Modern CSS

Version Baseline 2026 · verified 2026-11-02

The 2026 CSS primitives — layers, nesting, container queries, :has(), popover, anchors, view transitions — each with its Baseline status.

Contents

Mental model 2

Cascade layers 2

Nesting and custom properties 3

Logical properties and sizing 4

Flexbox and grid 5

Container queries 6

:has() and modern selectors 7

Popover, dialog and anchors 8

View transitions 10

Color functions 11

Further reading 12

Mental model

Modern CSS is component-oriented: a component styles itself from the space its container gives it and from the state of its own content, not from the viewport width and a pile of modifier classes. Layers decide which stylesheet wins before specificity is consulted, so specificity wars stop being a design tool. Check Baseline before shipping: a feature in one engine is a progressive enhancement, not a primitive.

Cascade layers

Syntax	Does	Baseline
`@layer a, b;`	Declares order; later wins	Widely
`@layer b { … }`	Adds rules to layer `b`	Widely
`@import url(x.css) layer(a)`	Imports into a layer	Widely
`:where(.x)`	Zero specificity	Widely
`@scope (.card) to (.slot)`	Scoped, bounded rules	Newly 2026

Status column: Baseline as of 2 Nov 2026. *Widely* means every core browser has shipped it for 30 months or more; *Newly (year)* means the last engine shipped it that year; *Limited* means at least one engine lacks it. Between layers, the later-declared layer wins regardless of specificity, so a one-line `@layer` statement at the top fixes the order for the whole app.

```
@layer reset, base, components, utilities;

@layer components {
  #nav .link { color: navy; }
}
@layer utilities {
  .muted { color: gray; } /* wins: later layer */
}
```

> **Gotcha:** Unlayered styles beat every layer. A third-party sheet you import without `layer()` overrides your whole layered system; for `!important` the order reverses and earlier layers win.

Nesting and custom properties

Syntax	Does	Baseline
`.a { .b { } }`	Nested descendant	Widely
`.a { &:hover { } }`	`&` is the parent	Widely
`var(--x, 1rem)`	Read, with fallback	Widely
`@property --x { … }`	Typed, animatable	Newly 2024

Nested rules desugar to `:is()`, so a nested selector takes the specificity of the most specific selector in the parent list. Custom properties inherit and resolve at computed-value time, which is what makes them a theming API: set one on a container and every descendant reads it.

```
.card {
  --pad: 1rem;
  padding: var(--pad);
  & .title { font-weight: 600; }
  &:hover { --pad: 1.25rem; }
  @media (width > 40rem) { --pad: 2rem; }
}
@property --angle {
  syntax: "<angle>";
  inherits: false;
  initial-value: 0deg;
}
```

> **Gotcha:** Sass habits break here: `&__title` and `&-title` do not concatenate in native CSS, so the whole rule is dropped. Write the full class name.

Logical properties and sizing

Physical	Logical	Baseline
`margin-left/right`	`margin-inline`	Widely
`padding-top/bottom`	`padding-block`	Widely
`width` / `height`	`inline-size` / `block-size`	Widely
`100vh`	`100dvh`, `100svh`	Widely
`min()`, `max()`, `clamp()`	Fluid values	Widely

Logical properties follow the writing mode, so one stylesheet serves left-to-right and right-to-left text. `inline` is the text direction, `block` is the stacking direction.

```
.prose {
  max-inline-size: 65ch;
  margin-inline: auto;
  padding-block: 2rem;
  font-size: clamp(1rem, 0.9rem + 0.5vw, 1.25rem);
}
.hero { min-block-size: 100dvh; }
```

> **Gotcha:** `100vh` on mobile is the height with the browser toolbar retracted, so content hides behind it. Use `100dvh` (tracks the toolbar) or `100svh` (smallest, stable).

Flexbox and grid

Syntax	Does	Baseline
`display: flex; gap: 1rem`	One axis	Widely
`flex: 1`	Grow, basis 0	Widely
`repeat(auto-fill, minmax(…))`	Responsive tracks	Widely
`grid-template-areas`	Named regions	Widely
`grid-template-rows: subgrid`	Inherit parent tracks	Widely

Use flexbox when content decides size along one axis; use grid when the layout decides both. `subgrid` lets a card's rows align with its siblings' rows, which used to need fixed heights.

```
.cards {
  display: grid;
  grid-template-columns:
    repeat(auto-fill, minmax(16rem, 1fr));
  gap: 1rem;
}
.card {
  display: grid;
  grid-row: span 3;
  grid-template-rows: subgrid;
}
```

> **Gotcha:** Flex and grid items default to `min-width: auto`, so a long word or `<pre>` refuses to shrink and overflows. Set `min-width: 0` (or `minmax(0, 1fr)` in grid) on the item.

