Why CSS is difficult to get right? - Break the CSS Stigma: Don't be afraid of CSS

Written by Seth Corker on Benevolent Bytes

Cascading Style Sheets (CSS) is the language for turning a webpage from a designer's nightmare into something aesthetically pleasing. It has helped craft the vibrant and usable web we know and use every day. As a web developer, I write a lot of CSS along with JavaScript and HTML. These are the core technologies of the web and are the foundations of a web developer. I like CSS, it's not perfect but I've become competent in it and it's easy for everyone to get involved. Not all web developers learn CSS. Many think they don't need to. CSS has a stigma for two reasons. The first is the difficulty of learning and using CSS effectively. Read *The reason developers avoid CSS* to understand why the second problem is systemic, developers don't always respect design. CSS isn't everyone's favourite language, but it's as inescapable as JavaScript for web developers.

The core concept of CSS is to select HTML elements and apply styling to them. Although it's easy to get started, it can become more complex with larger teams or older browsers. If you're a programmer and used to having a lot of flexibility, then CSS will seem like a very constrained sandbox. By looking at some stumbling blocks, we can overcome them and become more comfortable with CSS.

Overloaded Terminology

css snippet showing composition of .header classes to .header.darkmode CSS can be intimidating if you're unfamiliar with the core concepts but it's not that difficult (most of the time). If you're a programmer, the first challenge is understanding overloaded terminology. Class and id do not mean quite the same thing in CSS as they do in JavaScript.

> "I used an ID so it didn't break the site"

A CSS class is a selector that styles everything with that class. Give an HTML a class attribute and you can *select* that class to apply rules that style the element. An element can have multiple classes which will merge rule-sets and cover more individual cases. This is the most flexible and direct methods of styling elements, your elements share a direct relationship with your rules defined in CSS.

Common Pitfalls A common mistake I see developers making when styling elements is giving the element an *id* attribute to style a single element.

An *id* selector gets the job done, you've selected and styled a specific element but it's not a scalable approach. The problem with doing this is you've circumvented a core principle in CSS, the Cascading part. You can only style a single element like this, you're not enjoying any code reuse. If you want to have multiple elements that share the rule-set then a unique *id* needs adding to each element and the rule-set.

Selector specificity

using the > selector in CSS to select child elements Specificity is easy to learn but due to the cascading nature of CSS can become difficult to manage. Inherited properties are a powerful but often maligned feature that gets developers into trouble.

> "The only way is to use !important, should be fine."

The idea is simple, apply a rule-set to a parent HTML element, and the child element will inherit some of those properties for styling. An element is selected based on its parent elements. Child elements inherit properties to avoid defining every property again.

Common Pitfalls The specificity problems I see developers run into is two-fold. Notice I said *some* when referring to the properties a child will inherit from its parent element. Not all properties are inherited and what if you don't want to inherit a property?

One key thing to learn in CSS is what gets inherited. Although not always obvious, but it's important to learn to understand where to add properties and how this visibly affects elements.

An example of an inherited property is `*color*`. A parent with `color: red;` will mean any children will inherit the color property as red unless it's overridden by a more specific rule.

A property that isn't inherited is `background-color`. Any children of a parent with this property set will continue to have a transparent background by default. To inherit the parent's background-color property, create a new rule `background-color: inherit;`.

Another mistake I see inexperienced web developers make is overusing `!important`. This keyword meant to be used in dire situations and is almost always a complete hack. The problem is often solved by embracing CSS and selecting the element by other means. Specificity is a skill that's developed over time once you know that specificity follows the following pattern:

Type > Class > Id *Increases in specificity from left to right*

Reasoning about CSS becomes much easier when you know how specificity works. Over time you'll learn how best to select HTML elements. It will be much easier to avoid using hacks to force a property on an element with `!important`.

Constraints

As far as languages go CSS has many constraints. It's designed with a single goal in mind to adjust the presentation of HTML. Its focus is admirable but these constraints can feel encumbering compared with other languages. The constraints make CSS easy to parse for designers and developers alike but can hinder development.

> "CSS is not a real programming language"

There are common features that CSS lacks which confounds new developers learning CSS. The biggest gripes are no variables, limited ways for code reuse and lack of advanced selectors. This might seem like a big deal but it can be overcome. The key is to understand the symbiotic relationship between CSS, HTML and JavaScript to pick up the slack where one falls down.

Common Pitfalls Complaints I hear from developers usually go like this.

> **"I can't get this to look like the design because I can't select this element!"**

If you can't select an element then ask yourself why? If it's not obvious, then maybe you need to consider changing the HTML. To position elements in CSS is not always intuitive or straightforward. If you can't do what you need with a single element, then look to the humble `<div>` tag. It is often necessary to group elements with `<div>` tags for styling. Don't be afraid to change the structure of your HTML to make it easier to style.

> **"I want to use the same colour for all these elements but I can't use variables!"**

CSS doesn't have variables because CSS prefers sharing properties through inheritance. It's not as flexible but for many common cases, it's preferable to make a class that has the intended style and add that class to all the elements you need. The alternative is to inherit the property from parent elements. The key is to embrace the constraints CSS has and change your way of thinking.

> **"How do I make this button do X in CSS?"**

Some things in CSS just aren't intuitive or possible. There is a lot you can do with CSS but sometimes it needs some help. A common case is changing the look of a button depending on different states. Mostly, CSS does this but the most intuitive and flexible way is to use JavaScript. You can manage the state of the button in JavaScript and update the class depending on the state. This way, you give CSS the ability to do what it does best and style the element appropriately based on the class.

It's vital to embrace the constraints of CSS but know its limitations. Sometimes it's necessary to supplement CSS with one of the other core web technologies. Each technology has a focus and it's better to embrace its strengths to avoid swimming upstream.

Conclusion

CSS gets a lot of flak from developers but it's not going anywhere for the time being. It's an integral part of the web and is essential for breathing life into web pages and web apps. Although it's not always the easiest to wield effectively, it's important to learn and improve. The worst mistake as a web developer is to think CSS is not important and not worth investing time in. CSS has its flaws but it shouldn't be something to battle with. Take the time to:

- Figure out how best to use CSS with HTML to have the best experience.
- Learn the ins and outs of selectors and specificity and avoid common pitfalls such as adding `!important` when it's not needed.
- Embrace the constraints of CSS and recognize the limitations. Delegate unsuited tasks to HTML or JavaScript instead.

There's no need to fear CSS. Get your hands dirty and learn CSS now so you don't dread the next time the designer wants to make some small tweaks.