DOM(Document Object Model)

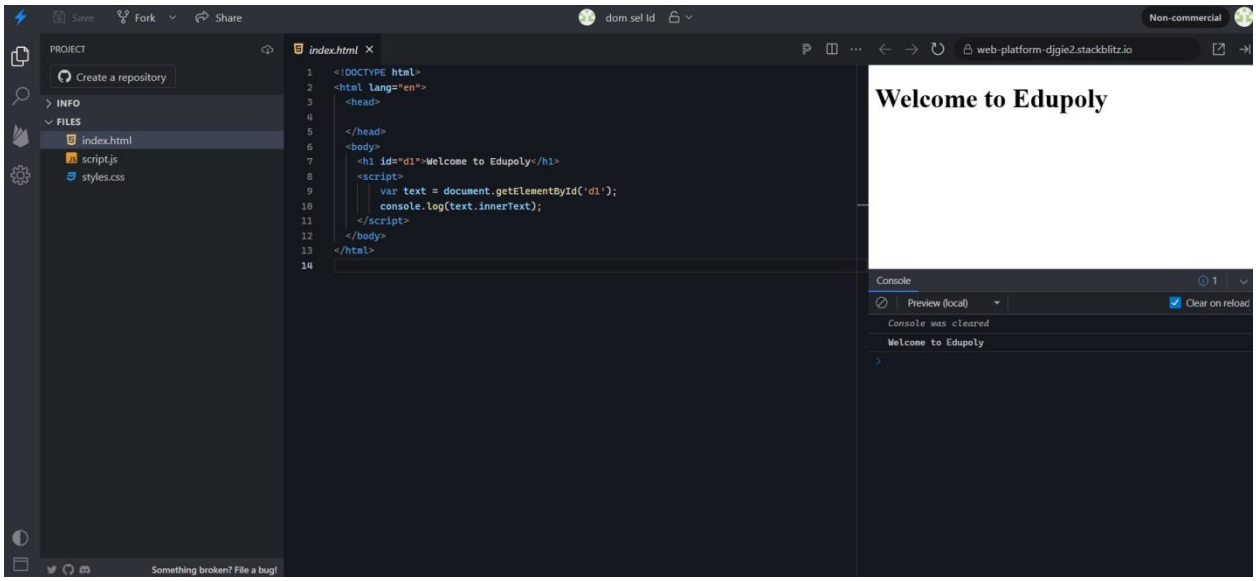
The document object in JavaScript is a key part of the Document Object Model (DOM), which represents the structure of a web page. It allows you to interact with and manipulate the content and structure of the HTML document displayed in the browser. Here's a comprehensive overview of the document object and its commonly used properties and methods:

