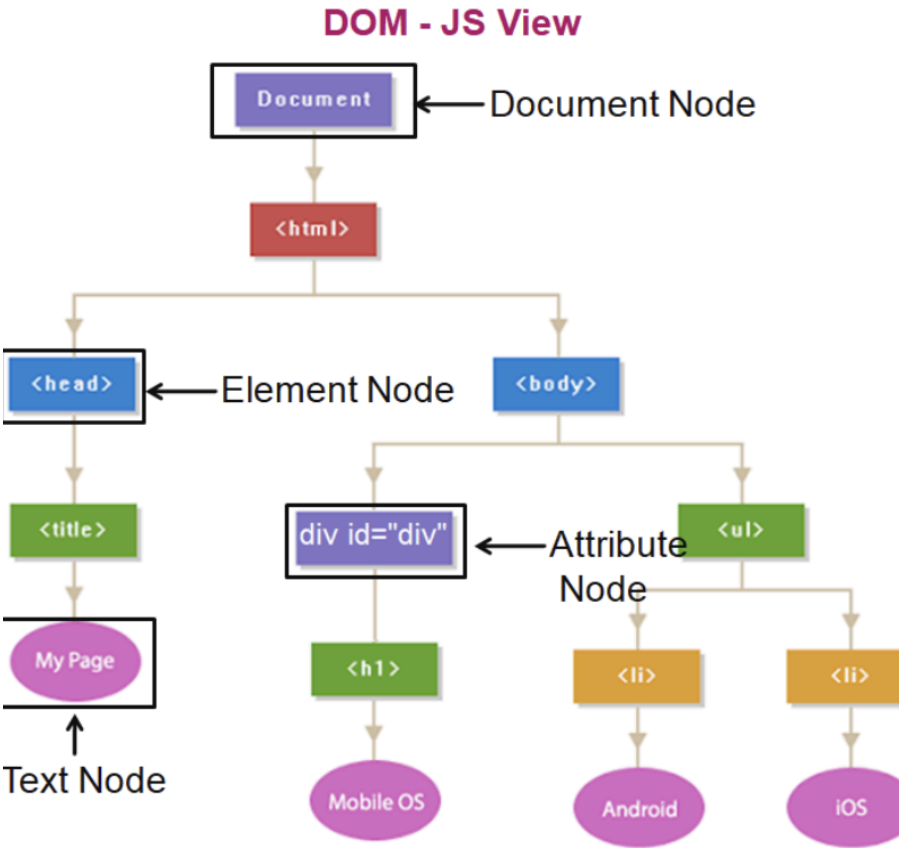
Class 15 - Introduction to DOM

title: Introduction to Document Object Model

1. The Document Object Model (DOM) is a programming interface for web documents.
2. Lets break this down
    a. Document here is the web document
    b. Object is the in memory representation of the web document as object in browser
3. It represents the page so that programs can change the document structure, style, and content.
    a. We call this the 'Document' because it represents the whole webpage.
    b. Now, just like a family tree branches out, the website tree does too. Each branch is called an 'Element,' and these branches represent the different parts of the webpage, like paragraphs, images, or buttons—basically, anything you can add to a webpage using HTML tags.
    c. Some items have special tags that tell us more about them. In web development, these are called 'Attributes.' Attributes can tell you what class an element belongs to, what its ID is, or how it should look (its style tag).
    d. Lastly, there's the actual words you read on a website. These words are like leaves on the tree. In the website

world, we call this 'Text.' It's the content within the elements that you can see and read.

e.



4.

5. So, when you're looking at a website, you're actually looking at a big family tree of elements, attributes, and text. This family tree is what web developers work with to add, change, or remove parts of a webpage."

Importance of DOM

1. Understanding the DOM is crucial for web developers because it:
    a. Provides a structured representation of the document.

    b. Allows JavaScript to access and manipulate the content and structure of a webpage dynamically.
    c. Enables the creation of rich, interactive web applications by providing **methods** to change the document content, structure, and styles.
2. JavaScript is used to put interactivity in DOM elements.

Manipulating the DOM with JavaScript

1. JavaScript can be used to interact with the DOM to:
    a. Add, remove, or modify elements: You can create new elements, delete existing ones, or change their properties.
    b. Change styles: Modify the CSS of elements to change their appearance.
    c. Respond to events: Set up **event listeners** to respond to user actions like clicks, keyboard input, or page loading.

