CSS Cheat Sheet: Inheritance, Cascade, Specificity

By: Zoe Gillenwater

Even seasoned CSS developers need a quick refresher course in CSS concepts and techniques from time to time. This CSS Cheat Sheet is designed for just that need. Use it as a reference for topics that you've already learned about in-depth but need a few reminders on. If you're still a beginner to CSS, use it to learn the nuts and bolts of working with CSS, then use our other articles, listed at the end of this one, to extend your learning and practice your new skills.

This Cheat Sheet reviews how the fundamental CSS concepts of inheritance, cascading and specificity work.

Inheritance

Inheritance is the process by which the value of certain CSS properties applied to an element get passed down to all elements nested within it. Inheritance relies on the **document tree**, which is the hierarchy of (X)HTML elements in a page based on nesting. Descendant elements may inherit CSS property values from any ancestor elements enclosing them.

In general, text-related properties are inherited by descendant elements, and box-related properties are not inherited. The *complete* list of CSS 2.1 properties indicating which inherit and which do not is found in the W3C CSS 2.1 specification (http://www.w3.org/TR/CSS21/propidx.html) . Here is a summary of that information:

Properties that inherit:

- `color`
- `font` (and related properties)
- `letter-spacing`
- `line-height`
- `list-style` (and related properties)
- `text-align`
- `text-indent`
- `text-transform`
- `visibility`
- `white-space`
- `word-spacing`

Properties that don't inherit:

- `background` (and related properties)
- `border` (and related properties)
- `display`
- `float` and `clear`
- `height` and `width`
- `margin` (and related properties)
- `min-` and `max-height` and `-width`
- `outline`
- `overflow`
- `padding` (and related properties)
- `position` (and related properties)
- `text-decoration`
- `vertical-align`
- `z-index`

Inheritance prevents certain properties from having to be declared over and over again in a style sheet, allowing the developers to write less CSS, the users to view faster-loading pages, and the clients to save money on bandwidth and development costs.

The Cascade

The **cascade** is the system that manages styles from multiple sources and determines what CSS rules should take precedence when there's a conflict.

Styles for a single element can come from multiple sources:

- **Browser style sheets** are built into each browser and set the default presentation for (X)HTML elements.
- **User style sheets** can be set by each individual web user to control the presentation of all or certain web pages.
- **Author style sheets** are created by you to alter the browser defaults and user styles.

In addition to these three sources of styles, remember that each page can have multiple author style sheets that you've applied to the page, either **external**, **embedded**, or **inline**. Plus, each author style sheet and user style sheet can contain multiple declarations that apply to the same element, either through inheritance or simply by the repetition of rules. Finally, declarations can be appended with `!important` to override styles that don't have this extra weight set.

If there is no conflict between any of these rules from any of these sources, they are all applied. When there is conflict, the cascade lets these multiple styles "cascade down" to each element using the sorting order illustrated in Image 1. In this graphic, styles at each level lower down always take precedence over styles at higher levels.