1. Accessing Elements

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>DOM Selection Example</title>
</head>
<body>
  <h1 id="header">Welcome to My Page</h1>
  <p class="item">First item description</p>
  <p class="item">Second item description</p>
  <p class="item">Third item description</p>
  <p>Just another paragraph.</p>
  <button class="btn-primary">Click Me</button>
  <script>
    // Using getElementById
    const header = document.getElementById('header');
    console.log(header.innerText);  // Output: Welcome to My Page
    // Using getElementsByClassName
    const items = document.getElementsByClassName('item');
    console.log(items[0].innerText);  // Output: First item description
    // Using getElementsByTagName
    const paragraphs = document.getElementsByTagName('p');
    console.log(paragraphs.length);  // Output: 4 (including all <p> elements)
    // Using querySelector
    const firstItem = document.querySelector('.item');
    console.log(firstItem.innerText);  // Output: First item description
    // Using querySelectorAll
    const allItems = document.querySelectorAll('.item');
    console.log(allItems.length);  // Output: 3 (number of elements with class
"item")
  </script>
</body>
</html>
```

I. getElementById(id)

Returns the element with the specified ID.



Explanation of how the above code works:

<h1 id="d1">Welcome to Edupoly</h1>

Above line creates a heading element. The text "Welcome to Edupoly" is displayed on the webpage. The heading has an id attribute set to "d1", which uniquely identifies this element in the HTML document.

var text = document.getElementById('d1');

This line declares a variable named text. It uses the document.getElementById method to search for an HTML element with the id "d1". If found, it assigns a reference of the respective <h1> element to the variable text.

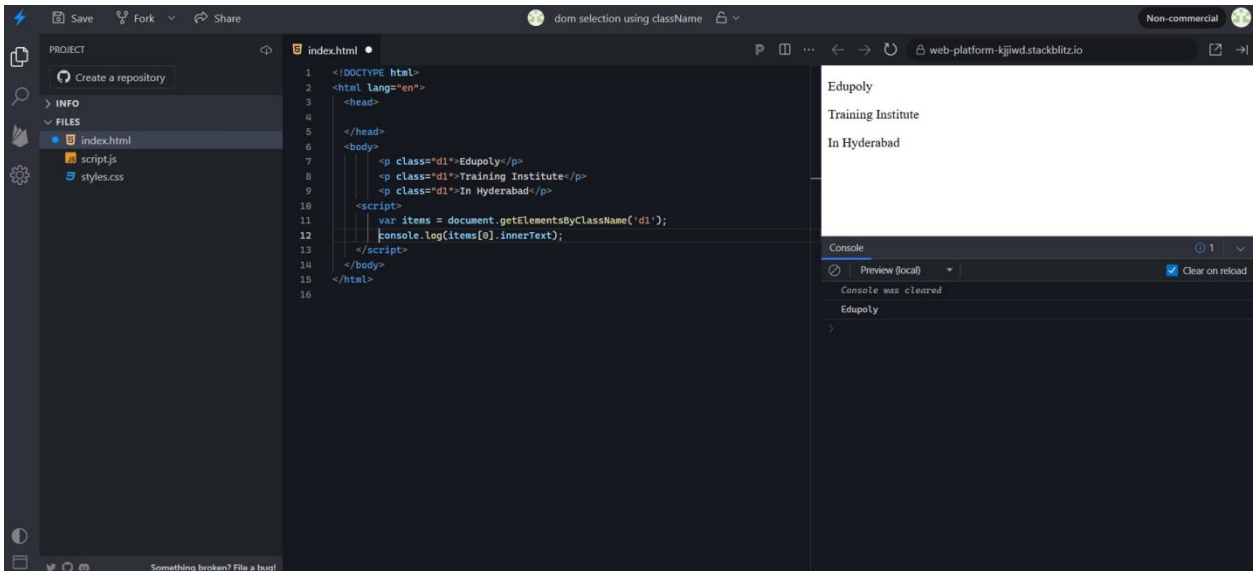
console.log(text.innerText);: This line outputs the text content of the text variable to the browser's console. The innerText property retrieves the actual text inside the <h1> element, which is "Welcome to Edupoly".

Open this url to check the code and try yourself:

https://stackblitz.com/edit/web-platform-djgie2?devToolsHeight=33&file=index.html

II. getElementsByClassName(className)

Returns a live HTMLCollection of elements with the specified class name.



Explanation of how the above code works:

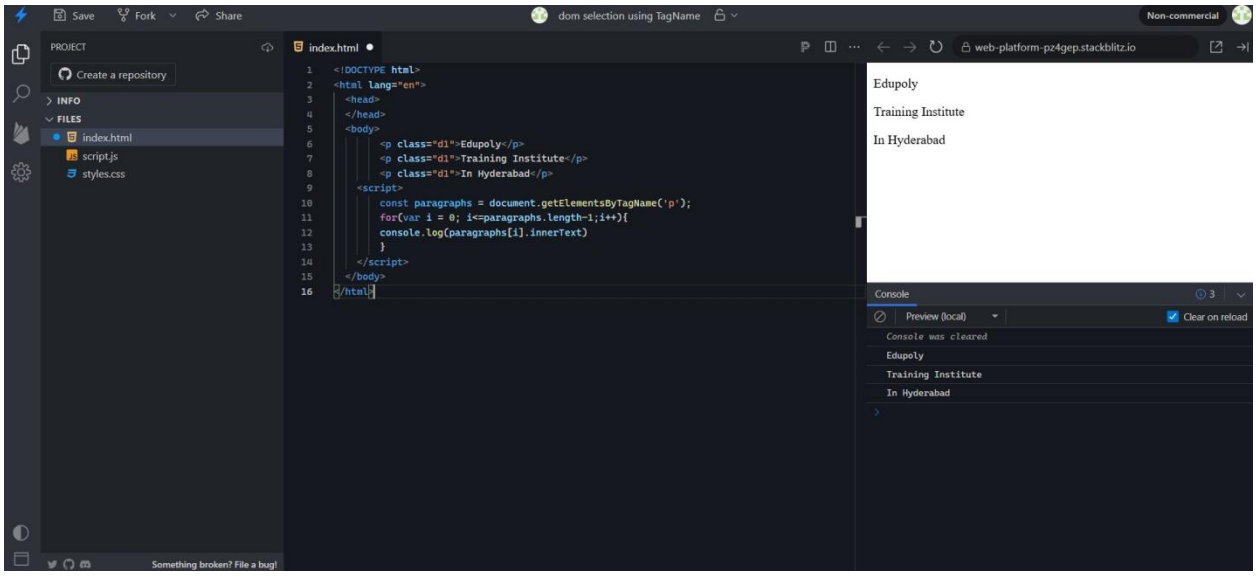
HTML Elements Creation: Three paragraph elements are created, each containing specific text. Each paragraph has a class attribute set to "d1," allowing them to be grouped together for selection in JavaScript.
Variable Declaration: A variable named items is declared. It uses a method to search the document for all elements that have the class name "d1." This method returns a collection of those elements.
Logging to Console: The code logs the text content of the first element in the collection to the browser's console. It retrieves the text from that paragraph, which is "Edupoly,"

Open this url to check the code and try yourself:

https://stackblitz.com/edit/web-platform-kjjiwd?devToolsHeight=33&file=index.html

III. getElementsByTagName(tagName)

Returns a live HTMLCollection of elements with the specified tag name.



Explanation of how the above code works:

HTML Elements Creation: Three paragraph elements are created, each containing specific text: "Edupoly," "Training Institute," and "In Hyderabad."

Variable Declaration: A variable named paragraphs is declared. It uses the document.getElementsByTagName method to search the document for all <p> elements. This method returns a live HTMLCollection of those paragraph elements.

For Loop Initialization: A for loop is initiated with a variable i starting at 0. The loop will continue as long as i is less than or equal to the total number of paragraphs minus one, ensuring that all paragraphs are processed.

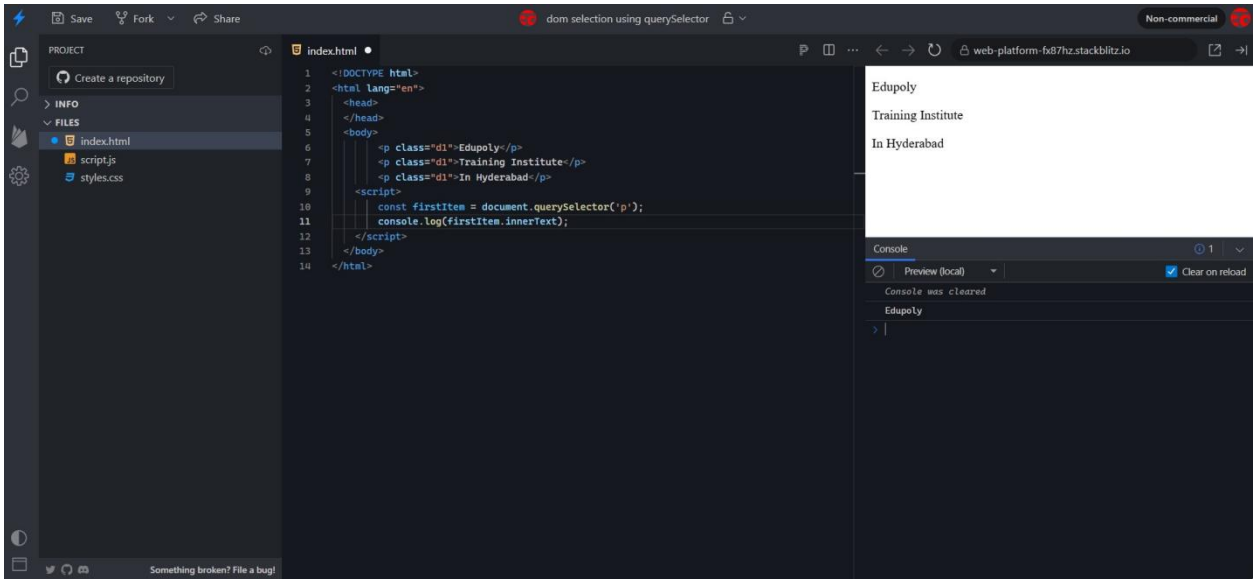
Logging to Console: Inside the loop, the code logs the text content of each paragraph to the browser's console. It retrieves the text from each paragraph using the innerText property, displaying "Edupoly," "Training Institute," and "In Hyderabad" sequentially in the console.

