Name: Lao Rana

Class: A3

Subject: WCT

Homework Day5: Task 1: CSSII

**Task 1:**

**CSS processor** is a tool that extends the capabilities of regular CSS by introducing additional features and syntax. These processors allow developers to write CSS in a more efficient and powerful way by providing features such as variables, nesting, functions, mixins, and more. Once written, the CSS processor compiles the code into standard CSS that browsers can interpret.

**Advantages of Using a CSS Preprocessor**

1. **Variables**: Store values like colors, fonts, or sizes in variables, which makes the CSS more reusable and easier to maintain.
2. **Nesting**: CSS preprocessors allow you to nest your styles, which mimics HTML structure and makes the CSS more readable.
3. **Mixins**: Reuse pieces of CSS code across multiple places without repeating it by using mixins.
4. **Mathematical Operations**: Perform operations such as addition, subtraction, multiplication, and division directly within your styles.
5. **Extensibility**: Preprocessors support the use of functions, loops, and conditionals, allowing for more dynamic and powerful stylesheets.

.container {

padding: 20px;

background-color: #3498db;

}

.container .header {

color: white;

font-size: 16px;

padding: 20px;

}

.container .header-title {

font-weight: bold;

}

.container .footer {

background-color: #2ecc71;

padding: 20px;

border-radius: 10px;

}

**Task 2:**

**CSS compilation** is the process of transforming code written in a CSS preprocessor language (like Sass or LESS) into standard CSS code that browsers can understand and render. Preprocessors like Sass and LESS introduce features such as variables, nesting, mixins, and functions to make CSS more dynamic and maintainable. However, since browsers only understand standard CSS, the preprocessor code needs to be **compiled** into plain CSS.

**How CSS Compilation Works with Preprocessors (Sass, LESS)**

CSS preprocessors like **Sass** or **LESS** allow developers to write more maintainable and feature-rich CSS. However, these preprocessors cannot be directly interpreted by browsers. Therefore, the preprocessor code is compiled into regular CSS code that the browser can render.

**Steps Involved in Compiling Sass to CSS:**

1. **Write Sass Code**: Developers write styles using the Sass syntax, which may include variables, nesting, mixins, etc.
2. **Run a Compiler**: After writing the Sass code, the next step is to compile it. This can be done using a Sass compiler, such as the official **Sass** CLI, **Node-sass**, or build tools like **Webpack** or **Gulp**.
3. **Compilation**: The Sass compiler processes the .scss (or .sass) file and converts it into standard .css code.
4. **Output**: The output is a plain CSS file that can be linked to HTML documents and understood by browsers.
5. **Auto Compilation**: In modern development environments, Sass can be set up to automatically recompile whenever changes are made to the Sass files, ensuring that the CSS stays up-to-date.

**Steps Involved in Compiling Sass to CSS**

1. **Write the Sass code**: The developer writes Sass code using the .scss extension (or .sass if using indented syntax).
2. **Compile the Sass code**: Using a command line tool (like sass), or build tools such as **Webpack** or **Gulp**, the Sass file is compiled into a .css file.
3. **Use the compiled CSS**: The generated .css file is used in the HTML or web application, where the browser can interpret and render it.

**Original file:**

// Variables

$primary-color: #3498db;

$secondary-color: #2ecc71;

$padding: 20px;

$font-size: 16px;

// Mixin for rounded corners

@mixin rounded-corners($radius) {

border-radius: $radius;

}

// Styling the container

.container {

padding: $padding;

background-color: $primary-color;

.header {

color: white;

font-size: $font-size;

padding: $padding;

&-title {

font-weight: bold;

}

}

.footer {

background-color: $secondary-color;

padding: $padding;

@include rounded-corners(10px);

}

}

**After Compiling**

.container {

padding: 20px;

background-color: #3498db;

}

.container .header {

color: white;

font-size: 16px;

padding: 20px;

}

.container .header-title {

font-weight: bold;

}

.container .footer {

background-color: #2ecc71;

padding: 20px;

border-radius: 10px;

}

**Task 3:**

Sass (Syntactically Awesome Stylesheets) is a powerful CSS preprocessor that extends regular CSS with additional features, making stylesheets more efficient and easier to maintain. Sass provides functionality like variables, nesting, mixins, functions, and more, which are not available in standard CSS.

