| **Method/Property** | **Description** | **Example** | **Usage** |
| --- | --- | --- | --- |
| document.querySelector() | Selects the **first** HTML element that matches a CSS selector. | document.querySelector('.class-name') | Used to select specific elements like classes, IDs, or tags. |
| document.querySelectorAll() | Selects **all** elements matching a CSS selector. | document.querySelectorAll('div') | Returns a NodeList (like an array) of matching elements. |
| document.getElementById() | Selects an element by its **ID**. | document.getElementById('unique-id') | Fast and direct way to select an element. |
| document.getElementsByClassName() | Selects all elements with a specified class name. | document.getElementsByClassName('class-name') | Returns an HTMLCollection (live list of elements). |
| document.getElementsByTagName() | Selects all elements with a specified tag name. | document.getElementsByTagName('p') | Useful to get all <p> or <div> tags. |
| document.createElement() | Creates a new HTML element in the DOM. | document.createElement('div') | Used for dynamically creating elements like div, button. |
| document.createTextNode() | Creates a new text node. | document.createTextNode('Hello World') | Adds textual content to the DOM. |
| .appendChild() | Appends a node (element or text) as a child to an existing element. | parent.appendChild(childElement) | Adds elements dynamically to the DOM. |
| .innerHTML | Gets or sets the HTML content inside an element. | element.innerHTML = '<h1>Hello</h1>'; | Fast way to update multiple inner elements but can overwrite existing content. |
| .innerText | Gets or sets the **visible text** inside an element. | element.innerText = 'Visible Text'; | Updates only the text inside the element without rendering HTML. |
| .textContent | Similar to innerText, but includes hidden text. | element.textContent = 'Hidden Text'; | Accesses text content, including hidden elements. |
| .style | Modifies the CSS style of an element. | element.style.color = 'red'; | Dynamically updates styles without external CSS. |
| .classList.add() | Adds a CSS class to an element. | element.classList.add('new-class'); | Easily add classes dynamically. |
| .classList.remove() | Removes a CSS class from an element. | element.classList.remove('old-class'); | Easily remove classes dynamically. |
| .classList.toggle() | Toggles a CSS class on or off. | element.classList.toggle('active'); | Adds a class if it's not present, removes it if it is. |
| .getAttribute() | Gets the value of an attribute from an element. | element.getAttribute('src'); | Retrieve attribute values like src, href. |
| .setAttribute() | Sets or updates the value of an attribute. | element.setAttribute('alt', 'Image Description'); | Dynamically updates or adds attributes to elements. |
| .removeAttribute() | Removes an attribute from an element. | element.removeAttribute('disabled'); | Useful for enabling or disabling form fields. |
| .remove() | Removes an element from the DOM. | element.remove(); | Deletes elements dynamically. |
| .insertBefore() | Inserts a new node before a specified child node. | parent.insertBefore(newElement, referenceChild); | Adds elements in specific positions relative to existing elements. |
| parentNode | Refers to the parent node of an element. | element.parentNode; | Navigate up the DOM tree. |
| childNodes | Returns all child nodes of an element (including text nodes). | parent.childNodes; | Access child elements, including text nodes. |
| children | Returns only element child nodes (no text nodes). | parent.children; | Used for manipulating only elements, ignoring text nodes. |
| .replaceChild() | Replaces one child node with another. | parent.replaceChild(newChild, oldChild); | Replace elements dynamically in the DOM. |
| addEventListener() | Attaches an event listener to an element. | button.addEventListener('click', () => console.log('Clicked!')); | Reacts to user actions or events like clicks, hover, or scroll. |
| .removeEventListener() | Removes an event listener from an element. | button.removeEventListener('click', eventHandlerFunction); | Useful for cleaning up event listeners. |
| scrollIntoView() | Scrolls an element into the visible area of the browser window. | element.scrollIntoView(); | Automatically scroll to specific elements. |

BEM Convention – <https://dev.to/waelhabbal/mastering-css-with-the-bem-naming-convention-279h#:~:text=The%20BEM%20naming%20convention%20consists,semantically%20tied%20to%20the%20Block>.

**Setting Up setTimeout**

* **Name of the Functionality**: This is called **asynchronous task scheduling** or simply using the **Timer API**.
* **Execution**: When setTimeout is called, the JavaScript engine registers the callback function () => { console.log("b"); } to execute **after the specified delay** (in this case, 1ms).

**Role of Web APIs**

* **What Happens?**: The callback function is not executed immediately. Instead:
  1. The setTimeout function registers the callback and sends it to the **Web APIs environment** (provided by the browser or Node.js runtime).
  2. A timer is started in the Web APIs environment.

**Moving the Callback to the Event Queue**

* Once the timer expires (in this case, after 1ms), the Web APIs environment **pushes the callback function** into the **Task Queue** (also called the **Callback Queue**).

**Execution via the Event Loop**

* The **Event Loop** monitors the Call Stack and the Task Queue. If the Call Stack is empty (i.e., all synchronous code is executed), the Event Loop picks the first task in the Task Queue (in this case, your callback function) and pushes it onto the Call Stack for execution.
* At this point, console.log("b"); runs.

A closure is a function that "remembers" the variables from its outer environment, even after the outer function has executed.