Open this url to check the code and try yourself:

https://stackblitz.com/edit/web-platform-pz4gep?devToolsHeight=33&file=index.html

IV. querySelector(selector)

Returns the first element that matches the CSS selector.



Explanation of how the above code works:

HTML Elements Creation: Three paragraph elements are created, each containing specific text: "Edupoly," "Training Institute," and "In Hyderabad."

Variable Declaration: A variable named firstItem is declared. It uses the document.querySelector method to search for the first <p> element in the document. This method returns the first matching element it finds.

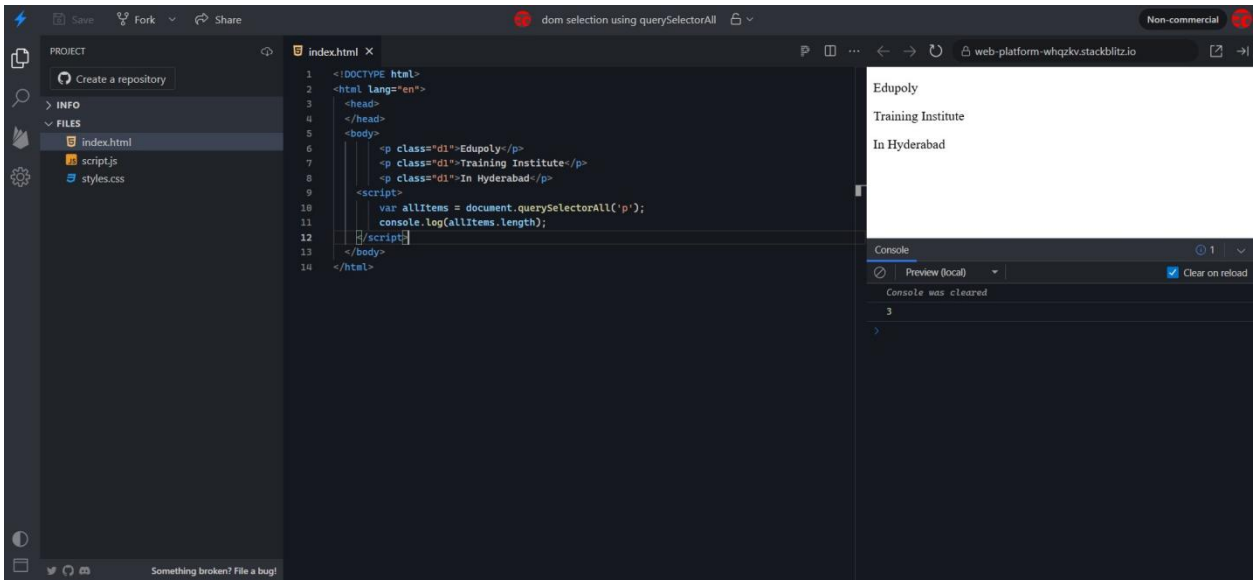
Logging to Console: The code logs the text content of the first paragraph element to the browser's console. It retrieves the text using the innerText property, which returns "Edupoly,"

Open this url to check the code and try yourself:

https://stackblitz.com/edit/web-platform-fx87hz?devToolsHeight=33&file=index.html

V. querySelectorAll(selector)

Returns a static NodeList of elements that match the CSS selector.



HTML Elements Creation: Three paragraph elements are created, each containing specific text: "Edupoly," "Training Institute," and "In Hyderabad." Each paragraph has a class attribute set to "d1,"

Variable Declaration: A variable named allItems is declared. It uses the document.querySelectorAll method to search for all <p> elements in the document. This method returns a NodeList containing all matching paragraph elements.

Logging to Console: The code logs the number of paragraph elements found to the browser's console. The length property of the NodeList is accessed, which returns the total count of <p> elements,

Open this url to check the code and try yourself:

https://stackblitz.com/edit/web-platform-whqzkv?devToolsHeight=33&file=index.html

2. Creating Elements

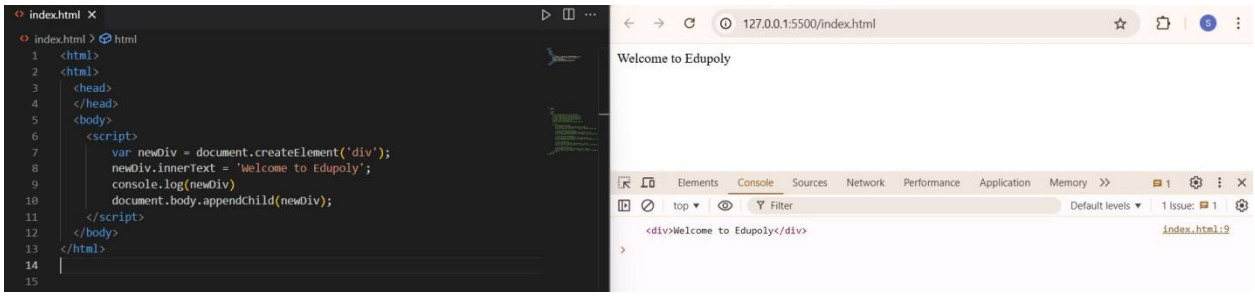
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>DOM Manipulation Examples</title>
  <style>
    .example {
      margin: 20px;
      padding: 10px;
      border: 1px solid #ccc;
    }
  </style>
</head>
<body>
  <h1>DOM Manipulation Examples</h1>
  <div id="output"></div>
  <script>
    // Example 1: Create a new div with text
    const newDiv = document.createElement('div');
    newDiv.innerText = 'Hello, world!';
    newDiv.classList.add('example'); // Add a class for styling
    document.getElementById('output').appendChild(newDiv);
    // Example 2: Create a new paragraph with text node
    const textNode = document.createTextNode('This is a new paragraph
created using createTextNode.');
    const newParagraph = document.createElement('p');
    newParagraph.appendChild(textNode);
    newParagraph.classList.add('example'); // Add a class for styling
    document.getElementById('output').appendChild(newParagraph);
    // Example 3: Create a list of items
    const ul = document.createElement('ul');
    const items = ['Item 1', 'Item 2', 'Item 3'];
    items.forEach(item => {
      const li = document.createElement('li');
      li.innerText = item;
      ul.appendChild(li);
    });
    ul.classList.add('example'); // Add a class for styling
    document.getElementById('output').appendChild(ul);
    // Example 4: Create static additional list items
    const additionalItems = ['Item 4', 'Item 5'];
```

```
        additionalItems.forEach(item => {
          const li = document.createElement('li');
          li.innerText = item;
          ul.appendChild(li);
        });
        // Example 5: Create a header and footer
        const header = document.createElement('h2');
        header.innerText = 'Header Section';
        header.classList.add('example');
        document.body.insertBefore(header, document.getElementById('output'));
        const footer = document.createElement('footer');
        footer.innerText = 'Footer Section';
        footer.classList.add('example');
        document.body.appendChild(footer);
      </script>
    </body>
    </html>
```

