

Unlimited asset downloads!

From \$16.50/m



envato**tuts**+



**POWER YOUR APPS WITH REAL-TIME
FX DATA**
Try OANDA's Exchange Rates API



Advertisement

[CODE](#) > [WEB DEVELOPMENT](#)

The 30 CSS Selectors You Must Memorize

by [Jeffrey Way](#) 9 Jun 2011

Difficulty: Intermediate Length: Long Languages: English

[Web Development](#) [Front-End](#) [HTML](#) [CSS](#) [CSS Selectors](#)



This post is part of a series called [CSS3 Mastery](#).

◀ [10 CSS3 Properties you Need to be Familiar With](#)

▶ [Getting to Work with CSS3 Power Tools](#)

Learn CSS: The Complete Guide

We've built a complete guide to help you [learn CSS](#), whether you're just getting started with the basics or you want to explore more advanced CSS.

CSS Selectors

So you learned the base `id`, `class`, and `descendant` selectors—and then called it a day? If so, you're missing out on an enormous level of flexibility. While many of the selectors mentioned in this article are part of the CSS3 spec, and are, consequently, only available in modern browsers, you owe it to yourself to commit these to memory.

And by the way, if you're having trouble with your CSS and want a pro to look over it and fix any errors, you can find some [qualified freelancers](#) on Envato Studio.

Advertisement

1. *

```
1  * {  
2    margin: 0;  
3    padding: 0;  
4  }
```

Let's knock the obvious ones out, for the beginners, before we move onto the more advanced selectors.

The star symbol will target every single element on the page. Many developers will use this trick to zero out the `margin`s and `padding`. While this is certainly fine for quick tests, I'd advise you to never use this in production code. It adds too much *weight* on the browser, and is unnecessary.

The `*` can also be used with child selectors.

```
1  #container * {  
2    border: 1px solid black;  
3  }
```

This will target every single element that is a child of the `#container` `div`. Again, try not to use this technique very much, if ever.

[View Demo](#)

Compatibility

- IE6+
- Firefox
- Chrome
- Safari
- Opera

2. #X

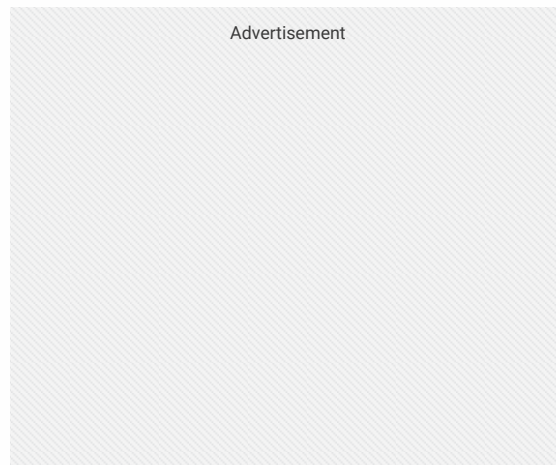
```
1 #container {  
2   width: 960px;  
3   margin: auto;  
4 }
```

Prefixing the hash symbol to a selector allows us to target by `id`. This is easily the most common usage, however be cautious when using `id` selectors.

Ask yourself: do I absolutely need to apply an `id` to this element in order to target it?

`id` selectors are rigid and don't allow for reuse. If possible, first try to use a tag name, one of the new HTML5 elements, or even a pseudo-class.

[View Demo](#)



Compatibility

- IE6+
- Firefox
- Chrome
- Safari
- Opera

3. .X

```
1 .error {  
2   color: red;  
3 }
```

This is a `class` selector. The difference between `id`s and `class`es is that, with the latter, you can target multiple elements. Use `class`es when you want your styling to apply to a group of

elements. Alternatively, use `id`s to find a needle-in-a-haystack, and style only that specific element.

