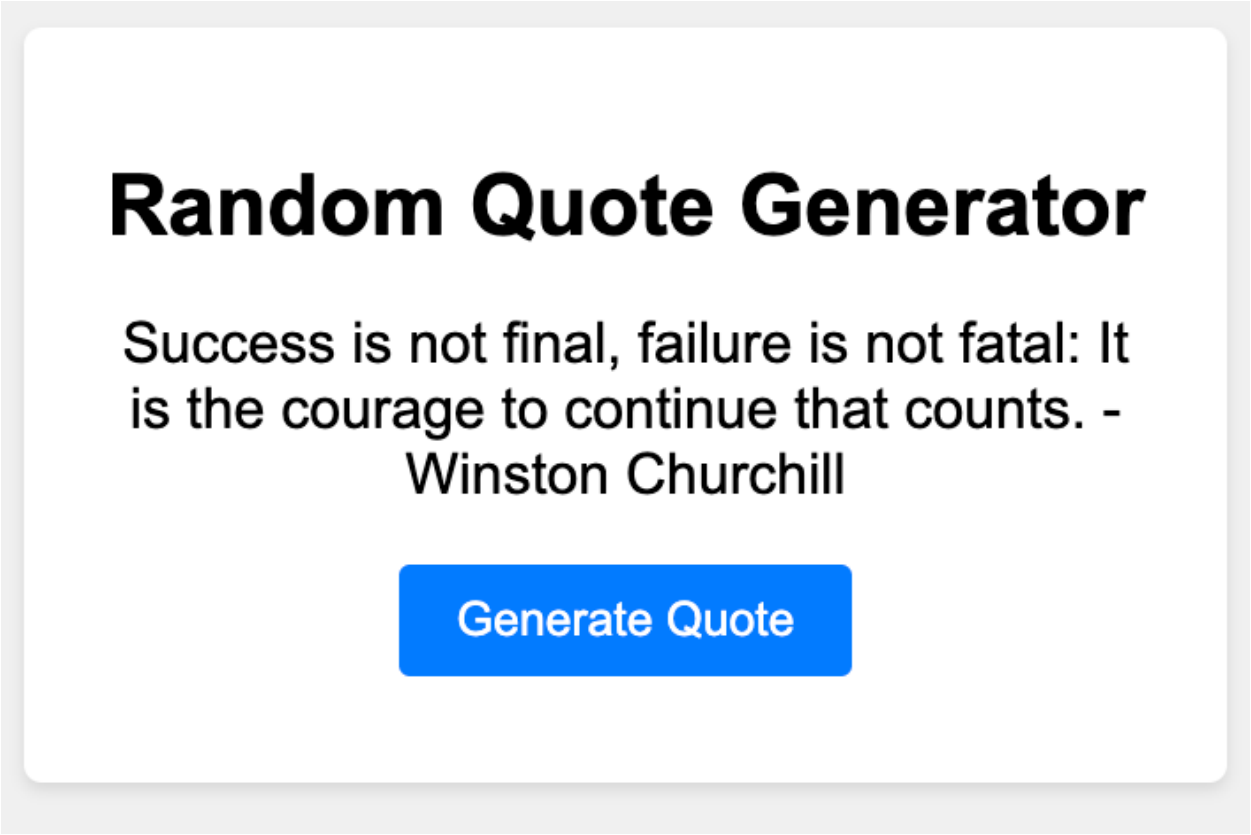
5 JavaScript Coding Projects

Learn JavaScript

Learn more about Google Apps Scripts with Examples and Source Code Laurence Svekis Courses https://basescripts.com/

JavaScript code Example Random Quote Generator Project: Random Quote Generator



Step 1: HTML Structure

Create an HTML file named index.html and set up the basic structure.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Random Quote Generator</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="container">
        <h1>Random Quote Generator</h1>
        <p id="quote">Click the button to generate a random quote.</p>
        <button id="generateButton">Generate Quote</button>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

Step 2: CSS Styling

Create a CSS file named styles.css for basic styling.

```
body {
    font-family: Arial, sans-serif;
    background-color: #f0f0f0;
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
}

.container {
    background-color: #fff;
    padding: 20px;
    border-radius: 5px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
    width: 300px;
    text-align: center;
}

h1 {
    font-size: 24px;
}
```

```
button {
    display: block;
    margin: 10px auto;
    padding: 8px 16px;
    background-color: #007bff;
    color: #fff;
    border: none;
    border-radius: 3px;
    cursor: pointer;
}
```

Step 3: JavaScript Logic

Create a JavaScript file named script.js for the application logic.

```
const quotes = [
   "The only way to do great work is to love what you do. - Steve Jobs",
   "Innovation distinguishes between a leader and a follower. - Steve Jobs",
   "Don't be afraid to give up the good to go for the great. - John D. Rockefeller",
   "Success is not final, failure is not fatal: It is the courage to continue that counts. - Winston Churchill",
   "The future belongs to those who believe in the beauty of their dreams. - Eleanor Roosevelt"
```

```
];

const generateButton = document.getElementById("generateButton");
const quoteElement = document.getElementById("quote");

generateButton.addEventListener("click", generateRandomQuote);

function generateRandomQuote() {
   const randomIndex = Math.floor(Math.random() * quotes.length);
   quoteElement.textContent = quotes[randomIndex];
}
```