I. createElement(tagName)

Creates a new element of the specified type.

```
const newDiv = document.createElement('div');
newDiv.innerText = 'Hello, world!';
document.body.appendChild(newDiv);
```



Explanation of how the above code works:

Creating a New Element:
var newDiv = document.createElement('div');
This line creates a new element and stores it in the variable newDiv. The document.createElement method is used to generate an HTML element that can be manipulated later.

Setting Inner Text:
newDiv.innerText = 'Welcome to Edupoly';
This line assigns the text "Welcome to Edupoly" to the innerText property of the newDiv. This defines the visible text that will appear inside the when rendered on the page.

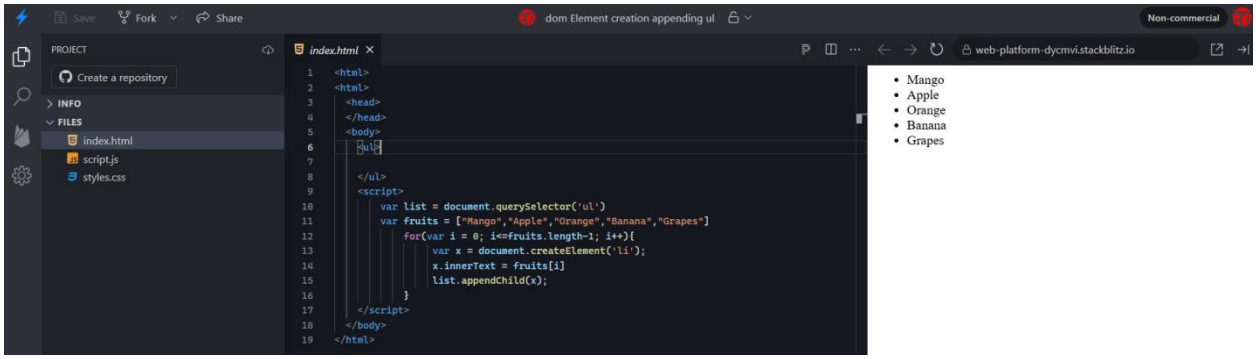
Logging the Element to the Console:
console.log(newDiv);
This line outputs the newDiv element to the console. This is useful as it allows you to inspect the properties and content of the element before it is added to the document.

