

CSS

CASCADE

# Types of style sheets

HTML documents may have **three types of style sheets** applied to them.

Type 1:  
User-agent styles

If you look at a web page in a browser, even a web page without any CSS file attached, **there are some styles already in place.**

Headings are often **larger in size**.  
Unvisited links are often blue,  
visited links are often purple and all  
general content is often black.

This is because all **browsers have their own style sheet** that is used to define the default behaviour of HTML elements.

This style sheet is more accurately referred to as a **“user-agent” style sheet.**

Type 2:  
User styles

“Users” are the **people who visit** our websites and interact with our content.

Many browsers have some sort of mechanism that **allows users to write their own style sheet or style sheets**. These style sheets are referred to as “user style sheets”.

User style sheets allows users to **override style sheets** for specific websites or for all websites that they visit.

An example could be **a user with a vision impairment** who writes a style sheet that turns all text white and the page background to black - to make content easier to read.

Type 3:  
Author styles

As web authors, we can use **three different methods** to apply CSS styles to any HTML document.

**Inline styles** are applied to elements in the HTML markup using the “style” attribute.

```
<!-- inline style -->  
<p style="color:blue;">  
    Content here  
</p>
```

**Header styles** are placed in the head of HTML documents using the `<style>` element.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Site name</title>
  <style>
    p { color: green; }
  </style>
</head>
<body>
</body>
</html>
```

**External style sheets** are applied using `<link>` element.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Site name</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
</body>
</html>
```

Normal vs important

Users and authors can define any declaration as **“more important”** than other declarations.

Important declarations are written with the “!” **delimiter token** and **“important” keyword** following the declaration.

```
/* example of important */  
h2 { color: red !important;}
```

As we will see soon, the CSS Cascade **treats important declarations differently** to normal declarations.

# Understanding Conflict

Browsers (or user-agents) have to deal with CSS rules coming from these three different origins - **user-agent, user and author.**

Browsers have to deal with CSS rules coming from different types of author style sheets - **inline vs header vs external.**

Browsers also have to deal with different types of declarations - **important vs normal.**

At some point, browsers have to deal with declarations that **conflict**.

Conflict is where more than one declaration refers to the **same element and property**.

```
/* conflicting rules */  
h2 { color: blue; }  
h2.intro { color: green; }  
.container h2 { color: red; }
```

When conflict occurs, browsers need to determine **which declarations will “win”** (be applied to an HTML document).

In CSS2.1, there are four steps to **determine which declaration will “win”**.

Step 1

Gather the  
declarations

Browsers must **gather all the declarations** that apply to an element and property from user-agent, user and author style sheets.

For example, browsers have to gather any declarations that match:

**element = h2**

**property = color**

```
/* user-agent styles */  
h2 { color: black; }  
  
/* user styles */  
h2 { color: green; }  
  
/* author styles */  
h2 { color: blue; }  
#nav h2 { color: lime; }
```

If there are declarations from **more than one of these three sources**, proceed to step 2.

## Step 2

Sort by origin and  
importance

For this second step, the gathered declarations are sorted according to **origin** (user-agent, user, author) and **importance** (normal or important).

In CSS2.1, there are **five steps** to determine how declarations are sorted.

## **From lowest to highest priority**

1. Normal user-agent declarations
2. Normal user declarations
3. Normal author declarations
4. Important author declarations
5. Important user declarations

If no other declarations exist, **user-agent declarations win.**

```
/* user-agent styles */  
h2 { color: black; }
```

**Normal user declarations** beat  
user-agent declarations.

```
/* user-agent styles */  
h2 { color: black; }
```

```
/* user styles */  
h2 { color: green; }
```

**Normal author declarations** beat  
user-agent declarations and normal  
user declarations.

```
/* user-agent styles */
```

```
h2 { color: black; }
```

```
/* user styles */
```

```
h2 { color: green; }
```

```
/* author styles */
```

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

**Important author declarations**  
beat all normal declarations.

```
/* user-agent styles */
```

```
h2 { color: black; }
```

```
/* user styles */
```

```
h2 { color: green; }
```

```
/* author styles */
```

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

```
h2 { color: purple !important; }
```

**Important user declarations** beat important author declarations and all normal declarations.

```
/* user-agent styles */  
h2 { color: black; }  
  
/* user styles */  
h2 { color: green; }  
h2 { color: red !important; }  
  
/* author styles */  
h2 { color: blue; }  
h2 { color: purple !important; }
```

But what if two declarations **have the same origin or importance?**

```
/* user-agent styles */  
h2 { color: black; }
```

```
/* user styles */  
h2 { color: green; }  
h2 { color: red; }
```

```
/* author styles */  
h2 { color: blue; }  
h2 { color: lime; }
```

If declarations have the **same origin or importance** then proceed to Step 3.

