tuts+

CODE > WEB DEVELOPMENT

The 30 CSS Selectors You Must Memorize

by Jeffrey Way 25 Sep 2020

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. You owe it to yourself to commit these advanced CSS and CSS3 selectors to memory.

Advertisement

Basic Selectors

1. `*`

```
* {
 margin: 0;
 padding: 0;
}
```

Let's knock the obvious ones out, for the beginners, before we move on to 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 never to use this in production code. It adds too much *weight* on the browser, and is unnecessary.

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

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

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.

2. `#x`

```
#container {
   width: 960px;
   margin: auto;
```

```
}
```

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.

3. `.x`

```
.error {
  color: red;
}
```

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.

4. `x`

```
a { color: red; }
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 {}`.

Advertisement

Live Demo of Basic Selectors

Combinator Selectors

5. `x y`

```
li a {
  text-decoration: none;
}
```

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.

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.*

Advertisement

6. `X + Y`

```
ul + p {
   color: red;
}
```

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.

7. `X > Y`

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

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.

```
<div id="container">
   <ul>
      <li> List Item

        <ul>
           <li> Child </li>
        </ul>
      </li>
      <li> List Item </li>
      <li> List Item </li>
      <li> List Item </li>
   </ul>
</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`.

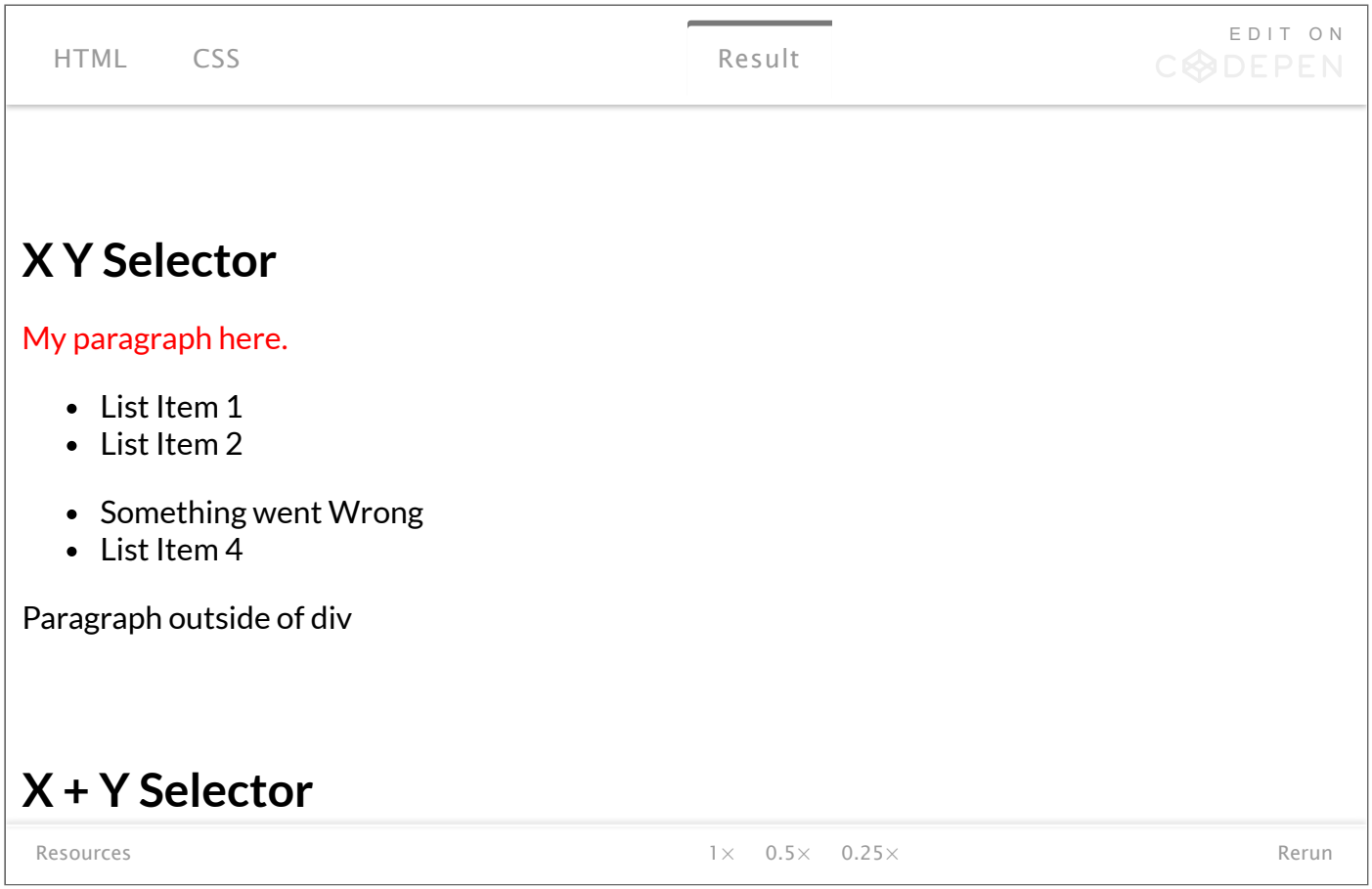
For this reason, there are performance benefits in using the child combinator. In fact, it's recommended particularly when working with JavaScript-based CSS selector engines.

