign it the value 25.
2. Declare a variable height. Assign it the value 5.
3. Log each variable to the console and hit the "Run" button in the console panel. Example: console.log(age)

Class Exercise

Part II

Write out an if / else if / else statement for the following conditions:

1. If a person is less than 8 years old, recommend the merry-go-round.
   console.log("Check out the Merry-Go-Round. You'll love it!");
2. Otherwise if a person is more than 8 years old AND less than 65 years old AND more than 4.5 feet tall, recommend the roller coaster.
   console.log("Check out the Roller Coaster. It's awesome!");
3. Otherwise recommend the lazy river
   console.log('Why not enjoy a float down the Lazy River?');

Class Exercise

Sentence Generator

1. Create three variables called noun verb and adjective and store one of each type.
2. Choose a short one sentence poem that includes the following variables:
   - Sample sentence: `My ${noun} leaps ${adjective} when I ${verb} a rainbow in the sky:`
3. Create five different versions of this sentence with different variables.
4. Style the HTML pages.

Class Exercise

Sentence Generator

```
1.  Make Make a list of at least five words for each variable:
    - Sample array: let nouns = [ ‘heart’, ‘rainbow’, ‘ocean’];.
2.  Create a randomly generated sentence by using the variables.
    - Sample sentence: `My ${noun} leaps ${adjective} when I
    ${verb} a rainbow in the sky:`
3.  Style the HTML page.

Hint: Formula for selecting a random element from an array
let item = arrayName[Math.floor(Math.random()*arrayName.length)];
```

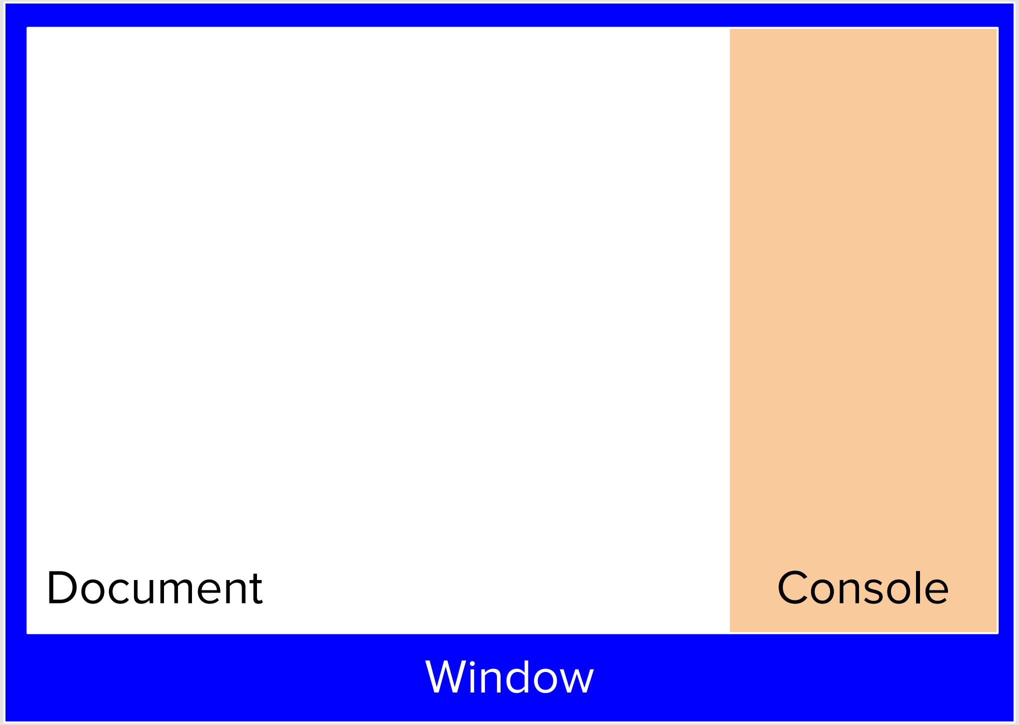
Document Object Model (DOM)

Everything you see in the browser is a JavaScript object.

Three Big Objects

Window	Document	Console
The whole web browser; mostly used for browser-level settings.	The current webpage. This object has the functionality we want to use when accessing elements on the page.	A scratch pad for development-related messages; highly useful in debugging.

