

Cheatsheet: Arbeiten mit dem DOM in JavaScript

JavaScript-Debugging, BOM- und DOM-Begriffe	Beschreibung	Codebeispiel
try{....} block	Der Code, der einen Fehler verursachen könnte, ist in einem try-Block eingeschlossen. Dieser Block hilft, Fehler zu überwachen.	<pre>const obj = undefined; try { const propertyValue = obj.property; // Attempting to access a property of an undefir 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.");</pre>
catch{....} block	Der catch-Block in JavaScript fängt Fehler ab und behandelt sie, die innerhalb eines try-Blocks auftreten.	<pre>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); }</pre>
getElementById() Methode	getElementById ist eine Methode in JavaScript, die verwendet wird, um auf ein bestimmtes HTML-Element im Document Object Model (DOM) basierend auf seinem einzigartigen id-Attribut zuzugreifen.	<pre><!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></pre>

getElementsByClassName() Methode	<p>getElementsByClassName ist eine Methode in JavaScript, die verwendet wird, um auf mehrere HTML-Elemente im Document Object Model (DOM) zuzugreifen, die denselben Klassennamen teilen.</p>	<pre><!DOCTYPE html> <html> <head> <title>getElementsByClassName Example</title> </head> <body> <p class="highlighted">This is a highlighted paragraph.</p> <p class="highlighted">This is another highlighted paragraph.</p> <p>This is a regular paragraph.</p> <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 c } </script> </body> </html></pre>
getElementsByTagName() Methode	<p>getElementsByTagName ist eine Methode in JavaScript, die verwendet wird, um mehrere HTML-Elemente im Document Object Model (DOM) basierend auf ihrem Tag-Namen zuzugreifen.</p>	<pre><!DOCTYPE html> <html> <head> <title>getElementsByTagName Example</title> </head> <body> <h2>Heading 2</h2> <p>This is a paragraph.</p> <p>This is another paragraph.</p> <script> const paragraphElements = document.getElementsByTagName('p'); console.log(paragraphElements); console.log(paragraphElements[0]); console.log(paragraphElements[1]); </script> </body> </html></pre>
querySelector	<p>querySelector ist eine Methode, die verwendet wird, um auf HTML-Elemente innerhalb des Document Object Model (DOM) basierend auf CSS-ähnlichen Selektoren wie Klasse, ID oder Tag-Namen zuzugreifen.</p>	<pre><!DOCTYPE html> <html> <head> <title>querySelector Example</title> </head> <body> <p class="highlighted">This is a highlighted paragraph.</p> <p id="my-paragraph">This is a paragraph with an ID.</p> <div>This is a regular paragraph.</div> <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 <p> element using querySelector const elementByTag = document.querySelector('p'); // Log the selected element to the console console.log(elementByTag); </script> </body> </html></pre>
querySelectorAll	<p>querySelectorAll ist eine Methode, die verwendet wird, um mehrere HTML-Elemente basierend auf CSS-ähnlichen Selektoren wie Klasse, ID oder Tag-Namen auszuwählen und</p>	<pre><!DOCTYPE html> <html> <head> <title>querySelectorAll Example</title> </head> <body> <p id="highlight">This is a highlighted paragraph.</p> <p class="highlighted">This is a highlighted paragraph.</p> <p class="highlighted">This is another highlighted paragraph.</p></pre>

	eine Sammlung von Array-Node-List-Elementen zurückzugeben, die dem angegebenen Selektor entsprechen.	<pre> <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> </pre>
textContent() Methode	Es kann den Text oder den HTML-Inhalt von Elementen ändern oder modifizieren.	<pre> <!DOCTYPE html> <html> <head> <title>textContent Example</title> </head> <body> <p id="my-paragraph">This is some text.</p> <script> const paragraph = document.getElementById('my-paragraph'); paragraph.textContent = 'This is updated text.'; </script> </body> </html> </pre>
setAttribute() Methode	Sie wird verwendet, um die Attribute (zum Beispiel src, href, class, id) von Elementen zu ändern, was ihr Verhalten oder Aussehen beeinflussen kann.	<pre> <!DOCTYPE html> <html> <head> <title>setAttribute Example</title> </head> <body> <script> const image = document.getElementById('my-image'); image.setAttribute('src', 'your-new-image.jpg'); </script> </body> </html> </pre>
Elemente Hinzufügen	Dynamisches Hinzufügen neuer Elemente zur Seite basierend auf Benutzerinteraktionen oder anderen Bedingungen.	<pre> <!DOCTYPE html> <html> <head> <title>createElement Example</title> </head> <body> <ul id="my-list"> Item 1 Item 2 <script> const list = document.getElementById('my-list'); const newItem = document.createElement('li'); newItem.textContent = 'Item 3'; list.appendChild(newItem); </script> </body> </html> </pre>