8. `X ~ Y`

```
ul ~ p {
   color: red;
}
```

This sibling combinator is similar to `X + Y`, but 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`.

Live Demo of Combinator Selectors



Attribute Selectors

9. `X[title]`

```
a[title] {
   color: green;
}
```

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? Check out the next example!

10. `X[href="foo"]`

```
a[href="https://code.tutsplus.com"] {
  color: #83b348; /* Envato green */
}
```

The snippet above will style all anchor tags which link to **https://code.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, although it's a bit rigid. What if the link does indeed direct to Envato Tuts+, but maybe the path is **code.tutsplus.com** rather than the full URL? In those cases, we can use a bit of the regular expressions syntax.

11. `X[href*="foo"]`

```
a[href*="tutsplus"] {
  color: #83b348; /* Envato green */
}
```

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 **tutsplus.com**, **code.tutsplus.com**, and even **webdesign.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 **tutsplus** in the URL? When you need to be more specific, use `^` and `$`, to reference the beginning and end of a string, respectively.

12. `X[href^="http"]`

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

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 an `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.

13. `X[href$=".jpg"]`

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

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 won't capture GIF and PNG images.

14. `X[data-*="foo"]`

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

How do we compensate for all of the various image types? Well, we could create multiple selectors, such as:

```
a[href$=".jpg"],
a[href$=".jpeg"],
a[href$=".png"],
a[href$=".gif"] {
   color: red;
}
```

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

attributes. What if we added our own `data-filetype` attribute to each anchor that links to an image?

```
<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.

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

15. `X[foo~="bar"]`

```
a[data-info~="external"] {
   color: red;
}

a[data-info~="image"] {
   border: 1px solid black;
}
```

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

Going along with our custom attribute from number 15, 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.

```
"<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.

```
/* Target data-info attr that contains the value "external" */
a[data-info~="external"] {
   color: red;
}

```

```
/* And which contain the value "image" */
a[data-info~="image"] {
  border: 1px solid black;
}
```

Pretty nifty, huh?

Live Demo of Attribute Selectors

HTML CSS Result EDIT ON CODEPEN

X[title] Selector

- List Item
  - Child
- List Item
- List Item
- List Item

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor.

Resources 1× 0.5× 0.25× Rerun

Pseudo Selectors

16. `X:visited` and `X:link`

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

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

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

on, or "visited".

17. `X:checked`

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

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.

18. `X:after`

The `before` and `after` pseudo classes are great. 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.

```
.clearfix:after {
    content: "";
    display: block;
    clear: both;
    visibility: hidden;
    font-size: 0;
    height: 0;
    }

.clearfix {
   *display: inline-block;
   _height: 1%;
}
```

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.*

19. `X:hover`

```
div:hover {
  background: #e3e3e3;
}
```

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 versions 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.