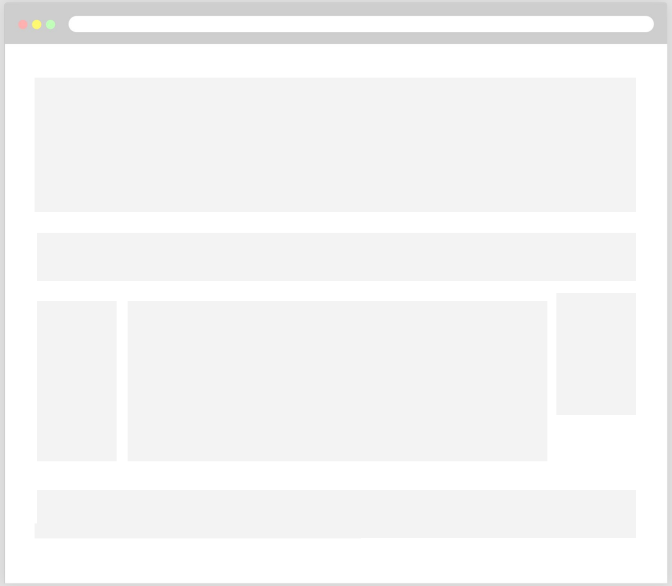
A Webpage Is One Giant JS Object



The Document Object Model (DOM)

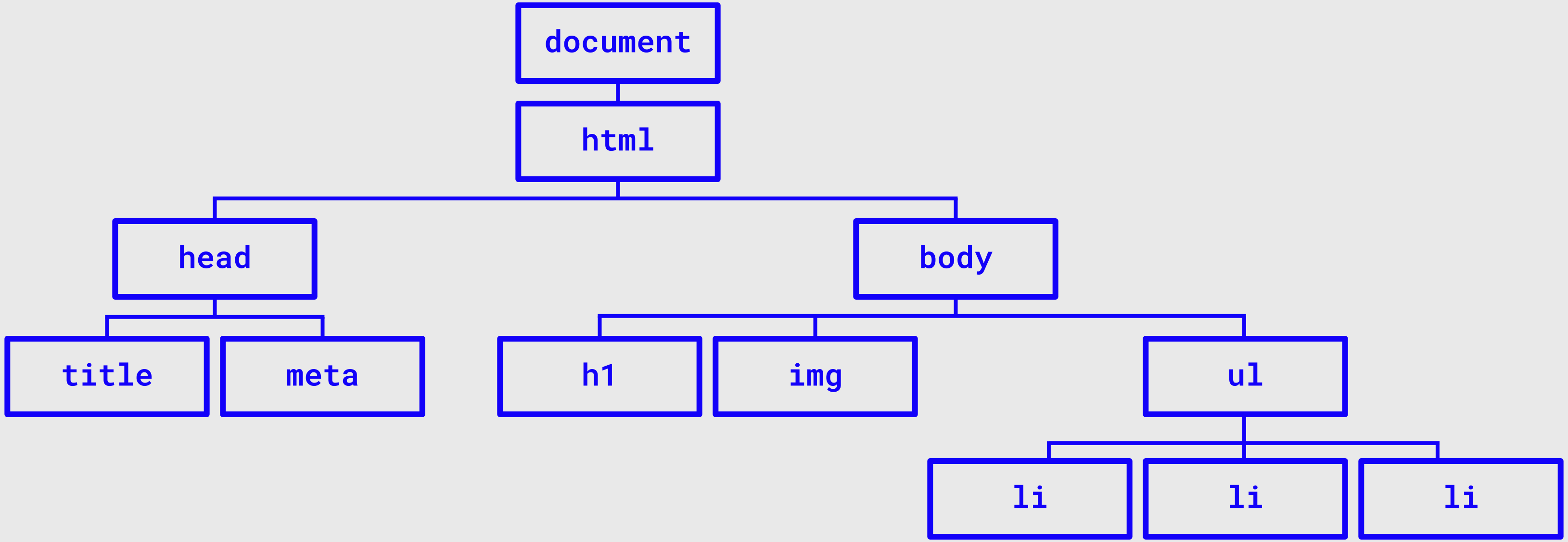
Browsers read your HTML and create an object in the computer's memory for each part. That HTML layout is called a "data model" because it describes the structure of your webpage.

The **Document Object Model** (DOM) is the browser's JavaScript representation of your HTML elements.



Working with the DOM

Document Object Model

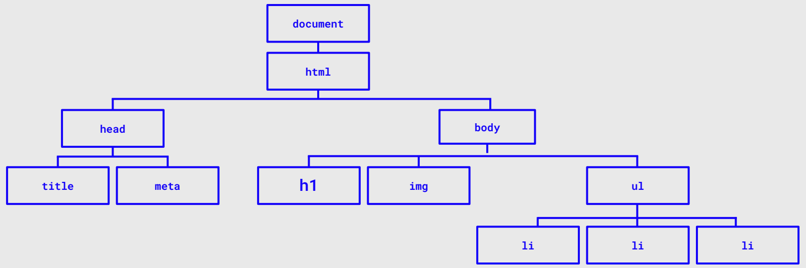


Working with the DOM

Document Object Model

It represents the page so that programs like JavaScript can change or manipulate its structure, style and content.