Problem: On clicking the button append hello to the page

1. Approach
    a. Step 1: Selecting the html element
        i. To select or identify a particular html element we have methods.
        ii. getElementById - If you want to select an element based on ID

iii. querySelector - If you want to select an element based on any selector like class id or a combination or a specific element

2. Lets say our button has an id and we will leverage that
3. Using getElementById

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF - 8" />
   <meta
     name="viewport"
     content="width = device - width, initial - scale =
1.0"
   />
   <title>Document</title>
 </head>
 <body>
   <button id="btn">Say Hello</button>

   <script>
     // on clicking the button append hello to the page

     let btnElem
 = document.getElementById("btn");
     console.log(btn); // shows the button
   </script>
 </body>
</html>
```

a. Recapping document - t's an object that gives you access to all the elements in the webpage, like buttons, images, text, etc.
b. getElementById - This is a method (a function that belongs to an object) you call on the document object. It looks through the entire webpage for an element that has an ID attribute equal to 'btn-1'

4. Code: Using querySelector by ID

```
let btn = document.querySelector("#btn");
console.log(btn);
```

a. Method Used: document.querySelector('#btn')
b. Purpose: Selects the first element that matches the specified CSS selector. In this case, it selects the element with the ID btn. querySelector is more versatile than getElementById because it can select elements based on classes, attributes, and more complex selectors. here you will just need to pass the exact identifier as well for id # for class . and so on

5. Using query selector by class

```
<body>
   <button id = "btn-1"> Say Hello </button>
   <button class = "btn-2"> Say Bye </button>
```

```
    <script>

        // on clicking the button append hello to the page

        let btn = document.querySelector('.btn-2')
        console.log(btn)

    </script>
</body>
```

a. Method Used: document.querySelector('.btn-2')
b. Purpose: Selects the first element that matches the specified class. This is useful when you're working with CSS classes instead of IDs or need to select elements based on other attributes or complex selectors.

6. How will the code change if I want to select an element tag using query selector

```
let btn = document.querySelector('button')
```

7. Now we know how to select element, let us make it do something
8. Before we dive into making the button functional, it's crucial to understand what an event is in the context of web development.

Events

1. What is an Event?

2. An event refers to any action that triggers a specific response . For instance, clicking a button is and event which will trigger the action to display "Hello" on the screen.
3. There are generally two main input devices (mouse and keyboard) who events we listen to
4. Event is much more than this but for now just understand this, We will deep dive into events moving forward
5. Now how to make a element respond to an event?
6. Comes into the picture event listeners!
7. Method - addEventListener:
    a. The addEventListener method allows us to attach an event to any DOM element. This method is essential for interactive web development, enabling dynamic responses to user actions.
8. code

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF - 8" />
   <meta
     name="viewport"
     content="width = device - width, initial - scale =
1.0"
   />
   <title>Document</title>
 </head>
 <body>
   <button id="btn">Click Me</button>
```

```
  <script>
    // Target the button and attach a click event listener
    const btnElem = document.querySelector("#btn");

    btnElem.addEventListener("click", function (e) {
      console.log("button Clicked");
    });
  </script>
 </body>
</html>
```

9. btn.addEventListener('click', ...): This method is called on the button element (btn). It listens for a "click" event, meaning it waits for the button to be clicked. When a click occurs, it executes the function provided as the second argument.
10. function(){ ... }: This is an anonymous function (a function without a name) that is executed when the button is clicked.
11. Infact this is a callback funciton which is called later when the event happens
12. See this whole thing in the Browser and the Browser's Console

Adding an element

1. We now know how to get an element, how to add listeners for event. The last part of the question was to add an element when the button is clicked
2. Step 3: CreateElement and AppendChild
3. So now to append hello on the click of the button we first need to create a element in which hello will be placed then we need to attach that element in our DOM tree in an appropriate position to get it appended and visible ,
4. We can do this by using two methods
    a. to Create an element where Hello will go we can use createElement method
    b. and to attach that element to DOM we can use appendChild , let's see how they work and how to use them
5. code

```
<script>
    /**
     * getElementById
     * querySelector
     */
    //    const btnElem = document.getElementById('btn');
    // selecting the element
    const btnElem = document.querySelector("#btn");
    console.log("elem", btnElem);
    // add a listener
    btnElem.addEventListener("click", function () {
      console.log("Button Clicked");
      // creating a new element
```

```
        const newElem = document.createElement("div");
        newElem.innerText = "Hello World";
        const body = document.querySelector("body");
        //append the new element inside body
        body.appendChild(newElem);
      });
      /**
       * createElement - create a new element
       * appendChild - add a new element to the parent
element
       * */
      //  const btnElem2 = document.querySelector('.btn-2');
      //    console.log("elem",btnElem2)
      // const btnElem3 = document.querySelector('button');
      //    console.log("elem",btnElem3)
    </script>
```

