Module 2: Website Layouts

Video 2.4: Three Languages HTML, CSS, and JavaScript

To create an interactive web page, you need to use HTML, CSS, and JavaScript together — HTML for the layout, CSS for the styles, and JavaScript for the logic.

Introduction to Interactive Web Pages

Interactive web pages are digital interfaces that allow users to actively engage with content and manipulate elements on a website. Unlike static web pages that only display information without user interaction, interactive web pages enable users to perform various actions, such as selecting buttons, filling out forms, and dragging elements. These pages use a combination of HTML, CSS, and JavaScript to create dynamic and engaging user experiences.

HTML (Hypertext Markup Language): HTML provides the structure and content of a web page. It defines elements such as headings, paragraphs, images, and links. Interactive web pages use HTML to structure the page's content and define the elements that users can interact with.

CSS (Cascading Style Sheets): CSS is used to control the visual appearance and layout of the web page. It enables developers to style elements with colors, fonts, spacing, and positioning. CSS is essential for creating visually appealing and user-friendly interactive elements.

JavaScript: JavaScript adds interactivity and functionality to web pages. It allows developers to manipulate HTML and CSS, handle user interactions, and dynamically update content without requiring a page reload. JavaScript can be used to respond to user actions such as selecting buttons, hovering over elements, and submitting forms.

Interactive web pages encourage users to actively participate, such as by selecting buttons, submitting forms, or inputting data. These interactions trigger specific actions and responses on the page, such as displaying new content, showing pop-up messages, or updating the page's appearance.

Some examples of popular interactive websites are:

1. Social Media Platforms:
   - Facebook
   - Twitter
   - Instagram
2. E-Commerce Websites:
   - Amazon
   - eBay
3. Collaboration and Communication
   - Slack
   - Microsoft Teams

Interaction with Web Pages

Following are various types of interactions commonly used in interactive web pages.

Buttons and Click Interactions:

- **Click Events:** Buttons are one of the most basic interactive elements. When a user selects a button, a JavaScript function can be triggered to perform an action such as submitting a form, toggling content visibility, or navigating to another page.

Forms and Input Interactions:

- **Form Submission:** Users can input data (text, numbers, selections) in form fields and submit the form. JavaScript can validate input and provide feedback before submission.

- **Drop-Downs and Selectors:** Users can choose from predefined options in drop-down menus or select elements.
- **Checkboxes and Radio Buttons:** Users can toggle checkboxes and radio buttons to select or deselect options.
- **Input Validation:** JavaScript can validate input in real time or before form submission, ensuring data integrity and providing error messages.

Animations and Transitions:

- **CSS Transitions:** Elements can smoothly change their appearance (e.g., color, size, position) over a specified duration.
- **CSS Animations:** More complex animations involving keyframes and intermediate steps can be created using CSS.
- **JavaScript Animations:** JavaScript libraries such as GreenSock Animation Platform (GSAP) allow for creating sophisticated animations with more control.

Hover and Mouse Interactions:

- **Hover Effects:** Elements can change appearance (e.g., color, opacity) when the mouse cursor hovers over them.
- **Tooltip Pop-Ups:** Additional information can appear as tooltips when the mouse hovers over certain elements.

Using JavaScript for Interaction

JavaScript is a powerful tool for creating interactive and dynamic web pages. The following is an example on how we can toggle the visibility of an element using JavaScript.

You can create a button that toggles the visibility of an element, such as a drop-down menu or a collapsible section.

```
<button id="toggleButton">Toggle Content</button>
<div id="content" style="display: none">This content can be toggled.</div>
<script>
    var toggleButton = document.getElementById("toggleButton");
    var content = document.getElementById("content");
    toggleButton.addEventListener("click", function () {
        if (content.style.display === "none") {
            content.style.display = "block";
        } else {
            content.style.display = "none";
        }
    });
</script>
```

You can create a button and a hidden content area. When the button is selected, JavaScript toggles the visibility of the content between hidden and visible by changing the display property.

