




CSS

(Cascading Style Sheets)




Specificity, Inheritance, and
the Cascade





Have you ever run into a situation where you're trying to apply a CSS style to an element, but it won't take?



Your page seems to be ignoring your CSS, but you are not sure why?





Three things control which CSS rule applies to a given HTML element:

- 
- **Specificity Calculations**
 - **Inheritance**
 - **the Cascade**
- 

Specificity


Process used to determine which rules take precedence in CSS.
Several rules could be applied to the same element in markup.
Every selector has its specificity.
If two selectors apply to the same element, the one with higher specificity wins.



Specificity hierarchy



Every selector has a place in the specificity hierarchy. There are four categories to define it:





1. Inline styles:



Lives within your HTML document

E.g. `<h1 style="color: #FF0000;"></h1>`






2. IDs:

(# of ID selectors)



ID is an identifier for your page elements.

E.g. `<div id="tree"></div>`



3. Classes, attributes and pseudo-classes

This group includes `.class`,
[attributes] and pseudo-classes such
as `:hover`, `:focus`, etc.



4. Elements and pseudo-elements



Type selectors, including for instance `:before` and `:after`






How to Calculate Specificity



There are several ways to calculate a selector's specificity. Let's look at the following picture:



Style
attribute

ID

Class,
psuedo-class,
attribute

Elements



,



,



,




Most
specificity
value

Least
specificity
value


Calculations:

- If the element has inline styling, that wins with value of = 1,0,0,0 points
- ID value is = 0,1,0,0 points
- Class value, pseudo-class, or attribute selector = 0,0,1,0 points
- Elements = 0,0,0,1 point



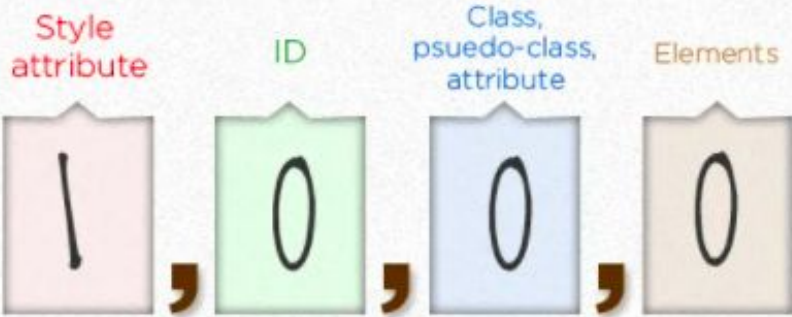
We can read the values as if they were just a number, 1,0,0,0 is “1000” and 0,1,0,0 is “100”.

The commas remind us that it's not really a “base 10” system.

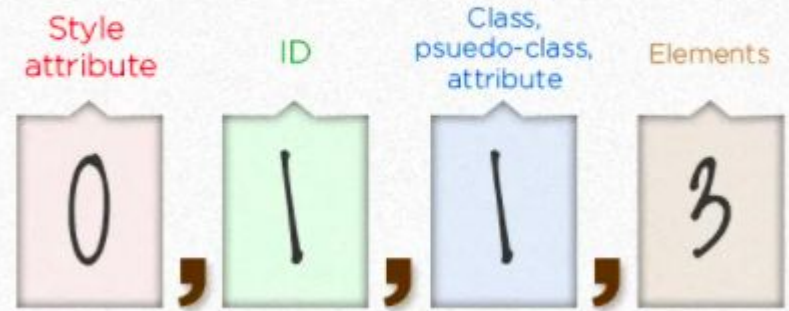


Sample calculations:

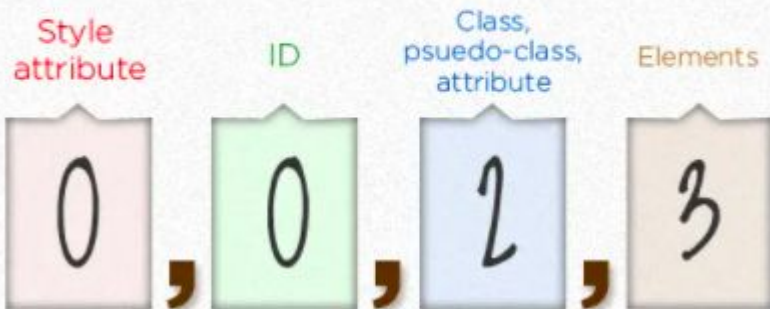
<li style="color: red;">



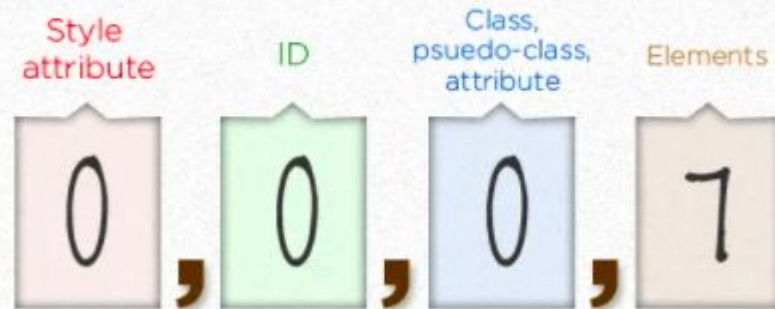
ul#nav li.active a

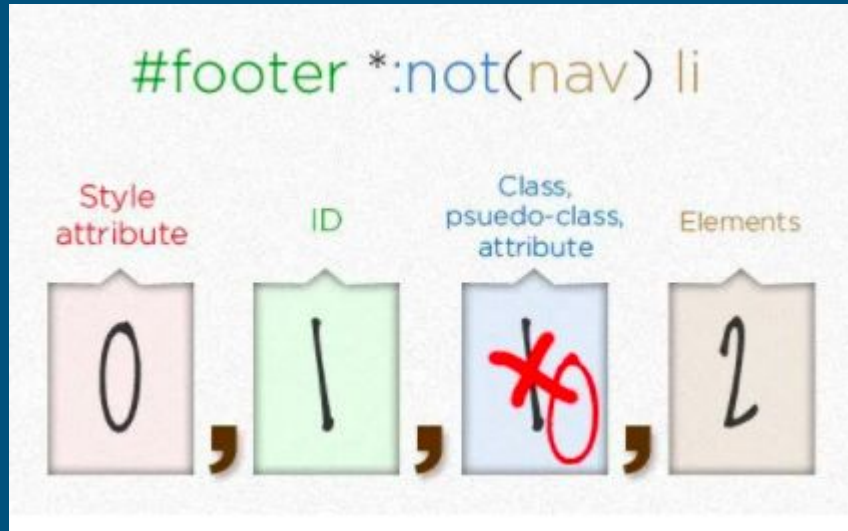


body.ie7.col_3 h2 ~ h2



ul > li ul li ol li:first-letter





Update: The `:not()` sort-of-pseudo-class adds no specificity by itself, only what's inside the parenthesis is added to specificity value.

IMPORTANT NOTES:

- The universal selector (*) has no specificity value (0,0,0,0)
- A class selector beats any number of element selectors.
- Pseudo-elements (e.g. :first-line) get 0,0,0,1 unlike pseudo-class which get 0,0,1,0

- The **!important** value appended a CSS property value is an *automatic win*. It overrides even inline styles from the markup. The only way an !important value can be overridden is with another !important rule declared later in the CSS and with equal or great specificity value otherwise. You could think of it as adding 1,0,0,0,0 to the specificity value.

Some Specificity Rules

- Rules with more specific selectors have a greater specificity.
- ID selectors have a higher specificity than attribute selectors.

For example:

1- `#p123{ color:#ddd;}`

2- `P[id="p123">{ color:#ccc; }`

1 more specific than 2

- The last rule defined overrides any previous, conflicting rules. For example, given these two rules

```
p { color: red; background: yellow }
```

```
p { color: green }
```

paragraphs would appear in green
text yellow background

INHERITANCE

The idea behind inheritance is relatively easy to understand.

Elements inherit styles from their parent container. Not all properties are inherited, but you can force ones to be by using the inherit value.

For example:

```
p {margin: inherit; padding: inherit}
```

How does Inheritance work?

When an element inherits a value from its parent, it is inheriting its computed value. What does this mean? Every CSS property goes through a four-step process when its value is being determined:

Specified value The user agent determines whether the value of the property comes from a style sheet, is inherited or should take its initial value.

Computed value The specified value is resolved to a computed value and exists even when a property doesn't apply. The document doesn't have to be laid out for the computed value to be determined.

Used value The used value takes the computed value and resolves any dependencies that can only be calculated after the document has been laid out (like percentages).

Actual value This is the value used for the final rendering, after any approximations have been applied (for example, converting a decimal to an integer).

If you look at any CSS property specification, you will see that it defines its initial (or default) value, the elements it applies to, its inheritance status and its computed value (among others).

For example: background

Name: background-color

Value: <color>

Initial: transparent

Applies to: all elements

Inherited: no

Percentages: N/A

Media: visual

Computed value: the computed color(s)

Why is this information relevant to inheritance?

When an element inherits a value from its parent, it inherits its computed value. Because the computed value exists even if it isn't specified in the style sheet, a property can be inherited even then: the initial value will be used. You can make use of inheritance even if the parent doesn't have a specified property.



Inherited properties



The initial value is used on the root of
the element only



Inherited types

Font Type: font-style, font-variant, font-weight, font-stretch, font-size, font-family, color, line-height;

Space Type: letter-spacing, word-spacing, white-space;

Letter Type: text-align, text-indent, text-shadow, text-transform;

List Type: list-style, list-style-type, list-style-position;

Others: visibility, cursor;

Example for inherited properties:

```
<span style="color: red">  hello,  
    <em>CSS</em> </span>
```


The result is the color of both `span` and `em` element are red. Because the color is an inherited property from the parent recursively.