Step 4: Testing

Open the index.html file in a web browser. You should see the "Random Quote Generator" with a button. Clicking the button will replace the quote with a random quote from the quotes array.

```
// An array of quotes to display
const quotes = [
    "The only way to do great work is to love what you
do. - Steve Jobs",
```

```
    "Innovation distinguishes between a leader and a
follower. - Steve Jobs",
    "Don't be afraid to give up the good to go for the
great. - John D. Rockefeller",
    "Success is not final, failure is not fatal: It is
the courage to continue that counts. - Winston
Churchill",
    "The future belongs to those who believe in the
beauty of their dreams. - Eleanor Roosevelt"
];

// Get references to the HTML elements
const generateButton =
document.getElementById("generateButton");
const quoteElement = document.getElementById("quote");

// Attach an event listener to the "Generate Quote"
button
generateButton.addEventListener("click",
generateRandomQuote);

// Define the function to generate a random quote
function generateRandomQuote() {
```

```
    // Generate a random index within the range of the
quotes array
    const randomIndex = Math.floor(Math.random() *
quotes.length);

    // Display the randomly selected quote in the
quoteElement
    quoteElement.textContent = quotes[randomIndex];
}
```

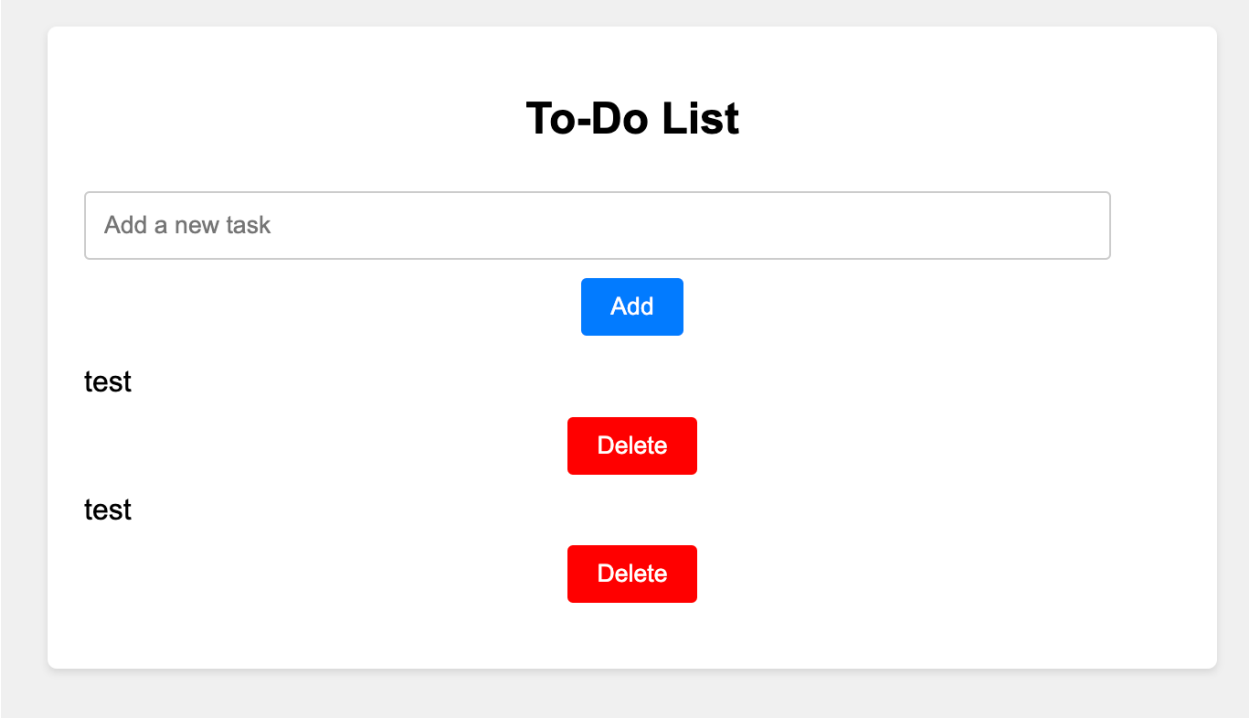
Step by Step Explanation:

1. We start by defining an array called quotes. This array contains a list of quotes as strings. Each quote includes both the quote itself and its attributed author.
2. We get references to the HTML elements we'll interact with: generateButton and quoteElement. The generateButton represents the "Generate Quote" button, and quoteElement represents the paragraph where the quote will be displayed.
3. We attach an event listener to the "Generate Quote" button (generateButton). This listener is set to call the generateRandomQuote function when the button is clicked.
4. The generateRandomQuote function is defined. This function generates a random quote and displays it on the webpage.

5. Inside the generateRandomQuote function:
    a. We use Math.random() to generate a random decimal between 0 (inclusive) and 1 (exclusive).
    b. We multiply the random decimal by the length of the quotes array using quotes.length to get a random number within the range of the array indices.
    c. We use Math.floor() to round down the random number to a whole number, which will be a valid index for the quotes array.
6. We set the textContent of the quoteElement to the randomly selected quote from the quotes array using the randomIndex.

By following these steps, the code creates a simple "Random Quote Generator" that displays a new random quote each time the "Generate Quote" button is clicked. This is a great example of how JavaScript can be used to create dynamic and interactive content on a webpage.

JavaScript code Example ToDo List JavaScript project that creates a basic to-do list application



Here's a simple JavaScript project that creates a basic to-do list application. I'll provide you with the step-by-step description and the full code for each step.

Project: Simple To-Do List App

Step 1: HTML Structure

Create an HTML file named index.html and set up the basic structure.

