CSS Custom (--*) Properties

Anumeha Gupta

Momina

Introduction to **CSS Custom properties**

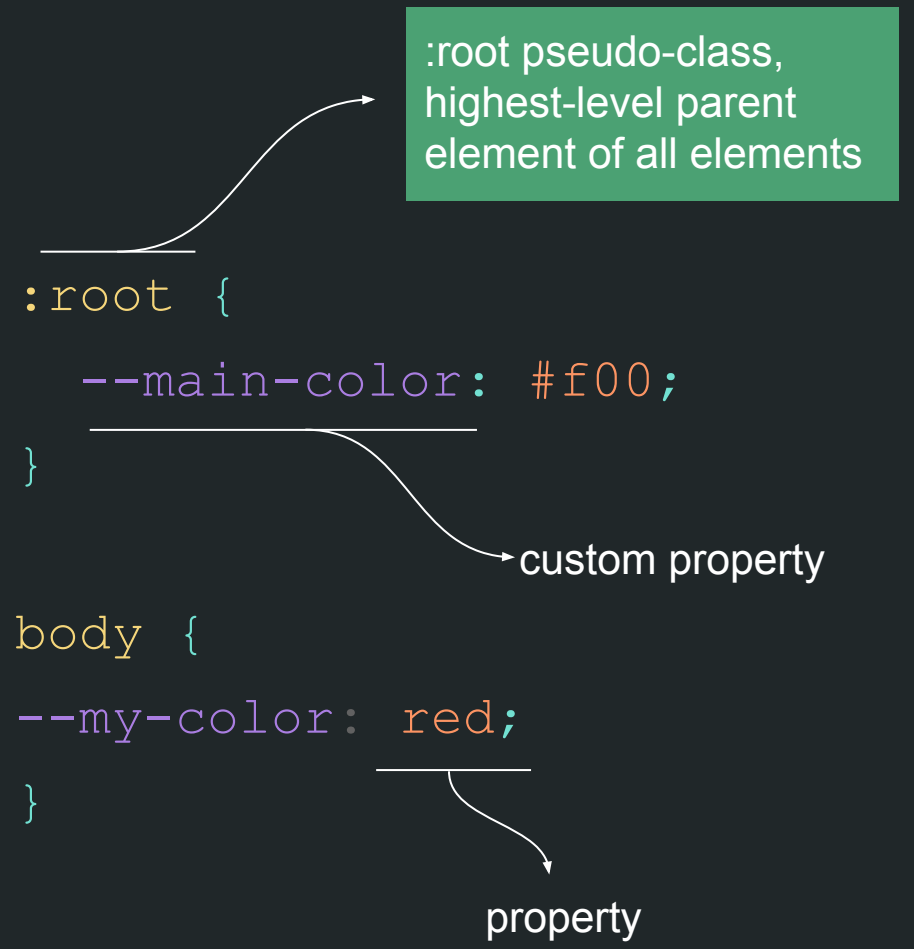
What is a custom property in CSS?

CSS custom properties, also known as CSS variables, are a CSS feature that allows you to assign custom values to properties in your stylesheet. They allow you to **store values that can be reused throughout your stylesheet**, making it easier to maintain and update your styles.

How to declare a custom property?

Custom properties are declared using the **--syntax**, followed by the property name and a value.

For example:



Declaring a custom property

Correct ways to declare custom property

General mistakes to be avoided in declaring custom property:

1. no double-dash (--) or one dash (-)
2. **case sensitive property** declaration
3. **no special characters** allowed

```
/* Nope, not within a selector */
 --foo: 1;

body {
/* No, 0 or 1 dash won't work */
foo: 1;
-foo: 1;

/* Yep! */
 --foo: 1;

/* OK, but they're different properties */
--FOO: 1;
--Foo: 1;


 /* Special characters are a no */
 --color@home: red;
--black&blue: black;
 --black^2: black;
}
```

Mistake generally made in declaring custom property

Valid values for custom property

The property will be declared as invalid and will either use the default property of browser or similar property defined in the CSS.