Non-inherited properties



The initial value is used on every
element.



Non-Inherited Types

Layout Type: float, position, left, right, top, bottom, z-index, display;

Box Type: width, max-width, min-width, height, max-height, min-height, margin, padding, border;

Background Type: background-size, background-image, background-clip, background-color, background-origin, background-position, background-repeat;

Others: overflow, text-overflow, vertical-align;

Example for Non-inherited properties:

```
<span style="border: 1px solid black">  
  hello, <em>CSS</em> </span>
```

The result is the border is only affected on `span`, not `em`.

The border is a non-inherited property and there is no border property specified on the `em`.

The “C” in CSS: The Cascade

Algorithm that defines how to combine property values originating from different sources.

Cascade follows 3 steps: Importance, Specificity, and Source order. They help determine which properties to assign to an element.

Importance

Style sheets can have different sources:

- **User agent:** browser's default style sheet.
- **User:** user's browser options.
- **Author:** CSS provided by the page
(whether inline, embedded or external)

By default this is the order in which the sources are processed, so the author's rules override those of the user and the user agent.

Source order

In ascending order of importance:

1. User agent declarations
2. User declarations
3. Author declarations
4. Author !important declarations
5. User !important declarations

Order of CSS Rules

CSS rules follow a certain order:

from left to right and top to bottom.

This means that CSS prioritizes the right-most CSS and the bottom-most CSS.

Here, the rightmost CSS is applied to the div, meaning that the div has a blue border:

```
<div style="border: 1px solid red; border: 2px solid blue;"></div>
```

Similarly, the CSS at the bottom is applied to the div, meaning that the div has a blue font:

```
div {  
  color: red;  
  color: blue;  
}
```



DEV TOOLS FOR CSS





To access the dev tools right click on any page element and select Inspect Element.

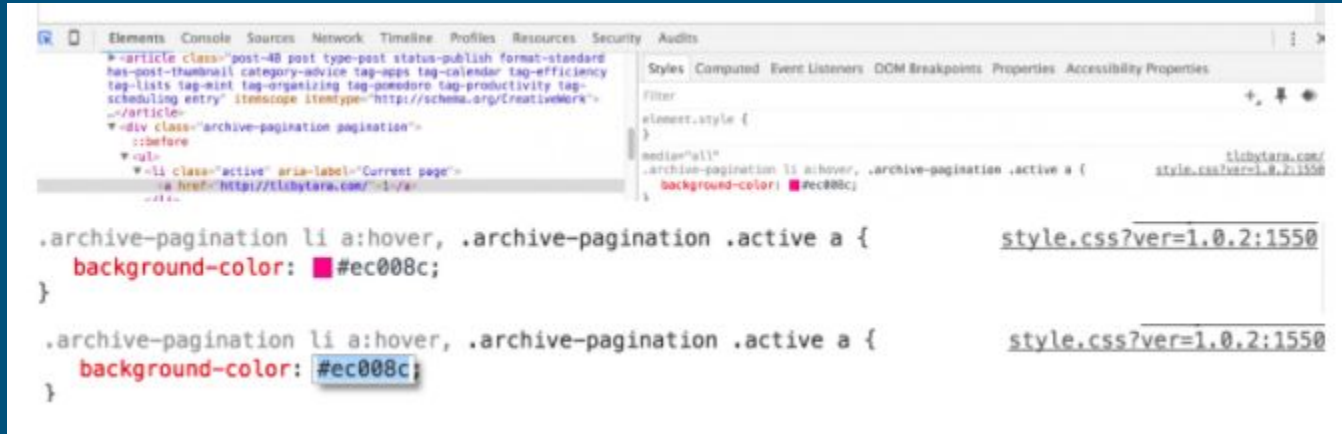
You can use the Elements panel for a variety of tasks:

- Inspect the HTML & CSS of a web page.
 - Test different layouts.
 - Live-edit CSS.
- 

Edit and create styles

You can add or edit styles within the Styles pane in the Elements panel. Unless the area containing the styling information is greyed out (as is the case with user agent stylesheets), all styles are editable. Edit styles in the following ways: Edit an existing property name or value. Add a new property declaration. Add a new CSS rule. To enable or disable a style declaration, check or uncheck the checkbox next to it.

Click an element in the page or the html element for the corresponding CSS



Click on a property value to edit the value. If you're editing a property name, press Tab or Enter to edit the property value. By default, your CSS modifications are not permanent, changes are lost when you reload the page.

SUMMARY

- In CSS, the order in which we specify our rules matters.
- The most important rule of the cascade: all other things being equal, the last defined style is the one that's used.
- when conflicts occur:
 1. Find all declarations whose selectors match a particular element.
 2. Sort these declarations by weight and origin
 3. Sort the selectors by specificity
 4. Sort by order specified
- If a style isn't working the way you want, you have a specificity or cascade problem, and you should fix that instead of using !important.
- Structure your code to respect the cascade. Put general element-based styles up top, followed by specific component/module styles, followed by modifiers.