**Assignment 1 JavaScript Project**

|  |  |
| --- | --- |
| Project Title | Write project title here |
| Students Name and Roll Number | MUHAMMAD UZAIR  ROLL NO.21014119-167 |
| GitHub Link | Paste GitHub link here. |
| YouTube Link | Paste YouTube video link here. |

**Note for GitHub Link:** (Remove this section before submission of this file)

You will push your project to your GitHub account and then paste the link in above section, so that it can be downloaded from the link to evaluate. You must ensure the link-working before submission of this file. You can attach a detail description about process for downloading of the project along this assignment-file. If the link is not working at the time of evaluation, then marks for the assignment will be 0.

**Note for YouTube Link:** (Remove this section before submission of this file)

You will make a video, upload on your YouTube account, and paste the video-link in above section. To make the video, you will not use mobile camera but you will record computer/mobile screen by any screen recording tool. On the screen, you will show how the project works, how different sections of code are linked with project-output.

**Project Description**

**Project Structure:**

* **HTML Structure:**
  + The HTML file (**index.html**) contains the structure of the webpage. It includes elements for displaying the stopwatch output, buttons for starting, stopping, resetting, and recording laps, and a container for displaying lap times.
* **CSS Styling:**
  + Basic CSS styles are applied to enhance the visual appearance of the stopwatch and buttons. The styles are kept minimal for clarity.
* **JavaScript Logic:**
  + The core functionality of the stopwatch is implemented in the embedded JavaScript code.

**JavaScript Implementation:**

1. **Stopwatch Object:**
   * A JavaScript object named **stopwatch** encapsulates the functionality of the stopwatch.
   * Properties include **startTime**, **running**, **elapsedTime**, and an array **lapTimes** to store recorded lap times.
2. **Start, Stop, and Reset Methods:**
   * The **start** method initializes the stopwatch by setting the start time and kicks off the update loop.
   * The **stop** method halts the stopwatch by clearing the update loop.
   * The **reset** method resets the elapsed time and clears recorded lap times.
3. **Lap Method:**
   * The **lap** method records lap times when the stopwatch is running.
   * It calculates the lap time based on the current time and the previous start time.
   * The lap time is then added to the **lapTimes** array, and the display is updated.
4. **Update Method:**
   * The **update** method is responsible for updating the elapsed time display at regular intervals.
   * It calculates the elapsed time and schedules the next update using **setTimeout**.
5. **Display Methods:**
   * The **displayTime** method formats and displays the elapsed time in minutes and seconds.
   * The **displayLapTimes** method generates HTML to display recorded lap times.
6. **Event Listeners:**
   * Event listeners are attached to HTML buttons to trigger the corresponding methods when the buttons are clicked.
   * The "Start," "Stop," "Reset," and "Lap" buttons are linked to the **start**, **stop**, **reset**, and **lap** methods, respectively.

**User Interaction:**

1. **Start Button:**
   * Clicking the "Start" button initializes the stopwatch, and the elapsed time begins to increment.
2. **Stop Button:**
   * Clicking the "Stop" button pauses the stopwatch, halting the elapsed time.
3. **Reset Button:**
   * Clicking the "Reset" button resets the stopwatch, clearing the elapsed time and lap times.
4. **Lap Button:**
   * Clicking the "Lap" button records the current lap time while the stopwatch is running.
5. **Display:**
   * The current elapsed time is continuously displayed at the top of the webpage.
   * Recorded lap times are displayed below the buttons.

### 

**JavaScript Implementation:**

1. **Stopwatch Object:**
   * A JavaScript object named **stopwatch** encapsulates the functionality of the stopwatch.
   * Properties include **startTime**, **running**, **elapsedTime**, and an array **lapTimes** to store recorded lap times.
2. **Start, Stop, and Reset Methods:**
   * The **start** method initializes the stopwatch by setting the start time and kicks off the update loop.
   * The **stop** method halts the stopwatch by clearing the update loop.
   * The **reset** method resets the elapsed time and clears recorded lap times.
3. **Lap Method:**
   * The **lap** method records lap times when the stopwatch is running.
   * It calculates the lap time based on the current time and the previous start time.
   * The lap time is then added to the **lapTimes** array, and the display is updated.
4. **Update Method:**
   * The **update** method is responsible for updating the elapsed time display at regular intervals.
   * It calculates the elapsed time and schedules the next update using **setTimeout**.
5. **Display Methods:**
   * The **displayTime** method formats and displays the elapsed time in minutes and seconds.
   * The **displayLapTimes** method generates HTML to display recorded lap times.
6. **Event Listeners:**
   * Event listeners are attached to HTML buttons to trigger the corresponding methods when the buttons are clicked.
   * The "Start," "Stop," "Reset," and "Lap" buttons are linked to the **start**, **stop**, **reset**, and **lap** methods, respectively.