6. Code breakdown
    a. Creating a New Element:
        i. let divElem = document.createElement('div')
        ii. document.createElement('div') creates a new element. This method takes one argument, the tag name of the element to be created.
        iii. The new element is stored in the variable divElem.
        iv. This method (createElement) is crucial for dynamically adding new elements to the DOM (Document Object Model).

b. Setting the Inner Text of the Element:
    i. divElem.innerText = 'Hello World'
    ii. The innerText property of divElem is set to 'Hello World'. This means the text content of the newly created will be "Hello".
    iii. innerText is used here to define what text the element will display.
c. Selecting the <body> Element:
    i. let body = document.querySelector('body')
    ii. document.querySelector('body') selects the <body> element of the page.
    iii. The selected <body> element is stored in the variable body.
d. Appending the to the <body>:
    i. body.appendChild(divElem)
    ii. appendChild is a method used to append a node as the last child of a node. In this case, divElem (the newly created with the text "Hello") is appended to body (the <body> element of the document).
    iii. After this method is executed, the new will appear on the webpage as part of the document's body. Each click on the button will create and append a new with "Hello World" to the body.

Problem

1. You are given an HTML document containing an unordered list (<ul>) with list items (<li>) representing numbers 1 through 10. However, there is item#7 missing between the numbers 6 and 8. Your task is to insert it appropriately into the list to maintain the sequential order using JavaScript.
2. Solution approach
    a. Create a new list item (<li>) with the missing number as its content.
    b. Insert the new list item into the correct position in the list to maintain the sequential numbering
3. Steps
    a. Step 1: Getting all the List Items (NodeList)
        i. Now as you can see the whole list , the number 7 is missing , so now to put 7 in between 6 and 8 we will need access to the whole list
        ii. till now we have seen methods taht return only one element
        iii. So to get the whole list t we can use the method **querySelectorAll** and can pass the element name inside it
        iv. It enables you to find all elements within the document that match a specified CSS selector(s). Here's a breakdown of its functionality:
        v.

vi. Return Value: querySelectorAll returns a **NodeList** of all elements within the document that match the specified group of selectors. A NodeList is a collection of DOM nodes that can be iterated over like an array **but it is not an array** it is a array like structure which is indexed and that's why it can be iterated over

vii. Extra notes: the reason we say that is not an array but array like because it is not derived from the base Array class in JS means it does not have array methods like map, filter and reduce. This was intentionally done to keep the dom methods lightweight

b. code

```
<body>
   <ul>
     <li>1</li>
     <li>2</li>
     <li>3</li>
     <li>4</li>
     <li>5</li>
     <li>6</li>

     <li>8</li>
     <li>9</li>
     <li>10</li>
   </ul>

   <script>
     // Fix the list by inserting the missing element
using querySelectorAll and insertBefor
```

```
    let allItems = document.querySelectorAll("li");
    console.log(allItems);
  </script>
</body>
```

c. Add missing li

d. Code

```
  <script>
    // Fix the list by inserting the missing element
using querySelectorAll and insertBefor

    const allItems = document.querySelectorAll("li");
    console.log(allItems);
    const indexThatHas8 = allItems[6];
    const sevenElement =
document.createElement("li");
    sevenElement.innerText = "7";
  </script>
```

e. Now i need to add the new li in my list, so i need the ul because inside ul , I will add the li

f. Now the task at hand is to add an element before element 8 in the list

g. Code

```
<script>
    const allItems = document.querySelectorAll("li");
    console.log("allItems", allItems);
    const indexThatHas8 = allItems[6]; // item at
index 6 i.e. 8
```

```
    const seventElement =
document.createElement("li");
    seventElement.innerText = "7";
    // get the ul node
    const ulNode = document.querySelector("ul");
    ulNode.insertBefore(seventElement,
indexThatHas8);
    // parentNode.insertBefore(newNode,
referenceNode);
  </script>
```

Problem: Double Clicking Cards

1. ClassList
    a. The classList property in the Document Object Model (DOM) is a read-only property that returns a collection of the class attributes of the element.
    b. It provides a convenient way to access and manipulate the class names of an element.
    c. With classList, you can easily add, remove, and check classes, which is particularly useful for dynamically changing the appearance or behavior of elements in response to user interactions.
2. Here are some of the most commonly used methods of classList:

a. add(className): Adds a specified class to the element. If the class already exists in the element's class list, it will not add it again.
b. remove(className): Removes a specified class from the element. If the class does not exist, it does nothing.
c. contains(className): Checks if a specified class exists in the element's class list, returning true if it does and false otherwise.

