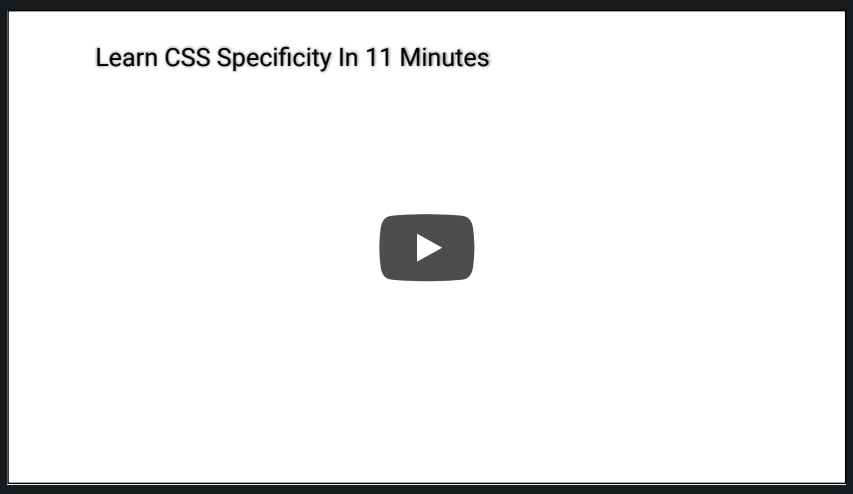
CSS Specificity Demystified

February 3, 2020

CSS

Creating amazing looking websites with CSS is a ton of fun, but as the CSS grows it becomes harder and harder to properly overwrite styles in the cascade. The CSS starts filling up with `!important` declarations and the cascade is nearly impossible to follow. This is a common problem that I have run into before and I am sure you have as well, so in this article I am going to explain CSS specificity in depth so you can start building well structured, easy to maintain CSS stylesheets.

If you prefer to learn visually, check out the video version of this article.



PS: There is an interactive specificity calculator at the bottom of the page you can checkout here.

What Is Specificity?

Specificity is the process that CSS uses to determine which styles overwrite other styles. This can be as simple as a selector's position in a stylesheet, but is often much more complex than that. The best way to think of specificity is as an array of four numbers per selector. These four numbers represent the four ways that CSS determines specificity.

!important

The first number in this array is the `!important` keyword. The `!important` keyword is used to override all styles no matter their specificity or position in the stylesheet. This is why `!important` is the first number in the array and the first thing CSS checks for specificity. Because of the power to overwrite all styles the `!important` keyword should be used very sparingly if ever at all since the `!important` keyword is the easiest way to ruin the cascade of a stylesheet and make maintaining a stylesheet impossible. In the below example the blue color will override the green color because it is declared with `!important`.

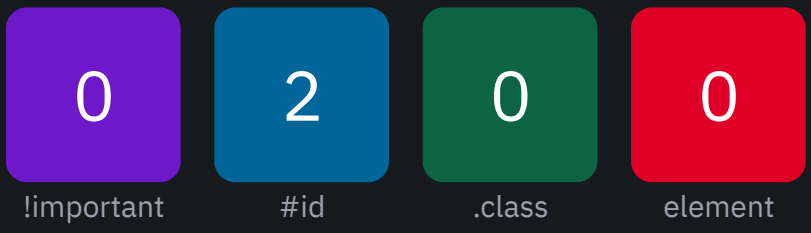
```
h1 {
  color: blue !important;
}

h1 {
  color: green;
}
```

#id

If the specificity of the `!important` keyword is the same between two selectors (either they both are `!important` or neither is `!important`) then CSS looks at the count of the id selectors to determine the specificity of an element. This is simply done by adding up the number of ids in a selector and using that as the number for the second position in the specificity array. For example the below selector would have an id specificity of two.

```
#id1 #id2 { ... }
```



This selector would only have an id count of one, though.

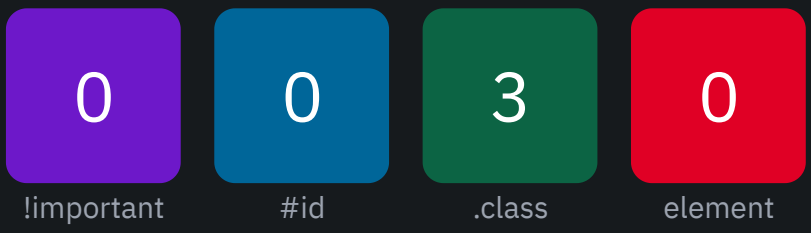
```
#id1 { ... }
```

If a selector has more ids in the selector then it will be more specific and override the less specific selectors with less ids.

