VIDEO TRANSCRIPT

# CSS Specificity Hierarchy

Hello and welcome to the lesson about the CSS specificity hierarchy. In the previous lessons, I've explained how you can define your CSS selectors. But in order to make sure our CSS works and to be able to easily define correct targets, we need to understand the CSS hierarchy. aka what trumps what.

## Selectors order

So obviously, first thing is that the order of selectors in CSS does play a role, and the further down one does in fact win when the specificity values are exactly the same, and that includes code written in one place within your theme options, for example. What is simply further down takes precedence. But what is also important is the order of the CSS sources in the HTML document if the CCS code in a separate file. In this simple example, the color of the heading will be blue. If two exactly same CSS selectors use different property values, the last one wins. Last one in the page source in the HTML structure will take precedence. So how does it work in Divi exactly? The Divi team style CSS file is loaded quite early in the head section, and the CSS you add in the theme options is added in the page later on, meaning that the theme option CSS will always trump the Divi style sheet CSS, and the same applies to CSS added in the Divi builder settings. But what if you cannot control the order of the CSS on your page? Let's say you have a plugin that loads its assets in a footer, long after the theme options custom CSS. You won't be able to overwrite some CSS used by this plugin if you use the

same exact selector.

## How the CSS Specificity works?

And that is when you need to know how the CSS specificity works. The best way to explain is to start with an example of where specificity gets confusing and perhaps doesn't behave like you would expect. So if you look here, I change our HTML structure a bit, and here is a sample unordered list, the ul with ID animals, and we've listed some animals. So let's say we want to mark one of these animals as our favorite with a custom class, and we want to change the styling. So let's say class: favorite. And now to add some styling for our new class. So I've added the CSS and it work. The selector is correct. We are using favorite CSS class, and that's the same class that is added to this list item, and that happens a lot. So we start to search through our code or we inspect the element in the browser and we find this bit of code ul ID animals li, which targets list items inside of ul with an ID of animals. So there is the trouble right there. Two different CSS selectors are telling that text what color and font weight to be. There is only one statement for font size, so clearly that one will take effect. And these aren't what we would call conflict, but the browser does need to decide which one of these States went to honor.

## Set of Specificity rules

And it does that by following a set of specificity rules. And the point here is that you want to be as specific as it makes sense to be every chance you get when you start adding your own CSS. But even with this simple example, it should be obvious from the start that simply using a class name to target that favorite animal isn't going to work, or even if it did work, it won't be very safe. The smart approach would be to use this

instead. That is being as specific as it makes sense to be. You could actually be way more specific and use something like HTML, body, ID, main content, ul and all that, but that is over the top. It makes your CSS harder to read and gives you no real benefits. Another approach would be to use an important statement. So if we simply add an important here, that would also work. But it is not recommended to use important for simple things like that. So if only there is a way to avoid it, you should. But if the element you are trying to overwrite already uses the important statement, you need to use it. So going back to the example, why is that our first attempt at changing the color and font weight failed? As we learned it was because simply using the class name by itself had a lower specificity value and was trumped by the other selector, which targeted the unordered list with the ID value and important words here are class and ID.

## Calculating CSS Specificity Values

CSS applies very different specificity weights to classes and IDs. In fact, an ID has infinitely more specificity value, meaning no amount of CSS classes alone can outweigh an ID. So let's take a look at how the numbers are actually calculated. If the element has inline styling, that would be 1.0.0.0 points. Inline styling is the style that is added right there in the HTML like this. That's the inline style, and it has 1.0.0.0 points. And for each ID value we would apply 0.1.0.0 points. And for each class value or pseudo class or attribute selector, we would apply 0.0.1.0 points, and for each element reference apply 0.0.0.1 point. You can generally read the values as if they were just a number like 1-0-0-0 is 1,000, and so clearly wins over a specificity of 0-1-0-0 or 100. But the commas are there to remind us that this isn't really a Base 10 system is that you could technically have a specificity value of 0 1 12 4 and at 12

doesn't spill over like a Base 10 system would. And the important value added to a CSS property is an automatic win it overwrites even inline styles from the markup. The only way an important value could be overwritten is with another important declared later in the CSS, and with equal or greater specificity value. And you can think of it as adding 1-0-0 to the specificity value. So once you understand the calculation, it becomes much easier to define your selectors. So let's look at some examples. Here we have an inline style attribute, so the specificity value is 1.0.0.0. And here we are targeting an A element that has two parents specified. We have three elements, 1 class and one ID. So it's 1-1-3 for specificity value. And this is a bit more complex example. We are targeting a pseudo element, and we are defining four parent elements, and we have specified one class, one pseudo element, and two IDs. So the specificity value is 2-2-4.

Hopefully, understanding that concept will make you creating your targets. Defining your selectors much easier.