We can use Input elements and buttons (<input>, <textarea>, <select>) outside a <form> element. In this case Input element accepts data,

However no data submission to server happens unless handled via JavaScript.

<input type="checkbox" id="switcher" >

<label for="switcher">Switch</label>

In the for attribute we mention id of element we want label for

The for attribute in a <label> must always reference an id, not a class.

We can wrap the element we need label for in label element in that we we won’t need an for attribute in label element and id for the input element

The <label> element is used to associate a descriptive text with an input element (like checkboxes, radio buttons, text fields, etc.).

Helps screen readers announce the purpose of the input field.

Users with disabilities can better navigate the form.

Without a label, users have to click directly on the small checkbox.

With a label, clicking on the text "Switch" also toggles the checkbox.

Easier for mobile users to tap large text rather than a small checkbox

The <label> element is specifically meant for form controls, mainly:

<input> (text, checkbox, radio, etc.)

<textarea>

<select>

It should not be used for non-form elements like <div>, <span>, <p>, etc.

body [data-theme="light"] {

--color-1: rgb(196,220, 241);

--color-2: white;

--color-3: rgb(200,200,200);

--color-4: rgb(80, 82, 110);

}

means that these variables are accessible in <body> only when data-theme="light".

body [data-theme="light"]{

.

Css styling

.

}

[data-theme="dark"] is an attribute selector.

It selects only those elements (in this case, <body>) that has a data-theme="dark" attribute. And appy css styling

For example we have 3 p elements and 2 of them has above attribute , so css styling will be applicable only on 2 p element which has that attribute

p[data-theme="dark"] {

color: white;

background-color: black;

padding: 5px;

}

Why use variable colors rather than color directly

--color-1:red;

Easier to Maintain & Update Colors

Suppose we have applied red color on more than 3 elements , now due to some reason I decided to change color from red to white for all of them , then I have to go and change for each element but if I’m using variable , I have to only update the value of variable from red to white

If variables are defined in :root (Global Scope)

Then it can be used even outside body , anywhere in the document, including header, footer, etc.

:root is a CSS pseudo-class selector, NOT an HTML element.

It selects the highest-level element in the document, which is the <html> element.

But if defined in a specific element or class (local scope) then can be used only in that one

They are Inheritable can be used in Variables child elements.

why Are CSS Variables Defined in <body> Not Accessible in <header> and <footer>?

Both <header> and <footer> are children of <body> in HTML, so it makes sense to assume that variables defined in body should be accessible in them. However, CSS handles variable inheritance differently than HTML structure.

CSS variable scope is determined by where they are declared in the stylesheet.

CSS variables are not automatically inherited like other CSS properties (e.g., color).

If header is styled separately before body in CSS, it won’t have access to the variable.

If you style <body> before <header>, then <header> and <footer> can inherit the variables.

CSS is applied in the order it appears in the stylesheet.

When the browser processes the CSS for <header>, it checks for --main-bg, but at that point, body's CSS has not been processed yet.

Since the variable doesn't exist yet, the value is undefined, and CSS ignores it.

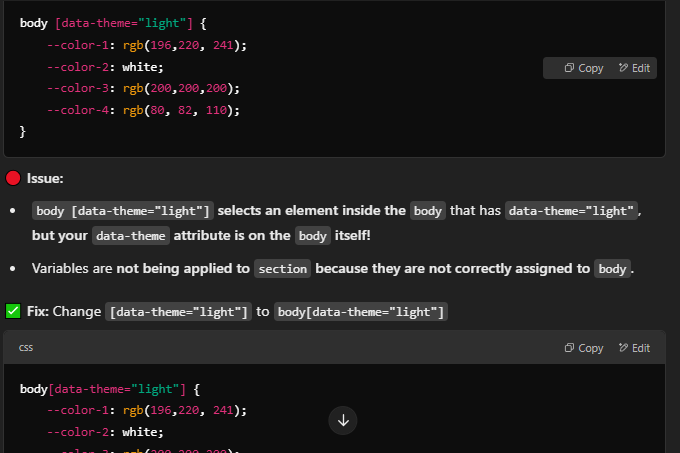
Later, when body is processed, the variable exists—but it’s too late for <header> to use it, unless <header> is styled again afterward.

Error

body [data-theme="light"] selects an element inside the body that has data-theme="light" atrribute, but your data-theme attribute is on the body itself!

Variables are not being applied to section because they are not correctly assigned to body.

Fix : Change [data-theme="light"] to body[data-theme="light"]



It’s crazy the way just a space can make that big of a difference

margin: 0 auto;

0 → Top & Bottom margin is 0 (No space above and below the element).

auto → Left & Right margins are automatically adjusted to center the element horizontally.

This works only if the element has a specified width.

Without a width, the element takes the full available space, and auto won’t center it.

The Element Must Be Block-Level

Doesn't work on inline elements (span, a, etc.).

Works on block-level elements (div, section, article, etc.).

The Element Must Have a Defined Width

Doesn't work if width is 100% or not set (default full-width behavior).

Works when width is smaller than the parent (width: 50%, 600px, etc.).

The Element Should Be Inside a Parent

Doesn't work properly if it's a direct child of <body> (depends on other styles).

Works best when inside a parent container with a larger width.

The Parent Should Allow Space for Centering

Doesn't work if the parent has display: flex; and justify-content: space-between;

Works with normal block flow or display: flex; justify-content: center;

Does Not Work for Vertical Centering

margin: 0 auto; only centers horizontally, not vertically.

we need to explicitly set border: none; because some elements have a default border applied by browsers.

