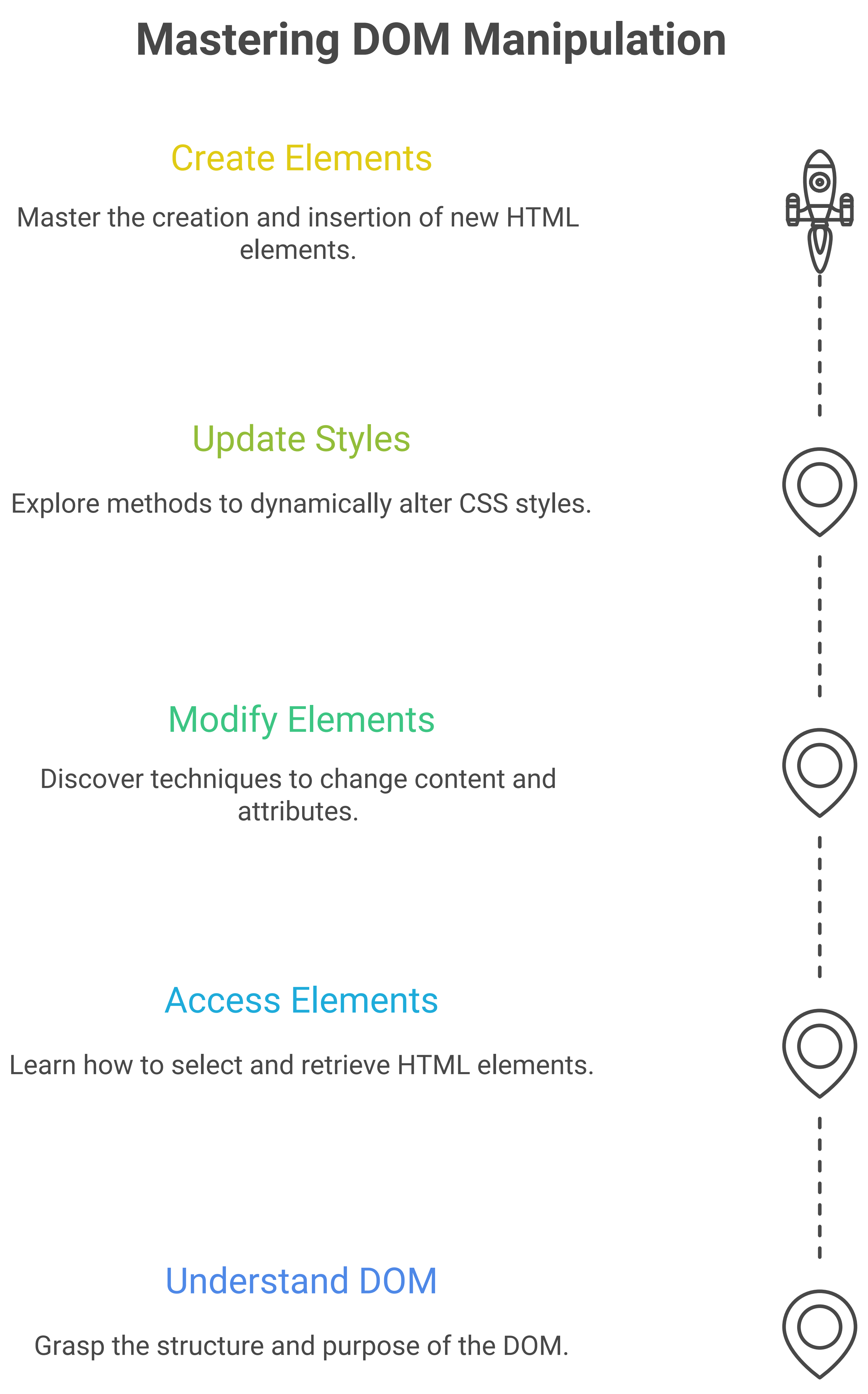
HTML DOM JavaScript: A Comprehensive Guide with Examples

This document provides a comprehensive overview of the HTML DOM (Document Object Model) and its interaction with JavaScript. We'll explore how JavaScript can be used to access, manipulate, and dynamically update HTML elements, attributes, and styles, empowering you to create interactive and dynamic web pages. Through clear explanations and practical examples, you'll gain a solid understanding of the core concepts and techniques involved in DOM manipulation.



