CSS Selectors Guide

Master the fundamental building blocks of CSS styling. Understanding selectors is essential for targeting HTML elements precisely and creating maintainable stylesheets. This guide covers the core selector types every web developer needs to know.

Universal Selector

Target every element on the page with the asterisk (*) selector. Use sparingly for global resets or base styling that applies universally across all elements.

Element Selector

Select all instances of a specific HTML tag like p, div, or h1. This is the most straightforward way to apply consistent styling to element types.

ID Selector

Target a unique element using the hash symbol (#) followed by the ID name. IDs should be unique per page and carry high specificity weight.

Class Selector

Apply styles to multiple elements using the dot (.) notation. Classes are reusable and form the backbone of modular, maintainable CSS architecture.

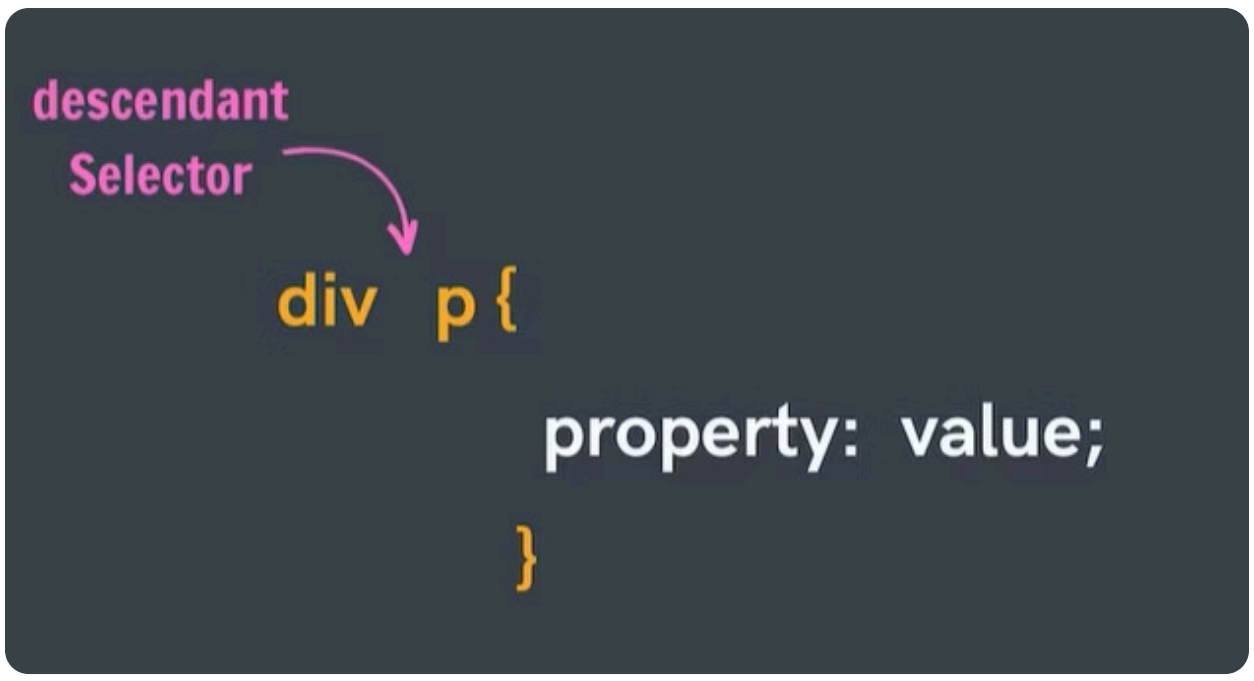
ADVANCED SELECTORS

CSS Combinators & Advanced Targeting

Move beyond basic selectors to target elements based on their relationships and attributes. These powerful selectors enable precise styling without cluttering your HTML with excessive classes.

Descendant Selector

Target elements nested within other elements, regardless of how deep. Use a space between selectors to match any descendant element in the hierarchy.

