Engineer

CSS

the language that says how a page looks

What it is

CSS, short for Cascading Style Sheets and still often called CSS3, is the rule language a browser applies to a document to decide how each element is drawn: its colours, size, spacing, position, typeface and movement, whether the document lands on a phone, a wide monitor or a sheet of paper.

A stylesheet is a list of rules. Each rule pairs a selector, which picks out elements, with declarations, each one a property and a value:

```
.note {
    color: #3a3a3a;
    margin-block: 1.5rem;
}
```

The first word of the name is the part that shapes daily work. Several rules routinely claim the same property of the same element, and the cascade decides which one wins. Its main steps, in order, are origin and importance (the browser's defaults, the reader's own settings, the site's stylesheet, and whether a declaration is marked `!important`), then cascade layer, then specificity, and finally order of appearance, where a later rule beats an earlier one of equal weight. Specificity is counted per selector as three numbers: IDs, then classes, attributes and pseudo-classes, then element names. The universal selector `*` adds nothing to any of them. Inheritance works alongside the cascade: some values, such as colour and font, pass down to an element's children, while others, such as margins and borders, stay where they were set.

CSS has no single version number. After CSS 2.1 the working group went modular: Level 3 builds on 2.1 one module at a time, and from there each module is levelled on its own, so Selectors is drafted at Level 4 while modules

with no older equivalent start at Level 1. “CSS3” is the popular name for that modular era, and the yearly snapshot that defines current CSS states plainly that there is no CSS Level 4.

The rule with the most practical weight is how the language treats errors. A browser that meets a property it does not know, or a value that is invalid for its property, must ignore that one declaration and apply the rest of the rule. The rule exists so that new properties and values can be added without breaking older browsers, and it is what lets a site use a feature only some browsers support. Its price is silence. The page renders, slightly wrong, and the only trace is a struck-through line in developer tools that someone has to go looking for.

Most of what is compared with CSS turns out to be a way of writing it. Sass and Less are preprocessors that compile to CSS. Utility frameworks such as Tailwind generate it from class names in the markup. CSS-in-JS libraries emit it from JavaScript components, and a `style` attribute puts declarations straight onto one element. Each changes how CSS is authored; each ends as declarations the cascade sorts in exactly the same way.

Use the concept in a decision

Put “What is CSS?” to work in a real decision. We can help relate the concept to the system you are building or buying. Interfaces that load quickly, work on a phone and behave the same in every browser your customers happen to use.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Directed full-stack GIS map development — full-stack mapping, from database to browser

Directed full-stack GIS map development Get in touch →

Why it matters

The decision CSS forces on a team is where a rule is allowed to fail silently. The language drops what it cannot apply and lets a stronger rule win without comment, so every stylesheet of any age holds declarations that do nothing. What a team controls is whether anyone finds out, and when.

Accessibility depends on it more than most people expect. Contrast, a visible focus ring, honouring a request for reduced motion, a usable forced-colours mode, text that survives being enlarged: each one is a handful of declarations, and each can be switched off by a rule elsewhere that happens to be more specific. The page looks fine to everyone who did not need the feature.

Maintenance cost depends on it too. A stylesheet whose behaviour nobody can predict only grows, because deleting a rule might break a page nobody checked. The usual end state is a file where most of the weight is overrides of earlier overrides, and every change is made by trying values until the screen looks right.

Verification depends on it most directly. A browser that accepts what it can and ignores the rest cannot serve as the test. A linter reading the source catches typos and invalid values before they ship. It cannot see what a minifier or any other build step does to the file afterwards, and it cannot see which of two valid rules wins on a given page. Checks against the served file, and visual comparison of rendered pages, cover what the linter cannot.

Organising the cascade early, with flat specificity, named layers and design tokens held in custom properties, costs a little discipline per change. Reorganising a stylesheet that grew without that discipline usually costs a rewrite.

