CHRIS LILLEY

Ask an expert: Why is CSS . . . the way it is?

A technical director of the W3C's interaction domain unpacks the histories and mysteries.

PART OF

ISSUE 13
MAY 2020

Frontend

As a technical director at the World Wide Web Consortium (W3C), I often give talks about CSS's great features and capabilities. But from time to time, I get questions about the bad parts. Why does CSS have feature X, when feature Y does all that and more? Why is feature Z so hard to use? Basically, "What was the CSS group thinking?"

CSS debuted way back in 1994, when web browsers were a very new, relatively undeveloped technology. People were excited just to see documents that lived on other computers across the world—and not just in plain text, but with headings and lists, too! Adding finer control over presentation was mostly seen as a secondary goal.

The two browsers then in common use, Internet Explorer and Netscape Navigator, had totally different Document Object Models, and there was

widespread disagreement over where to add presentational control: add a few attributes to the HTML, change the styling through script, or, the least popular option, create an entirely different presentation language. This last option was seen as too academic for a web that was expanding beyond universities to the general public.

As the unfashionable option, it was important to CSS's success that it be seen as easy and straightforward—a small addition, really, to what was mostly networking code. A more feature-rich proposal may well have failed at the first hurdle: If no one implemented it, then CSS would have no impact on the rapidly evolving web.

The question "Why is CSS the way it is?" is a reasonable one. And, having been involved in CSS since its very first days, I'm (sometimes painfully) aware of the answers. Growing slowly and incrementally over time, CSS has evolved into a full-featured and essential part of the modern web. But despite our best efforts, frustrating or confusing features exist, and in all likelihood will continue to be added. So, for present and future users, it's useful to know why CSS is the way it is, and to learn the backstories behind the features I'm asked about most often.

Basic beginnings

Some of CSS's earliest features, necessary at the time, caused myriad problems down the line and actively got in the way of improving CSS.

THE HUMBLE FLOAT

To the modern designer, familiar with `flexbox` and `grid`, the first CSS layout feature, `float`, is painfully primitive. Move an image all the way to the left (or right), and the text shows up to the right (or left), rather than skipping straight past the image and leaving an ugly white

space. That's it.

Crucially, you can't easily achieve this effect using HTML tables—at least, not if you still plan to modify the text—and tables were the main competitor for enhancing layout. When it was first developed, `float` had a real impact on how the page looked and was therefore widely used. That it offers little additional control and is badly underspecified in even slightly complex cases was a problem to solve later.

HOSTILE HEX

This "good enough for now" philosophy explains why colors in CSS were first specified in RGB—already known to be an unintuitive, user-hostile syntax—even though better systems existed. RGB was understood by engineers, required no device calibration, and was what you sent to the graphics driver. Small details like consistency, usability, and ease of getting the right color could be solved later. Much, much later, as it turned out.

The smallest improvements

Once a feature is in place, it's easier to slightly improve it than to add a new, better, but completely different feature that does the same thing.

LIST MARKERS

This explains, for example, why list markers were initially specified in CSS by expanding the role of `float`. (The list marker was floated left so the list item text wrapped around it to the right.) That effort was abandoned and replaced by the `list-style-position` property, whose definition currently has the following, not very confidence-inspiring inline issue: "This is handwavey nonsense from CSS2, and needs a

real definition."

TINY IMPROVEMENTS TO COLOR

This also explains why the first two improvements to specifying color in CSS—a named-color system and a hue-wheel, polar notation—were adopted over much better, but more complicated, systems proposed at the same time. They were slight improvements, seen as easy to implement.

MYSTERIOUS COLOR NAMES

What do you imagine "vivid deep blue," "very dark green," or "pale light reddish-orange" to look like? These are examples from the Color Naming System (CNS) described by Toby Berk and coauthors in a 1982 paper published by IEEE Computer Graphics and Applications, which I proposed CSS adopt in 1996.

Contrast these with "orchid," "gainsboro," or "burlywood." These are examples of X11 colors, which were added to SVG in 2000 and then to CSS in 2003. It isn't possible to modify or refine these color names with adjectives, as it is in CNS; you basically need to memorize, or refer to, the entire list.