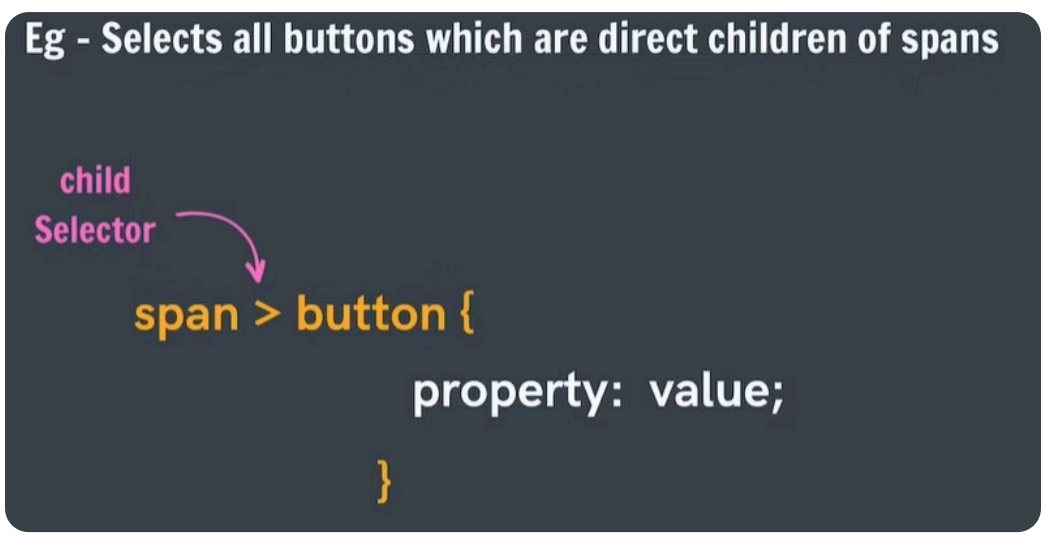


Adjacent Sibling Combinator

Style an element that immediately follows another specific element at the same level. Perfect for targeting heading3 elements that come directly after any paragraph using the plus (+) symbol.

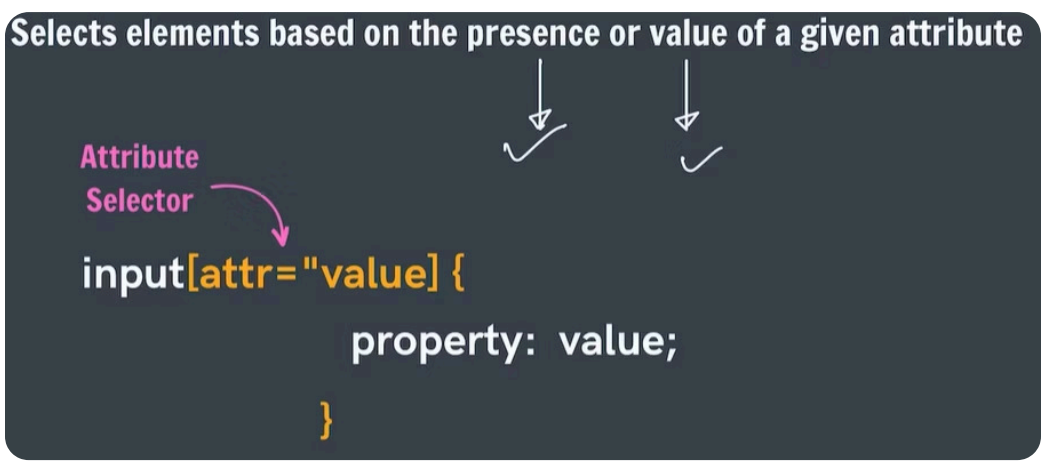
Child Combinator

Select only direct children using the greater-than symbol (>). This combinator provides more specificity than descendant selectors by limiting matches to immediate children only.



Attribute Selector

Target elements based on their attributes and values using square brackets. This powerful technique allows you to style elements by their href, data attributes, or any other HTML attribute.



DYNAMIC STYLING

Pseudo-Classes & Pseudo-Elements

Enhance user experience with dynamic styling that responds to element states and targets specific portions of content. These selectors unlock sophisticated interactions without JavaScript.