A web page is a document. This document can be displayed in the browser window or as the HTML source. In both cases it's the same document.



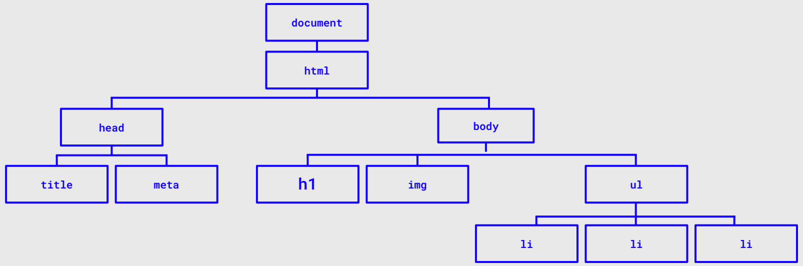
Working with the DOM

Document Object Model

Each web page loaded in the browser has its own document object. The document interface serves as an entry point to the web page's content.

To access the document object, call:

```
window.document
```

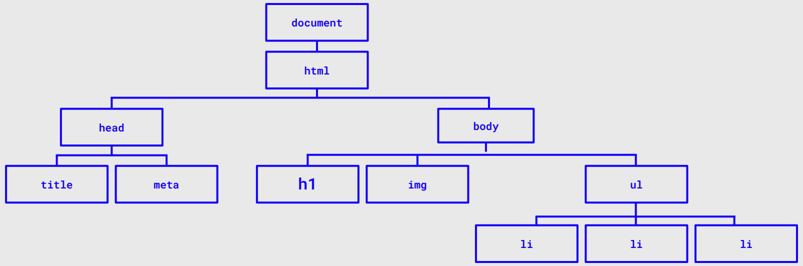


Working with the DOM

Document Object Model

Understanding the DOM is central to working in JavaScript. JavaScript uses the DOM to create dynamic HTML, including:

- adding new HTML elements or attributes,
- changing CSS styles on a page,
- removing existing elements or attributes, and more.

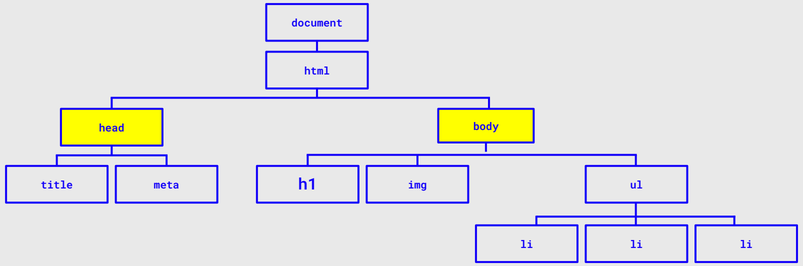


Working with the DOM

Nodes

Everything in the DOM exists as a node. In a standard HTML document you'll have an HTML object containing two nodes: head and body.

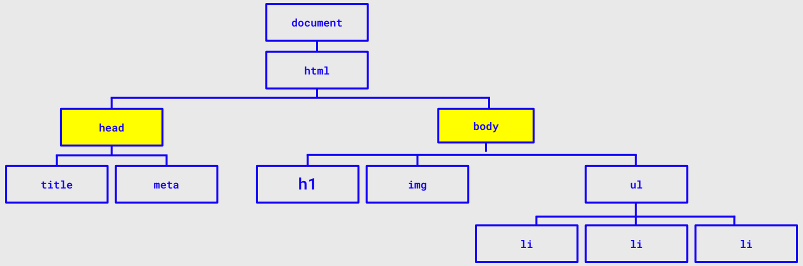
The <head> holds all the invisible objects like title, link, meta, script, while the <body> holds all the visible nodes in the viewport.



Working with the DOM

Nodes

HTML elements are called element nodes, attributes are called attribute nodes, the text inside elements are called text nodes.



Working with the DOM

Accessing Elements

There are two types of elements one would need to deal with, existing and non-existing.

Let’s start with existing elements.

Getters and Setters

The main thing we're doing with JavaScript is getting objects from the DOM and performing actions with them (moving, hiding, etc).

The methods that get something from a webpage are called getters.

The methods that change something on the webpage are called setters.

Properties

Objects often have metadata — information that describes the object (height, width, classes, etc). These pieces of information are called **properties**.