Anything can be created as a property in CSS, hence allowing us to provide a flexible and powerful way to create and manage styles in modern web development.

```css
body {
  --brand-color: #990000;
  --transparent-black: rgba(0, 0, 0, 0.5);

  --spacing: 0.66rem;
  --brandAngle: 22deg;

  --visibility: hidden;
  --my-name: "Chris Coyier";
}
```

```css
.hero-section {
  background-color: var(--brand-color);
}
```

```css
.hero-section {
  background-color: var(--spacing);
}
```

Properties as properties

Property inside a property

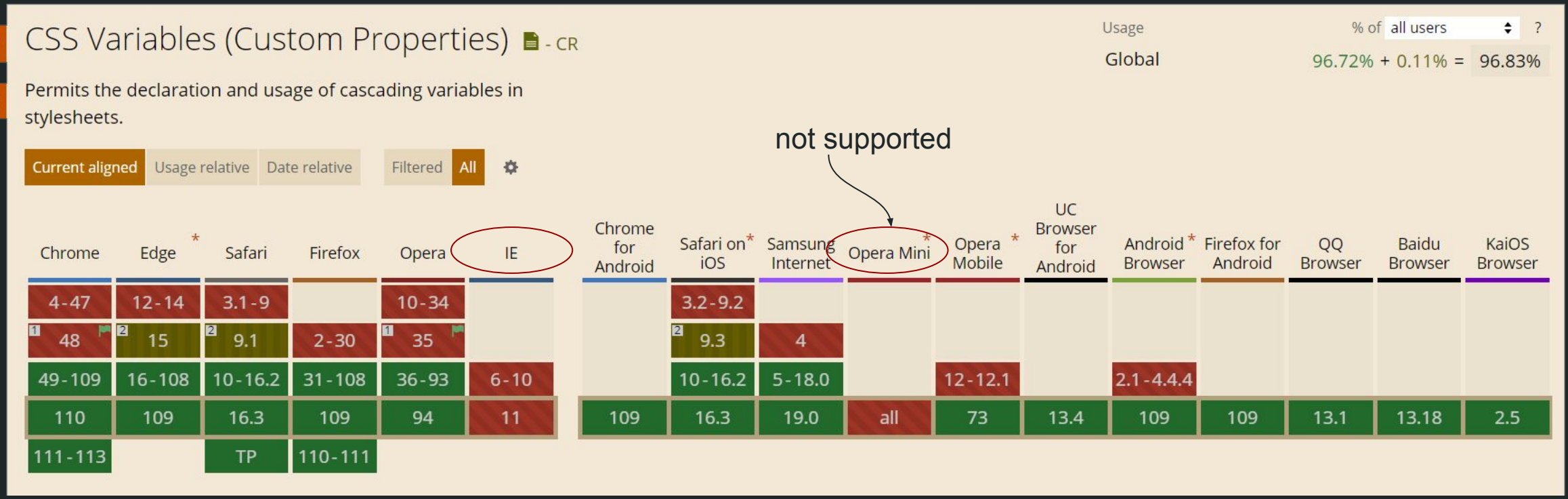
You can set the value of a custom property with another custom property:

```
html {
  --red: #a24e34;
  --green: #01f3e6;
  --yellow: #f0e765;
  --error: var(--red);
  --errorBorder: 1px dashed var(--red);
  --ok: var(--green);
  --warning: var(--yellow);
}
```

Declaring a custom property inside a property

Which browsers support custom properties?

source: caniuse.com



Custom properties are supported in modern browsers, including **Chrome, Firefox, Safari, and Edge**. However, they may not be supported in older browsers, so it's important to check the compatibility of your target audience before using them.