So what happened? In the early days, there was resistance to standardizing a big color list: Memory was small and expensive, especially on the newly emerging handheld devices that preceded smartphones. Then, by the time such resistance had faded, the Unix workstation X11 names had spread to the more popular Mac and PC implementations. Lastly, the full algorithm that made it possible to convert CNS colors to sRGB wasn't published in the original paper, and my efforts to reach out to the original authors were unsuccessful. The idea lost momentum, and we chose the path of least resistance instead.

HUE WHEELS

The concept of a circular, rainbow arrangement of colors has been familiar to artists for centuries. Extending this wheel to black, gray, and white in the center with progressively more vivid colors toward the edge has a similarly long history. Less frequent is when a color-wheel designer takes care to evenly space the colors around the wheel so they don't seem bunched up in any one area. The *Munsell Book of Color*, published in 1929, was the first to succeed in this respect.

In 1976, the International Commission on Illumination (or CIE, an abbreviation for Commission internationale de l'éclairage, its French name) standardized a three-dimensional representation of color derived from physical measurement, corresponding to how the human eye sees color. In this system, called CIELAB, the distance between two colors is directly related to how different they appear visually, a concept termed "perceptual uniformity." A hue-wheel derivative of this, CIE LCH (for Lightness, Chroma, Hue), combined the usability advantages of the Munsell color system with scientific rigor and direct physical measurement of any colored object.

Unfortunately, when the CSS Working Group added a hue wheel system to CSS in 2002, the greatly inferior HSL (for Hue, Saturation, Lightness) system, which lacked the CIE LCH's perceptual uniformity, was adopted instead. In it, bright yellow and dark blue have the same HSL lightness, and hues are bunched up in certain places but widely spaced elsewhere. Adoption was driven by a desire to have some sort of hue wheel system, the lack of dependence on display calibration, and math that made it somewhat simpler to convert HSL values to RGB than CIE LCH. Since it was already widely used in other programs, HSL seemed like it would be an easy addition to CSS, but the disadvantages—non-perceptual uniformity chief among them—would hit hard when developers used CSS preprocessors to implement design systems in CSS. (This is also

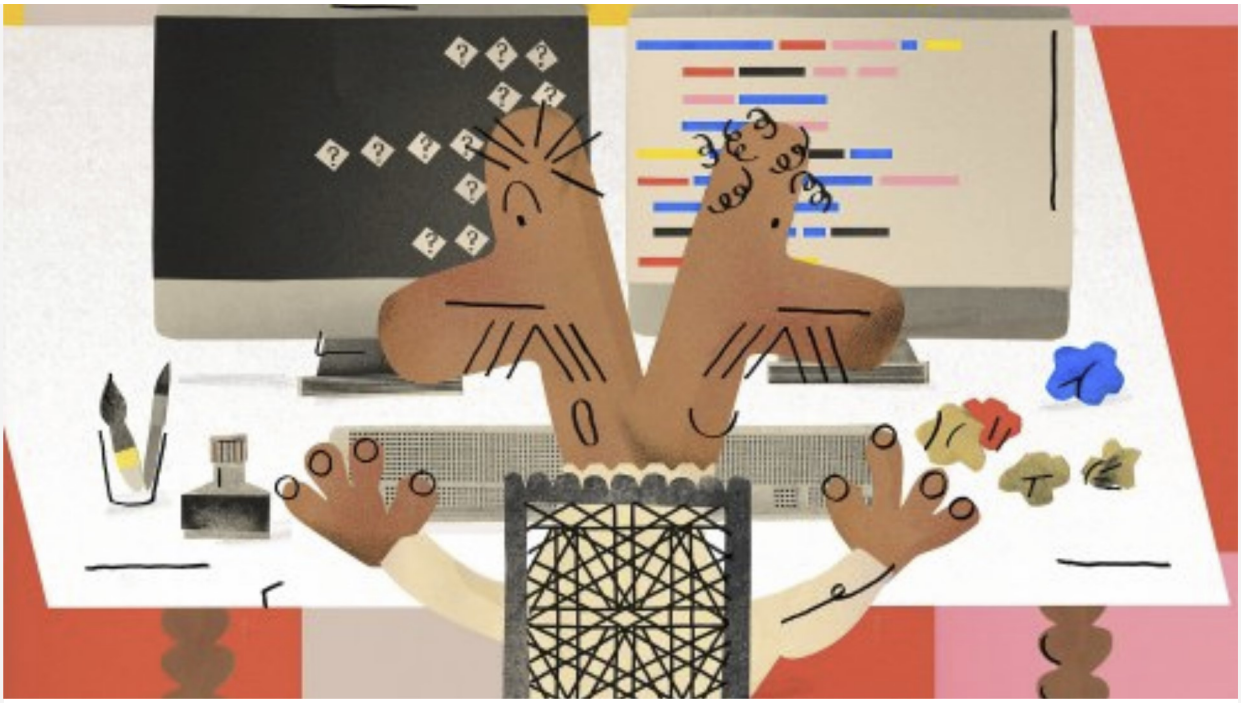
why CSS Color 5 was started, and why two of the four coauthors are design systems specialists.)

