Shoot to kill; CSS selector intent

csswizardry.com/2012/07/shoot-to-kill-css-selector-intent/

One type of CSS in particular makes me cringe every time I see it; poor selector intent. Poor selector intent means carpet bombed selectors whose key selector has way too broad a reach; a selector like .header ul{} as opposed to one like .main-nav{}; .widget h2{} instead of .widget-title; article > p:first-child{} as opposed to .intro{}. Selectors whose intent isn't specific enough.

It's worth noting that *selector intent* is something I completely made up at some point today; if you think there's a better name then please let me know!

Let's take a closer look at the .header ul{} example. Let's imagine that ul is indeed the main navigation for our website. It lives in the header, as you might expect, and is currently the only ul in there; .header ul{} is fine, right? Not *really*. I mean sure, it might work, but it's not very good. It's not very future proof and certainly not explicit enough. As soon as we add another ul to that header it will adopt the styling of our main nav and the the chances are we won't want it to. This means we either have to refactor a lot of code *or* undo a lot of styling on subsequent uls in that .header to remove the effects of the far reaching selector.

Your selector's intent must match that of your reason for styling something; ask yourself **'am I selecting this because it's a** ul **inside of** .header **or because it is my site's main nav?'**. The answer to this question will determine your selector.

It's all about the key selector…

What determines the impact of a selector is its *key selector*. The key selector is a very important thing in the world of CSS as *browsers read selectors* ***right to left***. This means the key selector is the last one before the opening {, for example:

```
.header ul { }
.ul li a { }
p:last-child { }
```

As I discuss in Writing efficient CSS selectors, the key selector plays a big part in CSS efficiency, so it's worth bearing that in mind, but where *selector intent* is concerned this is basically the place you need to be looking to see how far reaching your selector is. html > body > section.content > article span{} is a ridiculously convoluted and terrible selector that no one anywhere would ever write (right?) but despite how specific and disastrously over the top it is, its key selector (span) is still very, *very* broad. It doesn't matter so much what comes before your key selector, it's only the key that really matters.

As a *general* rule you should try and avoid any key selector that is a type selector (basically an element, like ul or span or whatever) or a base object (e.g. .nav or .media). Just because something is the only .media object in your content area it doesn't mean it always will be.

Let's keep looking at the .header ul{} example. Let's assume our markup is thus, as we're using the nav abstraction:

```
<div class=header>
 <ul class=nav>
 [links]
 </ul>
</div>
```

We could select this in one of several ways:

```
.header ul{
 [main nav styles]
}
```

This is bad because as soon as we add *any* other ul to our header it will look like our main nav. This is dangerous but thankfully easily avoidable.

Secondly we could use:

```
.header .nav{
 [main nav styles]
}
```

This is *marginally* better than the .header ul example, but barely so. We can now safely add another ul without risk, but we can't add anything else with a .nav class; that means adding sub-navs or breadcrumbs will be a nightmare!

Finally, our best solution would be to add a second class to the ul; a class of .main-nav:

```
<div class=header>
 <ul class="nav main-nav">
 [links]
 </ul>
</div>
.main-nav{
 [main nav styles]
}
```

This is good selector intent; we are selecting this element now for *exactly* the right reasons, not coincidental/circumstantial ones. Now we can add as many more uls and .navs to that .header and the scope of our main nav styles will never reach anything else. We're no longer carpet bombing!

Keep your key selector as explicit and specific as you possibly can, preferring for it to be a class over anything else. Applying specific styling through a vague selector is dangerous. A generic selector should always carry generic styling and if you want to target something in particular you should perhaps add a class to it. **Specific intent requires a specific selector.**

Real-life example

A really good example of where I messed up on this myself is on a project I did at Sky; I had a selector which was simply #content table{}. (Eww, I even used an ID!!!) This is a troublesome selector for three reasons; firstly it uses an ID which is a big no, secondly it has a lot higher specificity than it needs to and lastly—and most importantly—it has a poor selector intent. I wasn't wanting to style these tables *because* they were in #content, that was just how the DOM landed so that's how I chose to target them. *Entirely* my bad.

For the first few weeks this was fine but then all of a sudden we needed to add some tables inside #content that didn't want to look anything like the previous ones. Uh oh. My previous selector was far too far reaching, I was now having to undo a blanket style I'd set on *every* table in the #content div. If I'd had a better selector intent then instead of:

```
#content table{
 [foo styles]
}
#content .bar{
 [undoing foo styles]
 [bar styles]
}
```

I should/would have had:

```
.foo{
 [foo styles]
}
.bar{
 [bar styles]
}
```

And a *lot* less headaches. By thinking ahead and having a lot more considered selector intent then I would have had a much easier time...

Exceptions

Of course there are *always* exceptions. It's perfectly reasonable to have selectors like .main-nav > li where your key selector *is* a type selector. It also makes perfect sense to target every a inside something like this:

```
html{
 color:#333;
 background-color:#fff;
}
.promo{
 color:#fff;
 background-color:#333;
}
 .promo a{
 color:#fff;
 text-decoration:underline;
 }
```

That is a reasonably sensible far-reaching selector where it does make sense to style every a in a pretty carpet bombed manner.

Final word

In general, instead of carpet bombing your elements, shoot to kill; target them specifically and explicitly. Make sure your selector intent is accurate and targeted.

Think more carefully about why you want to target something and pick a more explicit and sensible selector; refine your selector intent. Do you mean:

```
.header em{}
```

or do you really mean:

```
.tagline{}
```

Do you want:

```
.footer p{}
```

or do you really want:

```
.copyright{}
```

Is it wise to select:

```
.sidebar form{}
```

or would a safer bet be:

```
.search-form{}
```

Consider your CSS selectors' intent; are you being specific enough? Are your selectors matching things for the right reasons, or is it just happy circumstance? Shoot to kill. Be a CSS sniper, not a CSS carpet bomber.

Incidentally, opting to switch out a longer selector like `.header ul` for something like `.main-nav` will also help reduce specificity and increase selector efficiency; win-win-win!

It is also worth noting that Jonathan Snook wrote something similar called the *depth of applicability*...

Did you enjoy this? **Hire me!**

Hi there, I'm Harry. I am a **Consultant Front-end Architect**, **designer**, **developer**, **writer** and **speaker** from the UK; I am **available** for work. I write, tweet, speak and share code about authoring and scaling CSS for big websites.

via Ad Packs