**User Interaction:**

1. **Start Button:**
   * Clicking the "Start" button initializes the stopwatch, and the elapsed time begins to increment.
2. **Stop Button:**
   * Clicking the "Stop" button pauses the stopwatch, halting the elapsed time.
3. **Reset Button:**
   * Clicking the "Reset" button resets the stopwatch, clearing the elapsed time and lap times.
4. **Lap Button:**
   * Clicking the "Lap" button records the current lap time while the stopwatch is running.
5. **Display:**
   * The current elapsed time is continuously displayed at the top of the webpage.
   * Recorded lap times are displayed below the buttons.

**Working**

**Detail about project’s component**

**Section 1: Objects**

**Stopwatch Object Explanation:**

The stopwatch object has the following properties:

**StartTime**: Represents the timestamp when the stopwatch started.

**Running**: A boolean flag indicating whether the stopwatch is currently running.

**elapsedTime:** Represents the total elapsed time in milliseconds.

**lapTimes:** An array used to store lap times recorded during the stopwatch operation.

The stopwatch object also includes the following methods:

**Start():** Initiates the stopwatch by setting the start time and starting the update loop.

**Stop():** Pauses the stopwatch by clearing the update loop.

reset(): Resets the stopwatch by setting the elapsed time to 0 and clearing recorded lap times.

**Lap():** Records lap times while the stopwatch is running, calculates lap time, and adds it to the lapTimes array.

**Update():** Continuously updates the elapsed time display at regular intervals using setTimeout.

**DisplayTime()**: Formats and displays the elapsed time in minutes and seconds.

**DisplayLapTimes()**: Generates HTML to display recorded lap times.

// Stopwatch object

const stopwatch = {

startTime: 0,

running: false,

elapsedTime: 0,

lapTimes: [],

start() {

if (!this.running) {

this.startTime = Date.now() - this.elapsedTime;

this.running = true;

this.update();

}

},

stop() {

if (this.running) {

this.running = false;

clearTimeout(this.timer);

}

},

reset() {

this.elapsedTime = 0;

this.lapTimes = [];

if (!this.running) {

this.update();

}

this.displayLapTimes();

},

lap() {

if (this.running) {

const lapTime = Date.now() - this.startTime;

this.lapTimes.push(lapTime);

this.startTime = Date.now();

this.displayLapTimes();

}

},

update() {

if (this.running) {

this.elapsedTime = Date.now() - this.startTime;

this.displayTime();

this.timer = setTimeout(() => this.update(), 10);

}

},

displayTime() {

const minutes = Math.floor(this.elapsedTime / 60000);

const seconds = ((this.elapsedTime % 60000) / 1000).toFixed(2);

document.getElementById("output").innerText = `${minutes}:${seconds}`;

},

displayLapTimes() {

const lapTimesElement = document.getElementById("lapTimes");

lapTimesElement.innerHTML = "<strong>Lap Times:</strong><br>";

this.lapTimes.forEach((lapTime, index) => {

const minutes = Math.floor(lapTime / 60000);

const seconds = ((lapTime % 60000) / 1000).toFixed(2);

lapTimesElement.innerHTML += `Lap ${index + 1}: ${minutes}:${seconds}<br>`;

});

}

};

// Event listeners

document.getElementById("start").addEventListener("click", () => stopwatch.start());

document.getElementById("stop").addEventListener("click", () => stopwatch.stop());

document.getElementById("reset").addEventListener("click", () => stopwatch.reset());

document.getElementById("lap").addEventListener("click", () => stopwatch.lap());

update() {

if (this.running) {

this.elapsedTime = Date.now() - this.startTime;

this.displayTime();

this.timer = setTimeout(() => this.update(), 10);

}

},

displayTime() {

const minutes = Math.floor(this.elapsedTime / 60000);

const seconds = ((this.elapsedTime % 60000) / 1000).toFixed(2);

document.getElementById("output").innerText = `${minutes}:${seconds}`;

},

displayLapTimes() {

const lapTimesElement = document.getElementById("lapTimes");

lapTimesElement.innerHTML = "<strong>Lap Times:</strong><br>";

this.lapTimes.forEach((lapTime, index) => {

const minutes = Math.floor(lapTime / 60000);

const seconds = ((lapTime % 60000) / 1000).toFixed(2);

lapTimesElement.innerHTML += `Lap ${index + 1}: ${minutes}:${seconds}<br>`;

});