[View Demo](#)

Compatibility

- IE6+
- Firefox
- Chrome
- Safari
- Opera

4. X Y

```
1 li a {  
2   text-decoration: none;  
3 }
```

The next most common selector is the `descendant` selector. When you need to be more specific with your selectors, you use these. For example, what if, rather than targeting *all* anchor tags, you only need to target the anchors which are within an unordered list? This is specifically when you'd use a descendant selector.

Pro-tip - If your selector looks like `X Y Z A B.error`, you're doing it wrong. Always ask yourself if it's absolutely necessary to apply all of that weight.

[View Demo](#)

Compatibility

- IE6+
- Firefox
- Chrome
- Safari
- Opera

5. X

```
1 a { color: red; }  
2 ul { margin-left: 0; }
```

What if you want to target all elements on a page, according to their `type`, rather than an `id` or `class` name? Keep it simple, and use a type selector. If you need to target all unordered lists, use `ul {}`.

[View Demo](#)

Compatibility

- IE6+
- Firefox
- Chrome
- Safari
- Opera

6. X:visited and X:link

```
1 a:link { color: red; }  
2 a:visited { color: purple; }
```

We use the `:link` pseudo-class to target all anchors tags which have yet to be clicked on.

Alternatively, we also have the `:visited` pseudo class, which, as you'd expected, allows us to apply specific styling to only the anchor tags on the page which *have* been clicked on, or *visited*.

[View Demo](#)

Compatibility

- IE7+
- Firefox
- Chrome
- Safari
- Opera

7. X + Y

```
1  ul + p {  
2    color: red;  
3  }
```

This is referred to as an adjacent selector. It will select *only* the element that is immediately preceded by the former element. In this case, only the first paragraph after each `ul` will have red text.

[View Demo](#)

Compatibility

- IE7+
- Firefox
- Chrome
- Safari
- Opera

8. X > Y

```
1  div#container > ul {  
2    border: 1px solid black;  
3  }
```

The difference between the standard `X Y` and `X > Y` is that the latter will only select direct children. For example, consider the following markup.

```
01  <div id="container">  
02    <ul>
```

```

03     <li> List Item
04         <ul>
05             <li> Child </li>
06         </ul>
07     </li>
08     <li> List Item </li>
09     <li> List Item </li>
10     <li> List Item </li>
11 </ul>
12 </div>

```

A selector of `#container > ul` will only target the `ul`s which are direct children of the `div` with an `id` of `container`. It will not target, for instance, the `ul` that is a child of the first `li`.

For this reason, there are performance benefits in using the child combinator. In fact, it's recommended particularly when working with JavaScript-based CSS selector engines.

[View Demo](#)

Compatibility

- IE7+
- Firefox
- Chrome
- Safari
- Opera

9. X ~ Y

```

1  ul ~ p {
2      color: red;
3  }

```

This sibling combinator is similar to `X + Y`, however, it's less strict. While an adjacent selector (`ul + p`) will only select the first element that is *immediately* preceded by the former selector, this one is more generalized. It will select, referring to our example above, any `p` elements, as long as they follow a `ul`.

[View Demo](#)

Compatibility

- IE7+
- Firefox
- Chrome
- Safari
- Opera

10. X[title]

```
1 a[title] {  
2   color: green;  
3 }
```

Referred to as an *attributes selector*, in our example above, this will only select the anchor tags that have a `title` attribute. Anchor tags which do not will not receive this particular styling. But, what if you need to be more specific? Well...

[View Demo](#)

Compatibility

- IE7+
- Firefox
- Chrome
- Safari
- Opera

11. X[href="foo"]

```
1 a[href="https://net.tutsplus.com"] {  
2   color: #1f6053; /* nettuts green */  
3 }
```

The snippet above will style all anchor tags which link to <https://net.tutsplus.com>; they'll receive our branded green color. All other anchor tags will remain unaffected.

Note that we're wrapping the value in quotes. Remember to also do this when using a JavaScript CSS selector engine. When possible, always use CSS3 selectors over unofficial methods.