JavaScript can be also used to validate user input in forms before submission.

```
<form id="myForm">
    <input type="text" id="username" placeholder="Username" />
    <input type="password" id="password" placeholder="Password" />
    <button type="submit">Submit</button>
</form>
<script>
    var form = document.getElementById("myForm");
    form.addEventListener("submit", function (event) {
        var username = document.getElementById("username").value;
        var password = document.getElementById("password").value;
        if (username === "" || password === "") {
            alert("Both fields are required.");
            event.preventDefault(); // Prevent form submission
        }
    });
</script>
```

You can create a form with input fields for a username and password. When the form is submitted, JavaScript validates that both fields are filled out. If either field is empty, it displays an alert and prevents the form from being submitted.

Modifying CSS Styles Dynamically Using JavaScript

Modifying CSS styles dynamically with JavaScript is a common way to create interactive and visually appealing web experiences. You can use JavaScript to change various aspects of an element's style, such as its color, size, and position.

First, you need to obtain a reference to the HTML element you want to modify. This can be done using methods such as getElementById, querySelector, or getElementsByClassName.

```
<div id="myDiv">This is a div.</div>
<button id="changeStyleButton">Change Style</button>
```


After getting a reference to the element, you can use JavaScript to modify its style properties using the style object.

```
<script>
    var myDiv = document.getElementById("myDiv");
    var changeStyleButton = document.getElementById("changeStyleButton");
    changeStyleButton.addEventListener("click", function () {
        // Modify the styles dynamically
        myDiv.style.color = "blue";
        myDiv.style.backgroundColor = "yellow";
        myDiv.style.fontSize = "20px";
        myDiv.style.border = "1px solid black";
    });
</script>
```

Instead of directly modifying individual style properties, you can also add, remove, or toggle CSS classes using the classList property. This is particularly useful for applying predefined styles.

```
<style>
    .highlight {
        color: red;
        font-weight: bold;
    }
</style>
<script>
    var myDiv = document.getElementById("myDiv");
    var changeStyleButton = document.getElementById("changeStyleButton");
    changeStyleButton.addEventListener("click", function () {
        // Add or remove a CSS class dynamically
        myDiv.classList.toggle("highlight");
    });
</script>
```

You can use JavaScript to create animations by changing style properties over time. For example, you can use the setInterval or requestAnimationFrame functions to gradually modify properties such as left or opacity.

```
<div id="animatedBox"></div>
<button id="animateButton">Animate</button>
<script>
    var animatedBox = document.getElementById("animatedBox");
    var animateButton = document.getElementById("animateButton");
    animateButton.addEventListener("click", function () {
        var position = 0;
        var animationInterval = setInterval(function () {
            position += 5;
            animatedBox.style.left = position + "px";
            if (position >= 200) {
                clearInterval(animationInterval);
            }
        }, 50);
    });
</script>
```

Form Validation with JavaScript

Now we'll learn how you can perform form validation using JavaScript with the help of an example. In this example, validate a form with a name and an email field. The form should not be submitted if either field is left empty.

```
<form id="myForm">
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" /><br />
    <label for="email">Email:</label>
    <input type="email" id="email" name="email" /><br />
    <button type="submit">Submit</button>
</form>
<script>
    var form = document.getElementById("myForm");
    form.addEventListener("submit", function (event) {
        var nameField = document.getElementById("name");
        var emailField = document.getElementById("email");
        if (nameField.value === "" || emailField.value === "") {
            alert("Both fields are required.");
            event.preventDefault(); // Prevent form submission
        }
    });
</script>
```

The form has two input fields: one for the name (id="name") and one for the email (id="email"). The script adds an event listener to the form's submit event.

Inside the event listener function:

- It retrieves references to the name and email fields using their ID attributes.
- It checks if either the name field or the email field is empty using an if statement.
- If either field is empty, it displays an alert and prevents the form from being submitted using event.preventDefault().

Let's now look through another example for a web page that changes container background color when you select a button.