Some Elements Have Default Borders

Certain elements come with built-in borders, even if you don’t specify one in CSS.

Example: Buttons & Inputs

Without border: none;, the button might have a default border depending on the browser.

why Hide the <input> (Checkbox)

input {

width: 0;

height: 0;

visibility: hidden;

}

Hides the checkbox while keeping it functional.

Why Not Just Remove It?

The checkbox still works behind the scenes.

We replace its appearance with a label, which we can design as a switch (toggle).

Clicking the label still toggles the checkbox because of for="id" in HTML.

Reason for Setting width: 0; height: 0;

✔ Completely hides the checkbox visually without removing its functionality.

✔ Prevents the default checkbox appearance from taking up space.

Setting width: 0; height: 0; already makes the checkbox invisible.

However, visibility: hidden; is added as an extra safety measure to ensure it is completely hidden and doesn't interfere with the layout.

label{

display: block;

text-indent: -9999px;

height: 30px;

width: 60px;

background-color: aliceblue;

border-radius: 50px;

}

text-indent: -9999px;

✔ Moves the text far off-screen, effectively hiding it but keeping it accessible to screen readers.

The text-indent property in CSS is mainly used to shift the starting position of text inside an element. However, in some cases, developers use it to hide text visually while keeping it accessible.

Often used for hiding text inside buttons, labels, or icons

A very large negative value moves text far off-screen, effectively hiding it.

text-indent: -9999px;

✔ Keeps text hidden visually but accessible for screen readers.

✔ Useful when replacing text with a background image or icon.

✔ The label text is hidden, but users can still click the switch.

✔ Instead of text, a background or toggle design will be showing

label remains functional and clickable even though the text inside it is hidden using text-indent. It still acts as a trigger for the theme switch.

If we had used display: none; or visibility: hidden; it would have would remove the text from the accessibility tree, making it invisible to screen readers.

text-indent: -9999px; hides the text visually but keeps it accessible.

Final Summary

✅ Hiding the input: We want functionality but not the default checkbox look.

✅ Using a label: It acts as a toggle switch while still controlling the hidden checkbox.

✅ Using text-indent: -9999px;: Hides text without breaking accessibility.

The display: block; property makes an element behave as a block-level element. This means:

By default, <label> is an inline element, meaning:

It only takes up as much space as its content.

You cannot set width, height, margin (top/bottom), or padding (top/bottom) effectively.

This property made a label behave like a button or switch and Created full-width clickable area

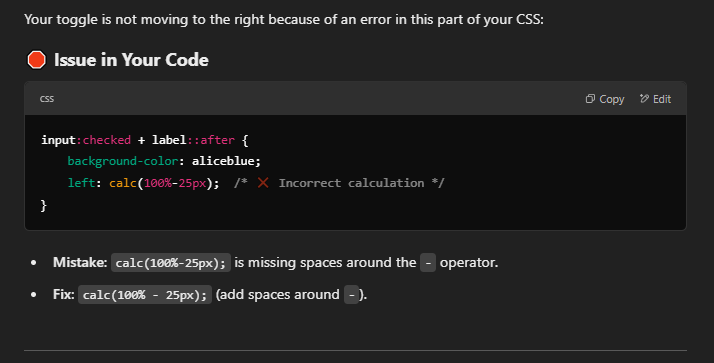
label::after {

position: absolute;

}

::after creates a pseudo-element that behaves like a child of label.

it's used to create the small circular toggle switch that moves when the checkbox is checked.



Okhhhhhhh

input:checked+label {

background-color: rgb(25, 26, 37);

border: 2px solid whitesmoke;

}

`input:checked` selects any input element (like a checkbox or radio button) that is currently checked (selected).

:checked: This is a pseudo-class that targets the <input> element only when it is in a checked state (e.g., a checkbox or radio button that has been selected).

`+`:This is an adjacent sibling combinator, which targets the element that immediately follows the previously targeted element.

In essence, this selector targets any label element that is directly adjacent to a checked input element.

A pseudo-class is a keyword that allows you to target specific states or conditions of an HTML element. Pseudo-classes are used to apply styles to elements based on their current state, such as:

1. Hover: When the user hovers over an element.

2. Active: When the user clicks on an element.

3. Focus: When an element receives focus (e.g., when a form field is selected).

4. Visited: When a link has been visited.

5. Checked: When a checkbox or radio button is selected.

Pseudo-classes are denoted by a colon (:) followed by the pseudo-class name. For example:

- a:hover targets anchor elements when they are hovered over.

- input:focus targets input elements when they receive focus.

- input:checked targets input elements when they are checked (selected).

Pseudo-classes allow you to create dynamic and interactive styles that respond to user interactions, without the need for JavaScript.

A pseudo-element in CSS is used to style a specific part of an element, rather than the entire element itself.

They are written with double colons (::) to differentiate them from pseudo-classes (:).

Common Pseudo-Elements

Pseudo-Element What It Does

::before Inserts content before an element.

::after Inserts content after an element.

::first-letter Styles the first letter of text.

::first-line Styles the first line of text.

::selection Styles the text when selected.

::placeholder Styles placeholder text in inputs.

const input = document.querySelector('.theme-switcher input')

Selects the <input> element inside the container with class name .theme-switcher .

if (e.target.checked) {

document.body.setAttribute('data-theme', 'dark')

If target element (input) value is checked , execute the following code

The target property in an event object (e.target) refers to the element that triggered the event. What it refers to depends on the type of event.