Pseudo-Classes

Apply different styling based on the particular state of an element. Pseudo-classes let you respond to user interactions like hover, focus, or visited states, creating dynamic and responsive designs.

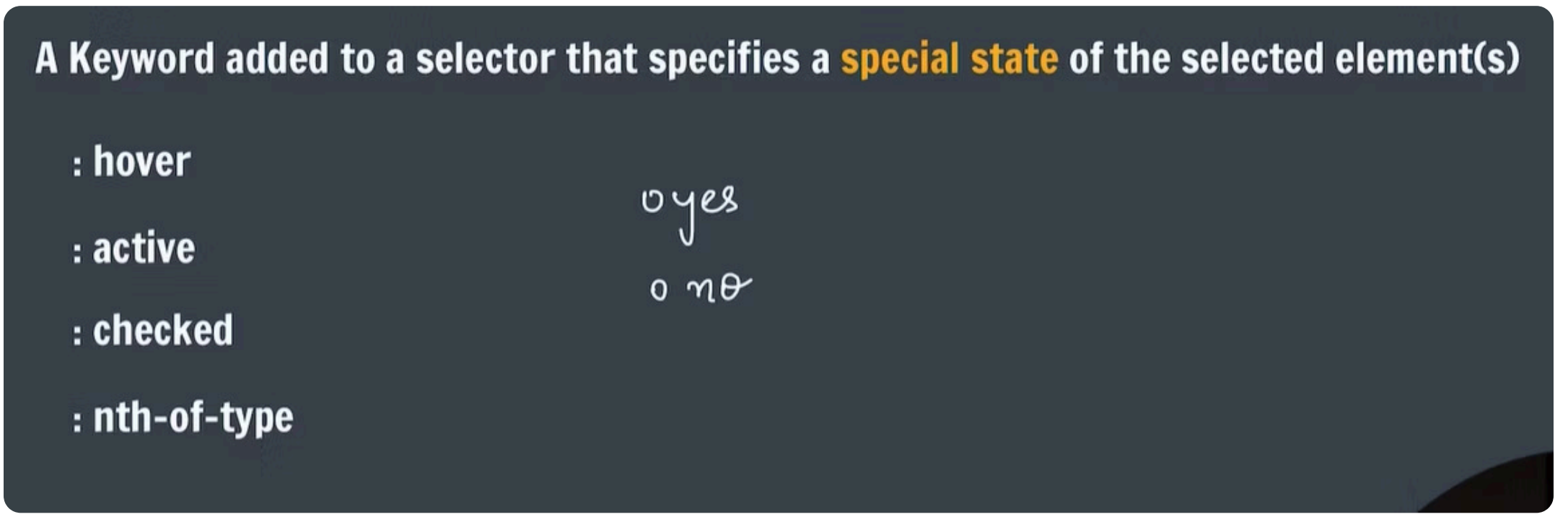
Common examples include `:hover` for mouse-over effects, `:focus` for keyboard navigation, `:first-child` for targeting the first element, and `:nth-child()` for pattern-based selection.