**How Sass Differs from Regular CSS**

* **Nesting**: Sass allows you to nest CSS rules inside one another to mirror HTML structure, which isn't possible in plain CSS.
* **Variables**: Sass introduces variables that let you store values (e.g., colors, font sizes) to reuse throughout the stylesheet, something CSS doesn't support natively.
* **Partials and Imports**: Sass allows splitting your CSS into smaller files (partials) and then importing them into one main file. This makes the codebase easier to maintain, unlike regular CSS where you typically write everything in one large file.
* **Mixins**: In Sass, mixins allow you to create reusable chunks of code with or without parameters, whereas CSS requires copying and pasting code.

**Task 4:**

Sass provides powerful features like \*\*variables\*\* and \*\*functions\*\* to help you write more maintainable and reusable CSS. These features allow you to store values (such as colors, font sizes, and spacing) and perform operations on them dynamically. Below, we will discuss how to declare and use variables and functions in Sass, and also explore the concept of inheritance in Sass.

Example:

// Declaring variables

$primary-color: #3498db;

$secondary-color: #2ecc71;

$font-size: 16px;

$padding: 20px;

body {

font-size: $font-size;

padding: $padding;

background-color: $primary-color;

}

.header {

color: white;

background-color: $primary-color;

padding: $padding;

}

.footer {

background-color: $secondary-color;

padding: $padding;

}

**Explanation:**

1. Variables Declared:

- `$primary-color`, `$secondary-color`, `$font-size`, and `$padding` are all Sass variables that store specific values (like colors and sizes).

2. Using Variables:

- These variables are used throughout the stylesheet in the `body`, `.header`, and `.footer` selectors. If you want to change the primary color or padding globally, you only need to modify the variable, and the change will reflect across the entire stylesheet.

**Using Functions in Sass**

Sass also allows you to define functions, which are used to perform operations on values (such as calculations or transformations) and return a result. Functions in Sass are defined using the `@function` directive.

**Declaring and Using Functions in Sass**

// Declaring a function to lighten a color

@function lighten-color($color, $percentage) {

@return lighten($color, $percentage);

}

$primary-color: #3498db;

.button {

background-color: $primary-color;

padding: 10px 20px;

border-radius: 5px;

color: white;

// Using the lighten-color function to adjust the button's hover state

&:hover {

background-color: lighten-color($primary-color, 20%);

}

}

```

**Explanation:**

1.Function Declaration:

- `@function lighten-color($color, $percentage)` is a custom function that takes two parameters: `$color` and `$percentage`. This function uses the built-in `lighten` function to lighten the color by the specified percentage.

2. Using the Function:

- The function is then used in the `&:hover` selector to lighten the background color of a button when hovered. By using the function, you can easily adjust the color transformation logic and reuse it throughout the stylesheet.

3. Inheritance in Sass

Inheritance in Sass allows one selector to inherit the styles of another, avoiding repetition and making styles more maintainable. Sass provides the `@extend` directive to enable inheritance between selectors. This helps to avoid duplicating styles that are shared among multiple elements.

// Defining a base class with common styles

%button-style {

padding: 10px 20px;

border-radius: 5px;

font-size: 16px;

text-align: center;

cursor: pointer;

}

.button-primary {

@extend %button-style;

background-color: #3498db;

color: white;

}

.button-secondary {

@extend %button-style;

background-color: #2ecc71;

color: white;

}

.button-danger {

@extend %button-style;

background-color: #e74c3c;

color: white;

}

**Explanation:**

1. Base Class with `%` Syntax:

- The `%button-style` is a \*\*placeholder selector\*\* (denoted by `%`) that contains common styles shared across multiple button classes.

- `%button-style` will not be compiled into the final CSS; it's just a template for other classes to inherit from.

2. Using `@extend`:

- The `@extend` directive allows the `.button-primary`, `.button-secondary`, and `.button-danger` classes to inherit all styles from `%button-style`. This reduces redundancy and makes the CSS more concise.

3. Final CSS Output:

The compiled CSS output will look like this:

```css

.button-primary, .button-secondary, .button-danger {

padding: 10px 20px;

border-radius: 5px;

font-size: 16px;

text-align: center;

cursor: pointer;

}

.button-primary {

background-color: #3498db;

color: white;

}

.button-secondary {

background-color: #2ecc71;

color: white;

}

.button-danger {

background-color: #e74c3c;

color: white;

}

As you can see, Sass has removed the redundancy by merging the common styles from `%button-style` into each button class.