Appending the Element to the Document Body:
document.body.appendChild(newDiv);
This line appends the newDiv to the body of the document. The appendChild method adds the new as the last child of the body, making it part of the rendered web page and visible to users.

Open this url to check the code and try yourself:

https://stackblitz.com/edit/web-platform-zsf3na?devToolsHeight=33&file=index.html

II. Creating Unordered-List



Explanation of how the above code works:

Selecting an Existing Element:
var list = document.querySelector('ul');
This line uses document.querySelector to select the first <ul> (unordered list) element in the document and assigns it to the variable list. This will be the container where the new list items will be added.

Defining an Array of Fruits:
var fruits = ["Mango","Apple","Orange","Banana","Grapes"];
This line creates an array named fruits that contains five strings, each representing a type of fruit. This array will be used to populate the list items.

Looping Through the Array:
for(var i = 0; i <= fruits.length - 1; i++)
This line starts a for loop that will iterate through each index of the fruits array. The loop runs from 0 to fruits.length - 1, ensuring that all elements in the array are processed.
Creating a New List Item:

var x = document.createElement('li');
Inside the loop, this line creates a new <li> (list item) element for each fruit and assigns it to the variable x. This allows you to create a list item dynamically for each fruit in the array.

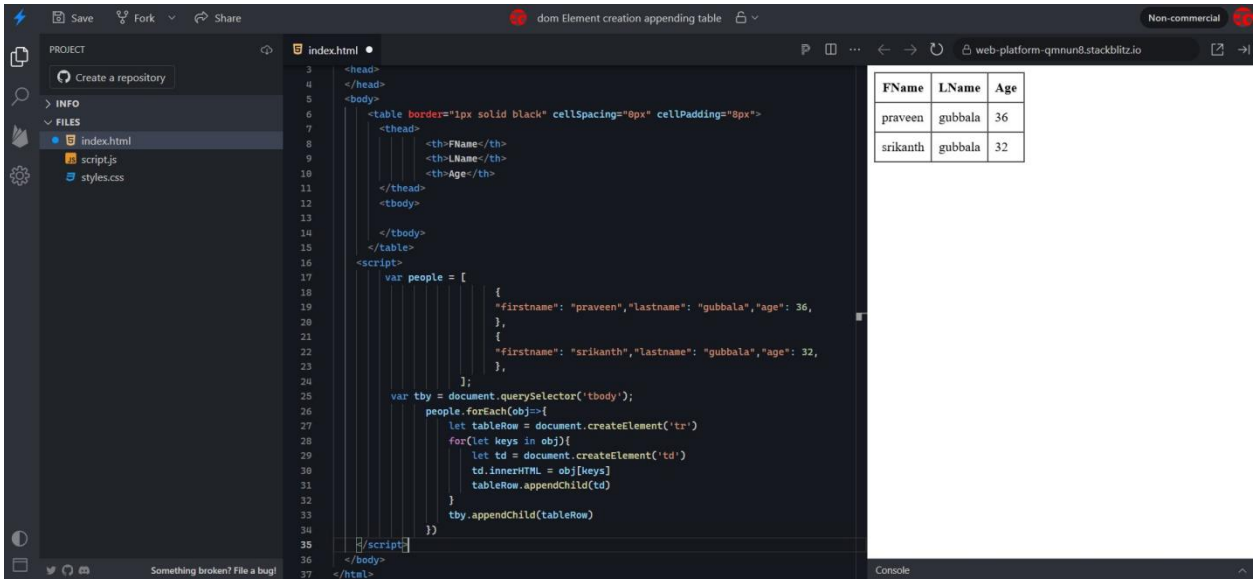
Setting Inner Text of the List Item:
x.innerText = fruits[i];
This line sets the text content of the newly created <li> element to the current fruit from the fruits array, using the loop index i.
Appending the List Item to the Unordered List:

list.appendChild(x);
Finally, this line appends the new <li> element (x) to the selected <ul> element (list). This adds each fruit as a new list item to the unordered list in the document.

Open this url to check the code and try yourself:

https://stackblitz.com/edit/web-platform-kvp1bz?devToolsHeight=33&file=index.html

III. Creating Table



Explanation of how the above code works:

Selecting the Table Body:
This line selects the <tbody> element of the table in the HTML using document.querySelector and assigns it to the variable tby. This is where the new rows will be added.

Iterating Over the Array:
The forEach method is called on the people array, allowing you to iterate through each object (obj) in the array.

Creating a New Table Row:
Inside the loop, a new <tr> (table row) element is created for each person and assigned to the variable tableRow. Each row will hold the person's details.

Iterating Over Object Properties:
A for...in loop iterates over each key (property name) in the current object (obj). This allows access to firstname, lastname, and age of each person.

Creating and Populating Table Cells:
Inside the inner loop, a new <td> (table cell) element is created for each property. The cell's inner HTML is set to the corresponding value from the object, and then this cell is appended to tableRow.

Appending the Row to the Table Body:
After all cells are added to tableRow,tableRow is appended to the <tbody> element by (tby). This makes the new row visible in the table on the web page.

