SkillNewton

CSS Cheat Sheet

Selectors, the box model, units, flexbox, grid, positioning, responsive design, custom properties and transitions on one page.

> This CSS cheat sheet collects the properties and selectors you reach for on every project, each with a short note on what it does. It suits beginners learning to style their first pages and developers who need a quick reminder of flexbox alignment, grid templates or `clamp()`. Work through it top to bottom when learning, or jump to a section while you code. All examples work in current versions of Chrome, Edge, Firefox and Safari.

How to use it

1. **Set sensible defaults.** Start your stylesheet with the box-sizing reset and custom properties from this sheet so every element sizes predictably.
2. **Choose the layout tool.** Use flexbox for a row or column of items and grid when you need rows and columns together.
3. **Build mobile first.** Write base styles for small screens, then add min-width media or container queries for larger layouts.
4. **Debug with DevTools.** Open the browser DevTools to inspect the box model, see which rule wins and toggle flex and grid overlays.

Selectors and specificity

`p { }`	Type selector: every `<p>`.
`.card { }`	Class selector: elements with `class="card"`.
`#main { }`	ID selector. High specificity, so prefer classes for styling.
`nav a { }`	Descendant: any `a` inside `nav`.
`ul > li { }`	Child: only direct children.
`h2 + p { }`	Adjacent sibling: the first `p` straight after an `h2`.
`a[href^="https"] { }`	Attribute selector: links whose href starts with https.
`a:hover, a:focus-visible { }`	Pseudo-classes for mouse hover and keyboard focus.
`li:nth-child(odd) { }`	Every odd list item.
`.card:has(img) { }`	Parent selector: cards that contain an image.
`:is(h1, h2, h3) { } / :where(h1, h2, h3) { }`	Group selectors. `:where()` adds zero specificity, `:is()` takes its most specific argument.
`p::first-line { } / .btn::before { }`	Pseudo-elements style part of an element or insert generated content.

```
/* Specificity is compared as (IDs, classes, types) */
p             /* 0,0,1 */
.intro p      /* 0,1,1 */
```

```
#hero .intro p /* 1,1,1  wins over both above */
/* Equal specificity: the rule written later wins.
   Inline styles beat all of these; !important beats inline. Avoid both. */
```

Box model

`*, *::before, *::after { box-sizing: border-box; }`	Width and height include padding and border. Add this to every project.
`margin: 1rem 2rem;`	Outer space: top and bottom 1rem, left and right 2rem.
`padding: 1rem 1.5rem 2rem 1.5rem;`	Inner space, clockwise: top, right, bottom, left.
`border: 1px solid #ccc;`	Width, style and colour of the border.
`border-radius: 8px;`	Rounded corners. `50%` on a square makes a circle.
`margin-inline: auto;`	Centres a block that has a width or max-width.
`max-width: 70ch;`	Limits line length for comfortable reading.
`display: block | inline | inline-block | none;`	How the element takes part in layout. `none` removes it completely.
`overflow: auto;`	Adds scrollbars only when content does not fit.
`outline: 2px solid; outline-offset: 2px;`	Focus ring that does not change layout. Never remove focus styles without a replacement.

Units

`px`	Absolute CSS pixel. Fine for borders and shadows.
`rem`	Relative to the root font size (usually 16px). Best for font sizes and spacing.
`em`	Relative to the current element font size. Handy for padding on buttons.
`%`	Relative to the parent (for widths) or to the element itself in some properties.
`vw / vh`	1% of the viewport width or height.
`dvh / svh / lvh`	Dynamic, small and large viewport height. `100dvh` avoids mobile address-bar jumps.
`ch`	Width of the "0" character. Good for line length.
`fr`	A fraction of the free space in a grid container.
`calc(100% - 2rem)`	Mix units in a calculation. Spaces around + and - are required.

Colours and typography

`color: #1a1a1a;`	Text colour as hex.
`background-color: rgb(255 255 255 / 0.9);`	Modern space-separated rgb with alpha.
`color: hsl(220 90% 45%);`	Hue, saturation, lightness. Easy to make lighter or darker shades.

`background: linear-gradient(135deg, #0ea5e9, #6366f1);`	Gradient background.
`font-family: system-ui, sans-serif;`	Font stack with a fallback generic family.
`font-size: 1.125rem; line-height: 1.6;`	Readable body text. Unitless line-height scales with the font size.
`font-weight: 600;`	Weight from 100 to 900; 400 is normal, 700 is bold.
`text-align: center;`	Horizontal text alignment.
`text-decoration: underline; text-underline-offset: 3px;`	Underline with some breathing room.
`text-transform: uppercase; letter-spacing: 0.05em;`	Uppercase labels with spacing.
`@font-face { font-family: "Inter"; src: url(inter.woff2) format("woff2"); font-display: swap; }`	Loads a web font and shows fallback text until it arrives.