Create an HTML file (e.g., 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>Interactive Page</title>
        <link rel="stylesheet" href="styles.css" />
    </head>
    <body>
        <div class="container">
            <h1>Interactive Web Page</h1>
            <button id="colorButton">Change Color</button>
        </div>
        <script src="script.js"></script>
    </body>
</html>
```

Create a CSS file (e.g., styles.css) to style the page.

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f0f0f0;
}
.container {
    text-align: center;
    padding: 100px;
}
button {
    padding: 10px 20px;
    font-size: 16px;
    background-color: #007bff;
    color: white;
    border: none;
    cursor: pointer;
}
```

Create a JavaScript file (e.g., script.js) for the interactive functionality.

```
document.addEventListener("DOMContentLoaded", function () {
    var colorButton = document.getElementById("colorButton");
    var container = document.querySelector(".container");
    colorButton.addEventListener("click", function () {
        var randomColor = getRandomColor();
        container.style.backgroundColor = randomColor;
    });
    function getRandomColor() {
        var letters = "0123456789ABCDEF";
        var color = "#";
        for (var i = 0; i < 6; i++) {
            color += letters[Math.floor(Math.random() * 16)];
        }
        return color;
    }
});
```

Link all the files together.

- The HTML file sets up the structure of the web page. It includes a button with the ID "colorButton" that users will select to change the background color.
- The CSS file styles the page, creating a centered container with a button.
- The JavaScript file adds interactivity to the page. It waits for the DOM to be loaded (DOMContentLoaded event) before setting up event listeners.
- When the button is selected, the JavaScript code generates a random color using the getRandomColor function and applies it as the background color of the container.

To see the project in action, create these three files (index.html, styles.css, and script.js) in the same directory, and then open the index.html file in a web browser. When you select the "Change Color" button, the background color of the container will change dynamically.

Why We Separate HTML, CSS, and JavaScript Code

The principle of separation of concerns (SoC) is a fundamental design principle in software development that encourages keeping different aspects of a program or system separate from one another. In the context of web development, it refers to keeping HTML, CSS, and JavaScript separate, each focusing on its own specific role. Here's why this principle is important:

Maintainability and Readability:

Separating HTML, CSS, and JavaScript makes your codebase more organized and easier to read. When each language is contained in its own file, it's simpler to find and update specific parts of your code without wading through unrelated code.

Collaboration:

In collaborative environments, developers often work on different aspects of a project simultaneously. By keeping HTML, CSS, and JavaScript separate, team members can work independently in their respective areas without stepping on each other's toes.

Modularity and Reusability:

Separation of concerns promotes modularity. CSS styles and JavaScript functions can be reused across different pages or components, enhancing code reuse and maintainability. This reduces duplication and promotes a consistent user experience.

Scalability:

As a project grows, maintaining a clear separation of concerns becomes even more crucial. It allows you to make changes or additions to one aspect (e.g., styling) without affecting others (e.g., functionality).

Debugging and Troubleshooting:

When issues arise, it's easier to identify the source of the problem if you have well-organized code. Debugging becomes more efficient when you can isolate the issue to a specific language or file.

Front-End and Back-End Separation:

Separation of concerns is also relevant when considering the distinction between front-end (HTML, CSS, JavaScript) and back-end (server-side) code. It ensures a clear separation between the presentation layer and the data-handling logic.

Exercises

Task 1: Toggle Button

Create a button that toggles the visibility of a hidden paragraph when selected.

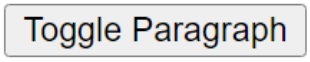
Starter Code (index.html):

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>Toggle Button Exercise</title>
        <link rel="stylesheet" href="styles.css" />
    </head>
    <body>
        <button id="toggleButton">Toggle Paragraph</button>
        <p id="hiddenParagraph" class="hidden">This is a hidden paragraph.</p>
        <script src="script.js"></script>
    </body>
</html>
```

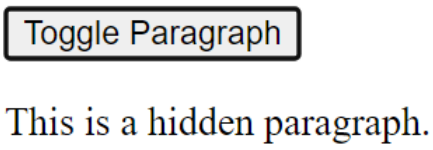
Starter Code (CSS):

