The Cascade

Foundations Course

Introduction

In the previous lesson, we covered basic CSS syntax and selectors. Now, it's time to combine our knowledge of selectors with the *C* of CSS – the cascade.

Lesson overview

This section contains a general overview of topics that you will learn in this lesson.

- What the cascade does.
- Specificity and combining CSS selectors.
- How inheritance affects certain properties.

The cascade of CSS

Sometimes we may have rules that conflict with one another, and we end up with some unexpected results. "But I wanted *these* paragraphs to be blue, why are they red like these other paragraphs?!" As frustrating as this can be, it's important to understand that CSS doesn't just *do* things against our wishes. CSS only does what we tell it to do. One exception to this is the default styles that are provided by a browser. These default styles vary from browser to browser, and they are why some elements create a large "gap" between themselves and other elements, or why buttons look the way they do, despite us not writing any CSS rules to style them that way.

So if you end up with some unexpected behavior like this it's either because of these default styles, not understanding how a property works, or not understanding this little thing called the cascade.

The cascade is what determines which rules actually get applied to our HTML. There are different factors that the cascade uses to determine this. We will examine three of these factors, which will hopefully help you avoid those frustrating "I hate CSS" moments.

Specificity

A CSS declaration that is more specific will take precedence over less specific ones. Inline styles, which we went over in the previous lesson, have the highest specificity compared to selectors, while each type of selector has its own specificity level that contributes to how specific a declaration is. Other selectors contribute to specificity, but we're focusing only on the ones we've gone over so far:

1. ID selectors (most specific selector)
2. Class selectors
3. Type selectors

Specificity will only be taken into account when an element has multiple, conflicting declarations targeting it, sort of like a tie-breaker. An ID selector will always beat any number of class selectors, a class selector will always beat any number of type selectors, and a type selector will always beat any number of less specific selectors. When there is no declaration with a selector of higher specificity, a rule with a greater number of selectors of the same type will take precedence over another rule with fewer selectors of the same type.

Let's take a look at a few quick examples to visualize how specificity works. Consider the following HTML and CSS code:

```html
<!-- index.html -->

<div class="main">
  <div class="list subsection">Red text</div>
</div>
```

```css
/* rule 1 */
.subsection {
  color: blue;
}

/* rule 2 */
.main .list {
  color: red;
}
```

In the example above, both rules are using only class selectors, but rule 2 is more specific because it is using more class selectors, so the `color: red` declaration would take precedence.

Now, let's change things a little bit:

```html
<!-- index.html -->

<div class="main">
  <div class="list" id="subsection">Blue text</div>
</div>
```

```css
/* rule 1 */
#subsection {
  color: blue;
}

/* rule 2 */
.main .list {
  color: red;
}
```

In the example above, despite rule 2 having more class selectors than ID selectors, rule 1 is more specific because ID beats class. In this case, the `color: blue` declaration would take precedence.

And a bit more complex:

```html
<!-- index.html -->

<div class="main">
  <div class="list" id="subsection">Red text on yellow background</c
</div>
```

```css
/* rule 1 */
#subsection {
  background-color: yellow;
  color: blue;
}

/* rule 2 */
.main #subsection {
 color: red;
}
```

In this final example, the first rule uses an ID selector, while the second rule combines an ID selector with a class selector. Therefore, neither rule is using a more specific selector than the other. The cascade then checks the number of each selector type. Both rules have only one ID selector, but rule 2 has a class selector in addition to the ID selector, so rule 2 has a higher specificity!

While the `color: red` declaration would take precedence, the `background-color: yellow` declaration would still be applied since there's no conflicting declaration for it.

> **Not everything adds to specificity**
>
> When comparing selectors, you may come across special symbols for the universal selector (`*`) as well as combinators (`+`, `~`, `>`, and an empty space). These symbols do not add any specificity in and of themselves.

```css
/* rule 1 */
.class.second-class {
  font-size: 12px;
}

/* rule 2 */
.class .second-class {
  font-size: 24px;
}
```

Here both rule 1 and rule 2 have the same specificity. Rule 1 uses a chaining selector (no space) and rule 2 uses a descendant combinator (the empty space). But both rules have two classes and the combinator symbol itself does not add to the specificity.

```css
/* rule 1 */
.class.second-class {
  font-size: 12px;
}

/* rule 2 */
.class > .second-class {
  font-size: 24px;
}
```

This example shows the same thing. Even though rule 2 is using a child combinator (`>`), this does not change the specificity value. Both rules still have two classes so they have the same specificity values.

```css
/* rule 1 */
* {
  color: black;
}

/* rule 2 */
h1 {
  color: orange;
}
```

In this example, rule 2 would have higher specificity and the `orange` value would take precedence for this element. Rule 2 uses a type selector, which has the lowest specificity value. But rule 1 uses the universal selector (`*`) which has no specificity value.

Inheritance

Inheritance refers to certain CSS properties that, when applied to an element, are inherited by that element's descendants, even if we don't explicitly write a rule for those descendants. Typography-based properties (`color`, `font-size`, `font-family`, etc.) are usually inherited, while most other properties aren't. You can find out if a property is inherited or not by going to its docs on MDN and heading to the **Formal Definition** section. For example, the [CSS `color` property formal definition] indicates that `color` is an inherited property, while the [`display` property formal definition] indicates that `display` is not.

The exception to this is when directly targeting an element, as this always beats inheritance:

```html
<!-- index.html -->

<div id="parent">
  <div class="child"></div>
</div>
```

```css
/* styles.css */

#parent {
  color: red;
}

.child {
  color: blue;
}
```

Despite the `parent` element having a higher specificity with an ID, the `child` element would have the `color: blue` style applied since that declaration directly targets it, while `color: red` from the parent is only inherited.

Rule order

The final factor, the end of the line, the tie-breaker of the tie-breakers. Let's say that after every other factor has been taken into account, there are still multiple conflicting rules targeting an element. How does the cascade determine which rule to apply?

Whichever rule was the *last* defined is the winner.

```css
/* styles.css */

.alert {
  color: red;
}

.warning {
  color: yellow;
}
```

For an element that has both the `alert` and `warning` classes, the cascade would run through every other factor, including inheritance (none here) and specificity (neither rule is more specific than the other). Since the `.warning` rule was the last one defined, and no other factor was able to determine which rule to apply, it's the one that gets applied to the element.

Assignment

1. Complete the exercise in our CSS exercises repository's `foundations/cascade` directory (remember that the instructions are in the README):
   - `01-cascade-fix`

   Note: Solutions for these exercises can be found in the `solution` folder of each exercise.

Knowledge check

The following questions are an opportunity to reflect on key topics in this lesson. If you can't answer a question, click on it to review the material, but keep in mind you are not expected to memorize or master this knowledge.

- Between a rule that uses one class selector and a rule that uses three type selectors, which rule has the higher specificity?

Additional resources

This section contains helpful links to related content. It isn't required, so consider it supplemental.

- The CSS Cascade is a great, interactive read that goes a little more in detail about other factors that affect what CSS rules actually end up being applied.
- CSS Specificity Explained from Kevin Powell goes through various specificity examples and gives some advice on avoiding wrestling with specificity.
- CSS Specificity Calculator allows you to fill in your own selectors and have their specificity calculated and visualized.
- Interactive Scrim on the CSS Cascade.