Property	**Description**
`someElement.classList`	A list of the classes belonging to a DOM element.
`someElement.id`	The ID of an element, if it has one.
`someElement.style.color`	The color of an element’s text.
`window.location.href`	The window object’s location details, including the page’s href (hypertext reference/URL).

What Does a Real Piece of Code Look Like?

$$\underbrace{\texttt{document}}_{\text{Object}}\texttt{.}\underbrace{\texttt{getElementById}}_{\text{Method (getter)}}\texttt{(‘}\underbrace{\texttt{gsapp}}_{\text{Parameter}}\texttt{’);}$$

Working with the DOM

Getters

Getters often fill in variables in your JavaScript. Once you get something from the DOM, you can use a variable to store it in memory for future manipulation.

Working with the DOM

Getters

```
// HTML
<div id='special'>my special element</div>

// JavaScript
let myElement = document.getElementById('special');
console.log(myElement);
```

Working with the DOM

Getters

Then, we pass in a string that matches the ID of an element in our HTML.

```
// HTML
<div id=”special”>my special element</div>

// JavaScript
let myElement = document.getElementById(“special”);
console.log(myElement);
```

Working with the DOM

Getters

Now that we have our element, myElement, we can access its properties:

```
// JavaScript
myElement.style.color;
myElement.innerText;
myElement.classList;
```

Guided Walk-Through:

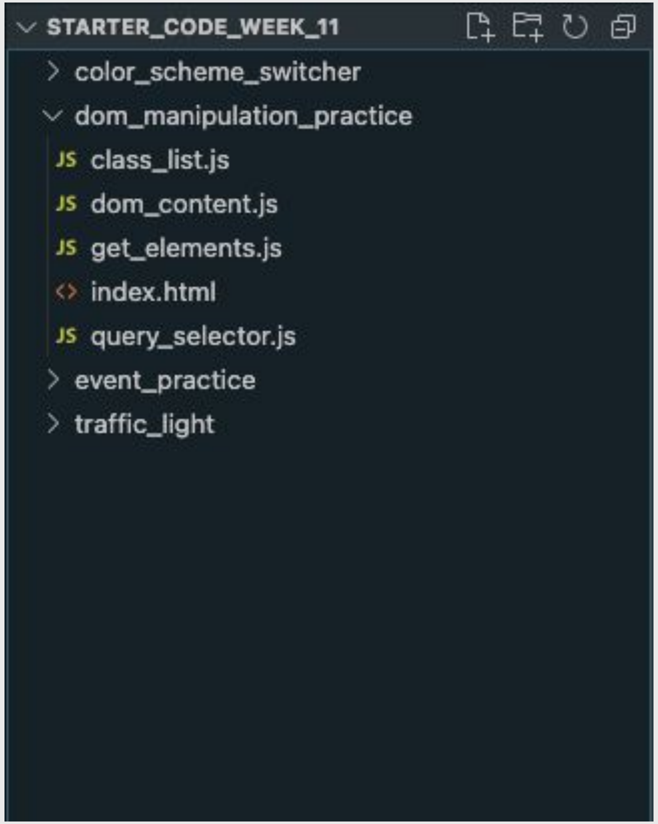
Working with the DOM

20 minutes

Practice getting elements from the DOM using getElement(s)By.

Instructions:

Go to the Google Drive course folder, **Week-11: DOM Manipulation** and download **starter_code_week_11** into your course folder. Open the file, **get_elements.js**


```
STARTER_CODE_WEEK_11
  > color_scheme_switcher
  v dom_manipulation_practice
      class_list.js
      dom_content.js
      get_elements.js
      index.html
      query_selector.js
  > event_practice
  > traffic_light
```

Working with the DOM

Query Selector

There are only two methods in this group: *querySelector* and *querySelectorAll*.

querySelector returns a single value, and *querySelectorAll* returns everything that it can find.

Working with the DOM

querySelector()

We pass in a CSS selector for the element we want to retrieve from the DOM.

The element that we get back will be the first element that matches that selector.

```
// HTML
<div class=”boxes”>
  <div class=”box”>first</div>
  <div class=”box”>second</div>
</div>
```

```
// JavaScript
let myElement =
document.querySelector(“.box”);
console.log(myElement);
```

Working with the DOM

querySelector()

We pass in a CSS selector for the element we want to retrieve from the DOM.