What it looks like in practice

The site's own credits page describes one CSS bundle, concatenated and fingerprinted at build time, with the hand-drawn arrows and margin notes drawn in CSS alone, and Stylelint holding the source to its rules in the gate. The same page links the W3C CSS Validator and warns the reader what it will say. As of August 2026 its profile stops short of `system-ui`, `@property`, `clamp()` in a `font-size` and scroll-driven animations, so against /terms/ it reported 176 complaints, 175 of them about features the browsers ship and the site uses. The remaining one was real. A resting `rotate: 0deg` lost its unit to the minifier, `rotate: 0` is not an angle, and the browser dropped the whole declaration, so an animation shipped without the rotation it was written with. The values specification allows a bare zero for an angle only in a few legacy places. Stylelint never saw the problem, because it reads the source and the minifier rewrites it later; a check of the company's own now reads what was actually served.

Specificity was the whole story in a piece of accessibility work. The site honoured the reduce-motion preference with one universal rule that turned every transition off, and under an emulated preference 11 selectors were still animating. A universal selector scores zero, so any rule carrying a class beat it. The fix was structural rather than a stronger selector: everything that moves now lives in one late stylesheet part, and a linter fails a transitioned transform anywhere else. Colour fades pass, since they are not motion, and motion must be

written as a transition rather than a keyframe animation, because a transition is the one that cancels cleanly. The lesson recorded with it is short: a rule that loses on specificity fails silently.

Colour is where stylesheets drift first, and the site's palette was closed at 28 colours, each documented with the contrast ratio it measures on the surface where it appears, and 11 brand values declared once as tokens. Two colours closer than a perceptual distance of 0.02 count as one colour spelled twice, and the linter found two near-duplicate spellings as soon as it ran. Transparency is written with relative colour syntax rather than the mixing function, for a reason that sums up the whole page: the guard cannot see through a mix.

Where it goes wrong

A declaration vanishes and the page still renders

One wrong character makes a declaration invalid, and the browser skips it without stopping:

```
.card {
    padding: 16; /* a length needs a unit: ignored */
    colour: #222; /* not a property name: ignored */
    margin-block: 1rem; /* applied */
}
```

The card renders with a margin and without its padding or its colour, and the browser raises no alarm. Lint the source in the build and fail on its errors. Where a minifier, preprocessor or bundler rewrites the CSS after the linter has run, check the output as well, because that file is the one the browser reads.

A rule loses to a stronger one

A component's rule meets a more specific selector from somewhere else, or an equal one loaded later, and loses. The usual reflex is a longer selector or an `!important`, which wins today and raises the price of the next change. Keep specificity flat, with classes rather than IDs and little nesting, and use cascade layers to put whole groups such as reset, base, components and utilities in an explicit order that specificity cannot overturn. For `!important` declarations the layer order runs the other way, with the earliest layer winning, which is worth knowing before reaching for it.

The stylesheet only ever grows

New work appends rules, because proving an old one unused takes longer than writing another. Values get retyped as literals beside the tokens that define them, and near-identical colours multiply until nobody can tell which is meant. Hold shared values in custom properties, measure which rules a set of pages actually uses, and delete as part of the work, with a size budget in the build to make growth visible.

The layout holds at one size only

A design checked at one width with short English text breaks under longer translations, enlarged text and narrow phones. Fixed heights clip their content, and fixed widths force sideways scrolling. Accessibility guidelines ask for text that can be enlarged to 200% and for content that reflows at a width of 320 CSS pixels. Size boxes from their content, use `min()`, `max()` and `clamp()` for fluid values, and test with long strings and zoom before calling a layout done.

Further reading

- CSS Snapshot 2026 — which modules make up CSS today, and why the language has levels instead of versions.
- CSS Cascading and Inheritance Level 5 — the full sort order, cascade layers and inheritance, step by step.
- Selectors Level 4 — every selector, and how its specificity is calculated.
- Syntax and basic data types — the CSS 2.1 rules for ignoring unknown properties and illegal values, with examples.
- CSS Values and Units Module Level 4 — lengths, angles, `clamp()` and where a bare zero is still allowed.



Glossary

Choose for your actual constraints

Describe the workflow, the existing interfaces and the users who need the change. We can define a testable delivery and a handover that lets your team maintain it.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is CSS?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Printed from https://engineer.company/glossary/css/

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.