

CSS
can do

what?!

👋 Hi! I'm Joran!

- 🕸 Web developer
- 🗣️ Author, Coach and Speaker
- 🖱️ I worked as Design System Tech Lead

joranquinten.nl | [@joranquinten](https://twitter.com/joranquinten)











I'm an average developer from an average company.

Organisational context

- 🛒 We've been doing groceries for longer than the internet existed
- 🧑 Tech is a means of fulfilling some of our goals
- 💰 We don't dedicate all or our companies resources into tech
- 🏢 We do our in house development as best we can
- ⚡ Development focus is aimed at JavaScript progression

Design System (component library part)

-  Built with Vue
-  Uses Sass in SFCs, and exposes mixins and variables
-  Multi team collaborative setup
-  History of about 5 years of development
-  We have been working on leveling up the design system
-  Closed resource



CSS

Global
HD

Meanwhile...

Browser vendors are killing it
with implementing and
supporting features

Working group is progressively
adding more and more useful
features

Crash course to the features of current CSS 

Or:

Confessions of a Web Developer

Things I didn't know CSS can do!

💡 But now I do!

Let's go! 🚀

Nesting

Organising code by nesting selectors, where the child inherit parents' specificity. The child rule is selector is calculated relative the the parent rule selector.

- Helps with readability, modularity and maintainability
- Additional benefit of reduced CSS file sizes
- Syntax is very familiar coming from sass like notation

Nesting

Default `h2` definition with nested rules in a `.card` element:

```
1  h2 {  
2    font-size: 2rem;  
3    color: red;  
4  }  
5  
6  .card {  
7    font-family: system-ui;  
8    font-size: 1.25rem;  
9    h2 {  
10     font-size: 4rem;  
11     color: blue;  
12   }  
13 }
```

Scoping

More control over the boundaries of the cascade and applied styles while preventing naming (and style) collisions!

- More freedom in sensible class naming systems
- Import styles from a lib and apply them to a scope!
- Compatible with component based approach, but just as compatible with cascading approach!
- Having control over both outer and inner boundaries of the scope is wild!

Scoping Example

Notation of a scope applied to a reusable pattern:

```
1 <div class="card">
2   <h3 class="title">My card title</h3>
3   <div class="slot">
4     <div class="slotted-content"></div>
5   </div>
6 </div>
7
8 <style>
9 @scope (.card) to (.slot) {
10  :scope {
11    padding: 1rem;
12  }
13  .title {
14    font-size: 1.4rem;
15  }
16 }
17 </style>
```


Donut scope 🍩

The bounds define a "ring" of elements around a "hole" of unscoped style. Ergo: donut scope

```
1 <div class="donut">
2   <h3 class="topping">Sprinkles ✨ </h3>
3   <span class="chocolate">Yum! 🍫 😊 </span>
4   <div class="hole">🕳️ </div>
5 </div>
6
7 <style>
8 @scope (.donut) to (.hole) {
9   :scope {
10     border-radius: 100%
11   }
12   .topping {
13   }
14   .chocolate {
15     background-color: chocolate;
16   }
17 }
18 </style>
```

Container Queries

Querying where it matters! Media queries' evolution

- Beyond the limits of media queries: target the container rather than the viewport
- Leads to more reusable components with fewer programmatic adjustments
- More and better control of visual representation

Responsive Elements using Container Queries

The anatomy of the query definition

```
1  .container {
2    container-type: inline-size;
3    container-name: cardContainer;
4  }
5
6  /* Define the behavior of elements in the container like this */
7  .card {
8    display: flex;
9    flex-direction: row;
10 }
11
12 @container cardContainer (max-width: 400px) {
13   .card {
14     flex-direction: column;
15   }
16 }
```

Container Queries in Practice

👉 One caveat though is the extra `.container`` class you'll see popping up depending on the layout.

```
1 <div class="container">
2   <article class="card">
3     <header class="card-hero">
4       
5     </header>
6     <div class="card-content">
7       <p>Mandatory lorem ipsum content
8         which you need to remove before
9         going to production.
10      </p>
11      <p>Definitely never happened to me 👉 </p>
12    </div>
13  </article>
14 </div>
```

```
1 .container {
2   container-type: inline-size;
3   container-name: cardContainer;
4 }
5
6 .card {
7   display: flex;
8   flex-direction: row;
9   gap: 1rem;
10  padding: 1rem;
11 }
12
13 header {
14   max-width: 400px;
15 }
16
17 @container cardContainer (max-width: 400px) {
18   .card {
19     flex-direction: column;
20   }
21 }
```

Container Units

New units for container spaces!