The element that we get back will be the first element that matches that selector.

```
// HTML
<div class="boxes">
  <div class="box">first</div>
  <div class="box">second</div>
</div>
```

```
// JavaScript
let myElement =
document.querySelector(".box");
console.log(myElement);
```

Working with the DOM

querySelectorAll()

The element method, *querySelectorAll* targets all the elements on the page matching the selector we pass in.

```
// HTML
<div class=”boxes”>
  <div class=”box”>first</div>
  <div class=”box”>second</div>
</div>

// JavaScript
let myElements =
document.querySelectorAll(“.box”);

console.log(myElements);
console.log(myElements[0]);
```

Working with the DOM

querySelectorAll()

The element method, *querySelectorAll* targets all the elements on the page matching the selector we pass in.

```
// HTML
<div class=”boxes”>
  <div class=”box”>first</div>
  <div class=”box”>second</div>
</div>

// JavaScript
let myElements =
document.querySelectorAll(“.box”);

console.log(myElements);
console.log(myElements[0]);
```

Guided Walk-Through:

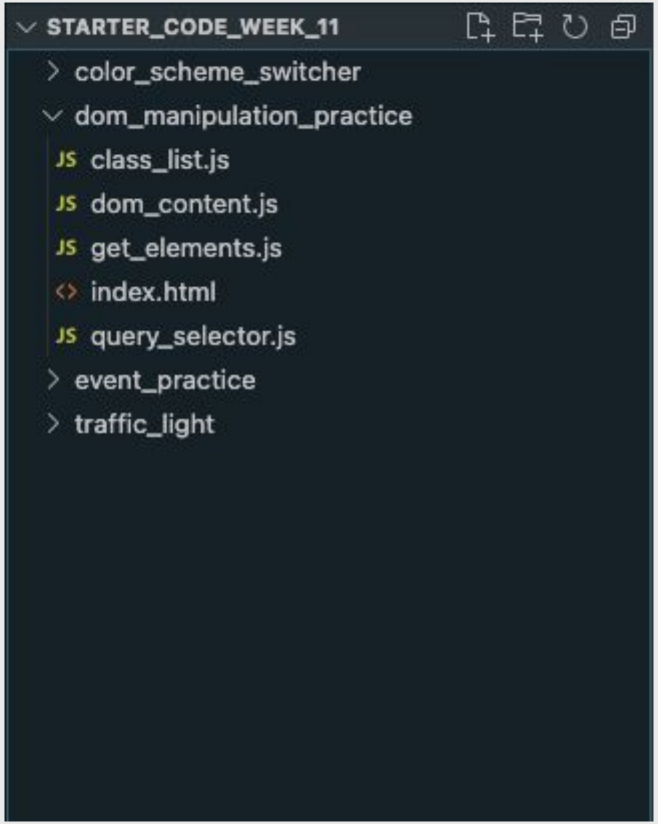
Working with the DOM

20 minutes

Practice getting elements from the DOM using querySelector(s)By.

Instructions:

Open the file, **query_selector.js**


STARTER_CODE_WEEK_11
- color_scheme_switcher
- dom_manipulation_practice
  - class_list.js
  - dom_content.js
  - get_elements.js
  - index.html
  - query_selector.js
- event_practice
- traffic_light

Working with the DOM

Manipulating Elements

Now that we are able to get elements from the DOM, let’s learn what we can do with them.

e.g. add or remove classes, replace the content with new content, move element from one part of the page to another.

Working with the DOM

Getting and Setting Attributes

Every node object has an attributes property where it lists it's attributes (like **href** and **src**).

You can get and set data using the getAttribute and setAttribute method.

```
// HTML
<div id=”special”>my special
element</div>
<div id=”regular”>my regular
element</div>

// JavaScript
let div =
document.querySelector(“div”);
div.setAttribute(“id”, “greeting”);
```

Working with the DOM

classList API

Every node has a classList property and there are methods we can use to add a class (addClass), remove a class (removeClass) or toggle a class (toggleClass).

```
// HTML
<div id=”special”>my special
element</div>
<div id=”regular”>my regular
element</div>

// JavaScript
let div =
document.querySelector(“div”);

div.classList.add(“visible”);
div.classList.remove(“visible”);
```

Working with the DOM

Content

Sometimes we have an element and want to change the text or HTML contained within that element.

We can use the node properties to reset the HTML or text of an element: innerHTML, outerHTML, innerText, outerText, textContent

```
// HTML
<div id=”special”>my special
element</div>
<div id=”regular”>my regular
element</div>

// JavaScript
let div =
document.querySelector(“div”);
div.innerHTML = “<p>Hey!</p>”;
```

Working with the DOM

Change the Style

Sometimes we may want to update or change the style of an element using JavaScript.

```
// HTML
<div id=”special”>my special
element</div>
<div id=”regular”>my regular
element</div>

// JavaScript
let div =
document.querySelector(“div”);
div.style[“width”] = “100px”;
```

Working with the DOM

Creating and Adding Elements

Now, let's focus on creating and manipulating new elements.

