Cheatsheet: Working with DOM in JavaScript

JavaScript Debugging, BOM and DOM Terminologies	Description	Code Example
try{....} block	The code that might generate an error is enclosed within a try block. This block helps to monitor for errors.	`const obj = undefined;` `try {` `  const propertyValue = obj.property; // Attempting to access a property of an undefined object` `  console.log("Property Value: " + propertyValue);` `  console.log("This message will be reached.");` `} catch (error) {` `  console.error("An error occurred while accessing the property:", error.message);` `}` `console.log("Program continues after error handling.");`
catch{....} block	The catch block in JavaScript catches and handles errors that occur within a try block.	`try {` `  // Code that might throw an error` `  const result = nondeclaredFunction(); // Assuming someFunction() is not defined` `  console.log(result); // This line won't execute due to the error` `} catch (error) {` `  // Code to handle the error` `  console.log('An error occurred:', error.message);` `}`
getElementById() Method	getElementById is a method in JavaScript used to access a specific HTML element within the Document Object Model (DOM) based on its unique id attribute.	`<!DOCTYPE html>` `<html>` `<head>` `  <title>getElementById Example</title>` `</head>` `<body>` `  <h1 id="main-heading">Welcome to the Example Page</h1>` `  <p id="content-paragraph">This is some content.</p>` `  <script>` `  const headingElement = document.getElementById('main-heading');` `console.log(headingElement)` `  </script>` `</body>` `</html>`
getElementsByClassName() Method	getElementsByClassName is a method in JavaScript that is used to access multiple HTML elements within the Document Object Model (DOM) that share the same class name.	`<!DOCTYPE html>` `<html>` `<head>` `  <title>getElementsByClassName Example</title>` `</head>` `<body>`

		This is a highlighted paragraph. This is another highlighted paragraph. This is a regular paragraph. <script> const highlightedElements = document.getElementsByClassName('highlighted'); // Modify the text content of each element for (let i = 0; i < highlightedElements.length; i++) { highlightedElements[i].textContent = `This paragraph is highlighted! for class ${i + 1}`; } </script> </body> </html>
getElementsByTagName() Method	getElementsByTagName is a method in JavaScript that is used to access multiple HTML elements within the Document Object Model (DOM) based on their tag name.	<!DOCTYPE html> <html> <head> <title>getElementsByTagName Example</title> </head> <body> Heading 2 This is a paragraph. This is another paragraph. <script> const paragraphElements = document.getElementsByTagName('p'); console.log(paragraphElements); console.log(paragraphElements[0]); console.log(paragraphElements[1]); </script> </body> </html>
querySelector	querySelector is a method used to access HTML elements within the Document Object Model (DOM) based on CSS-like selectors such as class, ID, or tag name.	<!DOCTYPE html> <html> <head> <title>querySelector Example</title> </head> <body> This is a highlighted paragraph. This is a paragraph with an ID. This is a regular paragraph. <script> const elementByClass = document.querySelector('.highlighted'); // Log the selected element to the console console.log(elementByClass); // Select the element with the ID "my-paragraph" using querySelector const elementByID = document.querySelector('#my-paragraph'); // Log the selected element to the console console.log(elementByID); // Select the first element using querySelector const elementByTag = document.querySelector('div'); // Log the selected element to the console console.log(elementByTag); </script> </body> </html>

querySelectorAll	querySelectorAll is a method used to select multiple HTML elements based on CSS-like selectors such as class, ID, or tag name and returns a collection of array Node-List elements that match the specified selector.	<!DOCTYPE html> <html> <head> <title>querySelectorAll Example</title> </head> <body> This is a highlighted paragraph. This is a highlighted paragraph. This is another highlighted paragraph. <section>This is a regular paragraph.</section> <script> const elementsById = document.querySelectorAll('#highlight'); const elementsByClass = document.querySelectorAll('.highlighted'); const elementsByTag = document.querySelectorAll('section'); // Log the selected elements to the console console.log(elementsById); console.log(elementsByClass); console.log(elementsByTag); </script> </body> </html>
textContent() Method	It can modify or change the text or HTML content of elements.	<!DOCTYPE html> <html> <head> <title>textContent Example</title> </head> <body> This is some text. <script> const paragraph = document.getElementById('my-paragraph'); paragraph.textContent = 'This is updated text.'; </script> </body> </html>
setAttribute() Method	It is used to alter the attributes (for example, src, href, class, id) of elements, which can affect their behavior or appearance.	<!DOCTYPE html> <html> <head> <title>setAttribute Example</title> </head> <body> <img id="my-image" src="your-old-image.jpg"> <script> const image = document.getElementById('my-image'); image.setAttribute('src', 'your-new-image.jpg'); </script> </body> </html>

