**Milestone 5 || Part 1 || DOM Tour || Event Handling**

**Resources**

[DOM Visualizer](https://bioub.github.io/dom-visualizer/)

* GitHub Repo:

<https://github.com/foy4748/session-milestone-5-play-with-dom>

* Minimal Shopping Cart (Template):

[Tailwind product gallery - Tailwind CSS Example](https://tailwindflex.com/@jaxstone/tailwind-product-gallery)

* For your own try:

[Tailwind Awesome](https://www.tailwindawesome.com/?price=free)

* Used Apps
  + gInk : ‣
  + NeoVim: <https://neovim.io/>
  + My NeoVim Config: ‣

**What is DOM?**

[DOM Visualizer](https://0xedward.github.io/dom-visualizer/)

[DOM Visualizer](https://bioub.github.io/dom-visualizer/)

DOM stands for *Document Object Model*.

Document refers to the HTML document. Object Model is the object that represents the HTML document, which

* Represents the Document as JavaScript Object, maintaining the parent-child relationship between the html elements/tags in a Tree-Like Structure. [ Click the [DOM Visualizer](https://bioub.github.io/dom-visualizer/) ]
* Includes all applied CSS styles to that document.
* Includes all necessary Functions/Methods that can CREATE, READ, UPDATE, DELETE the document Or, part of the document.

As I’ve said, DOM represent the document as JavaScript Object, then let’s see **Simplified JavaScript Object** equivalent of the HTML at the left column

<!DOCTYPE html>

<html lang="en">

<head>

<title>Learn DOM operations</title>

</head>

<body>

<header class="text-right">

<h1>Shopping Cart</h1>

</header>

<main>

<p>Some contents</p>

</main>

<footer class="text-center">

<small>Learn DOM manipulation</small>

</footer>

<script src="/assets/js/index.js"></script>

</body>

</html>

{

"html": {

"childNodes": [

{

"head": {

"childNodes": [

{

"title": {

"innerText": "Learn DOM operations"

}

}

]

}

},

{

"body": {

"childNodes": [

{

"header": {

"className": "text-right",

"childNodes": [

{

"h1": {

"innerText": "Shopping Cart"

}

}

]

}

}

]

}

}

]

}

}

*Now, at least we know what we are dealing with. (Object Mostly)*

**Common Grounds**

Whatever you do in programming, every action can be categorized as these operations (**CRUD**)

1. CREATE - new data/information as needed.
2. READ - already created data
3. UPDATE - the data you’ve read or replace it entirely
4. DELETE - already created data

*Since the document/HTML is already a Tree-Like data structure holding some information. That means, the CREATE phase is already DONE, even though we’ve not written a single line of JavaScript code. So, we will begin the READ operations.*

**Read Operations / Traverse**

***(Traverse in terms of Tree structure) [***[***Wiki page***](https://en.wikipedia.org/wiki/Tree_traversal)***]***

In JavaScript, there are several functions/methods that can READ / Traverse the DOM, and returns a specific element / node of the document we want to work with

| **Returns HTMLCollection / expects string** | **Returns NodeList / expects CSS selector as string** |
| --- | --- |
| getElementsByTagName() | querySelector() |
| getElementsByClass() | querySelectorAll() |
| getElementById() |  |
| **—— Returns HTMLCollection** | **—— Returns NodeList** |

**Also, there is a method called getAttribute() which can read attribute written on a html tag.**

Suppose you’ve read a DOM element using functions mentioned in the table above. You can access its parent, children or sibling DOM elements. For example

const aSingleCard = document.querySelector(".card")

const cardContainer = aSingleCard.parentElement;

const cardBtn = aSingleCard.children[2]

**# Pro Tip [✅]**

You can perform READ operations like get functions, querySelector(), or querySelectorAll() functions on any valid DOM element to read its children elements

const aSingleCard = document.querySelector(".card")

const cardContainer = aSingleCard.parentElement;

// [❌] Instead of doing these

// const cardBtn = aSingleCard.children[2]

// [✅] DO THIS

const cardBtn = aSingleCard.querySelector("button")

**Write Operations**

Let's consider you’ve read a **Single Element** using getElementById()

For example

const cardContainer = document.getElementById("card-container")

console.log(cardContainer)

/\* OUTPUT

<section id="card-container">

<div class="card card-1">Card 1</div>

<div class="card card-2">Card 2</div>

<div class="card card-3">Card 3</div>

<div class="card card-4">Card 4</div>

</section>

\*/

**Update**

As you can access any DOM elements using the functions mentioned in the *Read Operations / Traverse* section, you can alter its behavior or property simply by assigning values.