cloneNode() Methode	Erstellen von Kopien vorhandener Elemente, die an anderer Stelle im Dokument eingefügt werden können.	<pre> <!DOCTYPE html> <html> <head> <title>createElement Example</title> </head> <body> <ul id="my-list"> Item 1 Item 2 <script> const list = document.getElementById('my-list'); const firstItem = list.querySelector('li'); const clonedItem = firstItem.cloneNode(true); list.appendChild(clonedItem); </script> </body> </html> </pre>
window-Objekt	Das globale window-Objekt repräsentiert das Browserfenster oder den Tab und dient als Wurzel des BOM.	<p> window.alert(message): Displays a simple alert dialog with the specified message. window.confirm(message): Shows a confirmation dialog with "OK" and "Cancel" buttons ar 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. </p>
navigator Objekt	Das navigator Objekt liefert Informationen über den Browser des Clients, wie den Namen des Browsers, die Version und unterstützte Funktionen.	<pre> const browserName = navigator.appName; const browserVersion = navigator.appVersion; </pre>
Bildschirmobjekt	Das Bildschirmobjekt gibt Details über den Bildschirm des Benutzers an, einschließlich seiner Abmessungen und Farbtiefe.	<pre> const screenWidth = screen.width; const screenHeight = screen.height; </pre>
history-Objekt	Das history-Objekt repräsentiert die Sitzungsverlauf des Browsers und ermöglicht es Ihnen, im Verlauf der Browsersitzung des Benutzers vor- und zurückzunavigieren.	<pre> history.back(); // Navigates back one page history.forward(); // Navigates forward one page </pre>
Standortobjekt	Das Standortobjekt liefert Informationen über die aktuelle URL und ermöglicht es Ihnen, die URL zu manipulieren, indem der Benutzer auf andere Webseiten umgeleitet wird.	<pre> const currentURL = location.href; location.href = 'https://example.com'; // Redirects the user to a new URL </pre>

BOM-Beispiel	Dies stellt das kombinierte Beispiel der oben genannten BOM-Methoden dar.	<pre> <!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> </pre>
firstElementChild() und lastElementChild()	Es verwendet die Eigenschaften firstElementChild und lastElementChild, um auf die ersten und letzten Kindknoten eines Elements zuzugreifen.	<pre> <!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> </pre>
Container-Element	Um Elemente innerhalb eines Containers zu finden, verwendet man typischerweise Methoden, die es ermöglichen, Elemente basierend auf verschiedenen Kriterien abzufragen, wie z.B. Tag-Namen, Klassen oder anderen Attributen.	<pre> <!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> </pre>

element.style.property = value	Eine Möglichkeit, auf die Inline-Stile eines HTML-Elements zuzugreifen und diese zu ändern, indem man die style-Eigenschaft verwendet.	<pre> <!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> </pre>
element.classList	Sie können die Eigenschaft classList verwenden, um CSS-Klassen zu einem Element hinzuzufügen, zu entfernen oder umzuschalten.	<pre> <!DOCTYPE html> <html> <head> <title>DOM Styling Example</title> </head> <body> <div id="myDiv" class="active">This is a div</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 if (div.classList.contains("active")) { div.style.backgroundColor = "blue"; } else { div.style.backgroundColor = "red"; } } button.addEventListener("click", toggleClassAndColor); </script> </body> </html> </pre>
element.setAttribute	Eine Methode, um die setAttribute-Methode zu verwenden, um das Style-Attribut eines Elements festzulegen oder zu ändern, das eine Zeichenfolge mit Inline-CSS enthält.	<pre> <!DOCTYPE html> <html> <head> <title>DOM Styling Example</title> </head> <body> <p id="myParagraph" style="color: red;">This is a red paragraph.</p> <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> </pre>
element.style.cssText	Die cssText-Eigenschaft ermöglicht es Ihnen, den gesamten Inline-Stil eines	<pre> <!DOCTYPE html> <html> <head> <title>DOM Styling Example</title> </head> </pre>

	Elements als Zeichenfolge festzulegen.	<pre> <body> <p id="myText">This is a paragraph.</p> <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> </pre>
element.style.setProperty	Diese Methode ermöglicht es Ihnen, eine bestimmte CSS-Eigenschaft mit einer optionalen Priorität für den Inline-Stil eines Elements festzulegen.	<pre> <!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> </pre>
element.style.removeProperty	Sie können die Methode removeProperty verwenden, um eine bestimmte CSS-Eigenschaft aus dem Inline-Stil eines Elements zu entfernen.	<pre> <!DOCTYPE html> <html> <head> <title>DOM Styling Example</title> </head> <body> <p id="myParagraph" style="color: blue; font-size: 18px;">This is a styled paragraph</p> <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> </pre>



Skills Network