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Simple To-Do List</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="container">
        <h1>To-Do List</h1>
        <input type="text" id="taskInput"
placeholder="Add a new task">
        <button id="addButton">Add</button>
        <ul id="taskList"></ul>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

Step 2: CSS Styling

Create a CSS file named styles.css for basic styling.

```
body {
    font-family: Arial, sans-serif;
    background-color: #f0f0f0;
```

```
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
}

.container {
    background-color: #fff;
    padding: 20px;
    border-radius: 5px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
    width: 300px;
}

h1 {
    font-size: 24px;
    text-align: center;
}

input[type="text"] {
    width: 100%;
    padding: 10px;
```

```
    margin-top: 10px;
    border: 1px solid #ccc;
    border-radius: 3px;
}

button {
    display: block;
    margin: 10px auto;
    padding: 8px 16px;
    background-color: #007bff;
    color: #fff;
    border: none;
    border-radius: 3px;
    cursor: pointer;
}

ul {
    list-style: none;
    padding: 0;
}
```

Step 3: JavaScript Logic

Create a JavaScript file named script.js for the application logic.

```
const taskInput = document.getElementById("taskInput");
const addButton = document.getElementById("addButton");
const taskList = document.getElementById("taskList");

addButton.addEventListener("click", addTask);

function addTask() {
    const taskText = taskInput.value.trim();

    if (taskText !== "") {
        const li = document.createElement("li");
        li.textContent = taskText;

        const deleteButton =
document.createElement("button");
        deleteButton.textContent = "Delete";
        deleteButton.addEventListener("click", () => {
            taskList.removeChild(li);
        });

        li.appendChild(deleteButton);
        taskList.appendChild(li);
```

```
        taskInput.value = "";
    }
}
```

Step 4: Testing

Open the index.html file in a web browser. You should see the simple to-do list application with an input field and an "Add" button. Enter tasks and click the "Add" button to add them to the list. Each task will have a "Delete" button to remove it from the list.

JavaScript Code Details

```
// Get references to the HTML elements
const taskInput = document.getElementById("taskInput");
const addButton = document.getElementById("addButton");
const taskList = document.getElementById("taskList");

// Attach an event listener to the "Add" button
addButton.addEventListener("click", addTask);

// Define the function to add a task
function addTask() {
    // Get the trimmed value from the task input field
```

```
    const taskText = taskInput.value.trim();

    // Check if the task is not an empty string
    if (taskText !== "") {
        // Create a new <li> element to represent the
task
        const li = document.createElement("li");
        li.textContent = taskText;

        // Create a "Delete" button for each task
        const deleteButton =
document.createElement("button");
        deleteButton.textContent = "Delete";

        // Attach an event listener to the "Delete"
button
        deleteButton.addEventListener("click", () => {
            // Remove the task's corresponding <li>
element
            taskList.removeChild(li);
        });

        // Append the "Delete" button to the task's
<li> element
```

```
        li.appendChild(deleteButton);

        // Append the task's <li> element to the task
list
        taskList.appendChild(li);

        // Clear the input field after adding the task
        taskInput.value = "";
    }
}
```

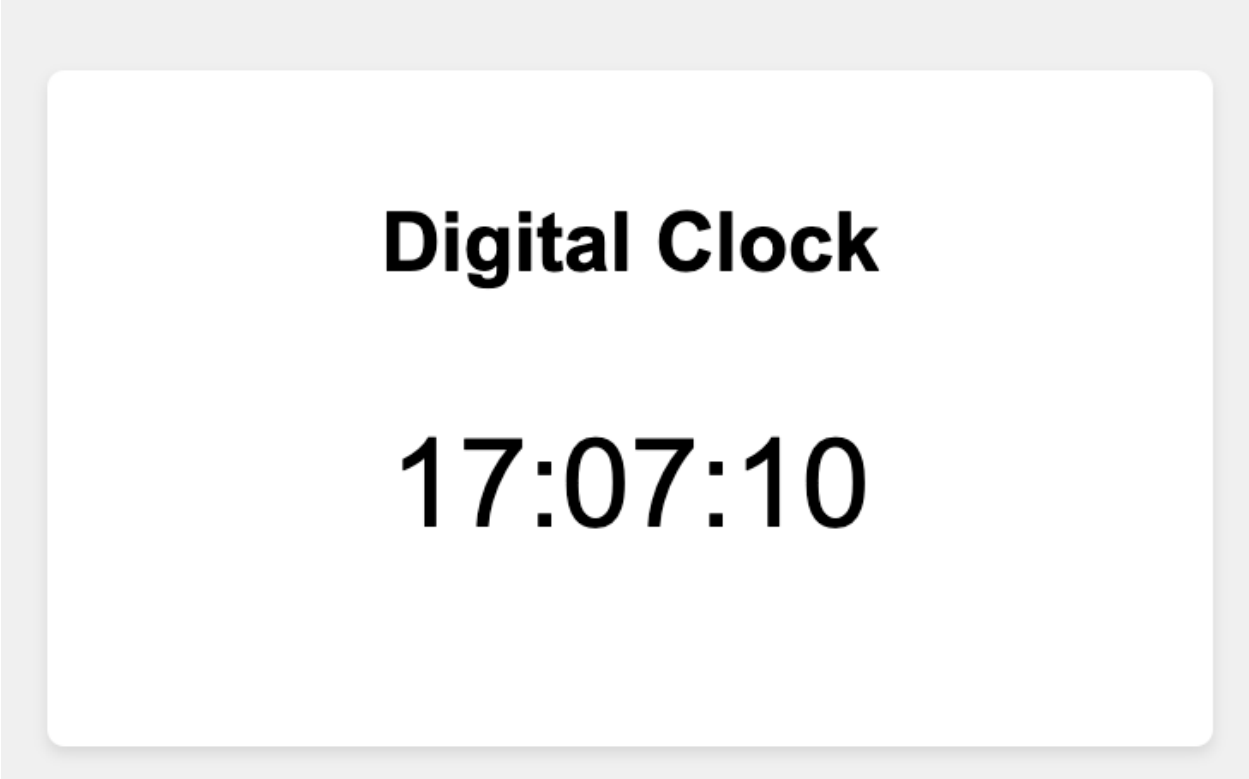
Step by Step Explanation:

1. We start by getting references to the HTML elements we'll be interacting with: taskInput, addButton, and taskList. These are obtained using the getElementById method, which fetches an element by its ID attribute.
2. We attach an event listener to the "Add" button (addButton). This listener is set to call the addTask function whenever the button is clicked.
3. The addTask function is defined. This function is responsible for adding a new task to the list.
4. Inside the addTask function, we retrieve the trimmed value from the task input field using taskInput.value.trim(). Trimming removes any leading or trailing spaces, ensuring we don't add empty tasks.

5. We check if the trimmed task text is not an empty string. If it's not empty, we proceed to create the HTML structure for the new task.
6. We create a new <li> element using document.createElement("li") to represent the task. We set its textContent to the value of the task input.
7. We create a "Delete" button for each task using document.createElement("button"). We set its textContent to "Delete".
8. We attach an event listener to the "Delete" button using deleteButton.addEventListener("click", ...). Inside the listener, we use the taskList.removeChild(li) method to remove the task's corresponding <li> element when the "Delete" button is clicked.
9. We append the "Delete" button to the task's <li> element using li.appendChild(deleteButton).
10. We append the task's <li> element, including the "Delete" button, to the task list (taskList.appendChild(li)).
11. After adding the task to the list, we clear the task input field by setting its value property to an empty string (taskInput.value = "").

By following these steps, the code creates a simple to-do list application where you can add tasks and delete them using the "Delete" button. This is a basic example, and you can build upon it by adding more features and functionality to create a more robust to-do list application.

JavaScript code Example Create a Digital Clock Project: Digital Clock



Step 1: HTML Structure

Create an HTML file named index.html and set up the basic structure.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
```

```
    <title>Digital Clock</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="container">
        <h1>Digital Clock</h1>
        <p id="time"></p>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

Step 2: CSS Styling

Create a CSS file named styles.css for basic styling.

```
body {
    font-family: Arial, sans-serif;
    background-color: #f0f0f0;
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
```

```
    height: 100vh;
}

.container {
    background-color: #fff;
    padding: 20px;
    border-radius: 5px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
    width: 300px;
    text-align: center;
}

h1 {
    font-size: 24px;
}

#time {
    font-size: 36px;
}
```

Step 3: JavaScript Logic

Create a JavaScript file named script.js for the application logic.

```
const timeElement = document.getElementById("time");

function updateTime() {
    const now = new Date();
    const hours = now.getHours().toString().padStart(2,
"0");
    const minutes =
now.getMinutes().toString().padStart(2, "0");
    const seconds =
now.getSeconds().toString().padStart(2, "0");
    const timeString =
`${hours}:${minutes}:${seconds}`;
    timeElement.textContent = timeString;
}

// Call updateTime every second (1000 milliseconds)
setInterval(updateTime, 1000);

// Initial call to display the time immediately
updateTime();
```

Step 4: Testing

Open the index.html file in a web browser. You should see a digital clock displaying the current time, updating every second.

Congratulations! You've successfully created a simple Digital Clock using HTML, CSS, and JavaScript. This project demonstrates how to manipulate time and update content dynamically on a webpage.

Here's the detailed breakdown of the JavaScript code:

We start by selecting the HTML element where we'll display the time using const timeElement = document.getElementById("time");.

We define the updateTime function which:

- Gets the current date and time using new Date().
- Extracts the hours, minutes, and seconds components and formats them with leading zeros using .padStart() method.
- Constructs a time string in the format "HH:MM:SS".
- Updates the content of timeElement with the constructed time string.

We use setInterval(updateTime, 1000); to call the updateTime function every 1000 milliseconds (1 second). This ensures that the clock updates every second.

We also call updateTime(); initially to immediately display the current time when the page loads.

Feel free to explore and expand this project further by adding features like displaying the current date, customizing the clock's appearance, or even implementing time zones.

```
// Get reference to the HTML element where the time
will be displayed
const timeElement = document.getElementById("time");

// Define the function to update the time
function updateTime() {
    // Create a new Date object to get the current time
    const now = new Date();

    // Extract hours, minutes, and seconds from the
Date object
    const hours = now.getHours().toString().padStart(2,
"0");
    const minutes =
now.getMinutes().toString().padStart(2, "0");
    const seconds =
now.getSeconds().toString().padStart(2, "0");

    // Create a formatted time string in "HH:MM:SS"
format
    const timeString =
`${hours}:${minutes}:${seconds}`;
```

```
    // Update the text content of the timeElement with
the new time string
    timeElement.textContent = timeString;
}

// Call updateTime every second (1000 milliseconds) to
keep the clock updated
setInterval(updateTime, 1000);

// Initial call to updateTime to display the time
immediately when the page loads
updateTime();
```

Step by Step Explanation:

1. We start by getting a reference to the HTML element where we want to display the time. In this case, it's the timeElement obtained using document.getElementById("time").
2. We define the updateTime function which is responsible for updating the displayed time.
3. Inside the updateTime function:
    a. We create a new Date object called now to capture the current date and time.
4. We extract the hours, minutes, and seconds components from the now object using the getHours(), getMinutes(), and getSeconds() methods. We

use the .toString().padStart(2, "0") chain to ensure that single-digit values are formatted with a leading zero.