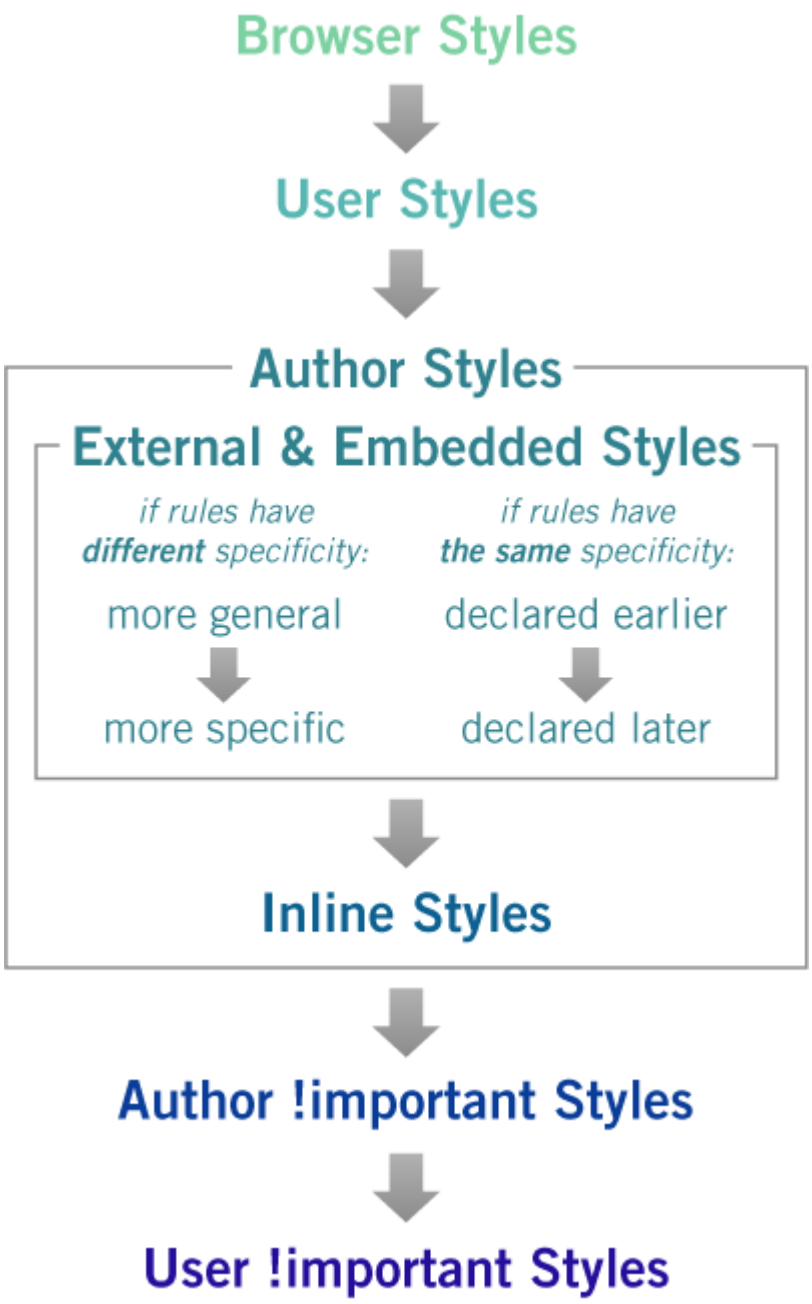

Image 1: *The cascade*

A printable PDF version of the above graphic named **cascade.pdf** is included in your download for this article.

Image 1 shows that the cascade is dependent on the style's **origin** (browser, user, external, embedded, inline), **weight** (normal or `!important`), **specificity**, and **order**. Origin and weight are straightforward, but specificity and order require additional explanation.

Specificity

As shown in the illustration of the cascade, **specificity** only comes into play when comparing rules within external and embedded author styles. Specificity is the amount of importance each rule has in comparison with others based on what its selector is made up of.

Specificity is calculated by counting the number of ids, classes, and element names (commonly called

tags) in each selector and creating a number in the form of *a-b-c*. Note that this is *not* the same as a three digit number, as a value of 0-15-11 is smaller than a value of 1-0-0. However, a much simpler way to calculate specificity is to ignore the number scheme and follow these steps:

1. Count the number of ids in the selectors being compared. The one with more ids wins.
2. If they have the same number of ids, the one with more classes and pseudo-classes wins.
3. If they have the same number of classes, the one with more element names and pseudo-elements wins.

This means that even if you have a selector with dozens of classes in it, for example, a selector with a single id in it would still be more specific.

Order

If selectors within external and embedded style sheets conflict but have the same specificity, the final tie-breaker is based on the order of apperance of the rules: the rule declared later wins. This applies not only to the order of rules within a single sheet, but also to the order that the sheets are linked, imported or embedded in the head of the (X)HTML page.

If you have a sheet that is imported into another sheet, all the rules in the imported sheet are considered to be declared earlier than the rules in the sheet that contains the `@import` directive. As the `@import` directive must be written before any of the actual rules in the sheet (or style element in the head), this is consistent with the concept that later rules override earlier ones.

