<!-- ! DOM -->

- **DOM (Document Object Model)**:

  - The DOM is a programming interface for web documents.
  - It represents the structure of a document as a tree of objects.
  - Each object corresponds to a part of the document (e.g., elements, attributes, and text).
  - The **DOM** allows JavaScript to manipulate the structure, style, and content of a webpage dynamically.

Key Points:

- **Browser's Object**: The DOM is created by the browser, not by JavaScript.
- **Tree Structure**: The document is structured as a tree with nodes representing elements, attributes, and text content.
- **Global Object**: The `window` object in JavaScript is the global object representing the browser window.

2. Accessing Elements in the DOM

Methods to Access Elements:

- **getElementById**: Retrieves an element by its ID.

```
  let element = document.getElementById("elementId");
```

- **getElementsByClassName**: Retrieves elements by their class name. Returns an HTMLCollection (array-like object).

```
  let elements =
document.getElementsByClassName("className");
```

- **getElementsByTagName**: Retrieves elements by their tag name. Returns an HTMLCollection.

```
let elements = document.getElementsByTagName("tagName");
```

- **querySelector**: Retrieves the first element that matches a CSS selector.

```
let element = document.querySelector(".className");
```

- **querySelectorAll**: Retrieves all elements that match a CSS selector. Returns a NodeList.

```
let elements = document.querySelectorAll(".className");
```

3. Manipulating Elements

Properties and Methods to Manipulate Elements:

- **innerText**: Sets or returns the text content of an element.

```
element.innerText = "New Text";
```

- **innerHTML**: Sets or returns the HTML content of an element.

```
element.innerHTML = "<p>New HTML Content</p>";
```

- **textContent**: Returns the text content of an element, including hidden text.

```
console.log(element.textContent);
```

- **getAttribute and setAttribute**: Gets or sets the value of an attribute on an element.

```
  let id = element.getAttribute("id");
  element.setAttribute("id", "newId");
```

- **style**: Modifies the inline CSS of an element.

```
  element.style.backgroundColor = "blue";
```

4. Inserting and Removing Elements

Creating and Adding Elements:

- **createElement**: Creates a new HTML element.

```
  let newElement = document.createElement("div");
  newElement.innerText = "Hello!";
```

- **append, prepend, before, after**: Methods to insert elements at different positions relative to a target element.

```
  parentElement.append(newElement);  // Adds as the last child
  parentElement.prepend(newElement); // Adds as the first child
  targetElement.before(newElement);  // Adds before the target element
  targetElement.after(newElement);   // Adds after the target element
```

Removing Elements:

- **remove**: Removes the element from the DOM.

```
  element.remove();
```

5. Managing Classes

classList Property:
- **add**: Adds a class to an element.
- **remove**: Removes a class from an element.
- **toggle**: Toggles a class (adds if not present, removes if present).
- **contains**: Checks if an element has a specific class.

Example:

```
element.classList.add("newClass");
element.classList.remove("oldClass");
element.classList.toggle("active");
console.log(element.classList.contains("newClass"));
```

<!-- ! Events -->

What is an Event?
An **event** in JavaScript is any action or occurrence recognized by the browser. These can be user actions (such as clicking or typing) or system-generated events (such as page load or media playback).

Event Types
Common event types include:
- **Mouse Events**: click, dblclick, mousedown, mouseup, mouseenter, mouseleave, mousemove, mouseover, mouseout.
- **Keyboard Events**: keydown, keyup, keypress.
- **Form Events**: submit, focus, blur, change, input.

Key Events
- keydown: Triggered when a key is pressed down.
- keyup: Triggered when a key is released.
- keypress: Triggered when a key is pressed (deprecated, use keydown or keyup).

Mouse Events
- click: Triggered when an element is clicked.
- dblclick: Triggered when an element is double-clicked.
- mouseover: Triggered when the mouse pointer is over an element.

Form Events
- submit: Triggered when a form is submitted.
- focus: Triggered when an element receives focus.
- blur: Triggered when an element loses focus.

Event Handling

- **Inline Event Handling**: You can define events directly in HTML attributes.
  html
  ```
  <button onclick="alert('Button clicked!')">Click me</button>
  ```

addEventListener()
- Used to attach event handlers to elements.
- Syntax: element.addEventListener(event, callbackfunction).
  - event: The type of event (e.g., click, submit).
  - callbackfunction: The function to be called when the event occurs.

```
element.addEventListener('click', () => {
  console.log('Element clicked');
});
```

Event Object

When an event occurs, an **Event object** is automatically passed to the event handler. This object contains details about the event:

- event.type: The type of the event (e.g., click).
- event.target: The element that triggered the event.
- event.preventDefault(): Prevents the default action (e.g., stopping a form submission).
- event.stopPropagation(): Stops the event from bubbling up to parent elements.

```
  button.addEventListener('click', function(event) {
    console.log(event.type);  // "click"
    event.preventDefault();   // Stops the default
behavior
  });
```

Event Propagation

Event propagation defines the order in which events are handled. It has two phases:

- **Bubbling Phase**: The event is first captured and handled by the innermost element, then propagated upwards to the outer elements.
- **Capturing Phase**: The event is first captured by the outermost element and propagated to the inner elements.