This works well, though, it's a bit rigid. What if the link does indeed direct to Nettuts+, but, maybe, the path is *nettuts.com* rather than the full url? In those cases we can use a bit of the regular expressions syntax.

[View Demo](#)

Compatibility

- IE7+
- Firefox
- Chrome
- Safari
- Opera

12. X[href*="nettuts"]

```
1 a[href*="tuts"] {  
2   color: #1f6053; /* nettuts green */  
3 }
```

There we go; that's what we need. The star designates that the proceeding value must appear *somewhere* in the attribute's value. That way, this covers *nettuts.com*, *net.tutsplus.com*, and even *tutsplus.com*.

Keep in mind that this is a broad statement. What if the anchor tag linked to some non-Envato site with the string *tuts* in the url? When you need to be more specific, use `^` and `$`, to reference the beginning and end of a string, respectively.

[View Demo](#)

Compatibility

- IE7+
- Firefox
- Chrome
- Safari
- Opera

13. X[href^="http"]

```
1 a[href^="http"] {  
2     background: url(path/to/external/icon.png) no-repeat;  
3     padding-left: 10px;  
4 }
```

Ever wonder how some websites are able to display a little icon next to the links which are external? I'm sure you've seen these before; they're nice reminders that the link will direct you to an entirely different website.

This is a cinch with the carat symbol. It's most commonly used in regular expressions to designate the beginning of a string. If we want to target all anchor tags that have a `href` which begins with `http`, we could use a selector similar to the snippet shown above.

Notice that we're not searching for `https://`; that's unnecessary, and doesn't account for the urls that begin with `https://`.

Now, what if we wanted to instead style all anchors which link to, say, a photo? In those cases, let's search for the *end* of the string.

[View Demo](#)

Compatibility

- IE7+
- Firefox
- Chrome
- Safari
- Opera

14. X[href\$=".jpg"]

```
1 a[href$=".jpg"] {  
2     color: red;  
3 }
```

Again, we use a regular expressions symbol, `$`, to refer to the end of a string. In this case, we're searching for all anchors which link to an image -- or at least a url that ends with `.jpg`. Keep in

mind that this certainly won't work for `gifs` and `pngs`.

[View Demo](#)

Compatibility

- IE7+
- Firefox
- Chrome
- Safari
- Opera

15. X[data-*= "foo"]

```
1 a[data-filetype="image"] {  
2   color: red;  
3 }
```

Refer back to number eight; how do we compensate for all of the various image types: `png`, `jpeg`, `jpg`, `gif`? Well, we could create multiple selectors, such as:

```
1 a[href$=".jpg"],  
2 a[href$=".jpeg"],  
3 a[href$=".png"],  
4 a[href$=".gif"] {  
5   color: red;  
6 }
```

But, that's a pain in the butt, and is inefficient. Another possible solution is to use custom attributes. What if we added our own `data-filetype` attribute to each anchor that links to an image?

```
1 <a href="path/to/image.jpg" data-filetype="image"> Image Link </a>
```

Then, with that *hook* in place, we can use a standard attributes selector to target only those anchors.

```
1 a[data-filetype="image"] {  
2   color: red;  
3 }
```

[View Demo](#)

Compatibility

- IE7+
- Firefox
- Chrome
- Safari
- Opera

16. X[foo~="bar"]

```
1  a[data-info~="external"] {  
2    color: red;  
3  }  
4  
5  a[data-info~="image"] {  
6    border: 1px solid black;  
7  }
```

Here's a special one that'll impress your friends. Not too many people know about this trick. The tilda (~) symbol allows us to target an attribute which has a spaced-separated list of values.