Insufficient review

Sometimes a feature is added to a CSS draft and no one has the time or the expertise to review it. Usually, troublesome features are caught by test implementations during the Candidate Recommendation (CR) phase of specification development. But not always.

MYSTERIOUS RANGES

One notable example is the syntax for `unicode-range`, which is used to indicate ranges of characters in a font that you don't want used. This was added to CSS2 following a request from the Unicode Consortium in May 1997. At the time, Unicode was brand new and somewhat experimental. To distinguish Unicodes from the more common ASCII or Latin-1 codes, people used the prefix U+. (No one does this anymore.) I came up with a compact syntax the following month to represent the ranges; it assumed you had a copy of the Unicode specification to refer to and were comfortable with hex notation and wildcards.



FROM ISSUE 5

Unplain text

A primer on text shaping and rendering non-Latin text in the shadow of an ASCII-dominated world.

This example specifies to only use the font for Japanese:

```
unicode-range: U+A5, U+4E00-9FFF, U+30??, U+FF00-FF9F;
/* yen, kanji, hiragana, katakana */
```

If you think this syntax is bad—and it is—consider that the only alternative proposal was a complete bitmap of Unicode 1.1:

```
unicode-range: 0x02037FBC4571000003100C0000001000100000300BDF74
```

After a week of mailing list discussion and no suggestions for

improvement, the consensus was that my syntax was good enough for now. (Ha!) We added it to the specification, thinking we could always improve it later. (This was some years before CR testing became a thing.) After 23 years, we're still using it, and I'm still apologizing for it.

The actual implementation testing occurred between 2013 and 2018 with CSS Fonts Level 3, which ironed out a lot of the bugs and precisely defined the corner cases. Even so, this syntax is unwieldy, especially for languages with discontinuous ranges, such as Chinese. It requires special handling in the CSS parser, and it puts the burden on the style sheet author to keep track of revisions to the Unicode specification as more characters are added. Not great.

Looked good enough at the time

Some suggestions are reviewed, seem okay, and get implemented. Years later, their shortcomings become increasingly obvious.

DODGY DISPLAY

A prime example is the `display` property. This is mainly used to specify whether a particular element should render like a paragraph, with new lines before and after (`block`), or like a run of styled text as part of a paragraph (`inline`). The first draft of CSS in August 1996 also added `none`, which disabled rendering entirely. More values have been added since then.

Though scripting languages and dynamic modification had not yet become common, it's easy to see the problem in retrospect. Suppose I use script to set the value to `none` to hide it. Later, I want to unhide it, but the original value of the property is now lost.

Also, newer values like `inline-block` have made it clear that this property is doing two things: changing what the element looks like to other elements surrounding it (`inline-block` looks just like `inline` on the outside), and changing what the element looks like to its children (`inline-block` looks just like `block` on the inside). Not to mention the hide/don't hide behavior I mentioned earlier.

Fortunately, CSS has a mechanism to handle this grouping of related properties: shorthands. A shorthand is a way to set the value of multiple longhand values at once. So in the future, CSS could add three longhand properties called, say, `display-outside`, `display-inside`, and `display-hiding`; the latter would take the place of the `none` and not-`none` values. The existing `display` property would then become a shorthand that sets the value of the three longhands.