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

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

20. `X:not(selector)`

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

The negation pseudo class is particularly helpful. Let's say I want to select all `div`s, 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:

```
*:not(p) {
  color: green;
```

```
}
```

21. `X::pseudoElement`

```
p::first-line {
   font-weight: bold;
   font-size: 1.2em;
}
```

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

```
p::first-letter {
   float: left;
   font-size: 2em;
   font-weight: bold;
   font-family: cursive;
   padding-right: 2px;
}
```

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

```
p::first-line {
   font-weight: bold;
   font-size: 1.2em;
}
```

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."—W3C Selectors Specs

Live Demo of Pseudo Selectors

HTML CSS Result EDIT ON CODEPEN

X:visited and X:link

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

X:checked

○ Radio Button

X:not(selector)

Resources 1× 0.5× 0.25× Rerun

Nth Child and Type Selectors

22. `X:nth-child(n)`

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

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, but 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.

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

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

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

This technique works almost identically to number 16 above. The difference is that it begins at the end of the collection, and works its way back.

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

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

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

Imagine markup 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.

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

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

And, so to remain consistent, we can also use `nth-last-of-type` to begin at the end of

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.

26. `X:first-child`

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

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.

27. `X:last-child`

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

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

`last-child` Selector Example

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

```
<ul>
   <li> List Item </li>
   <li> List Item </li>
   <li> List Item </li>
</ul>
```

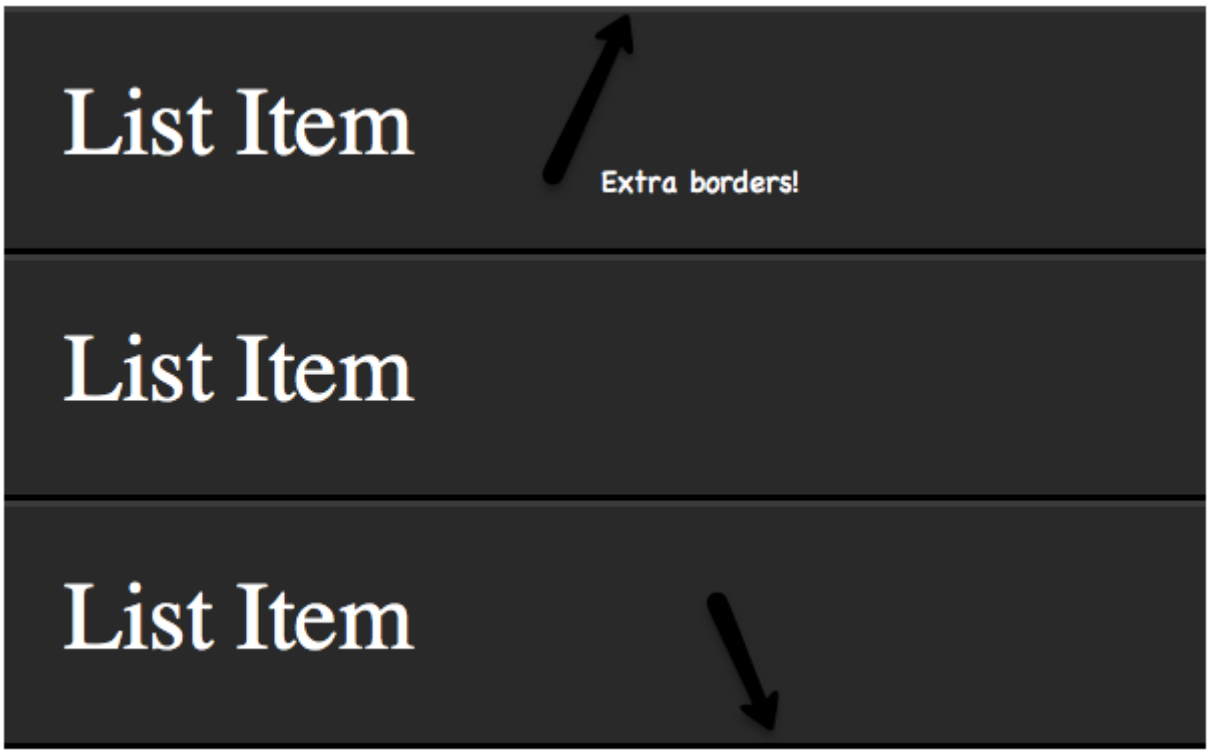
For the markup, there's nothing special: just a simple list.