Going along with our custom attribute from number fifteen, above, we could create a `data-info` attribute, which can receive a space-separated list of anything we need to make note of. In this case, we'll make note of external links and links to images -- just for the example.

```
1  "<a href='\"path/to/image.jpg\"' data-info='\"external image\"'> Click Me, Fool </a>"
```

With that markup in place, now we can target any tags that have either of those values, by using the ~ attributes selector trick.

```
1  /* Target data-info attr that contains the value "external" */  
2  a[data-info~="external"] {  
3    color: red;  
4  }  
5  
6  /* And which contain the value "image" */  
7  a[data-info~="image"] {  
8    border: 1px solid black;  
9  }
```

Pretty nifty, ay?

[View Demo](#)

[View Demo](#)

Compatibility

- IE7+
- Firefox
- Chrome
- Safari
- Opera

17. X:checked

```
1 input[type=radio]:checked {  
2     border: 1px solid black;  
3 }
```

This pseudo class will only target a user interface element that has been *checked* - like a radio button, or checkbox. It's as simple as that.

[View Demo](#)

Compatibility

- IE9+
- Firefox
- Chrome
- Safari
- Opera

18. X:after

The `before` and `after` pseudo classes kick butt. Every day, it seems, people are finding new and creative ways to use them effectively. They simply generate content around the selected element.

Many were first introduced to these classes when they encountered the clear-fix hack.

```
01 .clearfix:after {  
02     content: "";  
03     display: block;  
04     clear: both;  
05     visibility: hidden;  
06     font-size: 0;
```

```
07     height: 0;
08   }
09
10   .clearfix {
11     *display: inline-block;
12     _height: 1%;
13   }
```

This *hack* uses the `:after` pseudo class to append a space after the element, and then clear it. It's an excellent trick to have in your tool bag, particularly in the cases when the `overflow: hidden;` method isn't possible.

For another creative use of this, [refer to my quick tip on creating shadows](#).

According to the CSS3 Selectors specification, you should technically use the pseudo element syntax of two colons `::`. However, to remain compatible, the user-agent will accept a single colon usage as well. In fact, at this point, it's smarter to use the single-colon version in your projects.

Compatibility

- IE8+
- Firefox
- Chrome
- Safari
- Opera

19. X:hover

```
1  div:hover {
2    background: #e3e3e3;
3  }
```

Oh come on. You know this one. The official term for this is `user action pseudo class`. It sounds confusing, but it really isn't. Want to apply specific styling when a user hovers over an element? This will get the job done!

Keep in mind that older version of Internet Explorer don't respond when the `:hover` pseudo class is applied to anything other than an

anchor tag.

You'll most often use this selector when applying, for example, a `border-bottom` to anchor tags, when hovered over.

```
1 a:hover {  
2   border-bottom: 1px solid black;  
3 }
```

Pro-tip - `border-bottom: 1px solid black;` looks better than `text-decoration: underline;`.

Compatibility

- IE6+ (In IE6, :hover must be applied to an anchor element)
- Firefox
- Chrome
- Safari
- Opera

20. X:not(selector)

```
1 div:not(#container) {  
2   color: blue;  
3 }
```

The `negation` pseudo class is particularly helpful. Let's say I want to select all divs, except for the one which has an `id` of `container`. The snippet above will handle that task perfectly.

Or, if I wanted to select every single element (not advised) except for paragraph tags, we could do:

```
1 *:not(p) {  
2   color: green;  
3 }
```

[View Demo](#)

Compatibility

- IE9+
- Firefox

- Firefox
- Chrome
- Safari
- Opera

21. X::pseudoElement

```
1 p::first-line {  
2   font-weight: bold;  
3   font-size: 1.2em;  
4 }
```

We can use pseudo elements (designated by `::`) to style fragments of an element, such as the first line, or the first letter. Keep in mind that these must be applied to block level elements in order to take effect.

A pseudo-element is composed of two colons: `::`

Target the First Letter of a Paragraph

```
1 p::first-letter {  
2   float: left;  
3   font-size: 2em;  
4   font-weight: bold;  
5   font-family: cursive;  
6   padding-right: 2px;  
7 }
```

This snippet is an abstraction that will find all paragraphs on the page, and then sub-target only the first letter of that element.