Mastering DOM Manipulation

Create Elements

Master the creation and insertion of new HTML elements.

Update Styles

Explore methods to dynamically alter CSS styles.

Modify Elements

Discover techniques to change content and attributes.

Access Elements

Learn how to select and retrieve HTML elements.

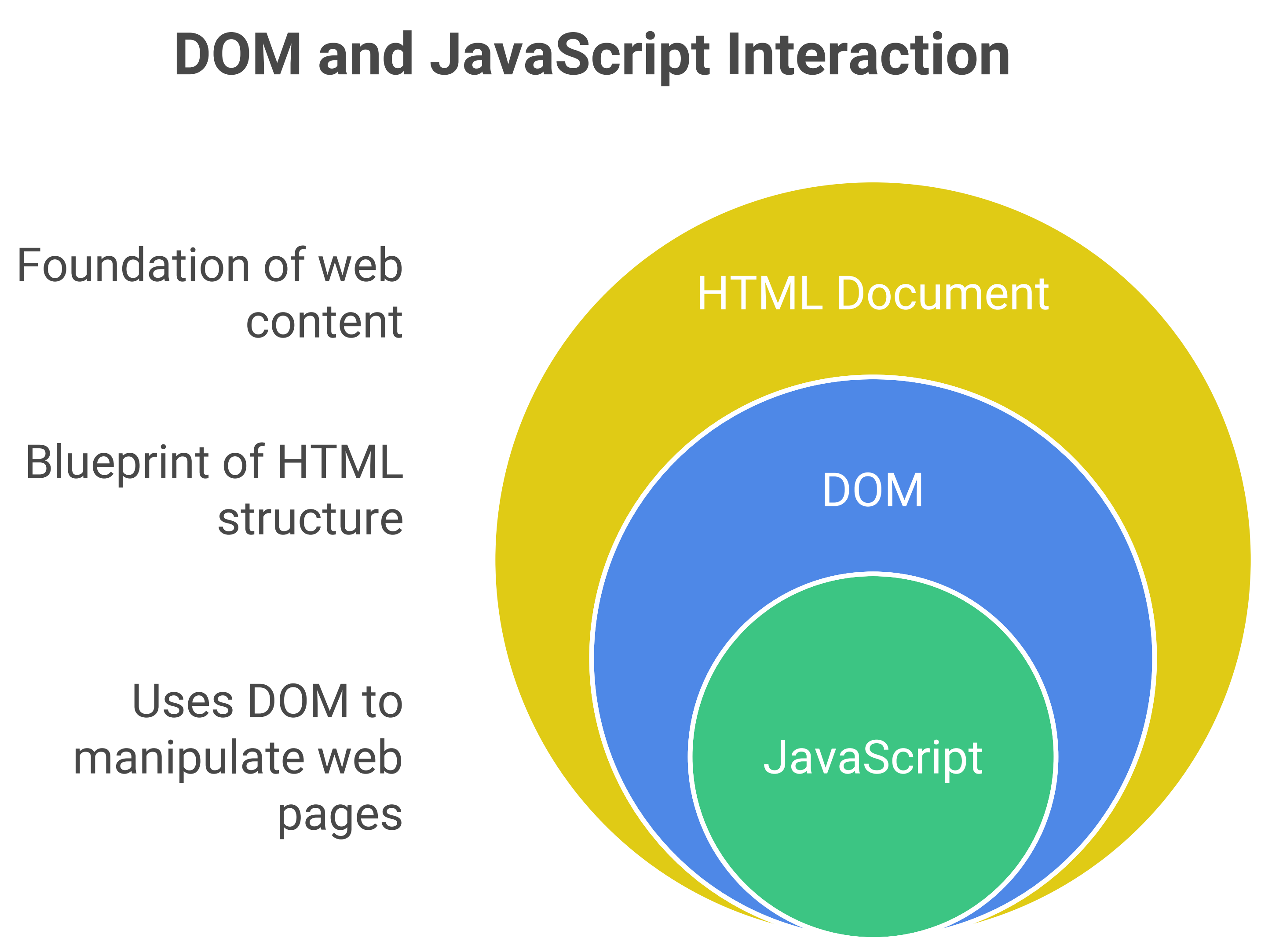
Understand DOM

Grasp the structure and purpose of the DOM.

Introduction to the DOM

The Document Object Model (DOM) is a programming interface for HTML and XML documents. It represents the page as a tree-like structure, where each HTML element, attribute, and text node is represented as an object. JavaScript uses the DOM to access and manipulate these objects, allowing you to dynamically change the content, structure, and style of a web page.

Think of the DOM as a map of your HTML document. JavaScript uses this map to find specific elements and make changes to them.


DOM and JavaScript Interaction
Foundation of web content
Blueprint of HTML structure
Uses DOM to manipulate web pages
HTML Document
DOM
JavaScript

Accessing HTML Elements

JavaScript provides several methods for accessing HTML elements within the DOM:

- document.getElementById(id)**:** This method retrieves an element based on its unique id attribute. It's the most efficient way to access a specific element.

~~~~
```html
~~~~

~~~~
<p id="myParagraph">This is a paragraph.</p>
~~~~

~~~~
<script>
~~~~

~~~~
  let paragraph = document.getElementById("myParagraph");
~~~~

~~~
  console.log(paragraph.textContent); // Output: This is a paragraph.
</script>
```
~~~

- document.getElementsByTagName(tagName): This method returns a collection (an HTMLCollection) of all elements with the specified tag name.

~~~
```html
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
</ul>
<script>
  let listItems = document.getElementsByTagName("li");