Pseudo-Elements

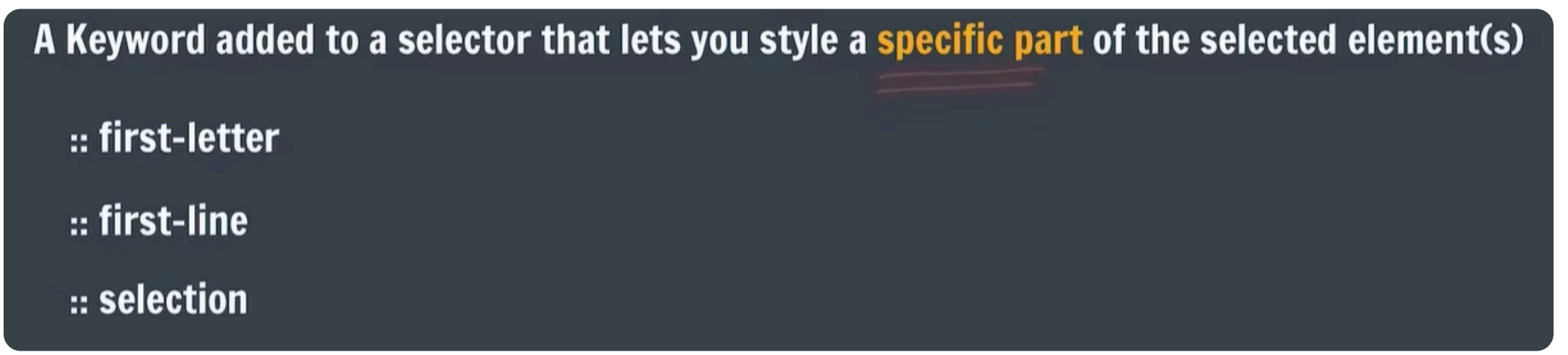
Apply styling to a specific part of a selected element rather than the entire element. Pseudo-elements create virtual elements that can be styled independently, enabling decorative effects and enhanced typography.

Use `::before` and `::after` to insert generated content, `::first-letter` for drop caps, and `::first-line` for special paragraph openings. Note the double-colon syntax that distinguishes them from pseudo-classes.

Pseudo-Class Example



Pseudo-Element Example



CORE CONCEPT

Understanding the CSS Cascade

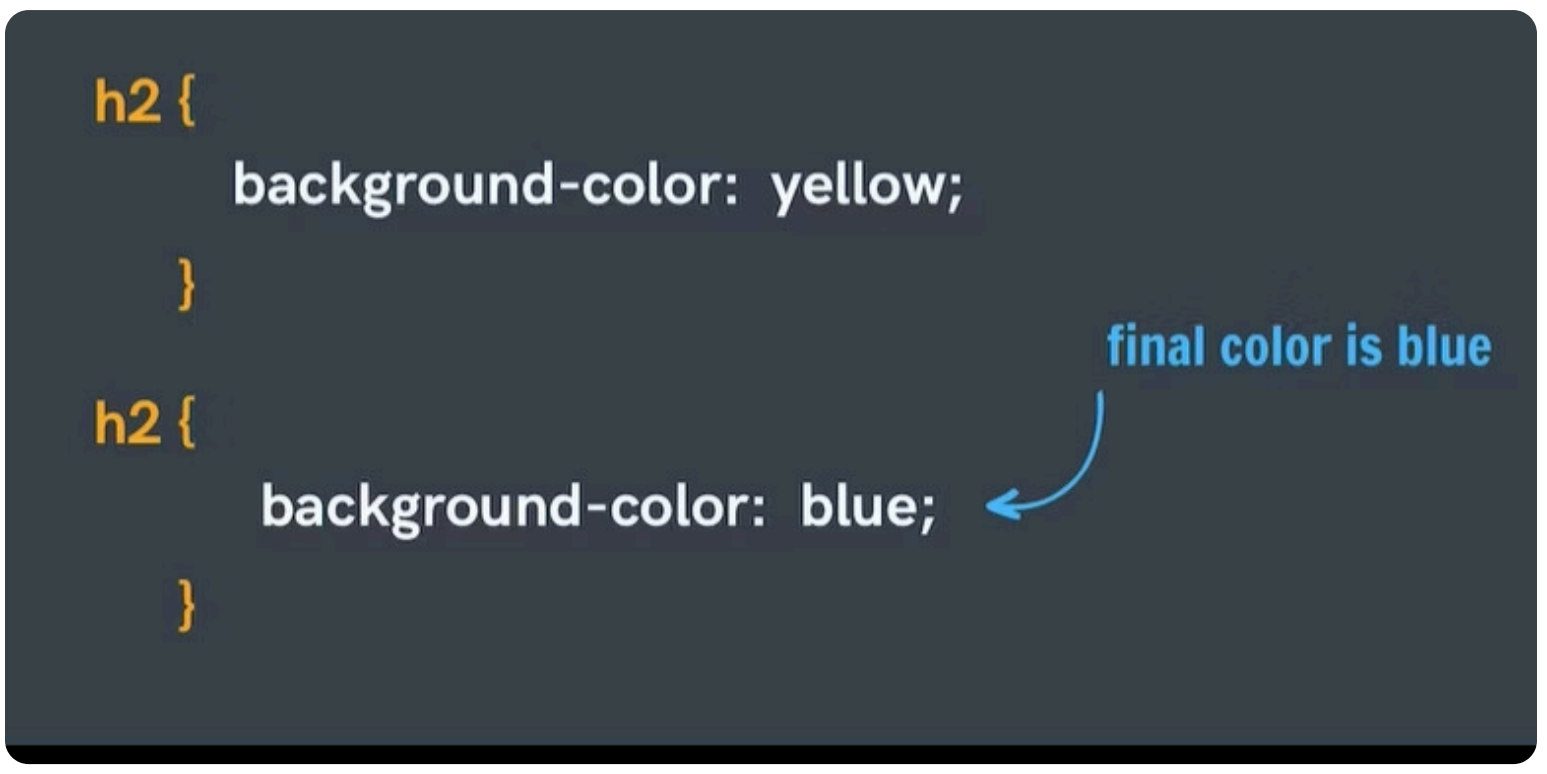
CSS stands for **Cascading Style Sheets**, but what does "cascading" actually mean? The cascade is the algorithm that determines which CSS declarations apply when multiple rules target the same element. Mastering this concept is crucial for writing predictable, maintainable stylesheets.