**Notable Differences and Benefits of Sass Features:**

- Variables: Sass variables help maintain consistency in your styles by storing frequently used values like colors, font sizes, and margins in a centralized location. If you need to change a value (e.g., changing the primary color), you only need to change it in one place, and the entire stylesheet will automatically update.

- Functions: Functions in Sass help perform operations like calculations or color transformations in a reusable manner, making it easy to adjust and apply consistent styles across a site.

- Inheritance: Using inheritance via the `@extend` directive allows for efficient, non-redundant CSS. Instead of rewriting the same styles for each button, you create a base set of styles and extend them wherever needed, keeping the CSS cleaner and easier to maintain.

**Task 4:**

**What is the Bootstrap Framework?**

**Bootstrap** is a popular front-end framework that provides a collection of tools, components, and utilities for developing responsive and mobile-first web designs. It includes predefined styles, JavaScript components, and a grid system that allow developers to quickly build modern, flexible layouts without having to write extensive CSS from scratch.

Bootstrap helps in developing responsive web designs by utilizing a **responsive grid system**, allowing web pages to adjust fluidly to different screen sizes (e.g., mobile, tablet, desktop). It also includes responsive utility classes, pre-built components like **navbars**, **buttons**, **forms**, **alerts**, and more, that help speed up the development process.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Bootstrap Layout</title>

<!-- Bootstrap CSS -->

<link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">

</head>

<body>

<!-- Navbar -->

<nav class="navbar navbar-expand-lg navbar-light bg-light">

<a class="navbar-brand" href="#">My Website</a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarNav">

<ul class="navbar-nav ml-auto">

<li class="nav-item active">

<a class="nav-link" href="#">Home</a>

</li>

<li class="nav-item">

<a class="nav-link" href="#">Features</a>

</li>

<li class="nav-item">

<a class="nav-link" href="#">Pricing</a>

</li>

</ul>

</div>

</nav>

<!-- Container for Grid Layout -->

<div class="container mt-5">

<div class="row">

<div class="col-md-4">

<div class="card">

<div class="card-body">

<h5 class="card-title">Card 1</h5>

<p class="card-text">This is some content inside card 1.</p>

<a href="#" class="btn btn-primary">Learn More</a>

</div>

</div>

</div>

<div class="col-md-4">

<div class="card">

<div class="card-body">

<h5 class="card-title">Card 2</h5>

<p class="card-text">This is some content inside card 2.</p>

<a href="#" class="btn btn-success">Learn More</a>

</div>

</div>

</div>

<div class="col-md-4">

<div class="card">

<div class="card-body">

<h5 class="card-title">Card 3</h5>

<p class="card-text">This is some content inside card 3.</p>

<a href="#" class="btn btn-danger">Learn More</a>

</div>

</div>

</div>

</div>

</div>

<!-- Bootstrap JS and dependencies -->

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>

<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.4/dist/umd/popper.min.js"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>

</html>

 **Navbar**:

* <nav class="navbar navbar-expand-lg navbar-light bg-light">: This creates a navigation bar. navbar-expand-lg ensures the navbar is collapsible on smaller screens. navbar-light and bg-light apply light-colored text and background.
* <button class="navbar-toggler">: This button appears when the navbar collapses on small screens, allowing users to toggle the menu.
* ml-auto: This class is used to align the navbar items to the right (auto margin-left).

 **Grid System**:

* <div class="container mt-5">: The container class creates a fixed-width container with responsive padding, while mt-5 adds margin-top spacing.
* <div class="row">: Defines a row inside the container, used to hold columns.
* <div class="col-md-4">: Defines a column that takes up 4 out of 12 columns on medium-sized screens (>= 768px).
* The grid system in Bootstrap uses a 12-column layout, and the column widths are responsive, meaning they adjust based on the screen size.

 **Cards**:

* <div class="card">: A card component used for grouping content. The card-body class is used to apply padding inside the card.
* <h5 class="card-title">: The title inside the card.
* <p class="card-text">: The paragraph text inside the card.
* <a href="#" class="btn btn-primary">: This is a Bootstrap button with predefined styles. The class btn applies the basic button style, and btn-primary gives it the primary color.

 **Buttons**:

* btn btn-primary, btn btn-success, btn btn-danger: These classes apply different button styles. btn-primary gives a blue button, btn-success gives a green button, and btn-danger gives a red button.

