CSS Specificity Hierarchy

Introduction

Welcome to the lesson about the CSS specificity hierarchy. In the previous lessons, I 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, the first thing to remember is that the order of selectors in CSS does play a role. The one further down 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 is in a separate file. In this simple example, the color of the heading will be

blue. If two exactly identical CSS selectors use different property values, the last one wins. The last one in the page source in the HTML structure will take precedence.

```
h2 {color: red;}
```

```
h2 {color: blue;}
```

Hello!

Welcome to my website

- Dog
- Cat
- Parrot

My links

- Page 1 - link 1
- Page 2 - link 2
- Page 3 - link 3
- Page 4 - link 4

So how does it work in Divi exactly?

The Divi Theme's 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 Options CSS will always trump the Divi style sheet CSS. The same applies to CSS added in the Divi Builder settings.

```
}
</style>
    <link rel='stylesheet' id='dashicons-css'  href='http://localhost:10058/wp-includes/css/dashicons.min.css?ver=5.7.
<link rel='stylesheet' id='admin-bar-css'  href='http://localhost:10058/wp-includes/css/admin-bar.min.css?ver=5.7.1' t
<link rel='stylesheet' id='wp-block-library-css'  href='http://localhost:10058/wp-includes/css/dist/block-library/styl
<link rel='stylesheet' id='dsa-styles-css'  href='http://localhost:10058/wp-content/plugins/dsa-functionality/assets/s
<link rel='stylesheet' id='divi-style-css'  href='http://localhost:10058/wp-content/themes/Divi/style.css?ver=4.9.4' t
<link rel='stylesheet' id='dsa-style-css'  href='http://localhost:10058/wp-content/themes/dsa-child/style.css?ver=1.0'
<link rel='stylesheet' id='divi-fonts-css'  href='http://fonts.googleapis.com/css?family=Open+Sans:300italic,400italic
<link rel='stylesheet' id='et-builder-googlefonts-cached-css'  href='http://fonts.googleapis.com/css?family=Montserrat
<script type='text/javascript' src='http://localhost:10058/wp-includes/js/jquery/jquery.min.js?ver=3.5.1' id='jquery-c
<script type='text/javascript' src='http://localhost:10058/wp-includes/js/jquery/jquery-migrate.min.js?ver=3.3.2' id='
<script type='text/javascript' src='http://localhost:10058/wp-content/plugins/dsa-functionality/assets/scripts.js?ver=
<link rel="https://api.w.org/" href="http://localhost:10058/wp-json/" /><link rel="alternate" type="application/json"
<link rel="wlwmanifest" type="application/wlwmanifest+xml" href="http://localhost:10058/wp-includes/wlwmanifest.xml" />
<meta name="generator" content="WordPress 5.7.1" />
<link rel="canonical" href="http://localhost:10058/" />
<link rel='shortlink' href='http://localhost:10058/' />
<link rel="alternate" type="application/json+oembed" href="http://localhost:10058/wp-json/oembed/1.0/embed?url=http%3A
<link rel="alternate" type="text/xml+oembed" href="http://localhost:10058/wp-json/oembed/1.0/embed?url=http%3A%2F%2Flo
<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=0" /><link rel=
    <style type="text/css" media="screen">
    html { margin-top: 32px !important; }
    * html body { margin-top: 32px !important; }
    @media screen and ( max-width: 782px ) {
        html { margin-top: 46px !important; }
        * html body { margin-top: 46px !important; }
    }
</style>
```

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 the 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 as one used by it.

How does CSS specificity work?