5. We construct a formatted time string using template literals (backticks) with the hours, minutes, and seconds components.
6. We update the text content of the timeElement with the newly formatted time string using timeElement.textContent = timeString.
7. We use the setInterval(updateTime, 1000) function to call the updateTime function every 1000 milliseconds (1 second). This ensures that the displayed time updates dynamically every second.
8. Finally, we call updateTime() initially to immediately display the current time when the page loads.

By following these steps, the code creates a simple digital clock that continuously updates with the current time. This project demonstrates how to use JavaScript to interact with date and time objects, format them, and dynamically update content on a webpage.

JavaScript Code Color Flipper Example Project: Color Flipper

Step 1: HTML Structure

Create an HTML file named index.html and set up the basic structure.

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Color Flipper</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="container">
        <h1>Color Flipper</h1>
        <div class="color-box" id="colorBox"></div>
        <button id="flipButton">Flip Color</button>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

Step 2: CSS Styling

Create a CSS file named styles.css for basic styling.

```
body {
    font-family: Arial, sans-serif;
    background-color: #f0f0f0;
```

```
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
}

.container {
    background-color: #fff;
    padding: 20px;
    border-radius: 5px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
    width: 300px;
    text-align: center;
}

h1 {
    font-size: 24px;
    margin-bottom: 10px;
}

.color-box {
    width: 150px;
```

```
    height: 150px;
    margin: 20px auto;
    border-radius: 5px;
}

button {
    display: block;
    margin: 10px auto;
    padding: 8px 16px;
    background-color: #007bff;
    color: #fff;
    border: none;
    border-radius: 3px;
    cursor: pointer;
}
```

Step 3: JavaScript Logic

Create a JavaScript file named script.js for the application logic.

```
const flipButton =
document.getElementById("flipButton");
const colorBox = document.getElementById("colorBox");
```

```
const colors = ["#007bff", "#28a745", "#dc3545",
"#ffc107", "#17a2b8", "#343a40"];

flipButton.addEventListener("click", flipColor);

function flipColor() {
    const randomIndex = Math.floor(Math.random() *
colors.length);
    colorBox.style.backgroundColor =
colors[randomIndex];
}
```

Step 4: Testing

Open the index.html file in a web browser. You should see a Color Flipper with a colored box and a "Flip Color" button.

Congratulations! You've successfully created a simple Color Flipper using HTML, CSS, and JavaScript. This project demonstrates how you can use JavaScript to change the appearance of an element dynamically based on user interaction.

Here's the detailed breakdown of the JavaScript code:

We start by getting references to the HTML elements we'll be interacting with: flipButton and colorBox.

We attach an event listener to the "Flip Color" button (flipButton). This listener is set to call the flipColor function when the button is clicked.

The flipColor function is defined. This function:

- Generates a random index using Math.random() and Math.floor() within the range of the colors array length.
- Sets the background color of the colorBox element to the randomly selected color from the colors array.

By following these steps, the code creates a simple Color Flipper that changes the background color of a box each time the button is clicked. This project demonstrates how JavaScript can be used to modify the visual properties of elements dynamically.

Step 1: Get References to HTML Elements

```
const flipButton =
document.getElementById("flipButton");
const colorBox = document.getElementById("colorBox");
```

In this step, we're using the document.getElementById() method to retrieve references to the HTML elements we'll be interacting with:

- flipButton: The "Flip Color" button.
- colorBox: The colored box element where we'll change the background color.

Step 2: Attach Event Listener

```
flipButton.addEventListener("click", flipColor);
```

Here, we're attaching an event listener to the "Flip Color" button. The flipColor function will be called when the button is clicked.

Step 3: Define the flipColor Function

```
function flipColor() {
    const randomIndex = Math.floor(Math.random() *
colors.length);
    colorBox.style.backgroundColor =
colors[randomIndex];
}
```

Here's a detailed breakdown of the flipColor function:

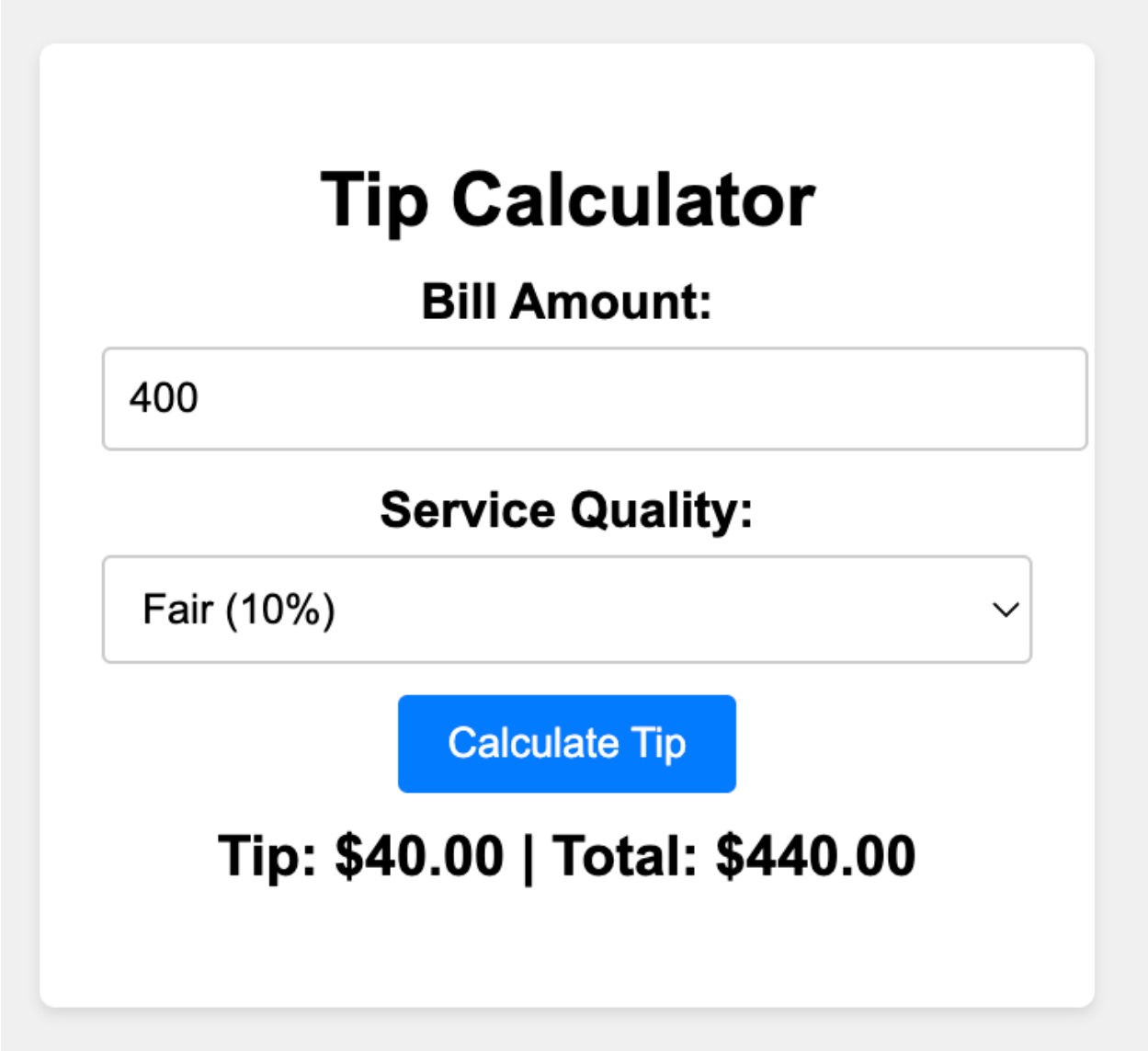
- When the "Flip Color" button is clicked, the flipColor function is executed.
- We use Math.random() to generate a random decimal number between 0 (inclusive) and 1 (exclusive). We then multiply this number by the length of the colors array to get a random index within the array.
- The Math.floor() function is used to round down the decimal index to the nearest integer, ensuring it's a valid index within the array.

- We access the colors array using the random index to get a randomly selected color.
- We set the backgroundColor property of the colorBox element to the randomly selected color. This changes the background color of the box dynamically.

Step 4: Testing

When the user clicks the "Flip Color" button, the flipColor function is executed. This function generates a random index, selects a color from the colors array, and changes the background color of the colorBox element to the selected color.
This project showcases how JavaScript can be used to modify the appearance of elements on a webpage based on user interactions. It's a simple yet effective demonstration of how JavaScript can create dynamic and visually engaging web experiences.

JavaScript Code Tip Calculator Example Project: Tip Calculator



Step 1: HTML Structure

Create an HTML file named index.html and set up the basic structure.

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Tip Calculator</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="container">
        <h1>Tip Calculator</h1>
        <label for="billAmount">Bill Amount:</label>
        <input type="number" id="billAmount"
step="0.01">
        <label for="serviceQuality">Service
Quality:</label>
        <select id="serviceQuality">
            <option value="0.2">Excellent
(20%)</option>
            <option value="0.15">Good (15%)</option>
            <option value="0.1">Fair (10%)</option>
            <option value="0.05">Poor (5%)</option>
        </select>
```

```
        <button id="calculateButton">Calculate
Tip</button>
        <p id="totalTip"></p>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

Step 2: CSS Styling

Create a CSS file named styles.css for basic styling.

```
body {
    font-family: Arial, sans-serif;
    background-color: #f0f0f0;
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
}

.container {
```

```
    background-color: #fff;
    padding: 20px;
    border-radius: 5px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
    width: 300px;
    text-align: center;
}

h1 {
    font-size: 24px;
    margin-bottom: 10px;
}

label {
    display: block;
    margin-top: 10px;
    font-weight: bold;
}

input, select {
    width: 100%;
    padding: 8px;
    margin-top: 5px;
    border: 1px solid #ccc;
```

```
    border-radius: 3px;
}

button {
    display: block;
    margin: 10px auto;
    padding: 8px 16px;
    background-color: #007bff;
    color: #fff;
    border: none;
    border-radius: 3px;
    cursor: pointer;
}

#totalTip {
    font-size: 18px;
    font-weight: bold;
    margin-top: 10px;
}
```

Step 3: JavaScript Logic

Create a JavaScript file named script.js for the application logic.

```
const calculateButton =
document.getElementById("calculateButton");
const billAmountInput =
document.getElementById("billAmount");
const serviceQualityInput =
document.getElementById("serviceQuality");
const totalTipElement =
document.getElementById("totalTip");

calculateButton.addEventListener("click",
calculateTip);

function calculateTip() {
    const billAmount =
parseFloat(billAmountInput.value);
    const serviceQuality =
parseFloat(serviceQualityInput.value);

    if (!isNaN(billAmount) && !isNaN(serviceQuality)) {
        const tipAmount = billAmount * serviceQuality;
        const totalAmount = billAmount + tipAmount;
```

```
        totalTipElement.textContent = `Tip:
$${tipAmount.toFixed(2)} | Total:
$${totalAmount.toFixed(2)}`;
    } else {
        totalTipElement.textContent = "Please enter
valid values.";
    }
}
```

Step 4: Testing

Open the index.html file in a web browser. You should see a Tip Calculator with input fields for the bill amount and service quality, as well as a button to calculate the tip.