Here's the CSS:

```
ul {
 width: 200px;
 background: #292929;
 color: white;
 list-style: none;
 padding-left: 0;
}

li {
 padding: 10px;
 border-bottom: 1px solid black;
 border-top: 1px solid #3c3c3c;
}
```

This styling will set a background, remove the browser default padding on the `ul`, and apply borders to each `li` to provide a bit of depth.



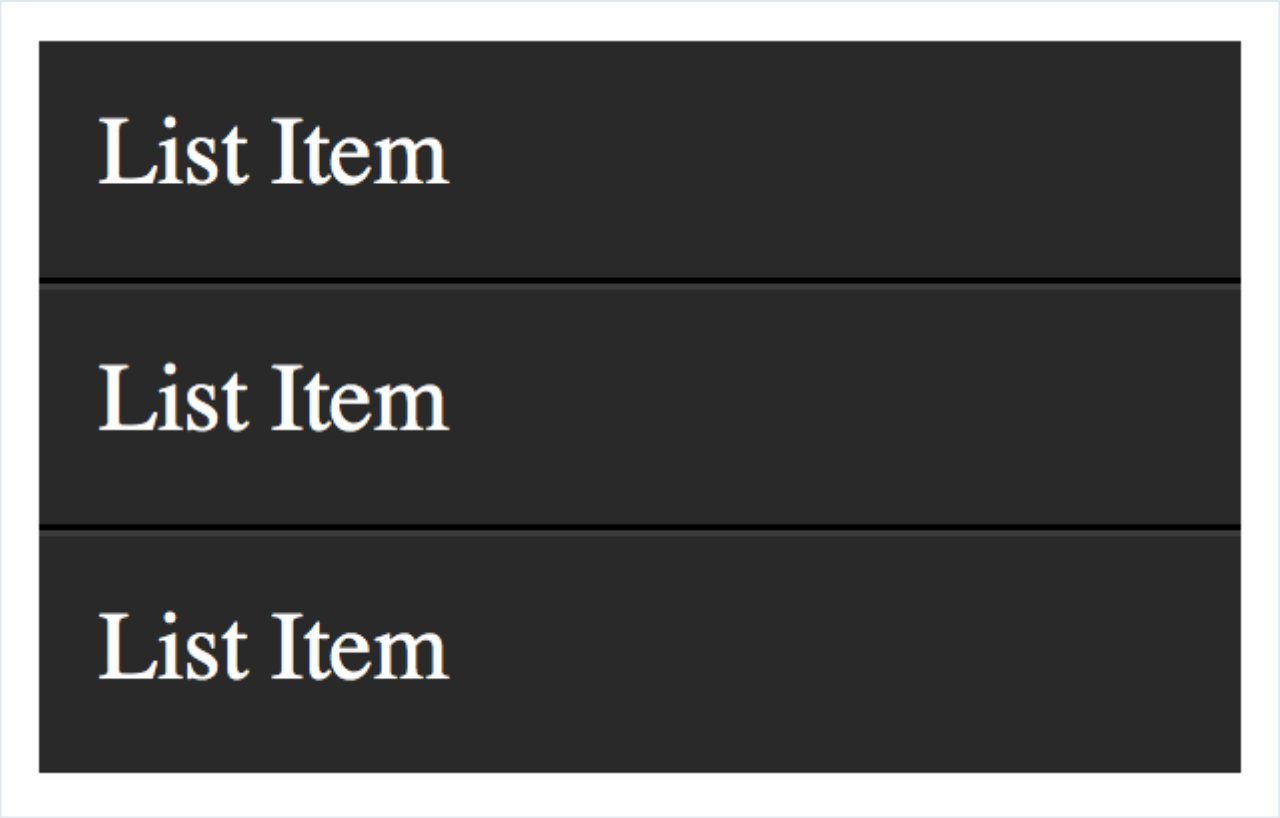
To add depth to your lists, apply a `border-bottom` to each `li` that is a shade or two darker than the `li`'s background color. Next, apply a `border-top` which is a couple of shades lighter.