# Step 3

Determine  
specificity

If declarations have the same origin or importance then the declaration's **selectors need to be scored** to see which declaration will “win”.

Four scores are concatenated (linked together as a chain) to create a final score. This score is referred to as a selector's specificity.

**selector specificity = a,b,c,d**

How is specificity  
calculated?

A. Is there an **inline style**?

```
<p style="color:red;"></p>
```

```
a = 1 x inline styles
```

```
b = 0 x IDs
```

```
c = 0 x classes, pseudo-classes, attributes
```

```
d = 0 x elements, pseudo-elements
```

```
Specificity = 1,0,0,0
```

B. Count the number of **ID selectors** in the selector.

```
#nav { }
```

```
a = 0 x inline styles
```

```
b = 1 x IDs
```

```
c = 0 x classes, pseudo-classes, attributes
```

```
d = 0 x elements, pseudo-elements
```

```
Specificity = 0,1,0,0
```

C. Count the number of **class selectors, attribute selectors or pseudo-class selectors** in the selector.

```
.intro { }
```

```
a = 0 x inline styles
```

```
b = 0 x ID
```

```
c = 1 x classes, pseudo-classes, attributes
```

```
d = 0 x elements, pseudo-elements
```

```
Specificity = 0,0,1,0
```

D. Count the number of **element type or pseudo-element selectors** in the selector.

p { }

a = 0 x inline styles

b = 0 x ID

c = 0 x classes, pseudo-classes, attributes

d = 1 x elements, pseudo-elements

Specificity = 0,0,0,1

E. Ignore the **universal selector**.

\* { }

a = 0 x inline styles

b = 0 x ID

c = 0 x classes

d = 0 x element

Specificity = 0,0,0,0

Some examples

```
#nav ul { }
```

```
a = 0 x inline styles
```

```
b = 1 x ID (#nav)
```

```
c = 0 x classes
```

```
d = 1 x element (ul)
```

```
Specificity = 0,1,0,1
```

```
blockquote.special { }
```

```
a = 0 x inline styles
```

```
b = 0 x IDs
```

```
c = 1 x class (.special)
```

```
d = 1 x element (blockquote)
```

```
Specificity = 0,0,1,1
```

```
a:link { }
```

```
a = 0 x inline styles
```

```
b = 0 x IDs
```

```
c = 1 x pseudo-class (:link)
```

```
d = 1 x elements (a)
```

```
Specificity = 0,0,1,1
```

```
input[required] { }
```

```
a = 0 x inline styles
```

```
b = 0 x IDs
```

```
c = 1 x attribute selector ([required])
```

```
d = 1 x elements (input)
```

```
Specificity = 0,0,1,1
```

```
p[class="a"] { }
```

```
a = 0 x inline styles
```

```
b = 0 x IDs
```

```
c = 1 x attribute selector ([class="a"])
```

```
d = 1 x elements (p)
```

```
Specificity = 0,0,1,1
```

```
div[id="news"] { }
```

a = 0 x inline styles

b = 0 x IDs

c = 1 x attribute selector ([id="news"])

d = 1 x elements (div)

Specificity = 0,0,1,1

```
::first-line { }
```

a = 0 x inline styles

b = 0 x IDs

c = 0 x classes

d = 1 x pseudo-element (::first-line)

Specificity = 0,0,0,1

```
#nav ul li a:hover { }
```

a = 0 x inline styles

b = 1 x ID (#nav)

c = 1 x pseudo-class (:hover)

d = 3 x elements (ul,li,a)

Specificity = 0,1,1,3

# A note on concatenation

**Generally speaking**, “A” will always beat “B”, which will always beat “C”, which will always beat “D”.

No matter how many IDs are used in a selector, **an inline style will always beat an ID selector.**

In the following example, **the inline style wins** due to specificity -  
1,0,0,0 beats 0,10,0,0.

```
/* author styles */  
#one #two #three #four #five #six #seven  
#eight #nine #ten  
{ color: green; }
```

```
<!-- HTML document -->  
<h2 style="color: purple;">
```

Theoretically, no matter how many classes are applied to a selector,  
**an ID will always beat classes.**

(Firefox browsers incorrectly implement the specification so that 256 classes will “beat” an ID)

In the following example, **the ID selector wins** due to specificity - 0,1,0,0 beats 0,0,10,0.

```
/* author styles */  
.one .two .three .four .five .six  
.seven .eight .nine .ten  
{ color: green; }  
  
#nav { color: lime; }
```

Theoretically, no matter how many element types are applied to a selector, **a class will always beat element types.**

(Firefox browsers incorrectly implement the specification so that 256 element types will “beat” a class)

In the following example, **the class selector wins** due to specificity -  
0,0,1,0 beats 0,0,0,10.

```
/* author styles */  
div  
{ color: green; }  
  
.intro { color: lime; }
```

What if there is still  
no clear winner?