Congratulations! You've successfully created a simple Tip Calculator using HTML, CSS, and JavaScript. This project showcases how you can interact with user input and perform calculations on a webpage.

Here's the detailed breakdown of the JavaScript code:

We start by getting references to the HTML elements we'll be interacting with: calculateButton, billAmountInput, serviceQualityInput, and totalTipElement.

We attach an event listener to the "Calculate Tip" button (calculateButton). This listener is set to call the calculateTip function when the button is clicked.

The calculateTip function is defined. This function:

- Retrieves the values entered by the user for the bill amount and service quality.
- Checks if the entered values are valid numbers using isNaN().
- If the values are valid, it calculates the tip amount and the total amount (bill amount + tip amount).
- Formats and displays the tip amount and total amount in the totalTipElement.

By following these steps, the code creates a simple Tip Calculator that calculates the tip and total amount based on the user's input. This project demonstrates how JavaScript can be used to perform calculations and dynamically update content on a webpage.

Step 1: Get References to HTML Elements

```
const calculateButton = document.getElementById("calculateButton");
const billAmountInput = document.getElementById("billAmount");
const serviceQualityInput = document.getElementById("serviceQuality");
const totalTipElement = document.getElementById("totalTip");
```

In this step, we're using the document.getElementById() method to retrieve references to various HTML elements we'll be interacting with:

- calculateButton: The "Calculate Tip" button.
- billAmountInput: The input field where the user enters the bill amount.
- serviceQualityInput: The dropdown select field where the user selects the service quality.
- totalTipElement: The paragraph element where we'll display the calculated tip and total amount.

Step 2: Attach Event Listener

```
calculateButton.addEventListener("click",
calculateTip);
```

Here, we're attaching an event listener to the "Calculate Tip" button. The calculateTip function will be called when the button is clicked.

Step 3: Define the calculateTip Function

```
function calculateTip() {
    const billAmount =
parseFloat(billAmountInput.value);
```

```
    const serviceQuality =
parseFloat(serviceQualityInput.value);

    if (!isNaN(billAmount) && !isNaN(serviceQuality)) {
        const tipAmount = billAmount * serviceQuality;
        const totalAmount = billAmount + tipAmount;

        totalTipElement.textContent = `Tip:
$${tipAmount.toFixed(2)} | Total:
$${totalAmount.toFixed(2)}`;
    } else {
        totalTipElement.textContent = "Please enter
valid values.";
    }
}
```

Here's a detailed breakdown of the calculateTip function:

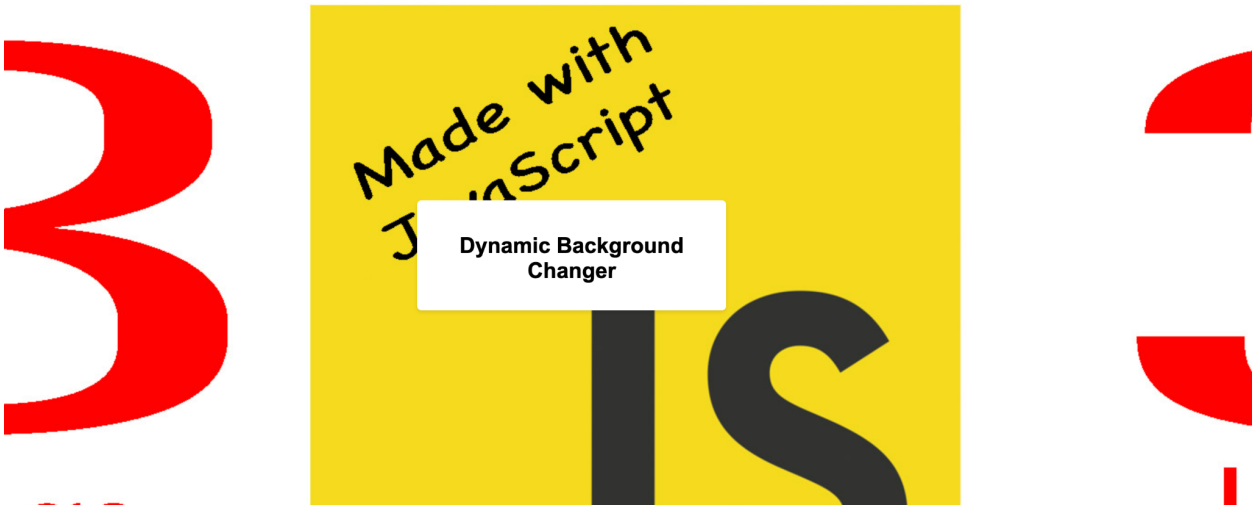
- We start by using parseFloat() to convert the values entered in the billAmountInput and serviceQualityInput fields from strings to floating-point numbers. This allows us to perform calculations with them.
- We use isNaN() to check if the entered values are valid numbers. If both the bill amount and service quality are valid numbers, we proceed with the calculations. If not, we display an error message in the totalTipElement.