Adding Elements	Dynamically adding new elements to the page based on user interactions or other conditions.	`<!DOCTYPE html>` `<html>` `<head>` `  <title>createElement Example</title>` `</head>` `<body>` `  <ul id="my-list">` `  <li>Item 1</li>` `  <li>Item 2</li>` `  </ul>` `  <script>` `  const list = document.getElementById('my-list');` `  const newItem = document.createElement('li');` `  newItem.textContent = 'Item 3';` `  list.appendChild(newItem);` `  </script>` `</body>` `</html>`
cloneNode() Method	Creating copies of existing elements that can be inserted elsewhere in the document.	`<!DOCTYPE html>` `<html>` `<head>` `  <title>createElement Example</title>` `</head>` `<body>` `  <ul id="my-list">` `  <li>Item 1</li>` `  <li>Item 2</li>` `  </ul>` `  <script>` `  const list = document.getElementById('my-list');` `  const firstItem = list.querySelector('li');` `  const clonedItem = firstItem.cloneNode(true);` `  list.appendChild(clonedItem);` `  </script>` `</body>` `</html>`
window Object	The global window object represents the browser window or tab and serves as the root of the BOM.	`window.alert(message): Displays a simple alert dialog with the specified message.` `window.confirm(message): Shows a confirmation dialog with "OK" and "Cancel" buttons and returns a Boolean value.` `window.open(url, name, specs, replace): Opens a new browser window or tab.` `window.close(): Closes the current window or tab.` `window.location: Provides information about the current URL and allows navigation.` `window.setTimeout(function, delay): Executes a function after a specified delay.` `window.localStorage and window.sessionStorage: Allow data storage on the client side.` `window.history: Provides access to the browser's session history.`

navigator Object	The navigator object provides information about the client's browser, such as the browser's name, version, and supported features.	`const browserName = navigator.appName;` `const browserVersion = navigator.appVersion;`
screen Object	The screen object gives details about the user's screen, including its dimensions and color depth.	`const screenWidth = screen.width;` `const screenHeight = screen.height;`
history Object	The history object represents the browser's session history, allowing you to navigate backward and forward in the user's browsing history.	`history.back(); // Navigates back one page` `history.forward(); // Navigates forward one page`
location Object	The location object provides information about the current URL and allows you to manipulate the URL, redirecting the user to other web pages.	`const currentURL = location.href;` `location.href = 'https://example.com'; // Redirects the user to a new URL`
BOM Example	This represents the combined example of above BOM methods.	`<!DOCTYPE html>` `<html>` `<head>` `  <title>BOM Example</title>` `</head>` `<body>` `  <button id="alertButton">Show Alert</button>` `  <button id="openWindowButton">Open Window</button>` `  <button id="navigateBackButton">Go Back</button>` `  <button id="changeURLButton">Change URL</button>` `  <script>` `  // Access HTML elements` `  const alertButton = document.getElementById('alertButton');` `  const openWindowButton = document.getElementById('openWindowButton');`

