Month 3: JavaScript, DOM Manipulation, and External APIs

Week 9: DOM Manipulation and Events

The DOM (Document Object Model) is a programming interface that represents a webpage as a structured tree of objects. This week focuses on understanding the DOM and learning how to manipulate it using JavaScript to create dynamic and interactive webpages.

Selecting and Modifying DOM Elements

Understanding the DOM

- The **DOM Tree** represents the HTML structure of a webpage as nested elements.
- Each HTML element is represented as a **node** in the tree, and JavaScript allows us to access and manipulate these nodes directly.

Selecting DOM Elements

To interact with elements on the page, you first need to select them. There are various methods available for this:

1. `document.getElementById(id)`
    - Selects a single element by its `id` attribute.
    - Fast and efficient but limited to targeting elements with unique `id`s.

    Example:

    ```
    const header = document.getElementById("main-header");
    console.log(header.textContent); // Logs the text inside the element with
    id="main-header"
    ```

2. `document.querySelector(selector)`
    - Selects the **first element** that matches a given CSS selector.
    - Flexible and supports complex selectors.

    Example:

    ```
    const firstParagraph = document.querySelector(".content p");
    console.log(firstParagraph.textContent); // Logs the text of the first
    <p> in the .content class
    ```

3. `document.querySelectorAll(selector)`
    - Selects **all elements** matching a given CSS selector and returns a NodeList (a collection of elements).
    - Useful for applying actions to multiple elements.

    Example:

```
const allButtons = document.querySelectorAll("button");
allButtons.forEach((button) => console.log(button.textContent)); // Logs
text of all buttons
```

Modifying DOM Elements

Once an element is selected, you can modify its content, styles, and classes dynamically.

1. **Change Content**
    - `innerHTML`: Gets or sets the HTML content inside an element.
    - `textContent`: Gets or sets the text inside an element, ignoring any HTML tags.

    Example:

```
const title = document.querySelector(".title");
title.textContent = "Welcome to JavaScript!"; // Changes the text of the
element
title.innerHTML = "<strong>Dynamic JavaScript Content</strong>"; // Adds
HTML content
```

2. **Modify Styles**
    - Use the `style` property to directly modify an element's inline styles.

    Example:

```
const box = document.querySelector(".box");
box.style.backgroundColor = "blue"; // Changes the background color
box.style.padding = "20px"; // Adds padding
```

> **Tip**: Use CSS classes for consistent styling and avoid adding many inline styles with JavaScript.

3. **Add and Remove Classes**

- The `classList` property provides methods to add, remove, toggle, and check for classes on an element.

Methods:

- `add("classname")`: Adds a class.
- `remove("classname")`: Removes a class.
- `toggle("classname")`: Adds the class if it doesn't exist; removes it if it does.
- `contains("classname")`: Checks if the element has a specific class.

Example:

```
const menu = document.querySelector(".menu");
menu.classList.add("active"); // Adds the 'active' class
menu.classList.remove("hidden"); // Removes the 'hidden' class
menu.classList.toggle("highlight"); // Toggles the 'highlight' class
```

Practical Exercises

1. **Exercise 1: Change the Heading**
   - Select a `<h1>` element by its `id` and change its text to "JavaScript is Awesome!" using `textContent`.

   Example Solution:

   ```
   const heading = document.getElementById("main-heading");
   heading.textContent = "JavaScript is Awesome!";
   ```

2. **Exercise 2: Modify Styles Dynamically**
   - Select a `<div>` with a class of `card`, and change its background color to `lightgray` and text color to `darkblue`.

   Example Solution:

   ```
   const card = document.querySelector(".card");
   card.style.backgroundColor = "lightgray";
   card.style.color = "darkblue";
   ```

3. **Exercise 3: Add and Toggle Classes**
   - Select a navigation menu and add a class `visible`. Use `classList.toggle()` to hide and show the menu when a button is clicked.

Example Solution:

```
const menu = document.querySelector(".nav-menu");
const toggleButton = document.querySelector("#menu-toggle");

toggleButton.addEventListener("click", () => {
  menu.classList.toggle("visible");
});
```

Summary

In this session, we learned to:

- Select DOM elements using methods like `getElementById`, `querySelector`, and `querySelectorAll`.
- Dynamically modify content, styles, and classes using JavaScript.
- Use the `classList` property to manipulate classes efficiently.