  console.log(listItems.length); // Output: 2
  console.log(listItems[0].textContent); // Output: Item 1
</script>
```
~~~

- document.getElementsByClassName(className): This method returns a collection (an HTMLCollection) of all elements with the specified class name.

~~~
```html
<div class="myClass">Div 1</div>
<div class="myClass">Div 2</div>
~~~

```
<script>

  let divs = document.getElementsByClassName("myClass");

  console.log(divs.length); // Output: 2

  console.log(divs[1].textContent); // Output: Div 2

</script>
```

- document.querySelector(selector)**:** This method returns the *first* element that matches a specified CSS selector.

```html
<p class="highlight">First paragraph</p>

<p>Second paragraph</p>

<script>

  let highlightedParagraph = document.querySelector(".highlight");

  console.log(highlightedParagraph.textContent); // Output: First paragraph

</script>
```

- document.querySelectorAll(selector)**:** This method returns a collection (a NodeList) of *all* elements that match a specified CSS selector.

```html
<p class="highlight">First paragraph</p>

<p class="highlight">Second paragraph</p>
```

~~~
<script>

  let highlightedParagraphs = document.querySelectorAll(".highlight");

  console.log(highlightedParagraphs.length); // Output: 2

  console.log(highlightedParagraphs[1].textContent); // Output: Second paragraph

</script>

```
~~~

Important Note: getElementsByTagName and getElementsByClassName return *live* HTMLCollections. This means that if the DOM is modified after the collection is created, the collection will automatically update to reflect the changes. querySelectorAll returns a *static* NodeList, which does not update automatically.

Modifying HTML Elements

Once you have accessed an element, you can modify its content, attributes, and styles.