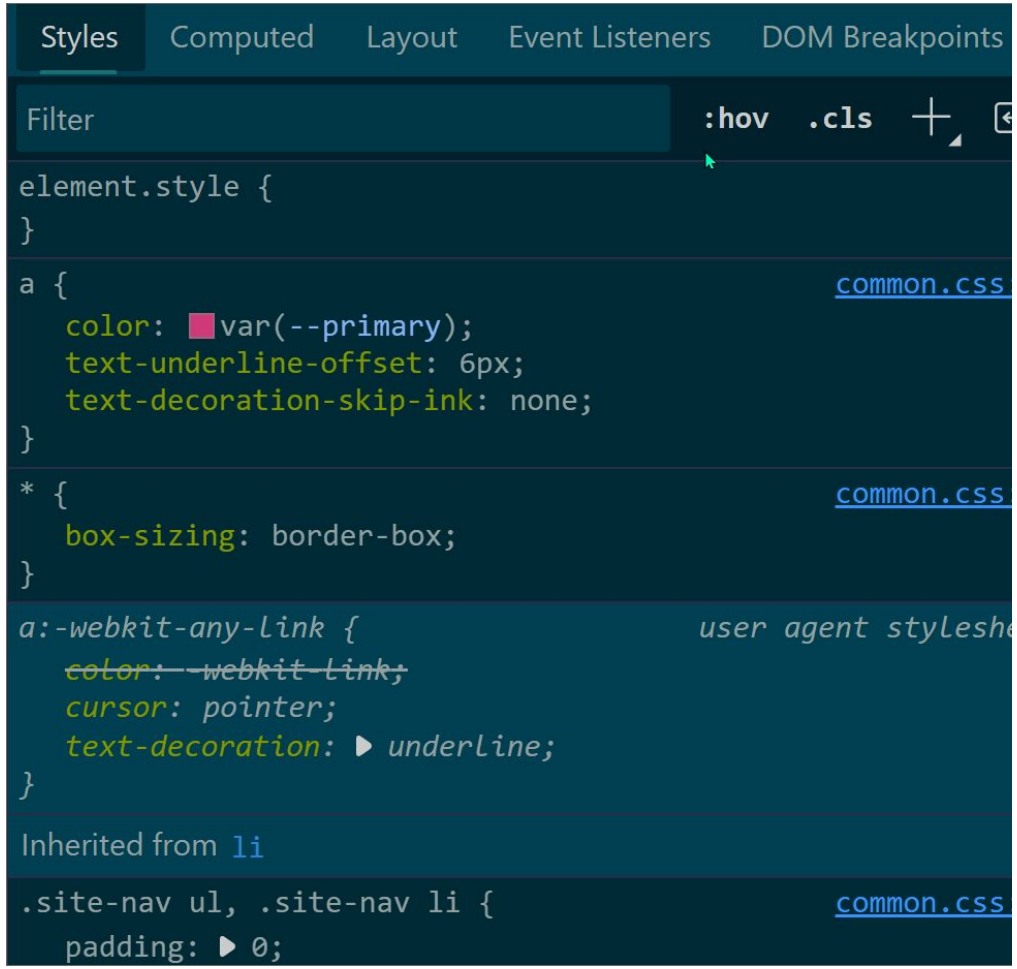
Use cases of **Custom properties**

Benefits of using custom property

1. **Scoping:** They help DRY up your CSS. That is “Don’t Repeat Yourself.” Custom properties can make code easier to maintain because **you can update one value and have it reflected in multiple places.**

2. **Inheritance:** Custom properties are inherited, so if you declare a custom property on a parent element, its value will be **inherited by its descendants.**

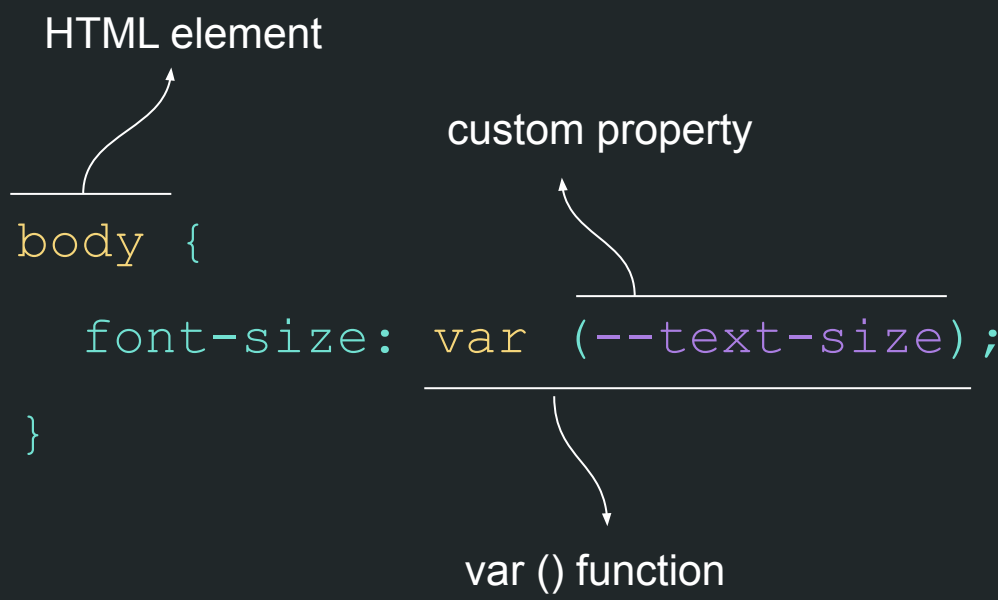
3. **Dynamic updates:** Custom properties can be updated dynamically using JavaScript, which makes it easy to create themes or switch between different styles. For example, colour themes of LIGHT & DARK MODE.