}

};

// Event listeners

document.getElementById("start").addEventListener("click", () => stopwatch.start());

document.getElementById("stop").addEventListener("click", () => stopwatch.stop());

document.getElementById("reset").addEventListener("click", () => stopwatch.reset());

document.getElementById("lap").addEventListener("click", () => stopwatch.lap());

**Section 2: Arrays**

### Arrays Explanation:

The **lapTimes** array is employed to keep a record of lap times as users interact with the stopwatch. Each time the "Lap" button is clicked, the current lap time is calculated and added to the array. This array then serves as a container for storing historical lap times, enabling the display of each recorded lap time on the webpage. Arrays are particularly useful in this context as they allow for dynamic and flexible storage of data

// Stopwatch object

const stopwatch = {

startTime: 0,

running: false,

elapsedTime: 0,

lapTimes: [], // The lapTimes array to store recorded lap times

// ... (other methods)

lap() {

if (this.running) {

const lapTime = Date.now() - this.startTime;

this.lapTimes.push(lapTime); // Recording lap time by adding it to the lapTimes array

this.startTime = Date.now();

this.displayLapTimes(); // Updating the display of lap times

}

},

displayLapTimes() {

const lapTimesElement = document.getElementById("lapTimes");

lapTimesElement.innerHTML = "<strong>Lap Times:</strong><br>";

// Iterating through the lapTimes array and displaying each recorded lap time

this.lapTimes.forEach((lapTime, index) => {

const minutes = Math.floor(lapTime / 60000);

const seconds = ((lapTime % 60000) / 1000).toFixed(2);

lapTimesElement.innerHTML += `Lap ${index + 1}: ${minutes}:${seconds}<br>`;

});

}

};

// Event listener for the "Lap" button

document.getElementById("lap").addEventListener("click", () => stopwatch.lap());

**Section 3: Callback Functions**

Here's the part of the JavaScript code that involves the use of callback functions:

// Stopwatch object

const stopwatch = {

// ... (other properties and methods)

// Callback function for starting the stopwatch

start() {

if (!this.running) {

this.startTime = Date.now() - this.elapsedTime;

this.running = true;

this.update();

}

},

// Callback function for stopping the stopwatch

stop() {

if (this.running) {

this.running = false;

clearTimeout(this.timer);

}

},

// Callback function for resetting the stopwatch

reset() {

this.elapsedTime = 0;

this.lapTimes = [];

if (!this.running) {

this.update();

}

this.displayLapTimes();

},

// Callback function for recording lap times

lap() {

if (this.running) {

const lapTime = Date.now() - this.startTime;

this.lapTimes.push(lapTime);

this.startTime = Date.now();

this.displayLapTimes();

}

},

// Callback function for updating the display

update() {

if (this.running) {

this.elapsedTime = Date.now() - this.startTime;

this.displayTime();

this.timer = setTimeout(() => this.update(), 10);

}

},

// Callback function for displaying the elapsed time

displayTime() {

const minutes = Math.floor(this.elapsedTime / 60000);

const seconds = ((this.elapsedTime % 60000) / 1000).toFixed(2);

document.getElementById("output").innerText = `${minutes}:${seconds}`;

},

// Callback function for displaying recorded lap times

displayLapTimes() {

const lapTimesElement = document.getElementById("lapTimes");

lapTimesElement.innerHTML = "<strong>Lap Times:</strong><br>";

this.lapTimes.forEach((lapTime, index) => {

const minutes = Math.floor(lapTime / 60000);

const seconds = ((lapTime % 60000) / 1000).toFixed(2);

lapTimesElement.innerHTML += `Lap ${index + 1}: ${minutes}:${seconds}<br>`;

});

}

};

// Event listeners with callback functions

document.getElementById("start").addEventListener("click", () => stopwatch.start());

document.getElementById("stop").addEventListener("click", () => stopwatch.stop());

document.getElementById("reset").addEventListener("click", () => stopwatch.reset());

document.getElementById("lap").addEventListener("click", () => stopwatch.lap());

},

// Callback function for displaying the elapsed time

displayTime() {

const minutes = Math.floor(this.elapsedTime / 60000);

const seconds = ((this.elapsedTime % 60000) / 1000).toFixed(2);

document.getElementById("output").innerText = `${minutes}:${seconds}`;

},

// Callback function for displaying recorded lap times

displayLapTimes() {

const lapTimesElement = document.getElementById("lapTimes");

lapTimesElement.innerHTML = "<strong>Lap Times:</strong><br>";

this.lapTimes.forEach((lapTime, index) => {

const minutes = Math.floor(lapTime / 60000);

const seconds = ((lapTime % 60000) / 1000).toFixed(2);

lapTimesElement.innerHTML += `Lap ${index + 1}: ${minutes}:${seconds}<br>`;

});

}

};

// Event listeners with callback functions

document.getElementById("start").addEventListener("click", () => stopwatch.start());

document.getElementById("stop").addEventListener("click", () => stopwatch.stop());

document.getElementById("reset").addEventListener("click", () => stopwatch.reset());

document.getElementById("lap").addEventListener("click", () => stopwatch.lap());

**In this code snippet:**

* Callback functions are defined for each method of the **stopwatch** object.
* These callback functions are then attached to the corresponding buttons using event listeners.