Container queries

Syntax	Does	Baseline
`container-type: inline-size`	Makes a container	Widely
`container-name: card`	Names it	Widely
`@container card (width > 30rem)`	Size query	Widely

Syntax	Does	Baseline
`cqi`, `cqb` units	% of container	Widely
`@container style(--x: y)`	Style query	Newly 2026

A component queries its nearest container, so the same card works in a sidebar and in a full-width grid without knowing which it is in.

```
.slot {
  container: card / inline-size;
}
.card { display: grid; gap: 1rem; }
@container card (width > 30rem) {
  .card { grid-template-columns: 1fr 2fr; }
}
.card h2 { font-size: clamp(1rem, 4cqi, 2rem); }
```

> **Gotcha:** An element cannot query itself, and an `inline-size` container ignores its children when sizing. A shrink-to-fit container (a flex item without `flex: 1`, a float) collapses to zero width. Put `container-type` on a wrapper that has a definite width.

:has() and modern selectors

Selector	Matches	Baseline
`.card:has(> img)`	Card with a direct img	Widely
`form:has(:invalid)`	Form with a bad field	Widely

Selector	Matches	Baseline
`:is(h1, h2) a`	Either, max specificity	Widely
`:where(h1, h2) a`	Either, zero specificity	Widely
`:focus-visible`	Keyboard focus ring	Widely
`li:nth-child(2 of .on)`	2nd `.on` item	Widely

`:has()` is the parent and previous-sibling selector CSS never had, so state that used to need a JavaScript class toggle becomes a selector.

```
.field:has(input:user-invalid) .hint {
  display: block;
}
body:has(dialog[open]) { overflow: hidden; }
h2:has(+ p) { margin-block-end: 0.25rem; }
:where(.btn) { padding: 0.5rem 1rem; }
```

> **Gotcha:** `:is()` takes the specificity of its most specific argument: `:is(#app, .x) a` costs an ID even when `.x` matched. Use `:where()` for defaults you want to override easily.

Popover, dialog and anchors

Syntax	Does	Baseline
`<dialog>` + `showModal()`	Modal, inert page	Widely
`popover` + `popovertarget`	Light-dismiss layer	Newly 2025
`command` / `commandfor`	Declarative buttons	Newly 2025
`anchor-name`, `position-area`	Anchor positioning	Newly 2026

Syntax	Does	Baseline
`<dialog closedby>`	Dismiss rules	Limited

A popover opens with no JavaScript; popovers and modal dialogs both render in the top layer, above every `z-index`. Anchor positioning tethers one element to another. Its core properties reached all engines in January 2026; the feature as a whole is still *Limited* (`position-visibility` values).

```
<button popovertarget="tip">Info</button>
<div id="tip" popover>Saved 2 min ago</div>
<style>
  button { anchor-name: --info; }
  #tip {
    position-anchor: --info;
    position-area: top;
    position-try-fallbacks: flip-block;
  }
</style>
```

> **Gotcha:** `dialog.show()` opens a non-modal dialog: no backdrop, no inert background, no Escape to close. Call `showModal()` for a modal. A popover is never modal either, so it does not trap focus.

View transitions

Syntax	Does	Baseline
`document.startViewTransition(fn)`	Same-document	Newly 2025
`view-transition-name: x`	Morph element pair	Newly 2025
`::view-transition-old(x)`	Style outgoing snapshot	Newly 2025
`@view-transition { navigation: auto }`	Cross-document	Limited

The browser snapshots the old state, runs your DOM update, then cross-fades or morphs between snapshots. Elements sharing a `view-transition-name` animate from one position to the other.

```
function show(next) {
  if (!document.startViewTransition) {
    render(next);
    return;
  }
  document.startViewTransition(() => render(next));
}
function render(n) {
  document.querySelector("main").textContent = n;
}
```

> **Gotcha:** Each `view-transition-name` must be unique on the page at snapshot time. Two visible elements with the same name abort the whole transition.

Color functions

Syntax	Does	Baseline
`oklch(70% 0.15 250)`	Perceptual color	Widely
`color-mix(in oklch, a, b 30%)`	Blend two colors	Widely
`oklch(from var(--c) l c h / 50%)`	Relative color	Newly 2024
`light-dark(#000, #fff)`	Per color scheme	Newly 2024

`oklch()` keeps lightness perceptually even across hues, so a palette built by changing only hue keeps equal contrast. Relative color syntax derives tints and alpha from one token.

```
:root {
  color-scheme: light dark;
  --brand: oklch(55% 0.18 250);
  --brand-soft: color-mix(
    in oklch, var(--brand), white 80%);
  --text: light-dark(#111, #eee);
}
.btn:hover {
  background: oklch(
```

```
    from var(--brand) calc(l + 0.1) c h);
}
```

> **Gotcha:** `light-dark()` does nothing unless `color-scheme` includes both `light` and `dark` on the element or an ancestor.

Further reading

- web.dev — Baseline
- web-features — Baseline data source
- MDN — CSS reference
- MDN — CSS container queries
- MDN — Popover API
- CSS Anchor Positioning Module Level 1