Changing Content

- element.textContent**:** Gets or sets the text content of an element and its descendants.

~~~
```html

<p id="myParagraph">Original text.</p>

<script>

  let paragraph = document.getElementById("myParagraph");

  paragraph.textContent = "New text!";

</script>

```
~~~

- element.innerHTML**:** Gets or sets the HTML content of an element. This allows you to insert HTML markup.

```html
<div id="myDiv"></div>
<script>
  let div = document.getElementById("myDiv");
  div.innerHTML = "<p>This is a new paragraph inside the div.</p>";
</script>
```

Caution: Using `innerHTML` can be a security risk if you are inserting user-provided content, as it can lead to cross-site scripting (XSS) vulnerabilities. Always sanitize user input before using it with `innerHTML`.

Changing Attributes

- element.getAttribute(attributeName)**:** Gets the value of an attribute.

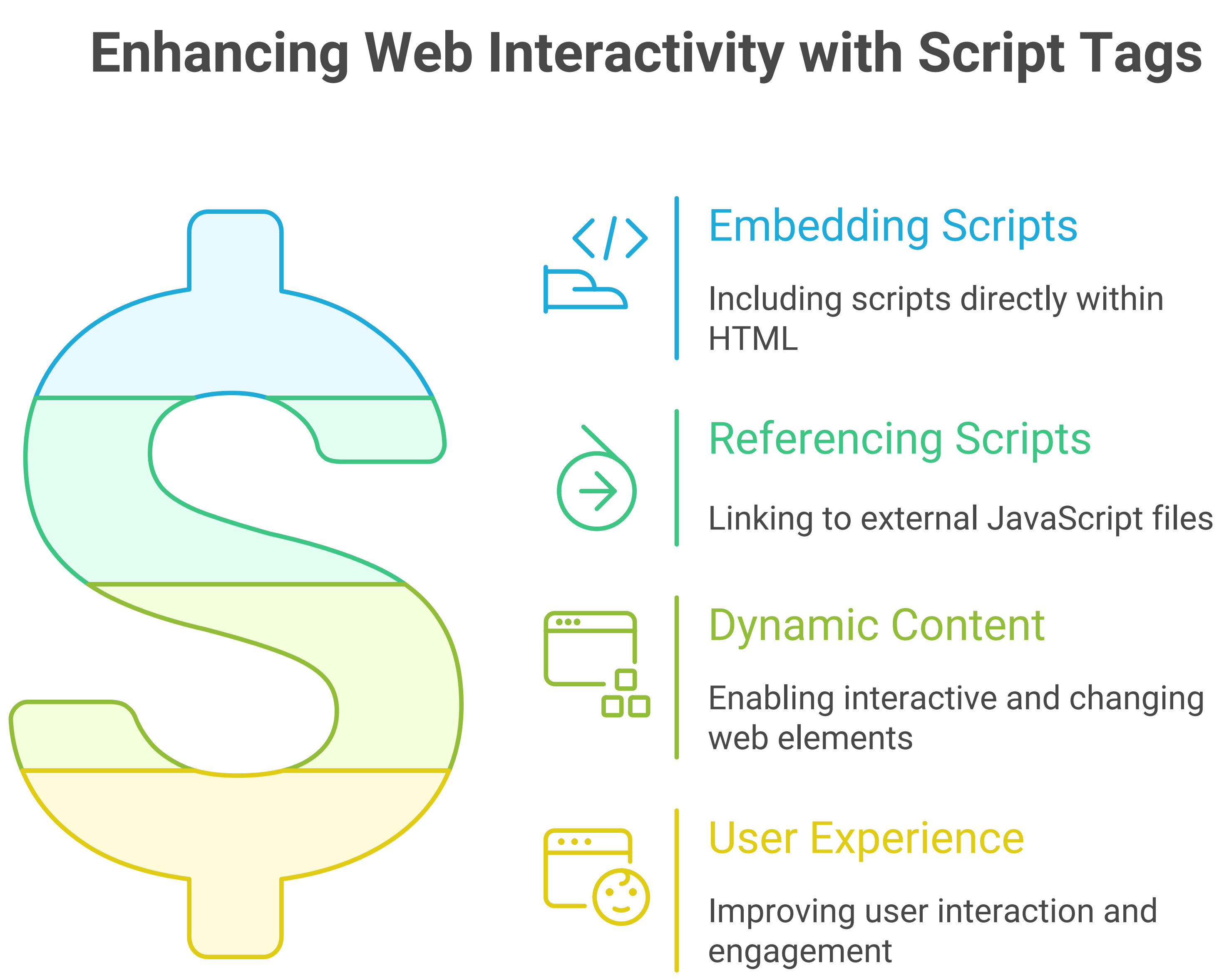
```html
<a href="https://www.example.com" id="myLink">Link</a>
<script>
  let link = document.getElementById("myLink");
  let href = link.getAttribute("href");
  console.log(href); // Output: https://www.example.com
</script>
```