3. example

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF-8" />
   <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
   <title>Document</title>
 </head>
 <body>
   <div id="myDiv" class="content visible">Hello,
world!</div>
 </body>
</html>
```

4. You can use the classList property to manipulate this element's classes in JavaScript:

```
<body>
   <div id="myDiv" class="content visible">Hello,
world!</div>
   <script>
```

```
    // Get the element
    var div = document.getElementById("myDiv");

    // Add a new class
    div.classList.add("highlight");

    // Remove a class
    div.classList.remove("visible");

    // Check if a class exists
    if (div.classList.contains("content")) {
      console.log('The div has a "content" class.');
    } else {
      console.log('The div does not has a "content"
class.');
    }
  </script>
 </body>
```

Problem - 3

1. Write a script which fetches the data-color attribute of the card and double clicking on them attaches the fetched class to that card and also changes the data-color attribute to "used"
2. Starter code

```
<!DOCTYPE html>
<html lang = "en">
 <head>
   <meta charset = "UTF - 8" />
```

```
   <meta http - equiv = "X - UA - Compatible" content = "IE
= edge" />
   <meta name = "viewport" content = "width = device -
width, initial - scale = 1.0" />
   <title>Document</title>
   <style>
     * {
       box-sizing: border-box;
     }

     body {
       display: flex;
       flex-wrap: wrap;
       justify-content: space-around;
       padding-top: 5rem;
     }

     .blue {
       background-color: blue;
       box-shadow: 0px 0px 6px 5px;
     }

     .green {
       background-color: green;
       box-shadow: 0px 0px 6px 5px;
     }

     .red {
       background-color: red;
       box-shadow: 0px 0px 6px 5px;
     }
```

```
    .card {
      border: 1px solid;
      height: 10rem;
      width: 10rem;
      margin: 2rem;
    }
  </style>
</head>

<body>
  <div class = "card" data-color = "blue"></div>
  <div class = "card" data-color = "red"></div>
  <div class = "card" data-color = "blue"></div>
  <div class = "card" data-color = "red"></div>
  <div class = "card" data-color = "red"></div>
  <div class = "card" data-color = "blue"></div>
  <div class = "card" data-color = "green"></div>
  <div class = "card" data-color = "blue"></div>
  <div class = "card" data-color = "green"></div>
  <div class = "card" data-color = "blue"></div>
  <script>
      /**
       * write a script to fetch the data-color attribute
of the card
       * double clicking on a card, attaches the
data-color attribute to the class list
       * changes the data-color attribute to used
       */
  </script>
  </body>
  </html>
```

Approach

1. Attach event listener to each card
    a. For that i need to get list of all cards
    b. Then iterate over the list and attach listener for dbl click
2. When clicked, pull the data attribute and add it to the class
3. Update the data attribute

```
<script>
    /**
       * write a script to fetch the data-color attribute of the card
       * double clicking on a card, attaches the data-color attribute to the class list
       * changes the data-color attribute to used
       *
       * Approach
       * 1. SELECTIONL: get list of cards
       * 2. ADDING EVENT LISTENER: then iterate over the cards and add a double click event listener
       * 3.1 LOGIC: - get the data-color attribute and add it to the class list
       * 3.2 LOGIC: change the data-color attribute to used
       */


      const cardsNodeList = document.querySelectorAll('.card');
      console.log('cardsNodeList', cardsNodeList);

     for(let i = 0; i < cardsNodeList.length; i++){
      cardsNodeList[i].addEventListener('dblclick', function(e){
```

```
    console.log("event:",e.target) // the card that was clicked

    const card = e.target;
    // get the attribute
    const classToBeAttached = card.getAttribute('data-color');
    console.log('classToBeAttached', classToBeAttached);
    card.classList.add(classToBeAttached);
    // set the attribute
    card.setAttribute('data-color', 'used');
   })
  }


</script>
```

4.

a. The e (Event) Object:

i. console.log(e): This logs the event object to the console. The event object e contains all the information about the event that occurred, including which element was clicked, the type of event, the position of the mouse, and more.

ii. e.target: This property of the event object refers to the element to which the event listener was attached. In this context, e.target is the card that was double-clicked.

iii. e.target.getAttribute('data-color'): This method gets the value of the data-color attribute from the clicked card. This value indicates the color class (blue, red, or green) that should be applied to the card.

iv. e.target.classList.add(classTobeAttached): This line adds the class (obtained from the data-color attribute) to the clicked card's class list, changing its appearance according to the CSS definitions for .blue, .green, or .red.

v. e.target.setAttribute('data-color', 'used'): Finally, this line changes the data-color attribute of the clicked card to "used", indicating that the card's color has been set and the attribute has been utilized.

5.