**# Style Update**

All the DOM elements/nodes have a property called style. You can set any CSS property using style object like below.

cardContainer.style.backgroundColor = "#eee" // light gray bg

cardContainer.style.display = "flex"

cardContainer.style.flexDirection = "column"

cardContainer.style.justifyContent = "center"

// Also you can read existying style property

// console.log(cardContainer.style.display) // prints "flex" in console

**Note:** You can see that, if CSS property which contains - , style uses it in CamelCase.

For example. CSS property justify-content will be accessed or used as justifyContent .

Also, you can add or remove or toggle CSS class to achieve styling of a DOM.

If you ensure proper TailwindCSS installation, you can use TailwindCSS classes

//cardContainer.style.backgroundColor = "#eee" // light gray bg

cardContainer.classList.add("bg-slate-50") // light gray bg

cardContainer.classList.add("flex")

cardContainer.classList.add("flex-col")

cardContainer.classList.add("justify-center")

// Or You can do this in ONE single line

cardContainer.classList.add("bg-slate-50","flex","flex-col","justify-center")

cardContainer.classList.toggle("active") // Toggle Between Active status

// Also you can read existying classes

// console.log(cardContainer.classList) // prints applied classes in console

**# # Pro Tips [✅]**

**Using style object Vs classList.add() / classList.remove() / classList.toggle()**

Generally, you shouldn’t use the style object, for styling a DOM element. Why? Becasue style applied using style object directly appield as Inline CSS. That’s why, the CSS written in .css file won’t work normally. “Handling styles in CSS file and handling JavaScript in JS file” - This simple good practice is violated.

**# Text Update**

You can update a text content like this

const cards = cardContainer.getElementsByClassName("card")

cards[0].innerText = "Changed Text of Card 1"

**# Attribute Update**

cards[0].width = "500"

// Updating DaisyUI Theme

// Turns on the light theme using JS || Yay!

const htmlTag = document.querySelector("html")

htmlTag.setAttribute("data-theme", "light")

**Create**

**# HTML Create, then Update**

Yes!! You can entirely throw out HTML code of a section, then add your own HTML!!

cardContainer.innerHTML = ""

// Made the cardContainer empty!!! Card Container will look like below

// <section id="card-section"></section>

// Creation of HTML tags

const newCard = `

<div class='card flex justify-between'>

<p> Chocolate </p>

<p> $32.00 </p>

<button> Buy </button>

</div>

`

cardContainer.innerHTML += newCard

// Now cardContainer will look like below

/\*

<section id="card-section">

<div class='card flex justify-between'>

<p> Chocolate </p>

<p> $32.00 </p>

<button> Buy </button>

</div>

</section>

\*/

Also, you can create elements using the createElement() function, then append it.

cardContainer.innerHTML = ""

// Made the cardContainer empty!!! Card Container will look like below

// <section id="card-section"></section>

// Creating Product Card Div

const newCardDiv = document.createElement("div")

newCardDiv.classList.add("card", "flex", "justify-between") // Styling

// Creating Product Name and Price <p> tags

const productName = document.createElement("p")

const productPrice = document.createElement("p")

const productBuyBtn = document.createElement("button")

// Populating texts within

productName.innerText = " Chocolate "

productPrice.innerText = " $32.00 "

productBuyBtn.innerText = " Buy "

// IMPORTANT

// Append the productName and productPrice

// within the Product Card Div

newCardDiv.appendChild(productName)

newCardDiv.appendChild(productPrice)

newCardDiv.appendChild(productByBtn)

// [!] Most IMPORTANT

// Append the newly created Product Card Div

// within the cardContainer

cardContainer.appendChild(newCardDiv)

/\*

<section id="card-section">

<div class='card flex justify-between'>

<p> Chocolate </p>

<p> $32.00 </p>

<button> Buy </button>

</div>

</section>

\*/

**# # Pro Tips [✅]**

Should I use createElement() / append() Or simply use innerHTML+= trick?

There is a trade-off between these two methods.

* createElement() / append() method provides a secure way of creating and appending a DOM element into the DOM tree, because these are created/appended using built-in JavaScript functions.

But the development experience (DX) or coding experience is bad this way. Because it requires calling many many functions just to create a simple thing.

* Using the innerHTML+= trick is way easy and fast way of creating a DOM element using Simple HTML syntax within JS string and add it to the DOM tree.