Step 1.

Create a new element, using .createElement()

```
// JavaScript
const listItem =
document.createElement(“li”);
```

Working with the DOM

Creating and Adding Elements

Step 2.

Do something with that element. e.g. we can add an attribute to it.

Using .classList.add() we've added a class to the list item element.

```
// JavaScript
const listItem =
document.createElement("li");

listItem.classList.add("list-item");
```

Working with the DOM

Creating and Adding Elements

Step 3. Append created element to HTML

The “listItem” element now needs to become part of the DOM in order to show up on the page. Therefore, we need to append it to HTML using .appendChild()

```
// JavaScript
const listItem =
document.createElement(“li”);

listItem.classList.add(“list-item
”);

ul.appendChild(listItem);
```

Working with the DOM

Removing Elements

Finally, to remove the HTML element we just created from the DOM:

```
// JavaScript
let deleteItem =
div.removeChild(listItem);
```

Guided Walk-Through:

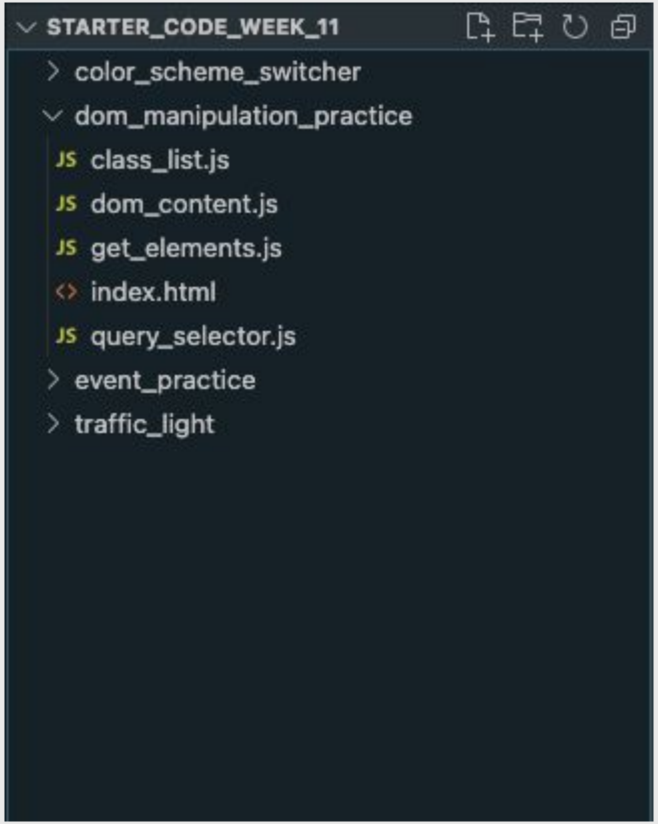
Working with the DOM

20 minutes

Practice getting elements from the DOM using node properties to reset the HTML or text innerText, innerHTML, and textContent.

Instructions:

Open the file, **dom_content.js**



```
STARTER_CODE_WEEK_11
  > color_scheme_switcher
  v dom_manipulation_practice
      class_list.js
      dom_content.js
      get_elements.js
      index.html
      query_selector.js
  > event_practice
  > traffic_light
```

Guided Walk-Through:

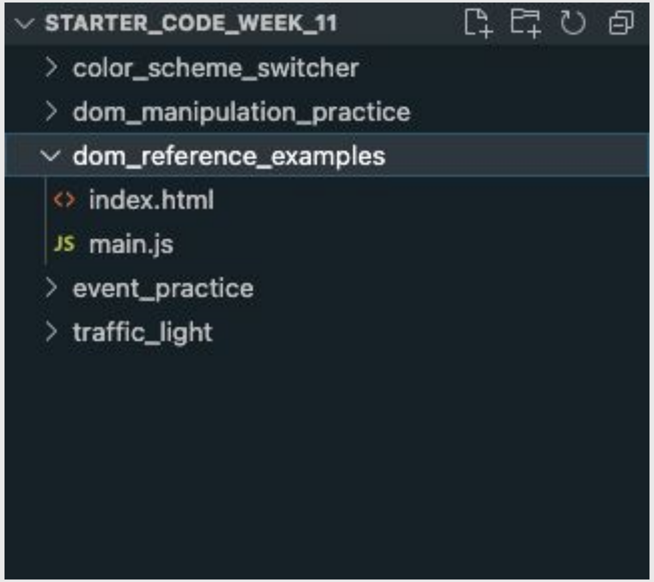
Working with the DOM

20 minutes

Let's look at some DOM properties and methods in action! There's no need to feel overwhelmed by details — the patterns are what matter.