In the following example, two author styles **have the same specificity.**

```
/* author styles */  
.intro h2 { color: blue; }  
h2.new { color: lime; }
```

```
<!-- HTML document -->  
<div class="intro">  
  <h2 class="new"></h2>  
</div>
```

If there is still **no clear winner** (the selectors have the same specificity) then proceed to Step 4.

# Step 4

Determining order  
specified

If two declarations have the same importance, origin and specificity,  
**the declaration that appears last in document order win.**

In the following example, the second rule will win **as it is written after the first rule.**

```
/* author styles */  
.intro h2 { color: blue; }  
h2.new { color: lime; }
```

```
<!-- HTML document -->  
<div class="intro">  
  <h2 class="new"></h2>  
</div>
```

Declarations from **imported style sheets** are ordered as if their style sheets are substituted in place of the @import rule.

In the following example,  
declarations are ordered from  
**“a.css”, then “b.css” then  
“c.css”**.

```
@import "a.css";  
@import "b.css";  
@import "c.css";
```

Declarations from **linked style sheets** are treated as if they were concatenated in linking order, as determined by the host HTML document.

In the following example,  
declarations are ordered from  
**“a.css”, then “b.css” then  
“c.css”**.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Title</title>
  <link rel="stylesheet" href="a.css">
  <link rel="stylesheet" href="b.css">
  <link rel="stylesheet" href="c.css">
</head>
<body>
  ...
</body>
</html>
```

Declarations from **header style sheets** are treated the same as linked style sheets - as if they were concatenated in linking order, as determined by the host HTML document.

In the following example,  
declarations are ordered from  
**“a.css”, then the header style,**  
**then “b.css”.**

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Title</title>
  <link rel="style sheet" href="a.css">
  <style>h2 {color:blue;}</style>
  <link rel="style sheet" href="b.css">
</head>
<body>
</body>
</html>
```

# Quiz 1: Origin and importance

In the following example, what color would the `<h2>` element be?

```
/* user-agent style sheet */  
h2 { color: black; }  
  
/* user style sheet */  
h2 { color: green; }  
h2 { color: lime !important; }  
  
/* author style sheet */  
h2 { color: blue; }  
h2 { color: red !important; }
```

**Answer:** The `<h2>` element will be lime because important user styles beat important author styles and all normal styles.

```
/* user-agent style sheet */  
h2 { color: black; }  
  
/* user style sheet */  
h2 { color: green; }  
h2 { color: lime !important; }  
  
/* author style sheet */  
h2 { color: blue; }  
h2 { color: red !important; }
```

# Quiz 2: Specificity

In the following example, what color would the `<p>` **element** be?

```
/* author style sheet */  
#nav p { color: lime; }  
p { color: blue; }  
div#container p { color: purple; }  
p.intro { color: green; }
```

**Answer:** The <p> element will be purple because this selector has more specificity (0,1,0,2) than the other selectors.

```
/* author style sheet */  
#nav p { color: lime; }  
p { color: blue; }  
div#container p { color: purple; }  
p.intro { color: green; }
```

# Quiz 3: Specificity

**Part 1:** Guess the specificity of the following selector:

```
#header h1 span a { }
```

```
#header h1 span a { }
```

```
a = 0 x inline styles
```

```
b = 1 x ID (#header)
```

```
c = 0 x classes
```

```
d = 3 x elements (h1,span,a)
```

```
Specificity = 0,1,0,3
```

**Part 2:** Guess the specificity of the following selector:

```
.intro :first-letter { }
```

```
.intro :first-letter { }
```

```
a = 0 x inline styles
```

```
b = 0 x IDs
```

```
c = 1 x class (.intro)
```

```
d = 1 x pseudo-element (:first-letter)
```

```
Specificity = 0,0,1,1
```

**Part 3:** Guess the specificity of the following selector:

```
a[href^="http:"] { }
```

```
a[href^="http:"] { }
```

```
a = 0 x inline styles
```

```
b = 0 x IDs
```

```
c = 1 x attribute selector ([href^="http:"])
```

```
d = 1 x element (a)
```

```
Specificity = 0,0,1,1
```

# CSS3 Cascade

In CSS3, there have been **several additions** to the overall cascade process.

# Origin and Importance

In CSS 2.1, there were five steps to determine origin and importance.  
In CSS3, **there are now some extra steps.**

1. Normal user-agent declarations
2. Normal user declarations
3. Normal author declarations
- 4. Animation declarations**
5. Important author declarations
6. Important user declarations
- 7. Important user-agent declarations**
- 8. Transition declarations**

Let's take a look at these **new steps** and what they mean!