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 changed our HTML structure a bit, and here is a sample unordered list, the ul, with ID animals, and we've listed some animals.

```
<body>
<div id="main-content">

    <div class="content sticky">

        <h1>Hello!</h1>

        <p class="subtitle">Welcome to my website</p>

        <ul id="animals">
            <li>Dog</li>
            <li>Cat</li>
            <li>Parrot</li>
        </ul>
    </div>

    <footer class="content">
        <h2>My links</h2>
        <ul>
            <li><a href="index.html">Page 1</a> - link 1</li>
            <li><a href="page2.html">Page 2</a> - link 2</li>
            <li><a href="page3.html">Page 3</a> - link 3</li>
            <li><a href="page4.html">Page 4</a> - link 4</li>
        </ul>
    </footer>

</div>
```

<body>

<div class=”content sticky”>
<h1>Hello!</h1>
Welcome to my website

```
<ul id=”animals”>
<li>Dog</li>
<li>Cat</li>
<li>Parrot</li>
</ul>
</div>
```

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.

```
<li class=”favourite”>Parrot</li>
```

So let's say class “favourite”. And now to add some styling for our new class.

```
.favourite {color: red; font-weight: bold;}
```

So I've added the CSS and it works. The selector is correct. We are using the “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#animals li {font-size: 18px; color:#000; font-weight: normal;}
```

It targets list items inside of the ul with the ID “animals”. That’s the problem 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 this isn’t what we would call a conflict, but the browser still needs to decide which one of these statements 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:

```
ul#animals li.favourite {color: red; font-weight: bold;}
```

This is being as specific as it makes sense to be.

You could actually be way more specific and use something like:

```
html body #main-content ul#animals li.favourite {color: red; font-weight: bold;}
```

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 to change the color to red:

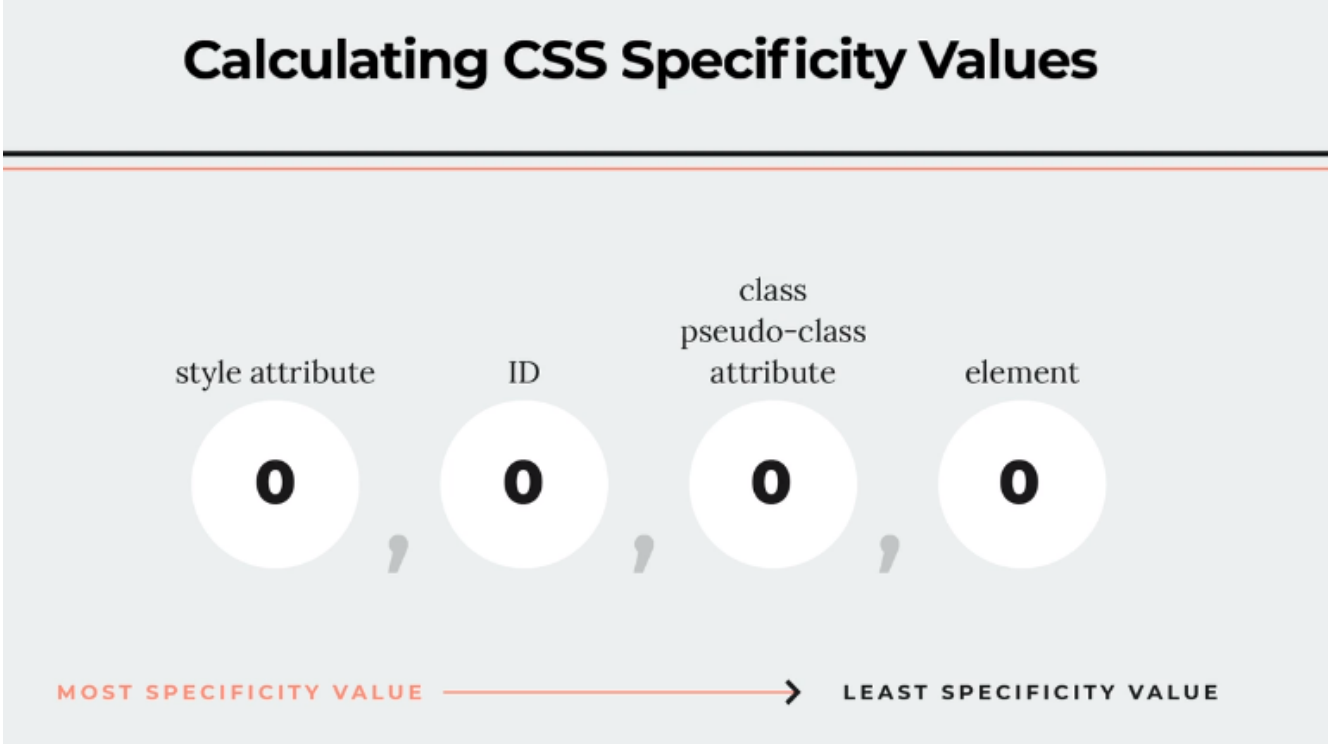
```
.favourite {color: red!important; font-weight: bold;}
```

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 it 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 the 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:

```
class="content sticky">

<h1>Hello!</h1>

<p class="subtitle">Welcome to my website</p>

<ul id="animals">
    <li>Dog</li>
    <li>Cat</li>
    <li class="favourite" style="color:green;">Parrot</li>
</ul>
```

Welcome to my

- Dog
- Cat
- Parrot

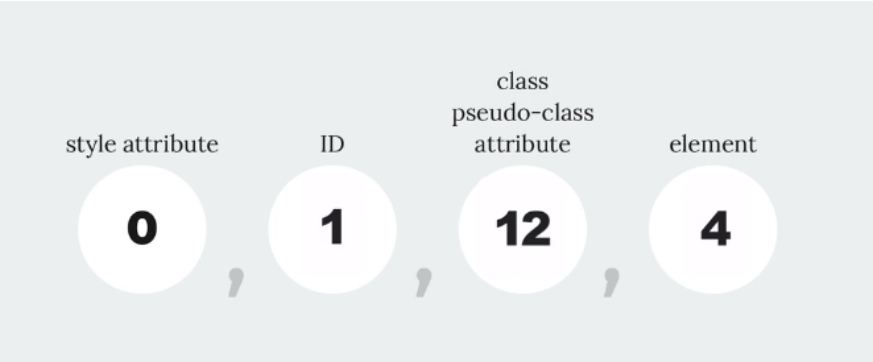
My links

<li class="favourite" style="color:green;">Parrot

That's the inline style, and it has 1,0,0,0 points.

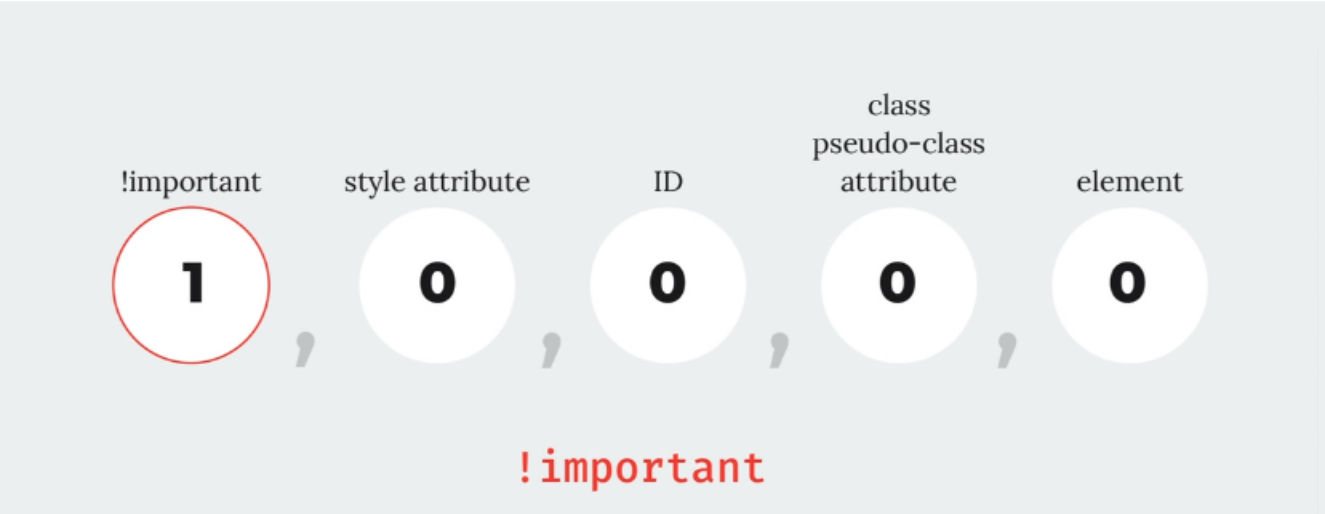
And for each ID value we would apply 0,1,0,0 points. And for each class value, or pseudo-class, or attribute selector, we would apply 0,0,1,0 points, and for each element reference we apply 0,0,0,1 point.

You can generally read the values as if they were just a number like 1,0,0,0 is 1000, 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, that is you could technically have a specificity value of 0,1,12,4, and the 12 doesn't spill over like a Base 10 system would.



Using the !important value

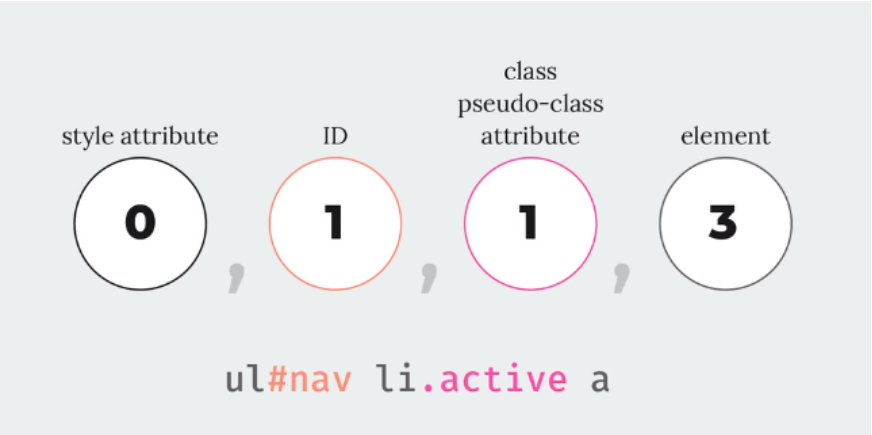