This means that if you have three style sheets, two linked in the head of the page and one imported into the second linked sheet, the rules in the first linked sheet will be overridden by the rules in the imported sheet, which in turn will be overridden by the rules in the second linked sheet. The rules at the bottom of the each sheet override the rules written farther up in the sheet. All of this, of course, is entirely dependent on the rules having the same specificity. Order is irrelevant otherwise.

The Cascade's Effect on Inheritance

The illustration of the cascade in Image 1 does not address what happens when conflicts between styles occur because of inheritance. There are a couple rules of the cascade that deal with what to do when inherited styles conflict, either with each other or with non-inherited, directly applied styles.

The nearest ancestor takes precedence

If you have two ancestors of an element trying to apply different values for the same property to their descendant using inheritance, the one that is closest to the descendant in the document tree wins. For example, if you set a different `font-family` on both the `body` and **`p`** elements, and you had a `strong` element nested inside a paragraph, that `strong` element would inherit the `font-family` from the **`p`** element over the one from the `body` element, because it is closer to the `p` element than the `body` element in the source.

At first glance this behavior may seem very obvious, but note that it holds true regardless of the order or specificity of the rules. So, continuing the same example, if the `body` rule's selector was amended to `body#home` so that it was more specific than the `p` rule, the `strong` text would still inherit the color

from the `p` rule despite its weaker specificity.

Directly applied styles take precedence over inherited ones

If you have a rule with properties that could be inherited by a descendant element, but the descendant element has its own explicit rule setting those properties, the explicit rule for the descendant element takes precedence even if all the other rules of the cascade say it shouldn't. For example, if you have an inline style (which Image 1 shows holds a lot of precedence in the cascade) on a paragraph setting the text color to gray, but also have a rule in an external style sheet setting the text color of a nested `strong` element to red, the `strong` element will be red and will not inherit the text color from the inline rule on its ancestor paragraph.

Additional Resources

The following resources provide more in-depth information about inheritance, the cascade and specificity, including practical applications of these concepts.

CMX Articles:

- Understanding the Cascade: Part One (http://www.communitymx.com/abstract.cfm?cid=A5A3E5C65CA5584B)
- Understanding the Cascade: Part Two - Specificity (http://www.communitymx.com/abstract.cfm?cid=2E5D2)
- Writing Efficient CSS (http://www.communitymx.com/abstract.cfm?cid=90F55)
- CSS An Introduction - Part Four: Type Selectors and Grouping (http://www.communitymx.com/abstract.cfm?cid=AF51B) (example of overriding a rule using the order property of the cascade)
- Common Coding Problems with HTML and CSS - Part One (http://www.communitymx.com/abstract.cfm?cid=67EEA) (the proper order to write link pseudo-classes due to the cascade)
- (Not So) Common Coding Problems with HTML and CSS — Part Six (http://www.communitymx.com/abstract.cfm?cid=A37EE) (browser bugs with specificity)
- The CSS Styles Panel in Dreamweaver 8: A Sneak Peek (http://www.communitymx.com/abstract.cfm?cid=81B54) (how to use Dreamweaver to view cascade information)
- Using Descendant Selectors to Create a Site Map (http://www.communitymx.com/abstract.cfm?cid=8572A) (example of the cascade and specificity at work)

External Resources:

- W3C CSS 2.1 specification: Inheritance (http://www.w3.org/TR/CSS21/cascade.html#inheritance)
- W3C CSS 2.1 specification: The cascade (http://www.w3.org/TR/CSS21/cascade.html#cascade)
- W3C CSS 2.1 specification: Calculating a selector's specificity (http://www.w3.org/TR/CSS21/cascade.html#specificity)

Keywords

CSS, cascading style sheets, styles, styling, inheritance, document tree, descendants, descendents, ancestors, nested elements, nesting, cascade, specificity, user styles, author styles, browser styles, overriding, external style sheets, embedded styles, inline styles, precedence, weight, origin, order,

!important, selectors