What is the Cascade?

The CSS cascade algorithm's job is to select CSS declarations in order to determine the correct values for CSS properties. It considers source order, specificity, and importance to resolve conflicts.

Source Order: When Selectors Match

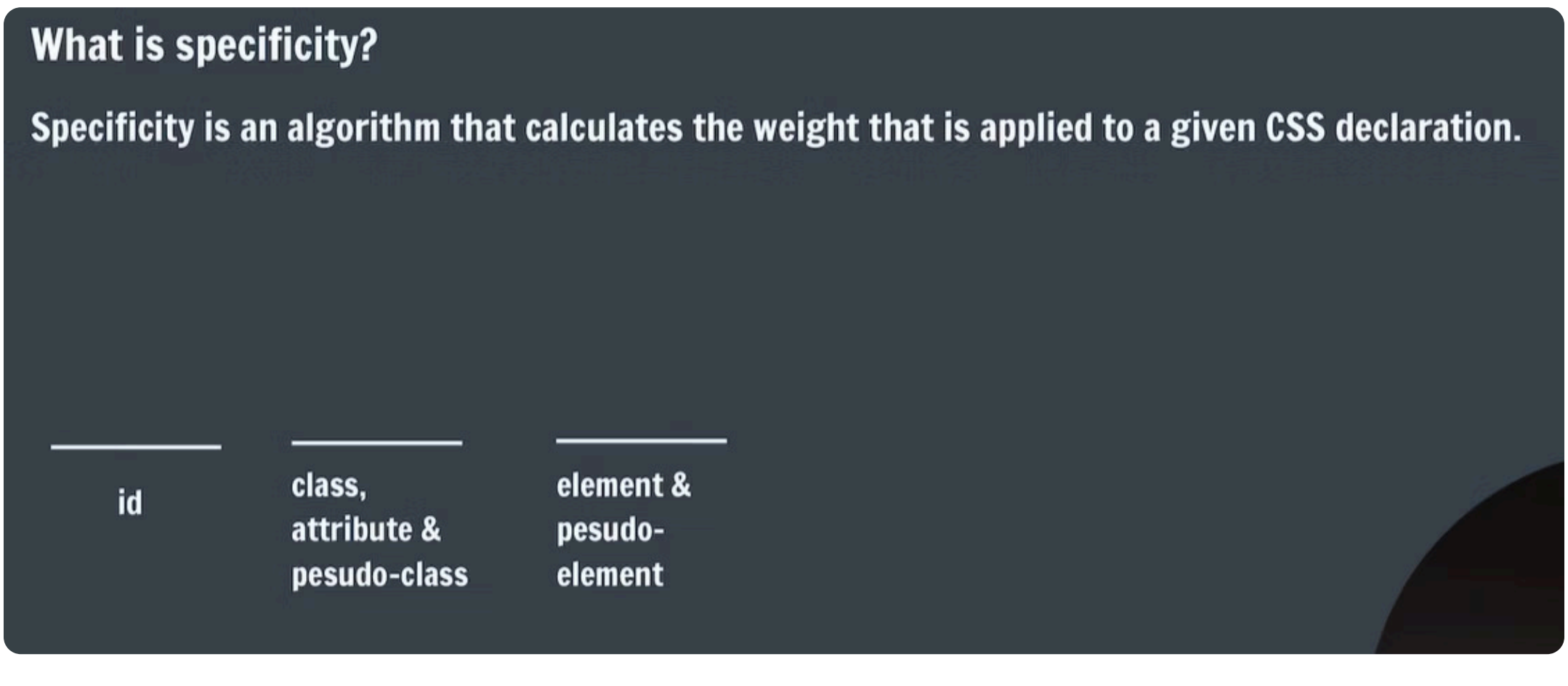
When multiple rules use identical selectors, the declaration that appears last in your stylesheet wins. The browser applies styles in order, with later rules overwriting earlier ones.