- Set dimensions based on current size of container element
- Allow for more flexible responsive designs
- Support for logical properties

```
1  .element {
2    width: 50cqi; /* or 50cqw */
3    height: 20cqb; /* or 50cqh */
4    font-size: 2cqmin; /* 2% of the smaller dimension */
5    border-radius: 1cqmax; /* 1% of the larger dimension */
6  }
```

- ``cqw`` Container query width units
- ``cqh`` Container query height units
- ``cqi`` Container query inline size units
- ``cqb`` Container query block size units
- ``cqmin`` Container query minimum size units
- ``cqmax`` Container query maximum size units

👉 Prefer ``cqi`` and ``cqb`` over width and height

Subgrid

Simplifying complicated layouts

Subgrid makes designing with grids easier in a few key ways:

- Consistency across breakpoints - Your designs stay the same across different screen sizes without extra work.
 - Flexible nesting - You can have grids within grids and they all work together nicely.
 - Granular control - You can tweak the smaller grids without messing with the big picture.
 - Easier maintenance - It's simpler to manage your layout with one set of rules for the whole thing.
- 👉 The subgrid is part of the parent. It's not a nested grid!

Aligning the content inside a grid

```
1 <div class="grid">
2   <article class="subgrid">
3     <h2>What sane person would
4       have such long headers
5       that they span more than
6       three lines anyway? 🤖
7     </h2>
8     <p>The paragraph text</p>
9   </article>
10
11  <article class="subgrid">
12    <h2>Short header</h2>
13    <p>The paragraph text</p>
14  </article>
15
16  <article class="subgrid">
17    <h2>Short Header</h2>
18    <p>The paragraph text</p>
19  </article>
20 </div>
```

```
1 .grid {
2   max-width: 60rem;
3   display: grid;
4   grid-template-columns: 1fr 1fr 1fr 1fr;
5   gap: 1rem;
6 }
7
8 .subgrid {
9   grid-row: auto / span 2;
10  display: grid;
11  grid-template-rows: subgrid;
12  gap: 0.4rem;
13 }
```

Cascade Layers

Cascade layers are not new!

They extend existing layers such as:

1. User-agent stylesheets
2. User settings
3. Author stylesheets

Use Cases for Cascades in the Author Layer

Breakers in the cascade where you can organize definitions within pools of specificity.

- Different types of collections of styles (i.e. reset, utilities or third party libraries)
- Different collections may have different needs of specificity
- Wrangling these collections on the top level may be challenging

Cascade How

Setting up layers

```
/* Nested layers */
@layer utils {
  @layer typography {
    h1 {
      font-family: "Comic Sans";
      color: red;
    }
  }
}
```

```
/* Shorthand nested */
@layer utils.typography {
  h1 {
    font-family: "Comic Sans";
    color: red;
  }
}
```

Combine styles with `:is()`

The `:is()` pseudo-class lets you apply the same CSS rules to different elements at the same time. You just list the elements inside the parentheses, separated by commas.

- Applying the same styles to different elements without repeating yourself
- Avoiding tricky issues with too-specific or invalid selectors
- Cutting down the amount of CSS you have to write

```
:is(button, .custom-input, ::unsupported) {  
  border-radius: 4px;  
  padding: 8px 12px;  
}
```

Flexible conditional styling with `:has()`

- Allows you to break out of the current element and inspect the state of child elements
- Changing how things are laid out if specific content is present
- Chain it for multiple use cases! It works with combinators and pseudo-classes

```
article {
  margin: 1rem;
}

article:has(.highlight, .hero-image) {
  border: 1px solid #fee599;
}

article:has(.highlight, .hero-image) .hero-image {
  background-size: cover;
}
```

Color mix

The `color-mix()` function allows to color mix on the fly! And yes, with CSS vars, you can build an entire color palette based on a single color definition.*

Combined with the clamp and min max is 🔥

```
1  :root {
2    --base-color: #bada55;
3    --primary-color: var(--base-color);
4    --secondary-color: color-mix(in srgb, var(--base-color) 80%, blue 20%);
5    --success-color: color-mix(in srgb, var(--base-color) 80%, green 20%);
6    --warning-color: color-mix(in srgb, var(--base-color) 80%, orange 100%);
7    --danger-color: color-mix(in srgb, var(--base-color) 80%, red 100%);
8  }
```

* This is probably not a good idea in practice: design is not mathematics.