.class

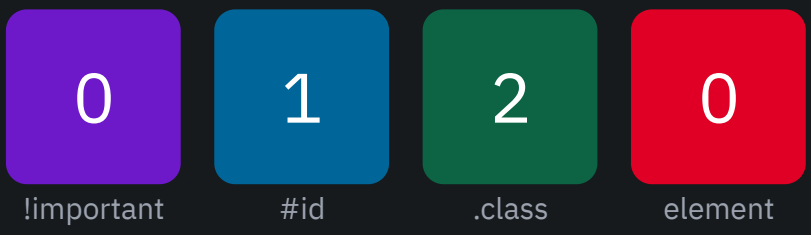
If the selectors being compared have the same `!important` specificity and they both have the same number of id selectors then the class selector specificity is used. This class specificity includes the count of all class, attribute, and pseudo-class's in the selector and fills the third number in the array. For example the following selector would have a class specificity of three.

```
.class1 .class2:hover { ... }
```



This selector would have a class specificity of two and an id specificity of one.

```
#form .textbox[type="text"] { ... }
```



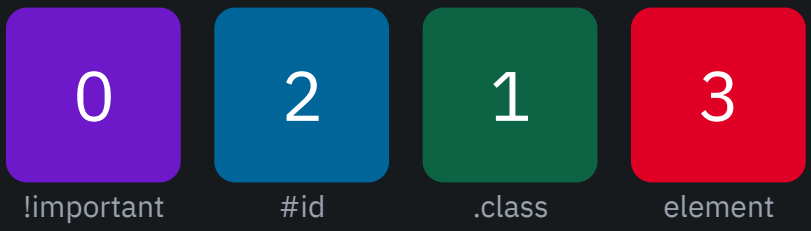
element

If after checking `!important`, ids, and classes, the specificity is still the same then the count of all element selectors is used. This count includes all HTML element selectors as well as all pseudo selectors such as `::before`. The count of all these elements goes into the fourth and final number in the array. For example the following selector would have an element specificity of one.

```
h1 { ... }
```

This selector would have an id count of two, a class count of one, and an element count of three.

```
#page1 header#main-header h1.title::before { ... }
```



Comparing Selectors

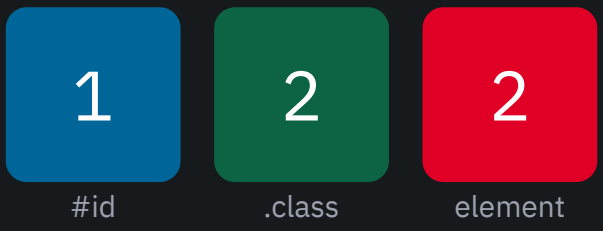
In order to compare selectors CSS uses this four number array and compares from left to right. This means that CSS first checks the `!important` keyword. If any of the CSS properties have the `!important` keyword then they are given a one in this array slot. CSS then picks all selectors that have the highest number in this slot (anything that has `!important` or all selectors if none are `!important`). If there is only one selector at this point then that is the selector that is used.

Next, CSS checks the second number in the array which is the id count. The same check is done where CSS chooses the selectors that have the most amount of id selectors or all selectors if none have any id selectors. Again if there is only one selector after checking ids then that is the selector that is used. This process is then repeated again for the next number in the array, class count, and finally again for the final number in the array, element count.

If at this point there are still more than one selector with the same specificity then the selector that is defined last in the stylesheet is the one that is used.

Calculator

Here is an example of a selector with its specificity listed (`!important` is left out since that is property specific and not selector specific). **You can modify the selector to play around with the specificity.**




```
#form input.custom-checkbox[type='checkbox']::before
```

1. #form

1. .custom-checkbox
2. [type='checkbox']

1. input
2. ::before

Inline Styles

`!important`.

Conclusion

CSS specificity is a difficult and complex topic, but if you understand the simple array method of calculating specificity it becomes a lot easier to understand and use.