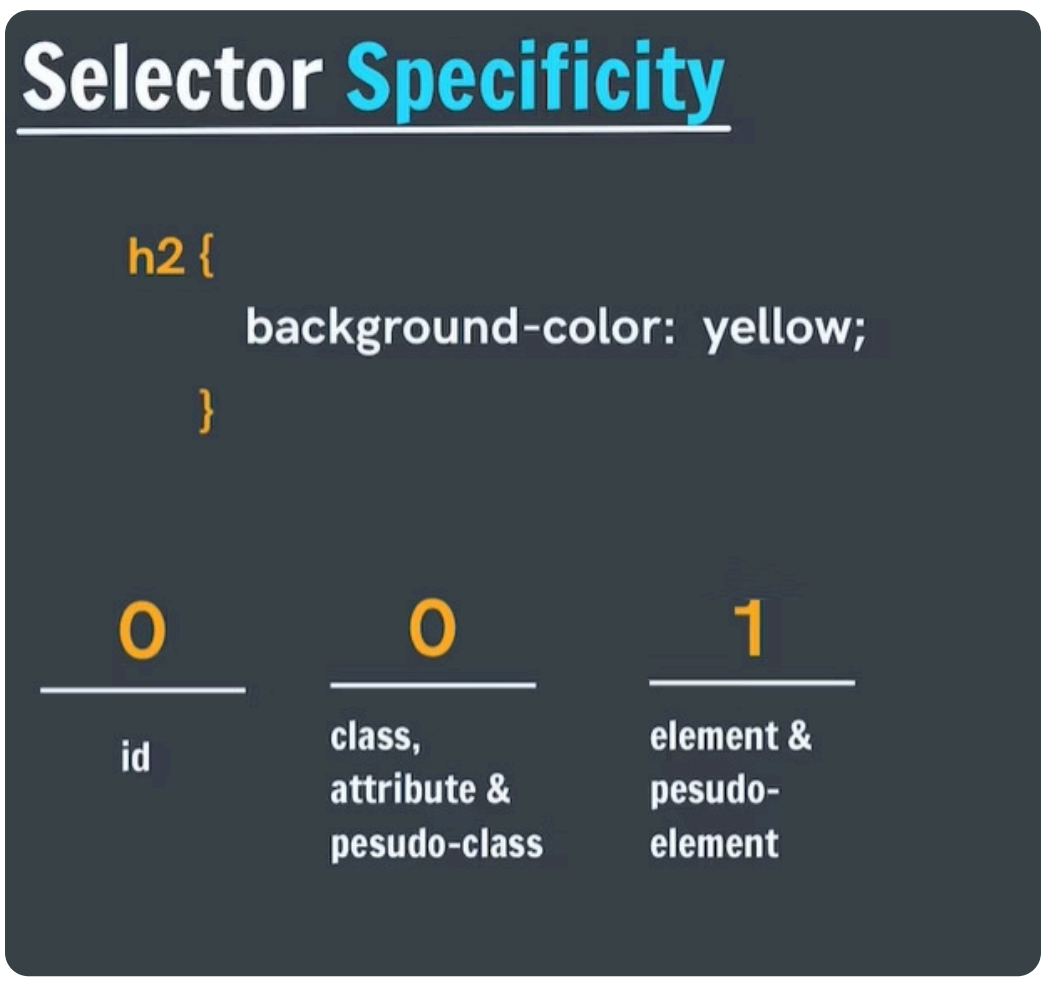
In the example above, if the selector is the same, the selector which comes below will be applied.