When a user clicks a button, the associated callback function is executed, carrying out the specified actions. This use of callback functions helps organize the code and makes it more modular, allowing each button to have its own set of actions without duplicating code

**Section 4: Promises**

Promises in JavaScript provide a way to work with asynchronous operations in a more organized and readable manner. They help mitigate the so-called "callback hell" by allowing you to write more linear and structured code.

**EXAMPLE OF PROMISES**

// Function that returns a promise to simulate an asynchronous operation

function fetchData() {

return new Promise((resolve, reject) => {

// Simulate fetching data after 2 seconds

setTimeout(() => {

const data = { message: 'Data fetched successfully!' };

// Simulate a successful response

resolve(data);

// Simulate an error

// reject(new Error('Failed to fetch data'));

}, 2000);

});

}

// Using the promise

console.log('Start fetching data...');

// Call the fetchData function and handle the promise

fetchData()

.then((data) => {

console.log('Fetched data:', data.message);

})

.catch((error) => {

console.error('Error fetching data:', error.message);

})

.finally(() => {

console.log('Fetching process completed.');

});

**The whole code of my project stopwatch**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Stopwatch</title>

    <style>

        body {

            text-align: center;

            font-family: 'Arial', sans-serif;

            margin-top: 400px;

            background-repeat: no-repeat;

            background-size: 100%,100%;

        }

        #output {

            font-size: 40px;

            margin-bottom: 20px;

        }

        button {

            font-size: 16px;

            padding: 10px 20px;

            margin: 10px;

            cursor: pointer;

        }

        #lapTimes {

            font-size: 18px;

            margin-top: 20px;

        }

    </style>

</head>

<body background="th.jpeg">

<div id="output">0:00</div>

<button id="start">Start</button>

<button id="stop">Stop</button>

<button id="reset">Reset</button>

<button id="lap">Lap</button>

<div id="lapTimes"></div>

<script>

    // Stopwatch object

    const stopwatch = {

        startTime: 0,

        running: false,

        elapsedTime: 0,

        lapTimes: [],

        start() {

            if (!this.running) {

                this.startTime = Date.now() - this.elapsedTime;

                this.running = true;

                this.update();

            }

        },

        stop() {

            if (this.running) {

                this.running = false;

                clearTimeout(this.timer);

            }

        },

        reset() {

            this.elapsedTime = 0;

            this.lapTimes = [];

            if (!this.running) {

                this.update();

            }

            this.displayLapTimes();

        },

        lap() {

            if (this.running) {

                const lapTime = Date.now() - this.startTime;

                this.lapTimes.push(lapTime);

                this.startTime = Date.now();

                this.displayLapTimes();

            }

        },

        update() {

            if (this.running) {

                this.elapsedTime = Date.now() - this.startTime;

                this.displayTime();

                this.timer = setTimeout(() => this.update(), 10);

            }

        },

        displayTime() {

            const minutes = Math.floor(this.elapsedTime / 60000);

            const seconds = ((this.elapsedTime % 60000) / 1000).toFixed(2);

            document.getElementById("output").innerText = `${minutes}:${seconds}`;

        },

        displayLapTimes() {

            const lapTimesElement = document.getElementById("lapTimes");

            lapTimesElement.innerHTML = "<strong>Lap Times:</strong><br>";

            this.lapTimes.forEach((lapTime, index) => {

                const minutes = Math.floor(lapTime / 60000);

                const seconds = ((lapTime % 60000) / 1000).toFixed(2);

                lapTimesElement.innerHTML += `Lap ${index + 1}: ${minutes}:${seconds}<br>`;

            });

        }

    };

    // Event listeners

    document.getElementById("start").addEventListener("click", () => stopwatch.start());

    document.getElementById("stop").addEventListener("click", () => stopwatch.stop());

    document.getElementById("reset").addEventListener("click", () => stopwatch.reset());

    document.getElementById("lap").addEventListener("click", () => stopwatch.lap());

</script>

</body>

</html>