This is most often used to create newspaper-like styling for the first-letter of an article.

Target the First Line of a Paragraph

```
1 p::first-line {  
2   font-weight: bold;  
3   font-size: 1.2em;  
4 }
```

Similarly, the `::first-line` pseudo element will, as expected, style the first line of the element only.

"For compatibility with existing style sheets, user agents must also accept the previous one-colon notation for pseudo-elements introduced in CSS levels 1 and 2 (namely, :first-line, :first-letter, :before and :after). This compatibility is not allowed for the new pseudo-elements introduced in this specification." - [Source](#)

[View Demo](#)

Compatibility

- IE6+
- Firefox
- Chrome
- Safari
- Opera

22. X:nth-child(n)

```
1 li:nth-child(3) {  
2   color: red;  
3 }
```

Remember the days when we had no way to target specific elements in a stack? The `nth-child` pseudo class solves that!

Please note that `nth-child` accepts an integer as a parameter, however, this is not zero-based. If you wish to target the second list item, use `li:nth-child(2)`.

We can even use this to select a variable set of children. For example, we could do `li:nth-child(4n)` to select every fourth list item.

[View Demo](#)

Compatibility

- IE9+
- Firefox 3.5+
- Chrome
- Safari

23. X:nth-last-child(n)

```
1 li:nth-last-child(2) {  
2   color: red;  
3 }
```

What if you had a huge list of items in a `ul`, and only needed to access, say, the third to the last item? Rather than doing `li:nth-child(397)`, you could instead use the `nth-last-child` pseudo class.

This technique works almost identically from number sixteen above, however, the difference is that it begins at the end of the collection, and works its way back.

[View Demo](#)

Compatibility

- IE9+
- Firefox 3.5+
- Chrome
- Safari
- Opera

24. X:nth-of-type(n)

```
1 ul:nth-of-type(3) {  
2   border: 1px solid black;  
3 }
```

There will be times when, rather than selecting a `child`, you instead need to select according to the `type` of element.

Imagine mark-up that contains five unordered lists. If you wanted to style only the third `ul`, and didn't have a unique `id` to hook into, you could use the `nth-of-type(n)` pseudo class. In the snippet above, only the third `ul` will have a border around it.

Compatibility

- IE9+
- Firefox 3.5+
- Chrome
- Safari

25. X:nth-last-of-type(n)

```
1  ul:nth-last-of-type(3) {  
2      border: 1px solid black;  
3  }
```

And yes, to remain consistent, we can also use `nth-last-of-type` to begin at the end of the selectors list, and work our way back to target the desired element.

Compatibility

- IE9+
- Firefox 3.5+
- Chrome
- Safari
- Opera

26. X:first-child

```
1  ul li:first-child {  
2      border-top: none;  
3  }
```

This structural pseudo class allows us to target only the first child of the element's parent. You'll often use this to remove borders from the first and last list items.

For example, let's say you have a list of rows, and each one has a `border-top` and a `border-bottom`. Well, with that arrangement, the first and last item in that set will look a bit odd.

Many designers apply classes of `first` and `last` to compensate for this. Instead, you can use these pseudo classes

these pseudo classes:

[View Demo](#)

Compatibility

- IE7+
- Firefox
- Chrome
- Safari
- Opera

27. X:last-child

```
1  ul > li:last-child {  
2    color: green;  
3  }
```

The opposite of `first-child`, `last-child` will target the last item of the element's parent.

Example

Let's build a simple example to demonstrate one possible use of these classes. We'll create a styled list item.

Markup

```
1  <ul>  
2    <li> List Item </li>  
3    <li> List Item </li>  
4    <li> List Item </li>  
5  </ul>
```

Nothing special here; just a simple list.

CSS

```
01  ul {  
02    width: 200px;  
03    background: #292929;  
04    color: white;  
05    list-style: none;  
06    padding-left: 0;  
07  }  
08  
09  li {
```