Flexbox

`display: flex;`	Turns the element into a flex container; children line up in a row.
`flex-direction: row | column;`	Main axis direction.
`justify-content: space-between;`	Distributes items along the main axis. Also `center`, `flex-start`, `space-evenly`.
`align-items: center;`	Aligns items on the cross axis (vertically in a row).
`gap: 1rem;`	Space between items without margins.
`flex-wrap: wrap;`	Lets items move onto new lines.
`flex: 1;`	On a child: grow to fill the free space equally.
`flex: 0 0 200px;`	On a child: fixed 200px, no growing or shrinking.
`align-self: flex-end;`	Overrides cross-axis alignment for one item.
`order: -1;`	Moves an item visually. Does not change reading or tab order.

```
.navbar {
  display: flex;
  justify-content: space-between;
  align-items: center;
  gap: 1rem;
  flex-wrap: wrap;
}
```

Grid

`display: grid;`	Turns the element into a grid container.
`grid-template-columns: 1fr 2fr;`	Two columns, the second twice as wide.
`grid-template-columns: repeat(3, 1fr);`	Three equal columns.
`grid-template-columns: repeat(auto-fit, minmax(240px, 1fr));`	Responsive card grid with no media queries.

`gap: 1.5rem 1rem;`	Row gap, then column gap.
`grid-column: 1 / -1;`	Child spans from the first to the last column line.
`grid-column: span 2;`	Child spans two columns.
`grid-template-areas: "head head" "side main";`	Names areas of the layout.
`grid-area: main;`	Places a child into a named area.
`place-items: center;`	Centres every child horizontally and vertically in its cell.

```
.layout {
  display: grid;
  grid-template-columns: 240px 1fr;
  grid-template-areas:
    "head head"
    "side main";
  gap: 1rem;
}
.layout > header { grid-area: head; }
.layout > aside  { grid-area: side; }
.layout > main   { grid-area: main; }
```

Positioning and stacking

`position: static;`	Default. The element follows normal flow.
`position: relative;`	Stays in flow but can be nudged; also becomes the anchor for absolute children.
`position: absolute; inset: 0;`	Removed from flow and placed against the nearest positioned ancestor.
`position: fixed; bottom: 1rem; right: 1rem;`	Fixed to the viewport, for example a chat button.
`position: sticky; top: 0;`	Scrolls normally, then sticks once it reaches the top.
`z-index: 10;`	Stacking order for positioned elements. Higher numbers sit in front.
`inset: 0;`	Shorthand for top, right, bottom and left.
`aspect-ratio: 16 / 9;`	Keeps a box at a fixed ratio, useful for video embeds.
`object-fit: cover;`	Fills an image box without stretching, cropping the overflow.

Responsive design

`@media (min-width: 768px) { }`	Applies rules on screens 768px and wider (mobile-first).
`@media (width >= 768px) { }`	Same query in the newer range syntax.
`@media (prefers-color-scheme: dark) { }`	Styles for visitors using dark mode.
`@media (prefers-reduced-motion: reduce) { }`	Turn off or reduce animation for visitors who ask for it.
`@media (hover: hover) { }`	Hover effects only on devices that can hover.
`container-type: inline-size;`	Makes an element a size container for container queries.
`@container (min-width: 400px) { }`	Styles a component by the width of its container, not the viewport.

`font-size: clamp(1rem, 0.9rem + 0.5vw, 1.25rem);`	Fluid size with a minimum, preferred value and maximum.
`img { max-width: 100%; height: auto; }`	Images never overflow their container.

```
.card-list { container-type: inline-size; }

.card { display: grid; gap: 1rem; }

@container (min-width: 500px) {
  .card { grid-template-columns: 160px 1fr; }
}
```

Custom properties and transitions

`:root { --brand: #2563eb; --space: 1rem; }`	Defines custom properties (CSS variables) globally.
`color: var(--brand);`	Uses a custom property.
`padding: var(--space, 1rem);`	Uses a fallback when the property is not set.
`.dark { --brand: #60a5fa; }`	Overrides a variable for a theme or component; children inherit it.
`transition: background-color 200ms ease;`	Animates a property change, such as on hover.
`transition: transform 150ms ease-out, opacity 150ms;`	Several properties with their own timings. `transform` and `opacity` are cheapest to animate.
`transform: translateY(-2px) scale(1.02);`	Moves or scales without affecting layout.
`@keyframes fade { from { opacity: 0; } to { opacity: 1; } }`	Defines a keyframe animation.
`animation: fade 300ms ease-in both;`	Runs the animation on an element.

```
:root { --brand: #2563eb; }

.btn {
  background: var(--brand);
  color: #fff;
  transition: transform 150ms ease-out;
}
.btn:hover { transform: translateY(-2px); }

@media (prefers-reduced-motion: reduce) {
  .btn { transition: none; }
}
```

Online version, with the matching free guide: skillnewton.com/resources/cheat-sheets/css-cheat-sheet/
Learn this hands-on: PHP Web Development: Foundation & Practical Implementation – skillnewton.com/courses/php-foundation-course/