**Task 6:**

In CSS, **components** are self-contained, reusable building blocks of the user interface. They typically represent individual UI elements like buttons, cards, navigation bars, or modals. By defining these components in CSS or Sass, you can create modular, maintainable, and reusable styles that make your code more efficient and scalable.

A **component-based approach** improves code reusability by:

1. **Encapsulating Styles**: Each component is styled independently, making it easier to maintain and update without affecting other parts of the design.
2. **Reusability**: Once a component is created, it can be reused in multiple places within the application, ensuring consistency in design.
3. **Modularity**: You can isolate different parts of the UI and work on them independently, which simplifies both styling and development.
4. A common project structure for Sass might look like this:
5. less
6. Copy code
7. /scss
8. /components
9. \_card.scss // Card component styles
10. \_button.scss // Button component styles
11. \_navbar.scss // Navbar component styles
12. /partials
13. \_variables.scss // Sass variables
14. \_mixins.scss // Sass mixins and functions
15. main.scss // Main Sass file that imports all partials
16. **Main Sass File Example (main.scss):**
17. scss
18. Copy code
19. // main.scss
20. // Import variables and mixins
21. @import 'partials/variables';
22. @import 'partials/mixins';
23. // Import component styles
24. @import 'components/card';
25. @import 'components/button';
26. @import 'components/navbar';

**Task 7:**

### \*\*CSS Grid System\*\*

The \*\*CSS Grid System\*\* is a powerful layout system that enables you to create complex web layouts by dividing the webpage into rows and columns. Unlike Flexbox, which is primarily one-dimensional (either row or column), Grid works in two dimensions, providing more flexibility to design both rows and columns simultaneously.

With CSS Grid, you can:

- Define a grid container and set up rows and columns.

- Place items into specific areas of the grid.

- Control the size, spacing, and alignment of grid items.

### \*\*Basic Structure of CSS Grid\*\*

A CSS grid layout consists of:

- \*\*Grid Container\*\*: The parent element that defines the grid.

- \*\*Grid Items\*\*: The child elements inside the grid container that are placed into the grid structure.

#### \*\*CSS Grid Syntax\*\*:

```css

/\* Grid Container \*/

.container {

display: grid; /\* Defines the element as a grid container \*/

grid-template-columns: repeat(3, 1fr); /\* Creates 3 equal columns \*/

grid-gap: 10px; /\* Adds space between grid items \*/

}

/\* Grid Items \*/

.item {

background-color: lightblue;

padding: 20px;

}

```

### \*\*Creating a Responsive Layout with CSS Grid\*\*

Here’s an example of how to create a \*\*responsive layout\*\* using CSS Grid that adapts to different screen sizes by using \*\*media queries\*\*.

#### \*\*Step 1: Base Layout with CSS Grid\*\*

```html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Responsive CSS Grid Layout</title>

<style>

/\* Grid Container \*/

.grid-container {

display: grid;

grid-template-columns: repeat(3, 1fr); /\* 3 equal columns \*/

grid-gap: 20px;

padding: 20px;

}

/\* Grid Items \*/

.grid-item {

background-color: lightblue;

padding: 20px;

text-align: center;

font-size: 1.2rem;

}

/\* Media Queries for Responsiveness \*/

@media (max-width: 1200px) {

.grid-container {

grid-template-columns: repeat(2, 1fr); /\* 2 equal columns on medium screens \*/

}

}

@media (max-width: 768px) {

.grid-container {

grid-template-columns: 1fr; /\* 1 column on small screens \*/

}

}

</style>

</head>

<body>

<div class="grid-container">

<div class="grid-item">Item 1</div>

<div class="grid-item">Item 2</div>

<div class="grid-item">Item 3</div>

<div class="grid-item">Item 4</div>

<div class="grid-item">Item 5</div>

<div class="grid-item">Item 6</div>

</div>

</body>

</html>

```

### \*\*Explanation of the Code\*\*

1. \*\*Grid Container\*\*:

- `.grid-container` is the parent element that defines the grid layout.

- `grid-template-columns: repeat(3, 1fr);` creates a grid with 3 equal-width columns. `1fr` is a flexible unit, dividing the available space equally between the columns.

- `grid-gap: 20px;` adds spacing between grid items.

2. \*\*Grid Items\*\*:

- `.grid-item` styles the individual grid items (the boxes inside the grid). It applies background color, padding, and text alignment to each item.

3. \*\*Media Queries\*\*:

- \*\*@media (max-width: 1200px)\*\*: When the screen width is less than or equal to 1200px (e.g., tablets), the grid changes to 2 columns.