Instructions:

Open the folder, **dom_reference_examples**


STARTER_CODE_WEEK_11
color_scheme_switcher
dom_manipulation_practice
dom_reference_examples
index.html
main.js
event_practice
traffic_light

Event Handling

Events and Listeners

Anytime a user interacts with a webpage, the browser classifies that action as an **event**.

In our JS code, we can listen for events in the browser and trigger functions in response using **event listeners**.

```
// When object is clicked, the actionFn function is called

object.addEventListener(‘click’, actionFn)
```

Get, Then Listen

We'll often **get** an element and then **set** an event listener on it. Once the event occurs, the listener will execute the function it was given.

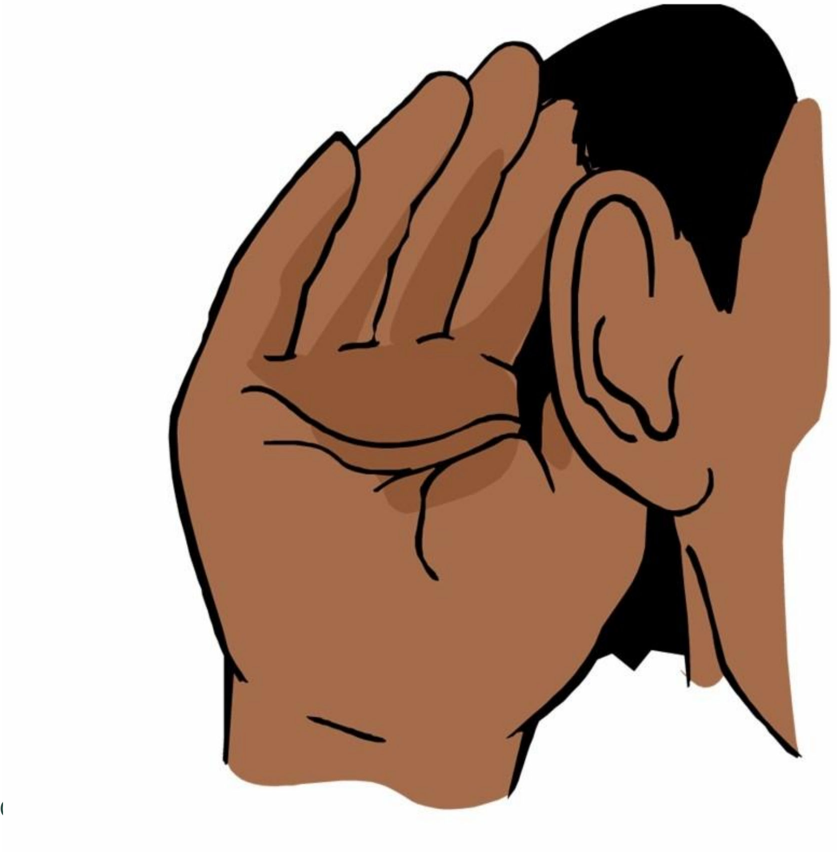
```
const gsapp = document.getElementById('gsapp');

function sayHello(){
    console.log("hello!");

}
gsapp.addEventListener('click', sayHello)
```

Discussion:

Events to Listen For



Click is the most common event; but what other events might we want to listen for?

Basics of Interaction

Types of Events

There are many events that can trigger a function. Here are some commonly used ones:

- click
- scroll
- keydown
- hover

Basics of Interaction

Types of Events

There are two steps to working with events:

1. We set up an event listener with .addEventListener
2. We define an event handler, a function that gets passed to .addEventListener

Basics of Interaction

Setting up an Event Listener

In order to listen for an event, we need to define an event listener. Below you’ll find a simple event listener associated with a 'click' event on a button element.

First, we target the button with a class name js-button:

```
const button = document.querySelector('.js-button');
```

Basics of Interaction

Setting up an Event Listener

Then, we tell JavaScript to listen for an event:

```
button.addEventListener('click', handleClickEvent);
```

* The first argument is the event, the second argument is the function (event handler) that will run once the button is clicked.

Basics of Interaction

Setting up an Event Handler

Now, we define the function that will be called whenever this event is emitted.

This is just a function, but it has a special name due to how it’s being used – a callback function:

```
function handleClickEvent() {
 console.log('I was clicked!');
}
```

Basics of Interaction

Putting it all together

```
<div class='js-button'></div>
```

html

```
const button =
document.querySelector('.js-button');

button.addEventListener('click', handleClickEvent);

function handleClickEvent() {
 console.log('I was clicked!');
}
```

javascript

select DOM node; tie event listener to DOM node; listen for click; run the function.