- Inside the calculation block:
    - We calculate the tip amount by multiplying the bill amount by the service quality. This gives us the tip amount in dollars.
    - We calculate the total amount by adding the bill amount and the tip amount. This gives us the total amount including the tip.
- We use the toFixed(2) method to format the calculated tip and total amounts to two decimal places, ensuring they look neat and precise.
- Finally, we update the text content of the totalTipElement with the calculated tip and total amounts, formatted as a string.

Step 4: Testing

When the user clicks the "Calculate Tip" button, the calculateTip function is executed. It takes the entered bill amount and service quality, calculates the tip and total amount, and updates the displayed result in the totalTipElement.
This project showcases how JavaScript can be used to interact with user input, perform calculations, and dynamically update content on a webpage. It's a great example of a practical application where JavaScript enhances user experience by providing instant feedback on calculations.

JavaScript Code Dynamic Background Images



Dynamic Background Changer project. This project will create a web page that cycles through captivating background images at a set interval. Here's a step-by-step breakdown along with the full JavaScript code and description.

Project: Dynamic Background Changer

Step 1: HTML Structure

Create an HTML file named index.html and set up the basic structure.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Dynamic Background Changer</title>
```

```
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="container">
        <h1>Dynamic Background Changer</h1>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

Step 2: CSS Styling

Create a CSS file named styles.css for basic styling.

```
body {
    font-family: Arial, sans-serif;
    background-color: #f0f0f0;
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    transition: background-image 1s ease-in-out;
```

```
}

.container {
    background-color: #fff;
    padding: 20px;
    border-radius: 5px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
    width: 300px;
    text-align: center;
}

h1 {
    font-size: 24px;
    margin-bottom: 10px;
}
```

Step 3: JavaScript Logic

Create a JavaScript file named script.js for the application logic.

```
const images = [
    "image1.jpg",
    "image2.jpg",
    "image3.jpg"
```

```
];

const body = document.body;

let currentIndex = 0;

function changeBackground() {
    body.style.backgroundImage =
`url('images/${images[currentIndex]}')`;
    currentIndex = (currentIndex + 1) % images.length;
}

setInterval(changeBackground, 5000);
```

Step 4: Images

Create a folder named images and place your background images (e.g., image1.jpg, image2.jpg, etc.) inside it.

Step 5: Testing

Open the index.html file in a web browser. You should see a web page with the title "Dynamic Background Changer." The background image will change automatically every 5 seconds.

Congratulations! You've successfully created a Dynamic Background Changer using HTML, CSS, and JavaScript. This project demonstrates how you can use JavaScript to create dynamic and visually engaging web experiences by cycling through a series of background images.

Description:

In this project, we've created a web page that showcases a captivating Dynamic Background Changer. Here's how it works:

1. We've defined an array called images that holds the filenames of our background images.
2. We access the body element using document.body.
3. The changeBackground function changes the background image of the body. It sets the backgroundImage property with the path to the current image in the images array.
4. We use the modulo operator (%) to cycle through the images in a loop. This ensures that when the index exceeds the number of images, it wraps around to the first image.
5. Finally, we use setInterval to call the changeBackground function every 5000 milliseconds (5 seconds), causing the background image to change automatically.

By following these steps, the code creates a captivating Dynamic Background Changer that keeps your web page visually engaging and ever-changing.

Step 1: Define the Background Images

```
const images = [
    "image1.jpg",
    "image2.jpg",
    "image3.jpg",
    "image4.jpg",
    "image5.jpg"
];
```

Here, we've defined an array named images that holds the filenames of the background images. You can replace these filenames with your own images. This array acts as a repository of images to cycle through.

Step 2: Access the Body Element

```
const body = document.body;
```

In this step, we're using document.body to get a reference to the <body> element of the HTML document. This will allow us to change the background of the entire page.

Step 3: Create a Counter Variable

```
let currentIndex = 0;
```

We initialize a currentIndex variable to keep track of the current image in the images array. This variable will help us cycle through the images.

Step 4: Define the changeBackground Function

```
function changeBackground() {
    body.style.backgroundImage =
`url('images/${images[currentIndex]}')`;
    currentIndex = (currentIndex + 1) % images.length;
}
```

Here's a detailed breakdown of the changeBackground function:

- When the changeBackground function is called, it does the following:
    - It sets the backgroundImage property of the <body> element using the body.style.backgroundImage property. The template literal ${images[currentIndex]} dynamically selects the current image based on the value of currentIndex.
    - It uses the modulo operator % to increment currentIndex and cycle through the images. When currentIndex reaches the length of the images array, it wraps around to 0, ensuring that the images loop.

Step 5: Set Up Automatic Background Changes

```
setInterval(changeBackground, 5000);
```

We use setInterval to repeatedly call the changeBackground function at a set interval of 5000 milliseconds (5 seconds). This interval will cause the background image to change automatically every 5 seconds.

Summary:

In summary, the JavaScript code for the Dynamic Background Changer project:

- Sets up an array of background image filenames.
- Retrieves the <body> element.
- Maintains a counter to track the current image.
- Defines a function to change the background image and cycle through the images.
- Uses setInterval to repeatedly call the function, resulting in automatic background changes.

By following these steps, the code creates a dynamic and visually engaging web experience where the background images cycle through at a set interval, providing an eye-catching effect on the web page.