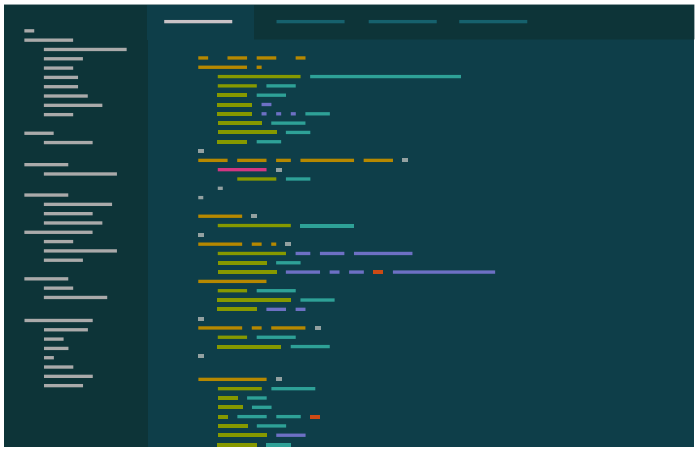
CSS for People Who Hate CSS

02.09.2016

CSS can quickly become an unmanageable nightmare of important rules and overly-specific selectors, don't let it.



Over my career as a designer & developer for the web, I've worked with some ridiculously intelligent developers. These are people who write databases for fun, people who can easily hash out the finer points of a complicated REST API and *then*

implement it that day. These are people who are solving ridiculously challenging problems with beautiful algorithms. They're writing in seriously hardcore languages like C. They're managing to work inside of insanely complicated legacy enterprise Java codebases to implement new features. These are people that earned *and use* a computer science degree. In short, these are people that are *way smarter than I am*.

And after meeting all of these people, there is one thing that they all have in common:

> They HATE writing CSS.

For a long time I wondered why. CSS seems way easier than what they're working on every day. You just add a class here, and a class there, and then a few rules, and BAM your web site is ready! No complicated algorithms or compiled binaries, just some visually improved HTML. Easy to preview and trouble shoot in the browser, no problems.

But thinking this way ignores one of the central problems with CSS:

> CSS is naturally drawn towards chaos, disorder, and complexity.

Think about it. If there is a problem with a site's style, rather than go in and fix the original style, you just add a new rule. Just add another class, with another selector, tack it at the end of the file and move on with your life. Easy.

But as more and more developers across an organization write more and more little rules and fixes, the complexity of the CSS grows dramatically. In a matter of months you can go from this:

```
.right {
  float: right;
}
```

to this:

```
.right,
.rtl .left,
.rtl #top .site-brand,
.rtl #top .site nav,
.rtl #top .site nav li,
.rtl .cui-carousel-item,
.rtl .case-study-group .block,
.rtl fieldset label,
.rtl .column-1, .rtl .column-2,
.rtl .column-3, .rtl .column-4,
.rtl .column-5, .rtl .column-6,
.rtl .column-7, .rtl .column-8,
.rtl .column-9, .rtl .column-10,
.rtl .column-11, .rtl .column-12,
.rtl .column-13, .rtl .column-14,
.rtl .column-15, .rtl .column-16,
.rtl .column-17, .rtl .column-18,
.rtl .column-19, .rtl .column-20,
.rtl .column-21, .rtl .column-22,
.rtl .column-23, .rtl .column-24,
.rtl .feature .feature-text,
.rtl .feature .feature-block,
.rtl .header .header-inner,
.rtl .navigation-bar header,
.rtl .navigation-bar nav,
.rtl dl.inline dt,
.rtl .toolbar .btn,
.rtl .toolbar .dropdown-wrapper,
.rtl .btn-group a,
.rtl .input-list li label,
.rtl input[type="checkbox"],
.rtl input[type="radio"],
.rtl .navigation-bar header h1,
.rtl .navigation-bar-nav ul li,
.rtl .tab-group .tab-nav .tab {
  float: right !important !important !important;
}
```

I would argue this is a big part of the reason so many great developers hate writing CSS. By the time they get to the CSS file it is a nightmare. "Don't change anything, just add stuff until it works", becomes the default mindset in this chaotic, buggy pile of CSS. At this point changes could have huge consequences that you can't really predict. Not only that, but it can be hard to know *if that CSS is even in use anymore*. So you don't even really know if you can delete the terrible CSS.