How to use a custom property in your CSS?

Custom properties can be used in your styles using the **var( )** function.

For example:



Use your defined custom property

Splitting up colours using custom properties

```
/* Button 1 */
```

Button

```
/* Button 2 */
```

Button

```
/* Change the lightness on hover */
```

Button

Button

```
/* Change the saturation on focus */
```

Button

Button

```
/* Changes opacity of button - DISABLED */
```

Button

Button

```
button {
  --h: 100;
  --s: 50%;
  --l: 50%;
  --a: 1;
```

defining custom property

```
background: hsl(var(--h) var(--s) var(--l) /
var(--a));
}
button:hover { /* Change the lightness on hover */
  --l: 75%;
}
button:focus { /* Change the saturation on focus */
  --s: 75%;
}
button[disabled] {  /* Make look disabled */
  --s: 0%;
  --a: 0.5;
}
```

Same values can be used for different elements

Shadows through custom properties

```
/* Button - HTML */
```

Button

```
/* Button STYLING - spread 5px */
```

Button

```
/* Button STYLING : HOVER - spread 10px */
```

Button

```
NOTE: /* It can work similarly on gradients, grids(eg. grid-gaps), etc. */
```

```
/* HTML */
<button> button </button>
```

```
/* CSS Styling */
button {
  --spread: 5px;
  box-shadow: 0 0 20px var(--spread) black;
}

button:hover {
  --spread: 10px;
}
```

only changing the required property

Does not require to type the whole property again

Use of custom property with `@media` queries

Media queries don’t change specificity, but they often come later (or lower) in the CSS file than where the original selector sets a value, which also means a **custom property will be overridden inside the media query.**

```
body {
  --bg-color: white;
  --text-color: black;

  background-color: var(--bg-color);
  color: var(--text-color);
}

/*DARK MODE*/
@media screen and (prefers-color-scheme:
dark) {
  body {
    --bg-color: black;
    --text-color: white;
  }
}
```

Can be used for accessibility purposes

Custom property with `!important`

The `!important` property works or not depends on the following factors:

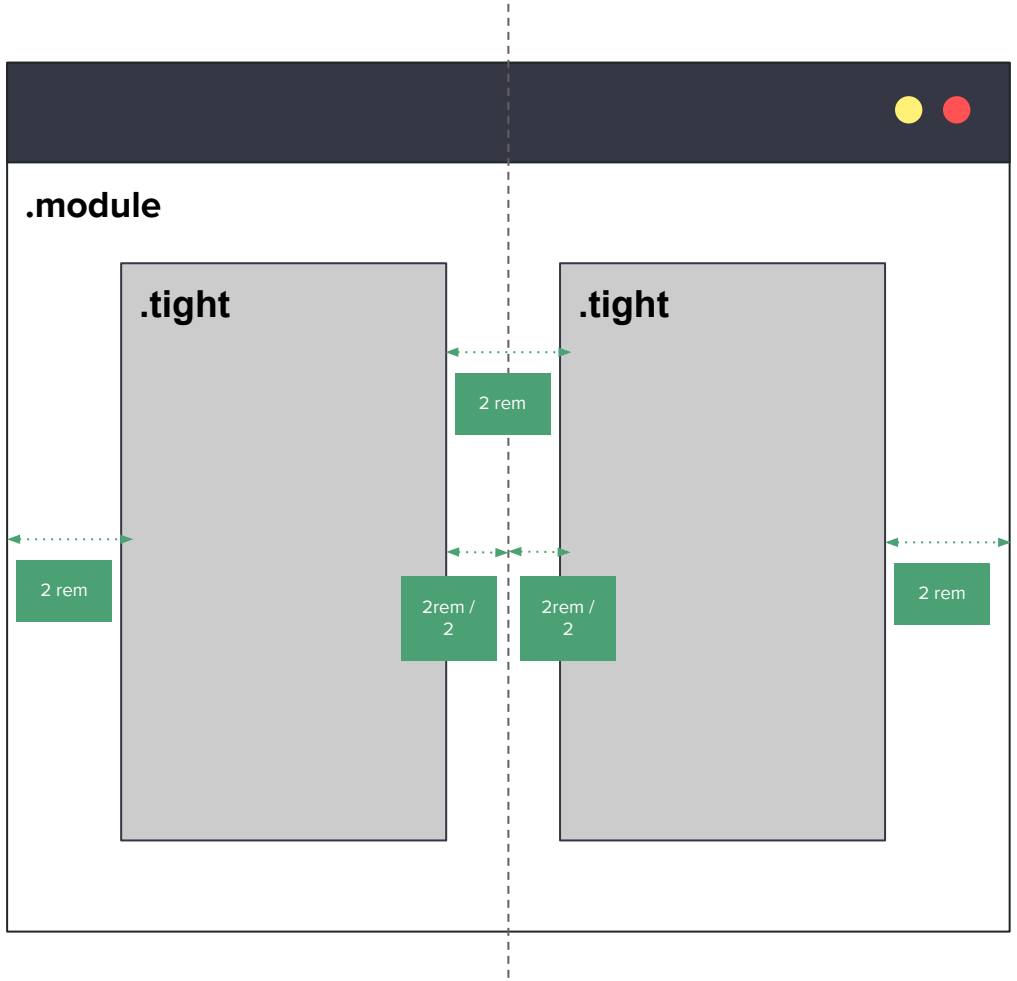
1. Ultimately, `!important` is stripped from the value of the custom property.
2. But it is used when determining which value wins when it is set in multiple places.

```
div {
  --color: red !important;
}

#id {
  --color: yellow;
}

/*If both of those selectors apply to an
element, you might think the #id value would
win because of the higher specificity, but
really red will win because of the
!important, but then ultimately be applied
without the !important. */
```