```
.hidden {
    display: none;
}
```

Desired Output:



When Button is pressed:



When Button is pressed again:

Instructions:

1. Start by creating the basic structure of your HTML document. You'll need to add a button and a hidden paragraph.
2. The button element with the ID toggle will be the toggle button. The p element with the ID hidden will be the paragraph that will be hidden or shown when the button is selected.
3. In the JavaScript file, start by getting references to the button and hidden paragraph using getElementById.
4. Store the references to the button and hiddenParagraph elements in the var button and var hiddenParagraph variables, respectively.
5. Add a click event listener to the button. When the button is selected, the event listener function is executed.
6. The button.addEventListener("click", function() { ... }) code snippet listens for the click event on the button element and executes the function inside the curly braces when the event is triggered.
7. The function should toggle the hidden class on the hiddenParagraph element, which hides or shows the paragraph depending on whether the class is already present or not.
8. Inside the event listener function, check the current display style property of the hidden paragraph.
9. Toggle the “hidden” class from the class list of p element. If the class is present, it is removed, else, if the class is not present, it is added.

You need to write the logic in script.js file. Go through the discussion above and try to implement the solution. Also, feel free to add styles if you want to. Don’t jump directly to the solution — try it out yourself first.

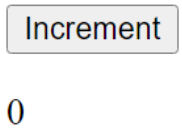
Task 2: Counter

Create a counter that increases when a button is selected.

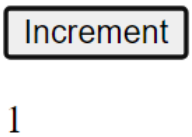
Starter Code (index.html):

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>Counter Exercise</title>
        <link rel="stylesheet" href="styles.css" />
    </head>
    <body>
        <button id="incrementButton">Increment</button>
        <p id="count">0</p>
        <script src="script.js"></script>
    </body>
</html>
```

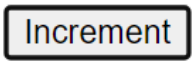
Desired Output:



When Button is pressed:



Pressed again:



2

Instructions:

1. Start by creating the basic structure of your HTML document.
2. You'll need an HTML file with a button to increment the counter and a place to display the current count.
3. The button element should have the ID “increment” so that we can recognize this button that increments the counter.
4. The p element should have the ID counter so that we can recognize the paragraph that displays the current value of the counter.
5. Implement the logic in a JavaScript file (script.js) to handle the counter functionality.
6. Store the references to the button and counter elements using getElementById() in the var button and var counter variables, respectively.
7. The button.addEventListener("click", function() { ... } code snippet listens for the click event on the button element.
8. It then executes the function inside the curly braces when the event is triggered.
9. The function should increment the value of the count variable and update the text content of the counter element to reflect the new value.

You need to write the logic in script.js file. Go through the discussion above and try to implement the solution. Also, feel free to add styles if you want to. Don’t jump directly to the solution — try it out yourself first.

Task 3: Interactive Form

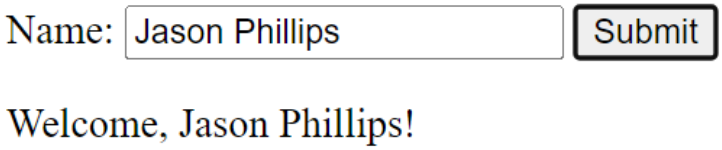
Create a form that displays a welcome message with the entered name when submitted.

Starter Code (index.html):

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>Interactive Form Exercise</title>
        <link rel="stylesheet" href="styles.css" />
    </head>
    <body>
        <form id="nameForm">
            <label for="name">Name:</label>
            <input type="text" id="nameInput" />
            <button type="submit">Submit</button>
        </form>
        <p id="welcomeMessage" class="hidden">
            Welcome, <span id="nameSpan"></span>!
        </p>
        <script src="script.js"></script>
    </body>
</html>
```

Desired Output:

When name is entered and submitted:



Instructions:

1. Start by creating the basic structure of your HTML document.

2. You'll need an HTML file with a form that includes an input field for the user to enter their name, a submit button, and a place to display the welcome message.
3. The form element should have the ID myForm, which will be the form that the user will interact with.
4. The name attribute of the form element is used to identify the form. This attribute is required if you want to submit the form data to a server.
5. The input element should have the type text and the name name which is the input field where the user will enter their name.
6. The input element should have the type submit which is the button that the user will select to submit the form.
7. The script element should contain the JavaScript code that will be executed when the form is submitted.
8. The preventDefault() method of the event object is used to prevent the default action of the form submission, which is to reload the page.
9. The form.addEventListener("submit", function(event) { ... } code snippet listens for the submit event on the form element
10. It executes the function inside the curly braces when the event is triggered.
11. The textContent property of the p element is used to set the text content of the paragraph.
12. The function gets the value of the name input field and sets the text content of the welcomeMessage paragraph to a welcome message that includes the name.

You need to write the logic in script.js file. Go through the discussion above and try to implement the solution. Also, feel free to add styles if you want to. Don’t jump directly to the solution — try it out yourself first.