Early misguided actions in the codebase start to have a ripple effect. One selector was too specific, and suddenly you're fighting the specificity graph all the way

down. A selector is generic, and all the sudden you're writing lots of CSS to fix the accidental collision everywhere else.

This, in addition to the language's... *eccentricities*... are why lots of really intelligent developers **hate** writing CSS.

It Doesn't Have to Be Terrible

There are a lot of things you can do to make CSS not terrible. In fact, CSS can be *easy* to work in. It can make sense. And writing new CSS can be obvious. There are a myriad of things which I won't cover in this article that can really help. Strategies like BEM, OOCSS, and SMACSS can give concrete rules instead of general guidelines. Preprocessors like Sass, Stylus, PostCSS, and Myth can help make it easier to write and organize CSS. Those tools and strategies were all written by really smart people and I encourage you to check them out and judge whether they would help in your application.

But instead of going over particular tools or organizational strategies, I'm going to instead offer a few guidelines for those developers who hate CSS and just want a few tips to make writing CSS more bearable.

1. Use Low-Specificity Selectors

This one is huge, and will probably get the you the most bang for your buck in terms of reducing future headaches. Several of the other rules are just more specific cases of this key guideline. For an example, it's pretty common to write css like:

```
.navigation a {
  color: blue;
}
```

But this selector is already pretty specific (and also relies on markup, see rule four). It would be much better to write this as:

```
.nav-link {
  color: blue;
}
```

Whenever possible (and it should pretty much always be possible), use *one class* for your selector. Don't nest, don't over-select, don't force future developers into a position where they have to work hard to overcome what you did.

If you're writing something to modify a style, use a single class and just put it after the original class. Like this:

```
.nav-link {
  color: blue;
  font-size: 2rem;
}

.nav-link-red {
  color: red;
}
```

2. Don't Use ID or Element Selectors

Variations on a theme with the guideline above, but try not to use id selectors *at all*. They are pretty specific, hard to modify, and you can't have multiple id's on an element. You also can't have the same id on multiple elements. Just don't use them in your stylesheets.

Also, while styling raw HTML elements is sometimes OK (think default typography, etc), try to avoid it wherever possible. These two rules together would change this:

```
#banner {
  background-color: blue;
}

#banner h1 {
  color: white
}
```

into this:

```
.banner {
  background-color: blue;
}

.banner-title {
  color: white;
}
```

Later, when you need the banner styles on an element without an id of banner, or you need to develop a new banner style, these rules will be easy to extend.

3. Don't Depend on a Certain Markup Structure

Again, this is really something we've already seen, but it is very common in CSS codebases, so it's worth mentioning again. Instead of depending on something being inside something else like this:

```
.header ul li a {
  margin-right: 20px;
}
```

Strive to use a one-class selector:

```
.header-link {
  margin-right: 20px;
}
```

Three reasons to do this:

1. **Markup structures change**. What happens when the above is rewritten as a `nav` element?

2. **You can accidentally select elements you didn't mean to**. What if you add a special promo in an expandable section of the `.header` element and it has a list with links inside. Oh no! Better write more CSS to fix that problem.
3. **It's hard to extend**. I want to override some style on a particular page. With the first way I now have to write a really gross selector. With one class I can also write one class and just depend on the cascade.

4. Don't Use Inline Styles

If you want somebody on your development team to really hate you at some point in the future, you should absolutely use inline styles. Inline styles are *rude*. The only thing in CSS land that can override them is an `!important` rule. You've just escalated things to a specificity brawl *immediately*.

But even more than that, you've selfishly prevented any of your styles from being used elsewhere **and** made the HTML markup look gross. Seriously don't do it.