- \*\*@media (max-width: 768px)\*\*: When the screen width is less than or equal to 768px (e.g., mobile devices), the grid switches to a single column layout.

These breakpoints (`1200px` and `768px`) allow the layout to adjust to different screen sizes, ensuring a \*\*responsive design\*\*.

### \*\*Using Bootstrap's Grid System\*\*

Bootstrap’s grid system is built on a 12-column layout. It uses a combination of \*\*container\*\*, \*\*row\*\*, and \*\*column\*\* classes to create a responsive grid.

Here’s how you can create a similar responsive layout using \*\*Bootstrap\*\*:

#### \*\*Step 2: Bootstrap Grid Layout\*\*

```html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Responsive Bootstrap Grid Layout</title>

<link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">

</head>

<body>

<div class="container mt-5">

<div class="row">

<div class="col-md-4 col-sm-6 col-12">

<div class="p-3 mb-3 bg-light text-center">Item 1</div>

</div>

<div class="col-md-4 col-sm-6 col-12">

<div class="p-3 mb-3 bg-light text-center">Item 2</div>

</div>

<div class="col-md-4 col-sm-6 col-12">

<div class="p-3 mb-3 bg-light text-center">Item 3</div>

</div>

<div class="col-md-4 col-sm-6 col-12">

<div class="p-3 mb-3 bg-light text-center">Item 4</div>

</div>

<div class="col-md-4 col-sm-6 col-12">

<div class="p-3 mb-3 bg-light text-center">Item 5</div>

</div>

<div class="col-md-4 col-sm-6 col-12">

<div class="p-3 mb-3 bg-light text-center">Item 6</div>

</div>

</div>

</div>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>

<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.4/dist/umd/popper.min.js"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>

</html>

```

### \*\*Explanation of Bootstrap Grid Classes\*\*

1. \*\*`.container`\*\*: Creates a fixed-width container to hold the grid.

2. \*\*`.row`\*\*: Defines a row within the grid to group columns together.

3. \*\*`.col-md-4`\*\*: Specifies that each column will take up 4/12 of the width on medium-sized screens (>= 768px). This means that 3 columns will be displayed side by side on medium and larger screens.

4. \*\*`.col-sm-6`\*\*: Defines the layout for smaller screens (>= 576px). The columns will take up 6/12 of the width, meaning 2 columns per row on small screens.

5. \*\*`.col-12`\*\*: On extra-small screens (mobile devices), each column will take up 12/12 of the width, which means a single column per row.

6. \*\*Responsive Behavior\*\*: Bootstrap automatically handles the layout based on the screen size. The grid adapts and stacks the columns on smaller screens.

### \*\*Media Queries in CSS vs Bootstrap Grid\*

- In the \*\*CSS Grid\*\* example, media queries are explicitly defined to handle layout changes at different screen sizes (e.g., switching from 3 columns to 2 columns, and then to 1 column).

- In \*\*Bootstrap\*\*, the \*\*grid system\*\* already includes predefined breakpoints (like `col-md-4`, `col-sm-6`, `col-12`), which automatically adjust the layout based on screen width without writing custom media queries.

**Task 8:**

### \*\*What is Theming in CSS?\*\*

\*\*Theming in CSS\*\* refers to the practice of defining a set of styles that can be easily swapped or modified to change the appearance of a website. This can be used to create multiple visual styles, such as light or dark themes, that users can toggle between. With \*\*Sass\*\*, theming becomes easier by leveraging variables, functions, and mixins to maintain consistency and manage styles across different themes.

### \*\*How Can Theming be Implemented with Sass?\*\*

In Sass, \*\*variables\*\* are especially useful for theming, as they allow you to define values like colors, fonts, and spacing once, and then reuse them across your stylesheets. By defining theme-specific variables, you can create different themes (e.g., light and dark) and switch between them easily.

A typical theme switcher works by:

1. Defining CSS variables or Sass variables for both themes.

2. Creating a mechanism (e.g., a button) to toggle between themes.

3. Changing the styles dynamically using JavaScript (or CSS alone with `prefers-color-scheme` for system-level dark mode support).

### \*\*Building a Light and Dark Theme Switcher Using Sass\*\*

Let's create a simple \*\*theme switcher\*\* that toggles between a light and dark theme. We'll use Sass to define the theme-specific styles, and then add a button to switch between the two themes.