**Animation declarations** are declarations that involve animating the relevant element.

```
h2 {  
    -webkit-animation: heading 3s;  
    animation: heading 3s;  
}  
  
@-webkit-keyframes heading { from {color:  
red;} to {color: lime;}  
}  
  
@keyframes heading { from {color: red;} to  
{color: lime;}  
}
```

**Animation declarations** beat  
normal user-agent, author and user  
declarations.

**Important user-agent declarations** are declarations within the user-agent style sheet.

**Important user-agent declarations** beat normal user-agent, author, user, and animation declarations. They also override important author and user declarations.

**Transition declarations** are declarations that transition an element to another state.

```
h2
{
  -webkit-transition: color 0.5s
ease;
  transition: color 0.5s ease;
}
```

```
h2:hover { color: purple;}
```

Where relevant, **transition declarations** beat all other types of declaration.

```
/* user-agent styles */  
h2 { color: black; }
```

```
/* user-agent styles */  
h2 { color: black; }
```

```
/* user styles */  
h2 { color: green; }
```

```
/* user-agent styles */  
h2 { color: black; }  
  
/* user styles */  
h2 { color: green; }  
  
/* author styles */  
h2 { color: blue; }
```

```
/* user-agent styles */  
h2 { color: black; }  
  
/* user styles */  
h2 { color: green; }  
  
/* author styles */  
h2 { color: blue; }  
h2 { animation: aaa 3s; }  
@keyframes aaa {from{color:red;} to {color:lime;}}
```

```
/* user-agent styles */
h2 { color: black; }

/* user styles */
h2 { color: green; }

/* author styles */
h2 { color: blue; }
h2 { animation: aaa 3s; }
@keyframes aaa {from{color:red;} to {color:lime;}}
h2 { color: purple !important; }
```

```
/* user-agent styles */
h2 { color: black; }

/* user styles */
h2 { color: green; }
h2 { color: red !important; }

/* author styles */
h2 { color: blue; }
h2 { animation: aaa 3s; }
@keyframes aaa {from{color:red;} to {color:lime;}}
h2 { color: purple !important; }
```

```
/* user-agent styles */
h2 { color: black; }
h2 { color: pink !important; }

/* user styles */
h2 { color: green; }
h2 { color: red !important; }

/* author styles */
h2 { color: blue; }
h2 { animation: aaa 3s; }
@keyframes aaa {from{color:red;} to {color:lime;}}
h2 { color: purple !important; }
```

```
/* user-agent styles */
h2 { color: black; }
h2 { color: pink !important; }

/* user styles */
h2 { color: green; }
h2 { color: red !important; }

/* author styles */
h2 { color: blue; }
h2 { animation: aaa 3s; }
@keyframes aaa {from{color:red;} to {color:lime;}}
h2 { color: purple !important; }
h2 { transition: color 0.5s ease; }
h2:hover { color: purple;}
```

Specificity

There are **no changes** to the specificity methodology defined in CSS2.1 Cascade.

Order of appearance

There are **no changes** to the specificity or order of appearance methodology defined in CSS2.1 Cascade.

For those of you who have read the CSS3 Cascade specification in detail, you may have noticed that I **simplified some of the steps.**

<http://www.w3.org/TR/2013/CR-css-cascade-3-20131003/>

Override

As part of the “origin and importance” process, the CSS3 Cascade specification mentions **override declarations**.

**Override declarations** allow authors to manipulate the DOM using JavaScript and change a style, without having to modify any associated style sheets.

“The **getOverrideStyle** allows DOM author to change to the style of an element without modifying the style sheets of a document.”

<http://www.w3.org/TR/DOM-Level-2-Style/css.html#CSS-DocumentCSS>

However, **none of the modern browsers support the “getOverrideStyle” method**, so this can be quite confusing for authors.

# Scoped styles

The specification also mentions an entirely new step in the cascade, which focuses on **scoped styles**.

The “scoped” attribute is applied to the <style> element. This method allows authors to include styles **inside the body of HTML documents**. These styles are designed to target a specific element and its children.

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Title</title>
</head>
<body>
<div>
  <style scoped>
    h1 { color:red; }
  </style>
</div>
</body>
</html>
```

There has been a lot of discussion within various W3C working groups about **removing scoped styles from the specification entirely**, so this is really not something that authors need to focus on.



# Russ Weakley

Max Design

**Site:** [maxdesign.com.au](http://maxdesign.com.au)

**Twitter:** [twitter.com/russmaxdesign](https://twitter.com/russmaxdesign)

**Slideshare:** [slideshare.net/maxdesign](https://slideshare.net/maxdesign)

**Linkedin:** [linkedin.com/in/russweakley](https://linkedin.com/in/russweakley)

Title S

Lesson

Lesson2

Section hi

Action

body highlight

body smaller

quote url

code c-h c-g

label

T