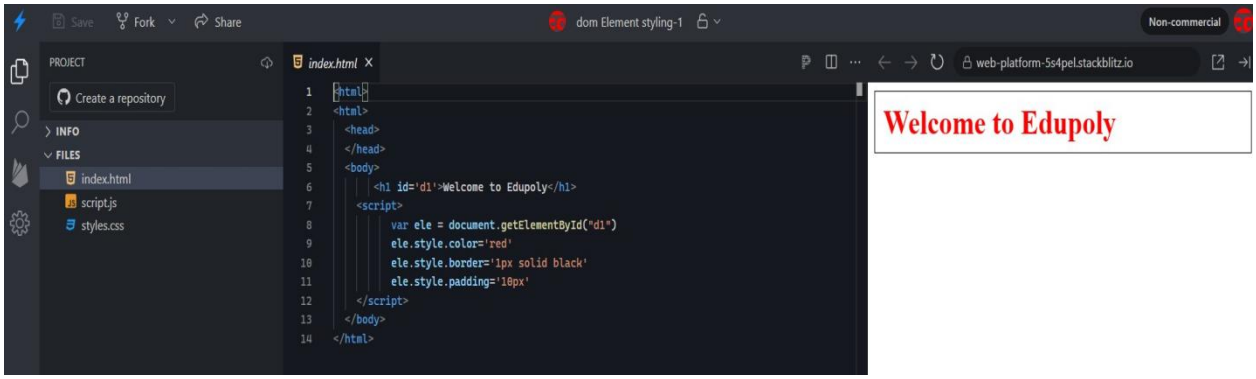
Open this url to check the code and try yourself:

https://stackblitz.com/edit/web-platform-qmnun8?devToolsHeight=33&file=index.html

3. Manipulating Styles

style

Allows you to set inline styles directly on an element.



Explanation of how the above code works:
var ele = document.getElementById("d1"):
This line selects an HTML element with the ID d1 and assigns it to the variable ele. This allows you to manipulate that specific element later in the code.

ele.style.color='red':
Here, the code changes the text color of the selected element (ele) to red. The style property accesses the inline CSS styles of the element.

ele.style.border='1px solid black':
This line adds a border to the element. It sets the border to be 1 pixel wide, solid, and black in color.

ele.style.padding='10px':
Finally, this line adds padding around the content of the element, giving it 10 pixels of space inside the border.

Open this url to check the code and try yourself:

https://stackblitz.com/edit/web-platform-5s4pel?devToolsHeight=33&file=index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Inline Styles Examples</title>
  <style>
    .example {
      margin-bottom: 20px;
      padding: 10px;
      border: 1px solid #ccc;
    }
  </style>
</head>
<body>
  <h2>Example 1: Changing Background Color and Padding</h2>
  <div id="example1" class="example">Hello, World!</div>
  <h2>Example 2: Setting Multiple Styles at Once</h2>
  <div id="example2" class="example">Hello, World!</div>
  <h2>Example 3: Applying Initial Styles on Load</h2>
  <div id="example3" class="example">Hello, World!</div>
  <h2>Example 4: Styling with a Function Call on Load</h2>
  <div id="example4" class="example">Hello, World!</div>
  <script>
    // Example 1: Changing Background Color and Padding
    const element1 = document.querySelector('#example1');
    element1.style.backgroundColor = 'lightgray';   // Change background color
    element1.style.padding = '20px';                // Add padding
```

```
        console.log('Example 1 styles:', element1.style.cssText);
        // Example 2: Setting Multiple Styles at Once
        const element2 = document.querySelector('#example2');
        element2.style.cssText = 'color: green; font-size: 25px; border: 2px solid
black;';
        console.log('Example 2 styles:', element2.style.cssText);
        // Example 3: Applying Initial Styles on Load
        const element3 = document.querySelector('#example3');
        element3.style.color = 'blue';                  // Change text color to blue
        element3.style.fontSize = '20px';               // Change font size to 20px
        element3.style.margin = '10px';                 // Add margin
        element3.style.padding = '15px';                // Add padding
        element3.style.border = '1px solid black';      // Add border
        console.log('Example 3 styles:', element3.style.cssText);
        // Example 4: Styling with a Function Call on Load
        const element4 = document.querySelector('#example4');
        function applyStyles() {
          element4.style.backgroundColor = 'lightblue'; // Change background color
          element4.style.color = 'darkblue';            // Change text color
          element4.style.fontSize = '24px';             // Change font size
          element4.style.textAlign = 'center';          // Center the text
        }
        // Apply styles immediately on load
        applyStyles();
        console.log('Example 4 styles applied:', element4.style.cssText);
      </script>

    </body>
    </html>
```

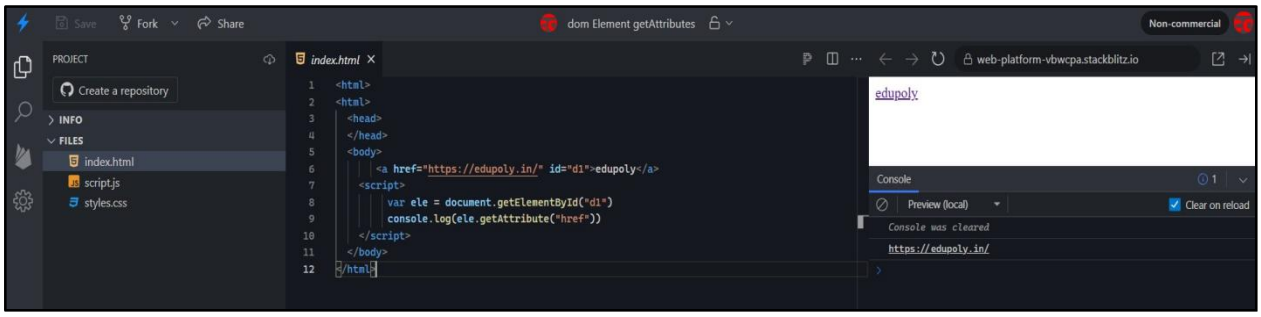
4. Manipulating Attributes