5. Don't Use !important

For all the reasons above, don't use `!important` unless you have to. The one case in which you might be forced to use important is if some (rude) third party JavaScript library adds inline styles to the DOM. `!important` can override inline styles and that's basically all it should be used for...

6. Prefix Modifier Classes

Now we're getting into the slightly more interesting stuff! This is a *really* common mistake. I have personally made and then subsequently paid the price for it. Essentially, the problem is best described in code for multiple styles of buttons:

```
.btn {
  padding: .25em;
  background-color: blue;
  color: white;
}
```

```
.btn.red {
  background-color: red;
}
```

That's not so bad, right? No, it's not that bad. And for smaller projects you might be OK doing this. But you've just used a very general word inside a stylesheet. Remembering that CSS is global, it's quite possible that down the road somebody else might write some code like this:

```
.top-navigation {
  font-size: 2rem;
  color: blue;
}

.top-navigation .red {
  color: red;
}
```

They mean well, but if you have a red button in `top-navigation`, they've inadvertently made that button red-on-red which will gain you zero style points. Then you're going to have to write more CSS to fix it. Instead, I've found it's best to prefix your modifier classes:

```
.btn {
  padding: .25em;
  background-color: blue;
  color: white;
}

.btn-red {
  background-color: red;
}
```

Now in your HTML, just use both: `<a href="/" class="btn btn-red">`. We've stayed at one level of specificity depth and avoided accidental name collisions from generic names, win!

7. Write Small Rules

In any other language, small, decoupled, easy-to-extend interfaces are expected, so it's a bit strange that in CSS we write the same rules over, and over, and over.

Ladies and gentlemen: the CSS you are about to read is real. Only the class names have been changed to protect the innocent.

```
.role-stat h1,
.role-stat h2,
.role-stat p,
.role-stat a,
.role-su h1,
.role-su h2,
.role-su p,
.role-su a {
  color: #FFF;
}

.features-1 h1,
.features-1 h2,
.features-1 p,
.features-2 h1,
.features-2 h2,
.features-2 p,
.features-3 h1,
.features-3 h2,
.features-3 p {
  color: #FFF;
}

.features-1 a,
.features-2 a,
.features-3 a {
  color: #FFF;
  text-decoration: underline;
}

.feature-4 h1,
.feature-4 h2,
.feature-4 p {
  color: #FFF;
}

.feature-4 a,
.feature-4 a,
.feature-4 a {
  color: #FFF;
  text-decoration: underline;
}
```

This is actually weirdly common in CSS. I see this quite a bit. And you can kind of understand how somebody would get here. There is a pattern, so they follow that pattern on subsequent pages, and pretty soon you end up with fifty lines of CSS to do two things. Not only that, but if you decide to make a title into an `h3` instead of an `h2` you'll break the styles. Also, if you want a link inside `.feature-4` to *not* be white anymore, you have to write more CSS to accomplish that.

You could rewrite this abstracting out the rules into something reusable, obvious, and simple:

```
.text-white {
  color: #FFF;
}

.text-underline {
  text-decoration: underline;
}
```

These classes are **very useful**. Other developers will instantly understand what `text-white` means when they see it on an HTML element. Developers who are creating new pages will just copy your HTML and won't need to write any CSS. This situation will make all parties much happier (especially the users)!

8. Use a Prefix for JavaScript Hooks

This is something I've started doing that has made my life a great deal easier. If you need a way to hook into the DOM from JavaScript I'd recommend using a class that is easily recognizable to future developers, such as `js-`. This communicates to other developers that your class is used by JavaScript and that they shouldn't attach styles to it or remove it without understanding the JS powering it.

If you hook into an existing CSS class that was presentational, and the developer wants to change the presentation, it's hard to know which styles have unintended JS consequences. Just make it as obvious as possible.

Hopefully these guidelines can help you hate CSS a bit less. Remember, it's not CSS's fault, *per se*. Rather, it's just people working fast, and not knowing there might be a better way.

As always, feel free to yell at me on twitter if you disagree about anything, or if anything didn't make sense. Happy styling!