👉 *Use the relative colors of OK-LCD based theming for this!* 🙋

P3 Color functions

Gotta have more colors!

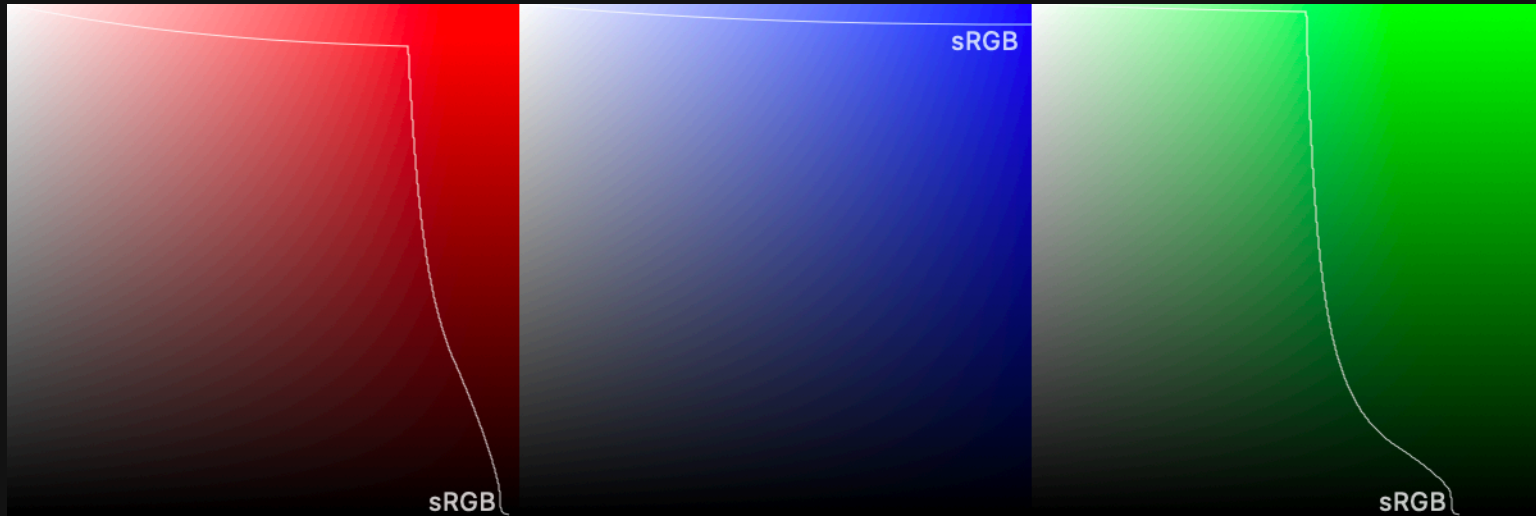
- More vivid colors that better match imagery
- More subtle and fluent gradients
- More control over color and contrasts



Side by side comparison of sRGB vs P3

P3 is a superset of sRGB

Visualisation of the range of P3



P3 Example

```
:root {  
  --hulk-green: color(display-p3 0 1 0);  
  --iron-man-gold: color(display-p3 0.85 0.65 0.1);  
  --thanos-purple: color(display-p3 0.45 0.2 0.65);  
}
```


Logical Properties

Clear sense of direction for everyone 🙋🙋🙋🙋

- CSS was developed with the English language in mind. In English language, the inline direction is horizontal, left to right and block direction is vertical, top to bottom.
- In languages such as Arabic, the inline direction is horizontal, but right to left.
- In languages like Chinese, Japanese, Korean or Mongolian, the inline direction is vertical!

All of a sudden, giving an element a `\margin-right`` or `\margin-top`` is not as straightforward as it seems! 🤖

Logical Properties

Logical properties aim to solve this problem by using properties that are not constraint to our cultural directional viewpoints.