```
    <!DOCTYPE html>
    <html lang="en">
    <head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <title>Attribute Manipulation Example</title>
    </head>
    <body>
      <a href="https://www.initial-link.com" id="myLink">Initial Link</a>
      <button id="changeLink">Change Link</button>
      <button id="removeLink">Remove Link</button>
      <script>
        const link = document.getElementById('myLink');
```

```
        // Get the value of the href attribute
        console.log('Initial href:', link.getAttribute('href')); // Outputs:
              https://www.initial-link.com
        // Set a new value for the href attribute
        document.getElementById('changeLink').addEventListener('click', () => {
          link.setAttribute('href', 'https://www.example.com');
          console.log('Updated href:', link.getAttribute('href')); // Outputs:
              https://www.example.com
        });
        // Remove the href attribute
        document.getElementById('removeLink').addEventListener('click', () => {
          link.removeAttribute('href');
          console.log('Href after removal:', link.getAttribute('href')); // Outputs: null
        });
      </script>
    </body>
    </html>
```

getAttribute(name)

Gets the value of the specified attribute.



Explanation of how the above code works:

var ele = document.getElementById("d1"):
This line selects the HTML element with the ID d1, and assigns it to the variable ele. This allows you to work with that specific element in your JavaScript code.

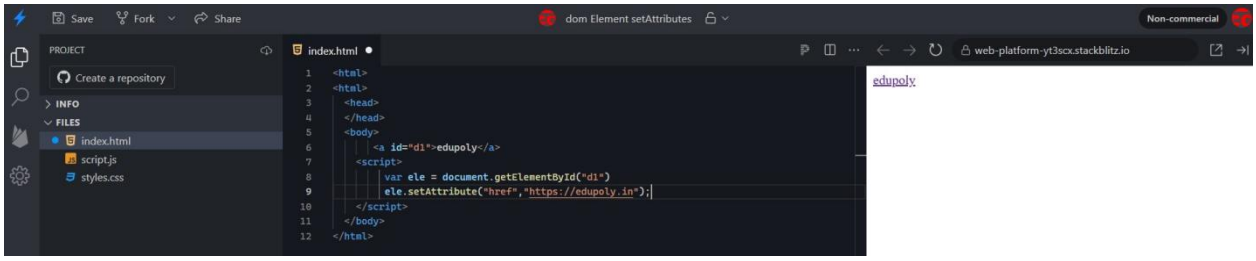
console.log(ele.getAttribute("href")):
This line retrieves the value of the href attribute from the selected element (ele). The getAttribute("href") method returns the URL linked to by the anchor tag (in this case, "https://edupoly.in/"). The console.log() function then outputs this value to the browser's console, allowing you to see the link associated with the anchor tag.

Open this url to check the code and try yourself:

https://stackblitz.com/edit/web-platform-vbwcpa?devToolsHeight=33&file=index.html

setAttribute(name, value)

Sets the value of the specified attribute.



Explanation of how the above code works:

var ele = document.getElementById("d1"):
This line selects the HTML element with the ID d1 and assigns it to the variable ele. This allows you to work with that specific element in your JavaScript code.

ele.setAttribute("href", "https://edupoly.in"):
This line sets the href attribute of the selected element (ele). The setAttribute("href", "https://edupoly.in") method updates the anchor tag to point to the specified URL. After this line executes, clicking the anchor will take the user to "https://edupoly.in".

Open this url to check the code and try yourself:

https://stackblitz.com/edit/web-platform-yt3scx?file=index.html

5. Manipulating Classes

classList

Provides methods to work with the class attribute of an element.

```
const element = document.querySelector('#myElement');
element.classList.add('newClass');  // Adds a class
element.classList.remove('oldClass');  // Removes a class
element.classList.toggle('toggleClass');  // Toggles a class
console.log(element.classList.contains('newClass'));  // Checks if a class exists
```

6. Traversing the DOM

parentNode

Gets the parent node of an element.

```
const child = document.querySelector('#child');
const parent = child.parentNode;
```

childNodes

Gets a live NodeList of child nodes.

```
const parent = document.querySelector('#parent');
const children = parent.childNodes;
```

firstChild and *lastChild*

Gets the first and last child nodes.

```
const firstChild = parent.firstChild;
const lastChild = parent.lastChild;
```

nextSibling and *previousSibling*

Gets the next and previous sibling nodes.

```
const nextSibling = parent.nextSibling;
const previousSibling = parent.previousSibling;
```

7. Working with Forms

form

Represents a form element, allowing you to interact with form controls.

```
const form = document.querySelector('form');
const input = form.querySelector('input');
console.log(input.value);  // Output: current value of the input field
```

submit()

Submits the form programmatically.

form.submit();

8. Manipulating the Document

document.title

Gets or sets the title of the document.

```
console.log(document.title);  // Output: current document title
document.title = 'New Title';
```

document.body

Represents the <body> element of the document.

document.body.style.backgroundColor = 'lightblue';

Summary

The document object is central to interacting with and manipulating web pages. It provides a wide array of methods and properties for selecting, creating, and modifying elements, handling events, and traversing the DOM. Understanding these capabilities allows you to build dynamic and interactive web applications.

Events in JavaScript are actions or occurrences that happen in the web browser, which JavaScript can respond to. They include user interactions, such as clicks and key presses, as well as other activities like page loading and form submissions. Understanding events is crucial for creating interactive and responsive web applications. Here's a comprehensive guide to handling events in JavaScript:

Document Object in JavaScript

The **Document Object** in JavaScript is a part of the **Document Object Model (DOM)**. It represents the entire web page (HTML document) loaded in the browser. Using the document object, you can manipulate the structure, style, and content of the web page dynamically.

Here's a breakdown of the **document object** and its key properties and methods:

---

1. Accessing the document Object

You can directly use the document object in JavaScript as it is globally available.

Example:

```
console.log(document); // Outputs the entire HTML document
```

---

2. Important Properties of the document Object

a. document.title

- Gets or sets the title of the document.