#### \*\*Step 1: Define the Sass Variables for Both Themes\*\*

We'll define separate variables for light and dark themes, so we can easily switch between them.

```scss

// \_variables.scss

/\* Light Theme Variables \*/

$light-bg: #fff;

$light-text: #333;

$light-primary: #3498db;

/\* Dark Theme Variables \*/

$dark-bg: #2c3e50;

$dark-text: #ecf0f1;

$dark-primary: #9b59b6;

```

#### \*\*Step 2: Set Up Global Styles Using Variables\*\*

Here we use the variables to define the basic layout and typography for both themes.

```scss

// \_base.scss

/\* Global Styles \*/

body {

font-family: Arial, sans-serif;

transition: background-color 0.3s, color 0.3s;

}

button {

padding: 10px 20px;

font-size: 1rem;

cursor: pointer;

transition: background-color 0.3s, color 0.3s;

}

/\* Light Theme Styles \*/

.light-theme {

background-color: $light-bg;

color: $light-text;

}

.light-theme button {

background-color: $light-primary;

color: $light-bg;

}

.light-theme button:hover {

background-color: darken($light-primary, 10%);

}

/\* Dark Theme Styles \*/

.dark-theme {

background-color: $dark-bg;

color: $dark-text;

}

.dark-theme button {

background-color: $dark-primary;

color: $dark-bg;

}

.dark-theme button:hover {

background-color: darken($dark-primary, 10%);

}

```

#### \*\*Step 3: Create the HTML Structure\*\*

Now, let's create the HTML structure with a button to toggle between the light and dark themes.

```html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Theme Switcher</title>

<link rel="stylesheet" href="styles.css">

</head>

<body class="light-theme">

<div class="container">

<h1>Welcome to the Theming Example</h1>

<p>This is a simple theme switcher using Sass.</p>

<button id="theme-toggle">Switch to Dark Theme</button>

</div>

<script>

const themeToggleButton = document.getElementById('theme-toggle');

themeToggleButton.addEventListener('click', () => {

document.body.classList.toggle('dark-theme');

document.body.classList.toggle('light-theme');

themeToggleButton.textContent = document.body.classList.contains('dark-theme') ? 'Switch to Light Theme' : 'Switch to Dark Theme';

});

</script>

</body>

</html>

```

### \*\*Explanation of the Code\*\*

#### \*\*Sass Code\*\*

1. \*\*Variables\*\*:

- We define separate color variables for the light and dark themes (`$light-bg`, `$dark-bg`, etc.) in a partial Sass file (`\_variables.scss`). This makes it easy to manage the theme values.

2. \*\*Base Styles\*\*:

- In the `\_base.scss` file, we define global styles for the `body` and `button` elements that will apply to both themes. We use the variables for the background color, text color, and button styles.

3. \*\*Theme-Specific Styles\*\*:

- The `.light-theme` and `.dark-theme` classes are used to apply the corresponding theme styles. These classes control the background color, text color, and button styles for the light and dark themes.

- The `transition` property is used to animate changes in background and text color smoothly when switching themes.

#### \*\*HTML Structure\*\*

1. The `<body>` element initially has the `light-theme` class applied, so the page loads with the light theme.

2. The `button` element toggles between light and dark themes by adding or removing the `.dark-theme` or `.light-theme` class on the `<body>` element.

3. The `textContent` of the button also changes based on the current theme, so users can easily know what action they can take next (e.g., "Switch to Dark Theme" or "Switch to Light Theme").

#### \*\*Step 4: CSS Output (After Sass Compilation)\*\*

After compiling the Sass code, the resulting CSS will look like this:

```css

/\* Global Styles \*/

body {

font-family: Arial, sans-serif;

transition: background-color 0.3s, color 0.3s;

}

button {

padding: 10px 20px;

font-size: 1rem;

cursor: pointer;

transition: background-color 0.3s, color 0.3s;

}

/\* Light Theme Styles \*/

.light-theme {

background-color: #fff;

color: #333;

}

.light-theme button {

background-color: #3498db;

color: #fff;

}

.light-theme button:hover {

background-color: #2980b9;

}

/\* Dark Theme Styles \*/

.dark-theme {

background-color: #2c3e50;

color: #ecf0f1;

}

.dark-theme button {

background-color: #9b59b6;

color: #2c3e50;

}

.dark-theme button:hover {

background-color: #8e44ad;

}