- Don't think ``horizontal`` but think: ``inline``
- Don't think ``vertical`` but think: ``block``
- Think about ``start`` and ``end`` to mark sides in any direction

```
body { direction: rtl; /* Changing the direction reflows the element without effort */ }

.card {
  width: 14rem;
  margin-right: .25rem;
  padding-top: 1rem;
}

.universalCard {
  inline-size: 14rem; /* "width" as a horizontal concept ← "inline-size" */
  margin-inline-end: .25rem; /* "left" and "right" as a horizontal concept ← "inline", "right" becomes "end" */
  padding-block-start: 1rem; /* "top" and "bottom" as a vertical concept ← "block", "top" becomes "start" */
}
```

Margin trim

The `margin-trim` property removes any margin in the direction specified from the selected container at the end of that direction.

```
<ul>
  <li>First Element</li>
  <li>Second Element</li>
  <li>Third Element</li>
</ul>

<style>
ul {
  padding: 1rem;
  margin-trim: block-end;
}

li {
  margin-block-end: 2rem;
}
</style>
```

Text wrapping

More balanced and pretty wrapping of text by requesting the browser to figure out wrapping.

- The ``balance`` property creates a visually more aesthetic balance between multiple lines.
- The ``pretty`` property gets rid of orphaned words that would spill into a new line.

```
1  .headline {
2    text-wrap: balance;
3  }
4
5  .excerpt {
6    text-wrap: pretty;
7  }
8
9  .extra {
10   word-break: auto-phrase;
11 }
```

💡 Good rule of thumb: ``balance`` for headlines and ``pretty`` for paragraphs.

Utils

- `min()` Sets the smallest value from a list of expressions as the CSS property value. Ensures styles stay within minimum constraints.
- `max()` Sets the largest value from a list of expressions as the CSS property value. Useful for preventing styles from becoming too constrained.
- `clamp()` Clamps a value between a minimum and maximum. Takes three parameters: `min`, preferred, and `max` values for flexible yet controlled styles. 🔥

```
1  :root {
2    --dynamic-margin: min(1rem, 5%);
3    --dynamic-padding: max(2rem, 4%);
4    --dynamic-font-size: clamp(1.5rem, 5vw, 3rem);
5  }
```

Animating discrete properties

Discrete properties are not interpolated or "tweened", but at the end of the animation are drawn in their final state.

- Allows for more accessible definitions without JS
- Useful when elements get visually added or removed from the page
- Define the initial state

```
1  .toast-message {
2    transition: opacity 0.5s, display 0.5s;
3    transition-behavior: allow-discrete;
4
5    @starting-style {
6      opacity: 0;
7      height: 0;
8      display: none;
9      z-index: -1;
10   }
11
12   .active {
13     opacity: 1;
14     height: auto;
15     display: block;
16     z-index: 10;
17   }
18
19   .dismissed {
20     opacity: 0;
21     height: auto;
22     display: none;
23     z-index: -1;
24   }
25 }
```

Scroll driven animations

Capture the attention of your visitors with added motion!

- Provides an enhanced user experience
- Offload animations to the GPU
- Keeps JS threads open for other operations

```
1  .container {
2      inline-size: 60vw;
3      block-size: 300px;
4      background-image: url('../image.jpg');
5      background-size: cover;
6      background-position: center;
7      transition: transform 0.3s ease;
8  }
9
10 @media (prefers-reduced-motion: no-preference) {
11     @scroll-timeline {
12         scroll: vertical 1 100%;
13     }
14     .container {
15         animation: scaleUp 1s linear both;
16         animation-timeline: scroll;
17     }
18     @keyframes scaleUp {
19         0% { transform: scale(1); }
20         100% { transform: scale(1.2); }
21     }
22 }
```

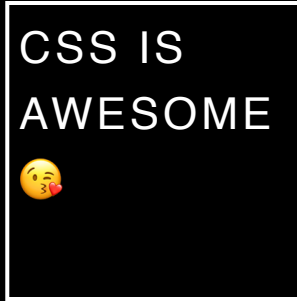

Now what? 🙄

Benefits 🙌

- Reduced build step and bundle size (do less and ship less!) 🚀
- Reduced generated CSS definitions reduce bundle size too! 📦
- More flexibility with runtime parsing (Lightning CSS / PostCSS) for optimization ⚡
- CSS Variables offer on the fly updates that cascade 🌊
- Nesting is a perfect match for organizing code 🧠

Considerations 🤔

- Adopting layers is more invasive but we expect pay off in the long run 📄
- Shifting to container queries perfectly matches component like approach on responsive behavior ↔
- Rewriting Sass to CSS codebase 🤝



joranquinten.nl

[@joranquinten](https://twitter.com/joranquinten)