the `!important` function will override the custom property

Can be used for accessibility purposes

Using `calc()` and custom properties

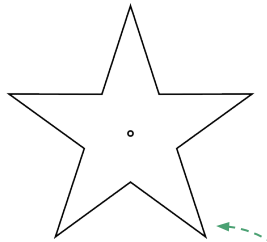



```
main {
  --spacing: 2rem;
}

.module {
  padding: var(--spacing);
}

.module.tight {
  /* divide the amount of spacing in half */
  padding: calc(var(--spacing) / 2));
}
```

custom property can be used in any unit or value

Use of custom property as a unit

This code defines a custom property, `--r`, and uses it to rotate an HTML element with the class `.star`.



Using `@property` with custom properties

In CSS, the `@property` feature allows authors to use JavaScript to create custom properties that can be used in CSS. It's sort of like **you're creating an actual CSS property** and have the ability to define what it's called, it's syntax, how it interacts with the cascade, and its initial value.

```
@property --r {
  inherits: false;
  initial-value: 0deg;
  syntax: "<angle>";
}

.star {
  --r: 0deg;
  transform: rotate(var(--r));
  animation: spin 1s linear infinite;
}

@keyframes spin {
  100% {
    --r: 360deg;
  }
}
```

property value will not be inherited by child elements.

Specifies that the value of the property must be an angle.

HTML element

rotates the element by the value of the `--r` property.

infinite animation loop

sets the value of the `--r` property to `360deg`

In 1 second, `--r` will rotate from 0deg to 360deg

Custom properties with JavaScript

Using custom properties with JavaScript involves accessing the custom properties using the `style` property of an element and setting new values for the custom properties using the `setProperty()` method. Here is the general syntax for accessing and updating custom properties in JavaScript:

```
const element =
document.querySelector("selector");
```

HTML element you want to access and update the custom properties for.

```
// Get the value of a custom property
const propertyValue =
element.style.getPropertyValue("--custom-property-name");
```

get the current value of the custom property

```
// Set the value of a custom property
element.style.setProperty("--custom-property-name", "new-value");
```

set a new value for the custom property

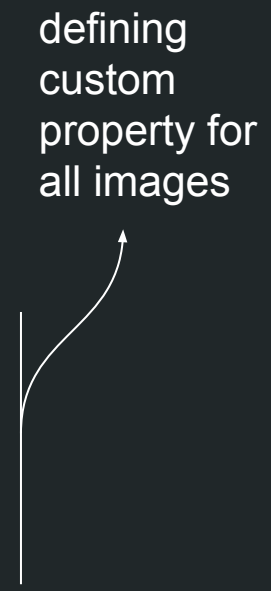
Custom properties with JavaScript

Using custom properties with JavaScript involves accessing the custom properties using the `style` property of an element and setting new values for the custom properties using the `setProperty()` method. Here is the general syntax for accessing and updating custom properties in JavaScript:

```
// get variable from inline style
element.style.getPropertyValue("--my-var");


// get variable from wherever
getComputedStyle(element).getPropertyValue("--my-var");


// set variable on inline style
element.style.setProperty("--my-var", jsVar + 4);
```