```
console.log(document.title); // Output: Current page title
document.title = "New Title"; // Changes the page title
```

b. document.body

- Represents the <body> of the document.

```
console.log(document.body); // Output: The body of the document
```

c. document.head

- Represents the <head> of the document.

```
console.log(document.head); // Output: The head section of the document
```

d. document.URL

- Gets the URL of the document.

```
console.log(document.URL); // Output: Current page URL
```

e. document.domain

- Gets the domain of the document.

```
console.log(document.domain); // Output: The domain name
```

f. document.documentElement

- Represents the <html> element of the document.

```
console.log(document.documentElement); // Output: The root `<html>` element
```

g. document.forms

- Returns a collection of all <form> elements in the document.

```
console.log(document.forms); // Output: HTMLCollection of forms
```

h. document.images

- Returns a collection of all <img> elements in the document.

```
console.log(document.images); // Output: HTMLCollection of images
```

---

3. Methods of the document Object

a. getElementById()

- Returns the element with the specified id.

```
let element = document.getElementById("header");
console.log(element); // Output: Element with id="header"
```

b. getElementsByClassName()

- Returns a collection of elements with a specific class name.

```
let elements = document.getElementsByClassName("card");
console.log(elements); // Output: HTMLCollection of elements with class="card"
```

c. getElementsByTagName()

- Returns a collection of elements with a specific tag name.

```
let paragraphs = document.getElementsByTagName("p");
console.log(paragraphs); // Output: HTMLCollection of <p> elements
```

d. querySelector()

- Returns the first element that matches a CSS selector.

```
let firstCard = document.querySelector(".card");
console.log(firstCard); // Output: First element with class="card"
```

e. querySelectorAll()

- Returns a NodeList of all elements that match a CSS selector.

```
let allCards = document.querySelectorAll(".card");
console.log(allCards); // Output: NodeList of all elements with class="card"
```

f. createElement()

- Creates a new HTML element.

```
let newDiv = document.createElement("div");
newDiv.textContent = "Hello, World!";
document.body.appendChild(newDiv); // Adds the new <div> to the body
```

g. removeChild()

- Removes a child element from its parent.

```
let header = document.getElementById("header");
document.body.removeChild(header); // Removes the header element
```

h. write()

- Writes content directly into the document.

```
document.write("<h1>Hello, World!</h1>"); // Writes directly into the document
```

i. appendChild()

- Appends a child element to a parent.

```
let newParagraph = document.createElement("p");
newParagraph.textContent = "This is a new paragraph.";
document.body.appendChild(newParagraph);
```

---

4. Event Listeners with the document Object

You can attach event listeners to the document to handle user interactions.

Example:

```
document.addEventListener("DOMContentLoaded", () => {
  console.log("The DOM is fully loaded!");
});
```

---

5. Modifying Document Content

a. Change the Inner HTML

- Use innerHTML to get or set the HTML content of an element.

```
let header = document.getElementById("header");
header.innerHTML = "<h2>Updated Header</h2>";
```

b. Change Text Content

- Use textContent to get or set the text inside an element.

```
let paragraph = document.getElementById("description");
paragraph.textContent = "Updated description text.";
```

c. Change Style

- Modify the inline styles of an element using style.

```
let box = document.getElementById("box");
box.style.backgroundColor = "lightblue";
box.style.padding = "20px";
```

---

6. Traversing the DOM

You can navigate through the DOM using the document object.

a. childNodes

- Returns a collection of all child nodes (including text nodes).

```
console.log(document.body.childNodes);
```

b. firstChild and lastChild

- Access the first and last child nodes.

```
console.log(document.body.firstChild);
console.log(document.body.lastChild);
```

c. parentNode

- Get the parent node of an element.

```
let header = document.getElementById("header");
console.log(header.parentNode); // Output: The parent node of the header
```

d. nextSibling and previousSibling

- Navigate to the next or previous sibling node.

```
let header = document.getElementById("header");
console.log(header.nextSibling); // Output: The next sibling node
```

---

7. Key Applications of the Document Object

1. **Dynamic Content Update:**
    - Modify text, images, or other elements based on user actions.
2. **Event Handling:**
    - Handle events like clicks, form submissions, and keyboard inputs.
3. **DOM Traversal:**
    - Navigate and manipulate the structure of the DOM.
4. **Dynamic Element Creation:**
    - Add or remove elements based on logic or user input.

Example: Dynamic List Creation

```
let items = ["Apple", "Banana", "Cherry"];

// Create a <ul> element
let list = document.createElement("ul");

// Add items to the list
items.forEach(item => {
  let li = document.createElement("li");
  li.textContent = item;
  list.appendChild(li);
});

// Append the list to the body
document.body.appendChild(list);
```

Summary of the Document Object:

Property/Method	Purpose
document.getElementById()	Get an element by its ID.
document.querySelector()	Get the first element matching a CSS selector.
document.createElement()	Create a new HTML element.
document.body	Access the <body> of the document.
document.title	Get or set the document's title.
document.addEventListener	Attach an event listener to the document.

Would you like a deeper dive into a specific concept, or do you need more examples?