		`` const navigateBackButton = document.getElementById('navigateBackButton'); const changeURLButton = document.getElementById('changeURLButton'); // Attach event listeners alertButton.addEventListener('click', () => { window.alert('Hello, this is an alert!'); }); openWindowButton.addEventListener('click', () => { window.open('https://example.com', '_blank'); }); navigateBackButton.addEventListener('click', () => { history.back(); // Navigates back one page in the user's browsing history. }); changeURLButton.addEventListener('click', () => { location.href = 'https://example.com'; // Redirects the user to a new URL. }); </script> </body> </html> ```
firstElementChild() and lastElementChild()	It uses the firstElementChild and lastElementChild properties to access the first and last child nodes of any element.	``` <!DOCTYPE html> <html> <head> <title>DOM Traversing Example</title> </head> <body> <div id="parent"> <p>Child 1</p> <p>Child 2</p> </div> <script> const parent = document.getElementById("parent"); const firstChild = parent.firstElementChild; const lastChild = parent.lastElementChild; console.log(firstChild.textContent); // Outputs: "Child 1" console.log(lastChild.textContent); // Outputs: "Child 2" </script> </body> </html> ```
container Element	To find elements within a container, you typically use methods that allow you to query elements based on various criteria, such as tag name, class, or other attributes.	``` <!DOCTYPE html> <html> <head> <title>DOM Traversing Example</title> </head> <body> <div id="container"> <p class="myClass">Paragraph 1</p> <p class="myClass">Paragraph 2</p> <p>Paragraph 3</p> </div> <script> const container = document.getElementById("container"); const singleElement = container.querySelector(".myClass"); const multipleElements = container.querySelectorAll(".myClass"); console.log(singleElement.textContent); // Outputs: "Paragraph 1" console.log(multipleElements[1].textContent); // Outputs: "Paragraph 2 </script> </body> </html> ```

element.style.property = value	A way to access and modify the inline styles of an HTML element using the style property.	<!DOCTYPE html> <html> <head> <title>DOM Styling Example</title> </head> <body> <button id="myButton">Click Me</button> <script> const button = document.getElementById("myButton"); button.style.backgroundColor = "blue"; button.style.color = "white"; button.style.fontSize = "16px"; </script> </body> </html>
element.classList	You can use the classList property to add, remove, or toggle CSS classes on an element.	<!DOCTYPE html> <html> <head> <title>DOM Styling Example</title> </head> <body> This is a div <button id="myButton">Toggle Class</button> <script> const div = document.getElementById("myDiv"); const button = document.getElementById("myButton"); function toggleClassAndColor() { div.classList.toggle("active"); div.classList.toggle("inactive"); // Check if the "active" class is present and change the background color accordingly if (div.classList.contains("active")) { div.style.backgroundColor = "blue"; } else { div.style.backgroundColor = "red"; } } button.addEventListener("click", toggleClassAndColor); </script> </body> </html>
element.setAttribute	A method to use the setAttribute method to set or modify the style attribute of an element, which is a string containing inline CSS.	<!DOCTYPE html> <html> <head> <title>DOM Styling Example</title> </head> <body>

		This is a red paragraph. <button id="btn">Click to change Color of above paragraph</button> <script> const paragraph = document.getElementById("myParagraph"); const btn=document.getElementById('btn'); btn.addEventListener('click',()=>{ paragraph.setAttribute("style", "color: blue; font-size: 18px;"); }) </script> </body> </html>
element.style.cssText	The cssText property allows you to set the entire inline style of an element as a string.	<!DOCTYPE html> <html> <head> <title>DOM Styling Example</title> </head> <body> This is a paragraph. <button id="btn">Click to change Color and bold</button> <script> const text = document.getElementById("myText"); const btn=document.getElementById('btn'); btn.addEventListener('click',()=>{ text.style.cssText = "color: red; font-weight: bold;"; }) </script> </body> </html>
element.style.setProperty	This method allows you to set a specific CSS property with an optional priority for an element's inline style.	<!DOCTYPE html> <html> <head> <title>DOM Styling Example</title> </head> <body> <h1 id="myHeading">This is a heading.</h1> <button id="btn">Click Here</button> <script> const heading = document.getElementById("myHeading"); const btn=document.getElementById('btn'); btn.addEventListener('click',()=>{ heading.style.setProperty("color", "violet", "important"); }) </script> </body> </html>

element.style.removeProperty	You can use the removeProperty method to remove a specific CSS property from an element's inline style.	<!DOCTYPE html> <html> <head> <title>DOM Styling Example</title> </head> <body> This is a styled paragraph. <button id="btn">Click Here</button> <script> const paragraph = document.getElementById("myParagraph"); const btn=document.getElementById('btn'); btn.addEventListener('click',()=>{ paragraph.style.removeProperty("color"); }) </script> </body> </html>

Skills Network