Comma separated values (like bgs)

```
/* Lots of backgrounds! */
background-image:
  url(./img/angles-top-left.svg),
  url(./img/angles-top-right.svg),
  url(./img/angles-bottom-right.svg),
  url(./img/angles-bottom-left.svg),

body {
  --bg1: url(./img/angles-top-left.svg);
  --bg2: url(./img/angles-top-right.svg);
  --bg3: url(./img/angles-bottom-right.svg);
  --bg4: url(./img/angles-bottom-left.svg);

background-image: var(--bg1), var(--bg2), var(--bg3);
}

@media (min-width: 1500px) {
  body {
    background-image: var(--bg1), var(--bg2),
var(--bg3), var(--bg4);
  }
}
```



Things to keep in mind while using **Custom properties**

Create fallback values for your custom property

Fallback values: Fallback values are required if your CSS properties don’t work.

You can provide a fallback value to the **var( ) function** in case the custom property is not defined, or the browser doesn’t support custom properties.

```
body {

  background-color: var
(--my-color, #fff);

}
```

custom property

fallback value for custom property

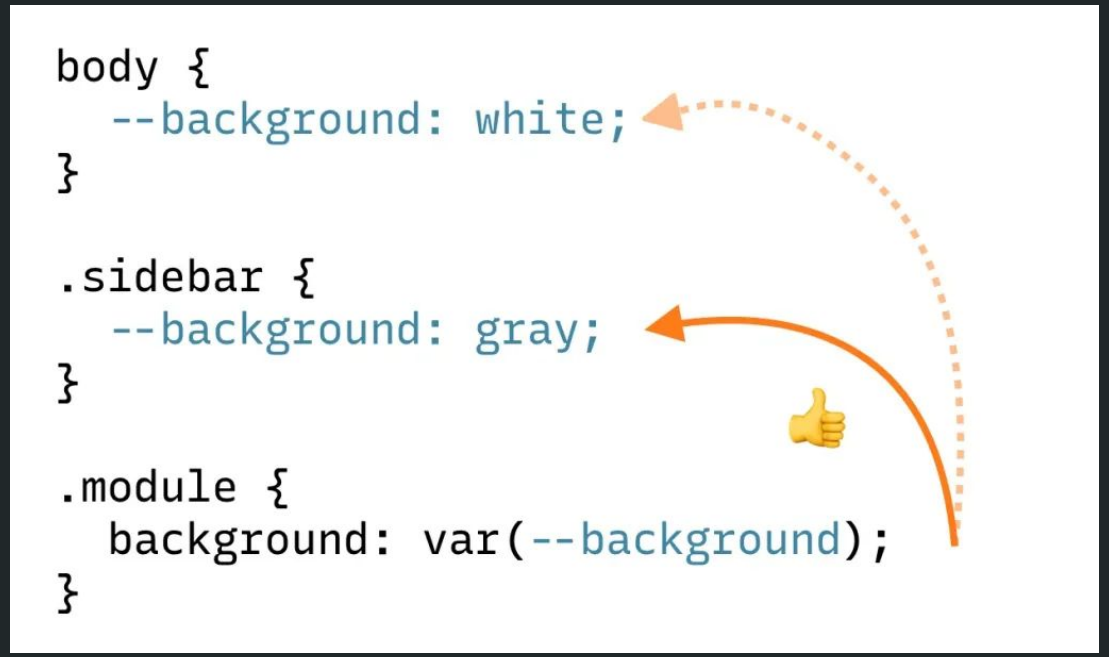
Creating fallback values of custom property

Effects of cascading on custom properties

Custom properties are inherited, so if you declare a custom property on a parent element, its value will be **inherited by its descendants.**

Make sure which parent element you are applying the property to. :)

source: CSS Tricks



Thank you!