Specificity: When Selectors Differ

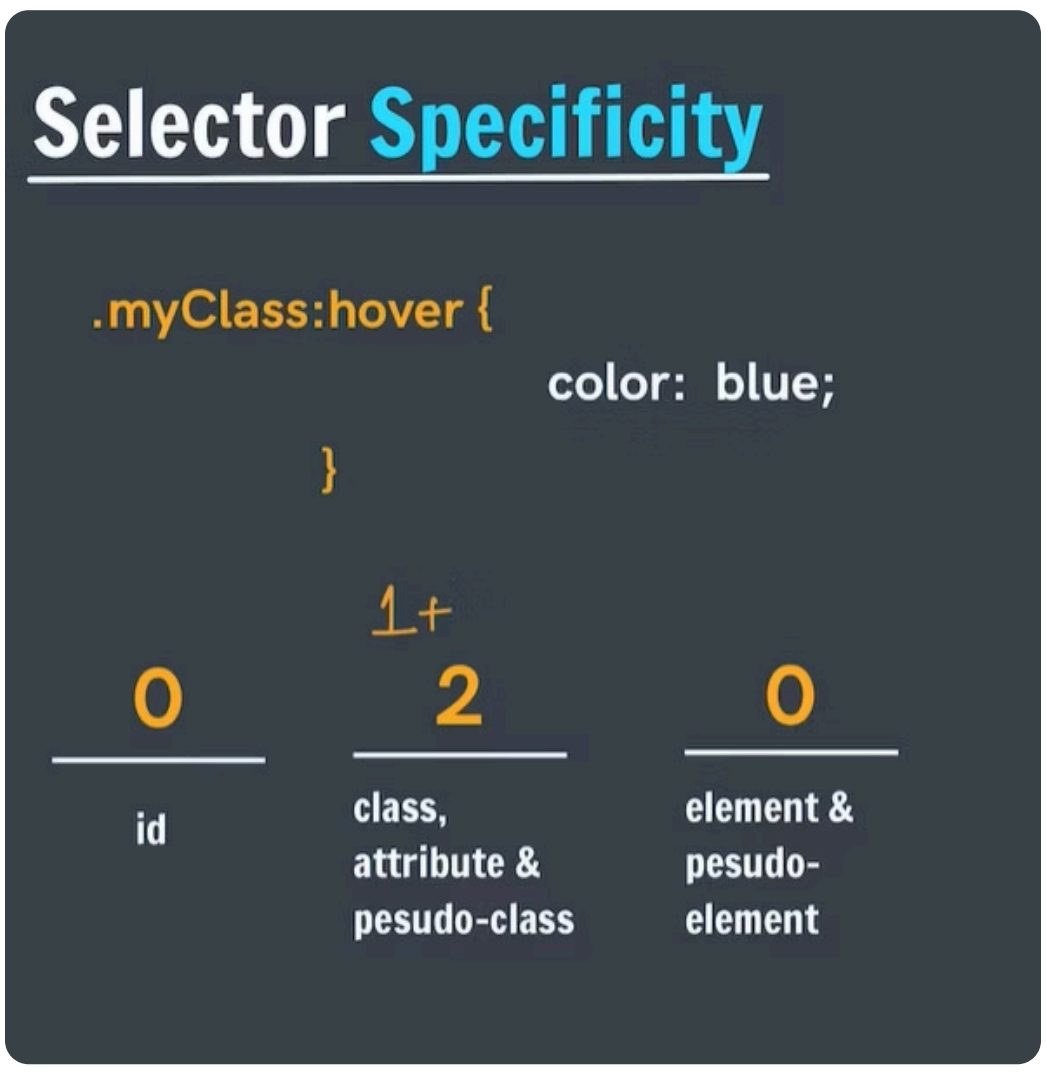
If selectors are different, we use **specificity** to decide which styling wins. Specificity is calculated based on the types of selectors used, creating a hierarchy that determines priority.