This knowledge lays the groundwork for creating interactive and dynamic web experiences. Next, we'll explore handling events to make webpages respond to user actions.

Handling Events — Click, Input, Submit

JavaScript events allow you to create interactive and dynamic user experiences. Events are triggered by user actions, such as clicking a button, typing in an input field, or submitting a form. In this session, you will learn how to handle these interactions using **event listeners** and control their behavior effectively.

1. Event Listeners: Responding to User Interactions

An **event listener** is a function that listens for a specific event and executes a callback function when that event occurs. The `addEventListener()` method is used to attach event listeners to DOM elements.

Common Event Types

1. **Click Event (`click`)**
    - Triggered when a user clicks on an element (e.g., buttons, links).

    Example:

    ```
    const button = document.querySelector("#myButton");

    button.addEventListener("click", () => {
      alert("Button clicked!");
    });
    ```

2. **Input Event (`input`)**

- Fired whenever the value of an input or textarea changes.

Example:

```
const inputField = document.querySelector("#nameInput");

inputField.addEventListener("input", (event) => {
  console.log(`Current value: ${event.target.value}`);
});
```

3. **Submit Event (`submit`)**

   - Triggered when a form is submitted.
   - Requires preventing the default form behavior to handle the submission manually.

   Example:

```
const form = document.querySelector("#myForm");

form.addEventListener("submit", (event) => {
  event.preventDefault(); // Prevents page reload
  console.log("Form submitted!");
});
```

2. The Event Object: Capturing Event-Specific Details

When an event occurs, the **event object** is automatically passed to the event listener's callback function. This object contains information about the event, such as the element that triggered it, mouse coordinates, or the input's current value.

Key Properties and Methods of the Event Object

1. `event.target`

   - Refers to the element that triggered the event.

   Example:

```
const list = document.querySelector("#itemList");

list.addEventListener("click", (event) => {
  console.log(`You clicked on: ${event.target.textContent}`);
});
```

2. `event.type`

- Returns the type of event that was triggered.

Example:

```
button.addEventListener("click", (event) => {
  console.log(`Event type: ${event.type}`); // Outputs: "click"
});
```

3. `event.preventDefault()`
    - Stops the browser's default behavior for the event (e.g., stopping a link from navigating or a form from submitting).

    Example:

```
const link = document.querySelector("#myLink");

link.addEventListener("click", (event) => {
  event.preventDefault(); // Prevents navigation
  console.log("Default behavior prevented.");
});
```

3. Preventing Default Behavior

Some user interactions have built-in browser behaviors, such as:

- Navigating to a new page when a link is clicked.
- Reloading the page when a form is submitted.

To take control over these actions, you can use `event.preventDefault()` to stop the default behavior.

Use Cases

1. **Custom Form Validation**
    - Prevent the form from submitting if the input fields don't meet specific requirements.

    Example:

```
const form = document.querySelector("#registrationForm");

form.addEventListener("submit", (event) => {
  const username = document.querySelector("#username").value;

  if (username === "") {
    event.preventDefault(); // Stops form submission
    alert("Username cannot be empty!");
```

```
  }
});
```

2. **Prevent Link Navigation**
    - Create a link that performs a JavaScript action instead of navigating.

    Example:

```
const actionLink = document.querySelector("#actionLink");

actionLink.addEventListener("click", (event) => {
  event.preventDefault();
  console.log("Link clicked, but navigation prevented!");
});
```

Practical Exercises 2

1. **Exercise 1: Button Click Handler**
    - Add a `click` event listener to a button that displays an alert with the text "Hello, World!".

    Solution:

```
const button = document.querySelector("#helloButton");
button.addEventListener("click", () => {
  alert("Hello, World!");
});
```

2. **Exercise 2: Dynamic Input Feedback**
    - Add an `input` event listener to a text field that updates a `<p>` element with the current input value.

    Solution:

```
const inputField = document.querySelector("#liveInput");
const output = document.querySelector("#output");

inputField.addEventListener("input", (event) => {
  output.textContent = event.target.value;
});
```

3. **Exercise 3: Form Validation**

- Add a `submit` event listener to a form that checks if an email input is valid. If invalid, prevent the form from submitting and show an error message.

Solution:

```
const form = document.querySelector("#emailForm");
const emailInput = document.querySelector("#email");

form.addEventListener("submit", (event) => {
  if (!emailInput.value.includes("@")) {
    event.preventDefault();
    alert("Please enter a valid email address!");
  }
});
```