But, in this case, the desired DOM is written in simple JavaScript String, where anyone can write wrong HTML code accidentally. **Debugging a broken HTML code is way harder than debugging a typical code.**

***[✅] To safely use the innerHTML+= trick, always write your HTML code in an HTML file first. Check it, style it, adjust it. Then copy it into a JavaScript file for populating DOM elements using that piece of code.***

**Delete**

To remove a DOM element

// [✅] This is more convenient

cardContainer.remove() // For self removal

// [❗] Rather than this 😐

cardContainer.removeChild(cardContainer.children[4]) // For child removal

// ✨ Magic Trick 💣

// Run this command in the browser console

// Then try to click anything on the Web Page

const allElements = document.querySelectorAll("\*")

for(let elm of allElements){

elm.addEventListener("click", function (event) {

// Overriding natural behavior of clicking a thing

event.preventDefault()

elm.remove() // Removing the Clicked Element !!

event.stopPropagation() // Also try without this line

// Then try to understand why everything just disappeared

// In One Click ❗❗

// Hint: Event Bubbl

})

}

**Event**

In terms of web development, all possible **User Interaction** is considered as **Event**. JavaScript can capture these events, then execute a function. Events usually happens in context of DOM, that means, it occurs on a DOM element. Some examples of events are given below.

| **click** | **input** | **focus** |
| --- | --- | --- |
| mouseenter | mouseover | change |
| mouseleave | mousedown | blur |

Feel free to visit <https://www.w3schools.com/js/js_events_examples.asp> or <https://www.javatpoint.com/javascript-events> for more.

**Event Listener**

DOM elements can be accessed using any methods mentioned in the *READ* section. Then, **on a single DOM element**, an event listener can be applied, which will listen to a **User Interaction / Event** on that element. So, **Event Listener** is simply a function which sits on a DOM element, and waits until the desired event happens, then it will assign the event to another function, where programmer can write code.

Using JavaScript, event occurred on a DOM element can be captured like this

// Accessing all the cards within cardContainers div

const cards = cardContainers.getElementsByClassName("card")

for(let singleCard of cards){

// Accessing the button within a singleCard

const cardBtn = singleCard.querySelector("button")

// Then this is event listener function

// listening to click events on that card

cardBtn.addEventListener("click", function(){

// Write your code

// This code will execute when

// the card button is clicked

console.log("Card Button Clicked")

})

}

Or you can do this

// Accessing all the cards within cardContainers div

const cards = cardContainers.getElementsByClassName("card")

for(let singleCard of cards){

// Accessing the button within a singleCard

const cardBtn = singleCard.querySelector("button")

// listening to click events on that card

cardBtn.onclick = function(){

// Write your code

// This code will execute when

// the card button is clicked

console.log("Card Button Clicked")

}

}

Or you can also do this

<section id="card-section">

<div class='card flex justify-between'>

<p> Chocolate </p>

<p> $32.00 </p>

<!-- Event is already listening here -->

<button onclick="buyProduct(32)"> Buy </button>

</div>

</section>

// But this is Event Handler

function buyProduct(price){

console.log(price)

console.log(this, button is clicked)

}

**Event Handler Function**

The function that executes when an event occur is the **Event Handler Function**. Event listener just listens to the event. It awaits till an event.

But, whenever an event occurs, the Event Listener execute the **Event Handler Function.** Event listener passes all the event information within the event object, then, it passes these to the **Event Handler Function**

// Event Listener || Event Handler Function

cardBtn.addEventListener("click", function(){

console.log(cardBtn)

})

Or, you can do this

// Event Handler Function

function handleButtonClick(btn){

console.log(btn)

}

// Event Listening || This is also 👇 a handler function

cardBtn.addEventListener("click", function(){

// Calling this handler

handleButtonClick(cardBtn)

})

**Session Outline Planning (Rough)**

/// SESSION PLAN

// Talk about DOM

// Read DOM Elements from `Read` section

// Show parent child relationship

// Go To Project 1 -------------------

// Grab images using document.getElementsByTagName("img")

// Grab all cards using document.getElementsByClassName("card")

// Remove Confusion about ArrayLikeObject and Singular DOM Element

// Change DaiyUI theme using getAttribute() and setAttribute() functions. Button Toggling

// Interview Question: Why HTML is not a Programming Language ?

// Before Going to Project 2 ----------

// Talk about Event Listener using Project 1

// Talk about Styling using JS

// At the beginning of the Project 3 ---

// Talk about Differences between HTMLCollection and NodeList

// Make sure teach data-attribute in Project 5 [Expense calculation]

// This session will ensure ===========

// Updating Style: Create a NavBar with Active Status

// Prevent selecting same thing twice.

// Limited selection of things.

// Apply something Only Once like - Discount Coupon

// Includes disabling button