Specificity Example 1



Specificity Example 2



ID beats Class beats Element

The hierarchy is clear: ID selectors have highest specificity, followed by class selectors, then element selectors

More Selectors = More Specificity

Combining selectors increases specificity. A selector with two classes beats one class

Equal Specificity? Use Cascade

When specificity scores match, the cascading algorithm applies and source order determines the winner

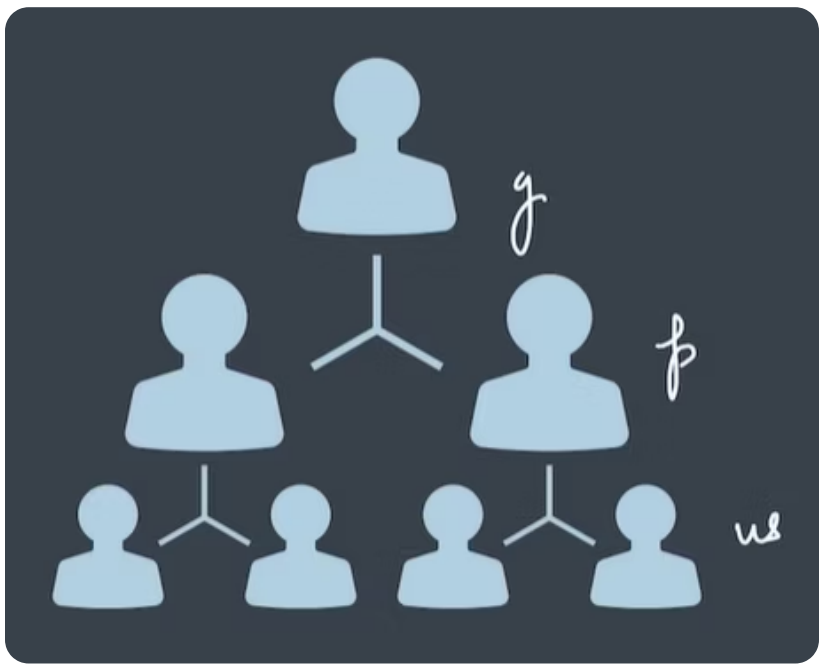
Inline Styles Win

Inline styles have the highest specificity and will override external or internal stylesheets

FINAL CONCEPT

CSS Inheritance Explained

Not all CSS properties are created equal. Some properties naturally flow from parent to child elements through inheritance, while others require explicit declaration. Understanding inheritance helps you write more efficient stylesheets with less repetition.



How Inheritance Works

Certain CSS properties automatically cascade down from parent elements to their children. This inheritance makes styling more efficient—set a font on the body element, and all text elements inherit it unless overridden.

`Typography properties` like font-family, font-size, color, and line-height typically inherit. `Layout properties` like margin, padding, and border do not inherit by default.

⚠️ Important Exception: Form Elements

Input boxes and buttons are special cases that **won't inherit most properties** from higher-level elements. You must explicitly style form controls if you want them to match your design system. This behavior exists for historical reasons and browser consistency.

Inheritable Properties

- Typography: font-family, font-size, font-weight
- Text: color, text-align, line-height
- Visibility: visibility, cursor

Non-Inheritable Properties

- Box model: margin, padding, border, width, height
- Positioning: position, top, left, z-index
- Background: background-color, background-image

Forcing Inheritance

- Use `inherit` keyword to force inheritance
- Use `initial` to reset to default value
- Use `unset` to inherit if possible, otherwise initial