- element.setAttribute(attributeName, attributeValue)**:** Sets the value of an attribute.

~~~
```html
~~~

```
<img src="old_image.jpg" id="myImage">
```

```
<script>
```


Enhancing Web Interactivity with Script Tags
Embedding Scripts
Including scripts directly within HTML
Referencing Scripts
Linking to external JavaScript files
Dynamic Content
Enabling interactive and changing web elements
User Experience
Improving user interaction and engagement

```
  let image = document.getElementById("myImage");
```

```
  image.setAttribute("src", "new_image.jpg");
```

```
</script>
```

```
```
```

- element.removeAttribute(attributeName)**:** Removes an attribute.

```
```html
```

~~~~
<div data-info="some data" id="myDiv"></div>

<script>

  let div = document.getElementById("myDiv");

  div.removeAttribute("data-info");

</script>

```
~~~~

Changing Styles

- element.style.propertyName**:** Gets or sets the inline style of an element. propertyName should be the CSS property name in camelCase (e.g., backgroundColor instead of background-color).

~~~~
```html

<p id="myParagraph">This is a paragraph.</p>

<script>

  let paragraph = document.getElementById("myParagraph");

  paragraph.style.color = "red";

  paragraph.style.fontSize = "20px";

</script>

```
~~~~

- element.classList**:** Provides methods for manipulating the CSS classes of an element.

~~~~
*   **`element.classList.add(className)`:** Adds a class.

*   **`element.classList.remove(className)`:** Removes a class.
~~~~

~~~~
*   **`element.classList.toggle(className)`:** Toggles a class (adds it if it's not present, removes it if it is).

*   **`element.classList.contains(className)`:** Checks if an element has a specific class.
~~~~

~~~~
```html

<p id=”myParagraph”>This is a paragraph.</p>

<script>

  let paragraph = document.getElementById(”myParagraph”);

  paragraph.classList.add(”highlight”); // Adds the class ”highlight”

  paragraph.classList.remove(”highlight”); // Removes the class ”highlight”

  paragraph.classList.toggle(”active”); // Adds ”active” if it's not there, removes it if it is.

  console.log(paragraph.classList.contains(”active”)); // Output: true or false

</script>

```
~~~~

Creating and Adding Elements

JavaScript can also be used to create new HTML elements and add them to the DOM.

- document.createElement[tagName]**:** Creates a new element with the specified tag name.

~~~~
```javascript

let newParagraph = document.createElement(”p”);

```
~~~~

- document.createTextNode[text]**:** Creates a new text node.

~~~
```javascript
let textNode = document.createTextNode("This is a new paragraph.");
```
~~~

- element.appendChild[childNode]**:** Appends a child node to an element.

~~~
```javascript
let newParagraph = document.createElement("p");
let textNode = document.createTextNode("This is a new paragraph.");
newParagraph.appendChild(textNode);

let myDiv = document.getElementById("myDiv");
myDiv.appendChild(newParagraph); // Adds the new paragraph to the div
```
~~~

- element.insertBefore[newNode, referenceNode]**:** Inserts a new node before a reference node.

~~~
```javascript
let newParagraph = document.createElement("p");
let textNode = document.createTextNode("This is a new paragraph.");
newParagraph.appendChild(textNode);

let myDiv = document
~~~