# Q) [**Relative positioning + Absolute positioning VS Floats Left an Float Right (Which approach do you use in your CSS)?**](http://stackoverflow.com/questions/1938676/relative-positioning-absolute-positioning-vs-floats-left-an-float-right-which)

A) I my experience using absolute positioning works well on elements you are in 100% control of in terms of size; like a logo, a meny, rss links and things like this. Then you can place these elements exactly where you want them.

I prefer using float when I am displaying dynamic content on a page, since the size of the element can change, and the placement of all elements in relation works really good with floats. Just remember to clear the floats when needed!

**Q) CSS Specificity**

A) Specificity is the means by which a browser decides which CSS property values are the most relevant to an element and therefore will be applied. Specificity is only based on the matching rules which are composed of [css selectors](https://developer.mozilla.org/en/CSS/CSS_Reference" \l "Selectors" \o "en/CSS/CSS_Reference#Selectors) of different type.

**Q) How is CSS specificity calculated?**

A) The specificity is a weight that is applied to a given CSS declaration based on the count of each [selector type](https://developer.mozilla.org/en-US/docs/Web/CSS/Specificity#selector-type). In the case of specificity equality, the latest declaration found in the CSS is applied to the element. Specificity only applies when the same element is targeted. CSS rules that [directly target an element](https://developer.mozilla.org/en-US/docs/Web/CSS/Specificity#directly-targeted-elements) will always take precedence over rules that an element inherits from an ancestor.

Inline style added to an element (e.g., style="font-weight:bold") always overwrites any styles in the CSS and thus can be though as having the biggest specificity.

Q) The!importantexception

When an !important rule is used on a style declaration, this declaration overrides any other declaration made in the CSS, wherever it is in the declaration list. Although, !important has nothing to do with specificity. Using !important is **bad practice** and should be avoided because it makes debugging more difficult by breaking the natural [cascading](https://developer.mozilla.org/en-US/docs/Web/CSS/Cascade) in your stylesheets. When two conflicting declarations with the !important rule are applied to the same element, the declaration with greater specificity will be applied.

Some rules of thumb

* **Always** look for a way to use specificity before even considering !important
* **Only** use !important on page-specific css that overrides site-wide or foreign css (from ExtJs or YUI for example).
* **Never** use !important when you're writing a plugin/mashup.
* **Never** use !important on site-wide css.

Instead of using !important, you can:

1. Make better use of CSS cascading properties
2. Use more specific rules. By indicating one or more elements before the element you're selecting the rule becomes more specific and gets higher priority:

<div id="test">

<span>Text</span>

</div>

div#test span { color: green }

div span { color: blue }

span { color: red }

No matter what the order, the text will be green because that rule is most specific. (Also, the rule for blue overwrites the rule for red, notwithstanding the order of the rules)

**You should use it when:**

A) Scenario one:

1. You have a global CSS file that sets visual aspects of your site globally
2. You (or others) use inline styles on elements themselves which is a very bad practice

In this case you could set certain styles in your global CSS file as important thus overriding inline styles set directly on elements.

Real world example: Some badly written **jQuery plugins** that use inline styles.

B) Another scenario

#someElement p {

color: blue;

}

p.awesome {

color: red;

}

How do you make awesome paragraphs always turn red, even ones inside #someElement? Without !important, the first rule will have more specificity and will win over the second rule.

**Q) How to override !important**

A) Simply add another CSS rule with !important, and either give the selector a higher specificity (adding an additional tag, id or class to the selector), or add a CSS rule with the same selector at a later point than the existing one (in a tie, the last one defined wins).

Some examples with a higher specificity:

table td {height: 50px !important;}

.myTable td {height: 50px !important;}

#myTable td {height: 50px !important;}

B) Or add the same selector after the existing one:

td {height: 50px !important;}

C) Or rewrite the original rule to avoid the use of !important altogether.

### **The :not exception**

The negation pseudo-class :not is not considered a pseudo-class in the specificity calculation. But selectors placed into the negation pseudo-class count as normal selectors when determining the count of [selector types](https://developer.mozilla.org/en-US/docs/Web/CSS/Specificity#selector-type).

Here is a CSS chunk:

div.outer p {

color:orange;

}

div:not(.outer) p {

color: lime;

}

when used with the following HTML:

<div class="outer">

<p>This is in the outer div.</p>

<div class="inner">

<p>This text is in the inner div.</p>

</div>

</div>

Shall appear on the screen as:

This is in the outer div.

This text is in the inner div.

### **Form-based specificity**

Specificity is based on the form of a selector. In the following case, the selector \*[id="foo"] counts as an attribute selector for the purpose of determining the selector's specificity, even though it selects an ID.

The following style declarations:

\*#foo {

color: green;

}

\*[id="foo"] {

color: purple;

}

when used with this markup:

<p id="foo">I am a sample text.</p>

Will end up looking like:

I am a sample text.

Because it matches the same element but the ID selector has a higher specificity.

### **Tree proximity ignorance**

The proximity of an element to other elements that are referenced in a given selector has no impact on specificity. The following style declaration:

body h1 {

color: green;

}

html h1 {

color: purple;

}

With the following HTML:

<html>

<body>

<h1>Here is a title!</h1>

</body>

</html>

Will render as:

Here is a title!

Because the two declarations have equal [selector type](https://developer.mozilla.org/en-US/docs/Web/CSS/Specificity#selector-type) counts, but the html h1 selector is declared last.

### **Directly targeted elements versus inherited styles**

Styles for a directly targeted element will always take precedence over inherited styles, regardless of the specificity of the inherited rule.

#parent {

color: green;

}

h1 {

color: purple;

}

With the following HTML:

<html>

<body id="parent">

<h1>Here is a title!</h1>

</body>

</html>

Will also render as:

Here is a title!

Because the h1 selector targets the element specifically, but the green selector is only inherited from the parent.

element.style{

}

It is an element's inline style attribute.

Here's the specificity value

Selector Specificity Specificity in large base

inline-style 1 0 0 0 1000

id selector 0 1 0 0 100

class,pseudo,attribute selector 0 0 1 0 10

type selector and pseudo elements 0 0 0 1 1

When do we prefer using this style attribute?

When you want to set/update the styles of an element dynamically using Javascript, you'll use this styles.

How selectors work?

<https://css-tricks.com/how-css-selectors-work/>

# **30 CSS Selectors**

<http://code.tutsplus.com/tutorials/the-30-css-selectors-you-must-memorize--net-16048>