And the !important value added to a CSS property is an automatic win. It overwrites even inline styles from the markup. The only way the !important value could be overwritten is with another important declared later in the CSS, and with equal or greater specificity value. And you can think of it as adding 1,0,0,0,0 to the specificity value.



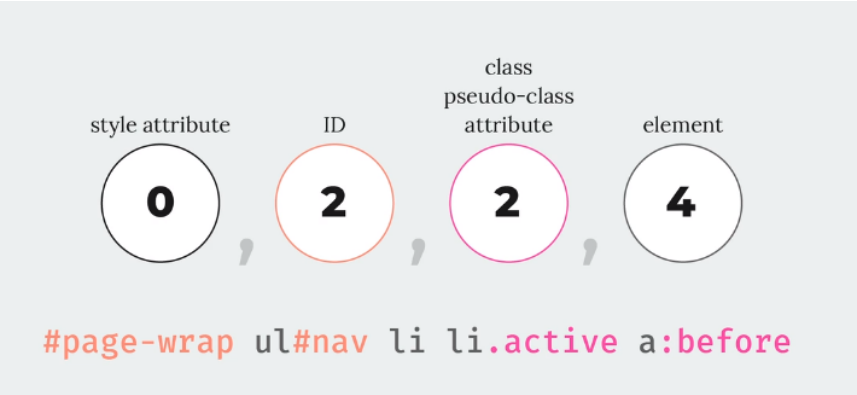
So once you understand the calculation, it becomes much easier to define your selectors. So let's look at some examples.

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 defining your selectors much easier.

Resources

DIG DEEPER:

Selectors Explained

CSS Specificity Guide (available in the Downloads section of this lesson)

Action Items

- ☐ Pick a selector from your stylesheet or by inspecting a page. Calculate its specificity value and use the Selectors Explained tool to check if your calculations are correct. Do this for as many selectors as you can to get familiar with the concept.
- ☐ Download and print the CSS Specificity Guide.