Guided Walk-Through:

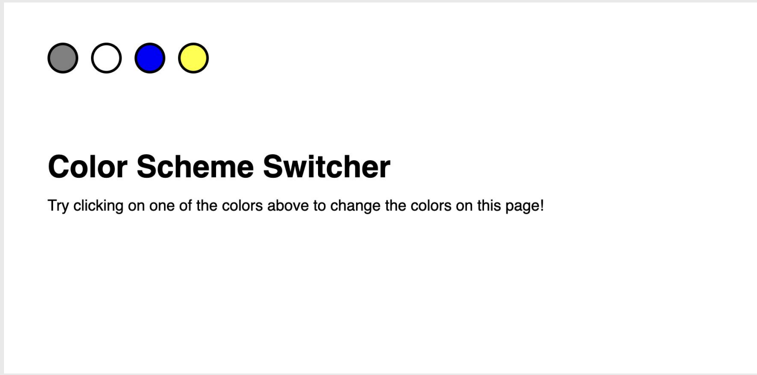
Color Scheme Switcher

20 minutes

This is an exercise in finding elements on the web page. Your task is to write JavaScript that selects each element. Then, add an event listener to each element. Finally, write the event handler that changes the color of the page.

Instructions:

Open the folder, **color_scheme_switcher**



Color Scheme Switcher

Try clicking on one of the colors above to change the colors on this page!

Guided Walk-Through:

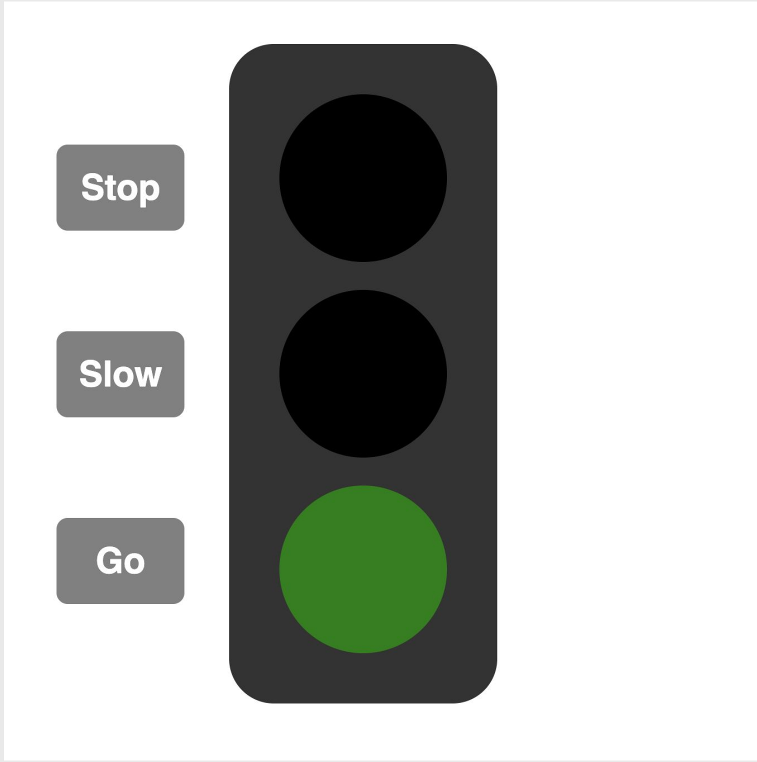
Traffic Lights

20 minutes

Try making this traffic light work - Think through the problem BEFORE you code! You'll use addEventListener and getElementById if you're doing it right.

Instructions:

Open the folder, **traffic_light**



Project

Project 03

Deliverable 03 << 11/19/2024 >>

Create a collection of 100 (visual) media items. These can be physical objects that you document, screenshots, found images, videos, original images, etc. If you want to collect quotes or text, it will have to be displayed with an image-based method.

We will add these into a database using Google Sheets, *(you have the option to swap collections with one of your peers)*. We will then make an online experience that connects with the database in order to learn how to pull structured data that is not yours and use it to populate a website. (What is structured data? For example, on a recipe page, what are the ingredients, the cooking time and temperature, the instructions, etc).

Project 03

Deliverable 03 << 11/21/2023 >>

1. Finally, choose your design direction and start building it out in code according to your chosen design – using the resources and lectures made available to you this semester.
2. Things to keep in mind:
   a. Select a platform for hosting your images (e.g. Google Drive, GitHub etc)
   b. Optimize your images for the web (remember: large images take long for your webpage to load)