**Self-Evaluation Document**

1. Inline Event Handler

* I have implemented an inline event handler for a button in my HTML code:

<button onclick="alert('Website still responsive')">Check for responsiveness</button>

Upon clicking this button, an alert is triggered, demonstrating how inline event handling works directly within HTML elements.

1. Listener Approach

* I have used the addEventListener method to handle button clicks. Here is the relevant JavaScript code:

highlightButton.addEventListener("click", () => {

document.querySelectorAll(".core-table tbody tr").forEach((row, index) => {

if (index % 2 === 0) {

row.style.backgroundColor = "#d3f9d8";

}

});

});

This method allows the button to change the background color of table rows when clicked, demonstrating the use of an event listener.

1. Event Types

* Statement: I have demonstrated multiple types of events:
  + A click event for the button that changes the table row colors.
  + A submit event for form validation before submission.

For example, here is the form submission handler:

document.querySelector("form").addEventListener("submit", (event) => {

event.preventDefault();

// form validation logic

});

Both click and submit events are used effectively to trigger actions in the application.

1. Validating Forms

* I have implemented form validation to ensure required fields are filled out correctly before submission. For example:

if (!dateInput.value) {

messages.push("Date is required.");

}

if (djInput.value.length < 3) {

messages.push("DJ name must be at least 3 characters long.");

}

This validation ensures that the form cannot be submitted unless both fields meet the criteria.

1. Modifying DOM Elements

* I have used JavaScript to modify the DOM. Specifically, I highlight even-numbered rows in the table by changing their background color:

document.querySelectorAll(".core-table tbody tr").forEach((row, index) => {

if (index % 2 === 0) {

row.style.backgroundColor = "#d3f9d8";

}

});

This demonstrates how to dynamically update the appearance of elements in the DOM.

6. Object Properties

* I have utilized object properties to store and manipulate data for the DJ assignments. For example:

let radioHost = { name: "Alex", showName: "Morning Melodies", yearsExperience: 5 };

radioHost.yearsExperience = 6; // Updated yearsExperience

radioHost.timeSlot = "8am - 10am"; // Added new property

This demonstrates both the access and modification of object properties.

7. Window Object

* I have used the window object for a delayed action using setTimeout:

setTimeout(() => {

console.log("This message is shown after 3 seconds");

}, 3000);

This shows how the window object is used for delayed execution of code.

8. Prototypes/Classes

* I have created a class, DJAssignment, to encapsulate the properties and methods for managing DJ assignments:

class DJAssignment {

constructor(date, timeslot, dj) {

this.date = date;

this.timeslot = timeslot;

this.dj = dj;

}

displayAssignment() {

console.log(`DJ ${this.dj} assigned to ${this.timeslot} on ${this.date}`);

}

}

This demonstrates the use of a class to structure the application logically and enables the creation of new DJ assignments.

9. Modules

* I have used modules to separate the functionality of different components of the application. For example, I created a separate module for the DJAssignment class:

// djAssignment.js

export class DJAssignment { ... }

// main.js

import { DJAssignment } from './djAssignment.js';

This modular approach helps to keep the code organized and reusable.

10. JavaScript Fundamentals

* I have utilized fundamental JavaScript concepts, such as:
  + Variables: Using let and const for variable declarations.
  + Comparison operators: For conditional checks (e.g., if statements).
  + Logical operators: Used for complex conditions.
  + Loops: Implemented forEach loops to iterate over DOM elements and apply actions like changing row colors.

This demonstrates my understanding and application of core JavaScript functionality.