FUNNY FONTS

Not that shorthand properties are a silver bullet. One of the original shorthand properties from the first CSS proposal, font, was meant to emulate "a traditional typographic shorthand notation to set multiple properties related to fonts." Take, for example:

```
font: 700 12pt/14pt "Times New Roman"
```

Here, 700 is the font weight on a 0 to 999 scale, 12pt is the font size, 14pt is the leading (interline size), and the string in quotes is the font family name. The weight can be omitted, defaulting to 400. It would've been nice to allow the size to be omitted as well, but we couldn't do that. Why? Because, unfortunately, CSS allowed the quoting to be omitted. Compare:

```
font: 50 Shades of Gray
```

with:

```
font-weight: 50;
font-family: Shades of Gray
```

or, alternatively:

```
font-family: 50 Shades of Gray
```

Because the family name happened to start with a number, it would have introduced ambiguity to the shorthand.

A glimmer of hope

Though I've highlighted many of CSS's shortcomings, don't feel despondent. I have cheering news to lift the spirits (at least a little). First, after over 20 years of being ignored, LCH in CSS Color 4 is being implemented by Apple in Safari right now. There are also moves to add it to Chrome. Color modification functions, which rely on a perceptually uniform color space, will finally be able to take advantage of LCH. Second, the CSS Working Group is currently designing a proposal to add Unicode script codes as well as numbers for `unicode-range`. It will allow developers to write easy-to-remember and maintenance-free CSS like `unicode-range: Japanese`. Third, all CSS Working Group specifications and their related issues have now been maintained on

GitHub for over five years. Any interested member of the public can contribute to solving issues or pointing out errors. Take it from an expert: You can help!

ABOUT THE AUTHOR

Chris Lilley is a technical director at the World Wide Web Consortium working on CSS, web audio, web fonts, and SVG.

@svgeesus

TOPICS

Ask an Expert

Learn Something New

CONTINUE READING

13 Frontend

RAMSEY NASSER

A frontend stack for video games

Tales of a powerful and expressive game engine built entirely from open-source, web-based technologies.

13 Frontend

IPSITA AGARWAL

Case study: Web components for screen readers

How Slack changed the way it designs accessible frontend components.

13 Frontend

CHRIS STOKEL-WALKER

The rise of React

On the social, cultural, and technological impacts of the increasingly ubiquitous

frontend framework.

ELIZABETH MINKEL

A frontend of our own

The true story of what happened when a group of fanfiction writers built a Hugo award–winning—and resolutely, delightfully amateur—web publishing platform.

13 Frontend

IPSITA AGARWAL

Case study: Mobile payments in India

How Google designed an app for users from big cities to rural areas on devices old and new.

3 Development

MATT KLEIN

Ask an expert: Has adopting microservice architecture changed the way we develop software?

We asked Matt Klein, Senior Software Engineer at Lyft.

6 Documentation

DAVID J. LUMB

Inside the complex world of life-saving software

Nuclear power plants. Medical devices. Airplanes. Self-driving cars. Developing

software for safety-critical projects takes documentation to the next level.

FREDERIK VOLLERT

Ask an expert: What's the best way to begin software localization?

PhraseApp's Frederik Vollert provides an overview of the essentials.

8 Internationalization

10 Testing

IPSITA AGARWAL

A test of meaning

For AARP, AutoCAD, Google, and Pinterest, qualitative research can include everything from focus groups to hand-drawn maps.

EXPLORE TOPICS

Learn Something New · Scaling & Growth · Ask an Expert
Interviews & Surveys · Guides & Best Practices
Essays & Opinion · Workplace & Culture

ALL ISSUES

ISSUE 19
NOVEMBER 2021
Planning

ISSUE 18
AUGUST 2021
Mobile

ISSUE 17
MAY 2021
Containers

ISSUE 16
FEBRUARY 2021
Reliability

ISSUE 15
NOVEMBER 2020
Remote

ISSUE 14
AUGUST 2020
APIs

ISSUE 13
MAY 2020
Frontend

ISSUE 12
FEBRUARY 2020
Software Architecture

ISSUE 11
NOVEMBER 2019
Teams

ISSUE 10
AUGUST 2019
Testing

ISSUE 9
MAY 2019
Open Source

ISSUE 8
FEBRUARY 2019
Internationalization

ISSUE 7
OCTOBER 2018
Security

ISSUE 6
AUGUST 2018
Documentation

ISSUE 5
APRIL 2018
Programming Languages

ISSUE 4
FEBRUARY 2018
Energy & Environment

@incrementmag

incrementmag

RSS Feed

ABOUT

Increment is a print and digital magazine about how teams build and operate software systems at scale. Learn more

WORK WITH US

Interested in joining the team at Stripe? View job openings

 Published by Stripe Privacy policy