The only problem, as shown in the image above, is that a border will be applied to the

The only problem, as shown in the image above, is that a border will be applied to the very top and bottom of the unordered list—which looks odd. Let's use the `:first-child` and `:last-child` pseudo classes to fix this.

```
li:first-child {
    border-top: none;
}

li:last-child {
   border-bottom: none;
}
```



There we go; that fixes it!

28. `X:only-child`

```
div p:only-child {
   color: red;
}
```

Truthfully, you probably won't find yourself using the `only-child` pseudo class too often. Nonetheless, it's available, should you need it.

It allows you to target elements which are the *only* child of its parent. For example, referencing the snippet above, only the paragraph that is the only child of the `div` will

Referencing the snippet above, only the paragraph that is the only child of the `div` will be colored red.

Let's assume the following markup.

```
<div><p> My paragraph here. </p></div>

<div>
   <p> Two paragraphs total. </p>
   <p> Two paragraphs total. </p>
</div>
```

In this case, the second `div`'s paragraphs will not be targeted; only the first `div`. As soon as you apply more than one child to an element, the `only-child` pseudo class ceases to take effect.

29. `X:only-of-type`

```
li:only-of-type {
   font-weight: bold;
}
```

This structural pseudo class can be used in some clever ways. It will target elements that do not have any siblings within its parent container. As an example, let's target all `ul`s which have only a single list item.

First, ask yourself how you would accomplish this task. You could do `ul li`, but this would target *all* list items. The only solution is to use `only-of-type`.

```
ul > li:only-of-type {
   font-weight: bold;
}
```

30. `X:first-of-type`

The `first-of-type` pseudo class allows you to select the first siblings of its type.

A Test

To better understand this, let's have a test. Copy the following markup into your code editor:

```
<div>
   <p> My paragraph here. </p>

   <ul>
      <li> List Item 1 </li>
      <li> List Item 2 </li>
   </ul>

   <ul>
      <li> List Item 3 </li>
      <li> List Item 4 </li>
   </ul>
</div>
```

Now, without reading further, try to figure out how to target only "*List Item 2*". When you've figured it out (or given up), read on.

Solution 1

There are a variety of ways to solve this test. We'll review a handful of them. Let's begin by using `first-of-type`.

```
ul:first-of-type > li:nth-child(2) {
   font-weight: bold;
}
```

This snippet essentially says, to find the first unordered list on the page, then find only the immediate children, which are list items. Next, filter that down to only the second list item in that set.

Solution 2

Another option is to use the adjacent selector.

```
p + ul li:last-child {
   font-weight: bold;
}
```

In this scenario, we find the `ul` that immediately proceeds the `p` tag, and then find the very last child of the element.

Solution 3

We can be as obnoxious or as playful as we want with these selectors.

```
ul:first-of-type li:nth-last-child(1) {
   font-weight: bold;
}
```

This time, we grab the first `ul` on the page, and then find the very first list item, but starting from the bottom!

Live Demo of Nth Child and Type Selectors

HTML CSS Result EDIT ON CODEPEN

X:nth-child(n)

- List Item
  - Child
  - Child
  - Child
- List Item
- List Item
- List Item

- Anchor Tag
- Anchor Tag
- Anchor Tag
- Anchor Tag

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do Nettuts tempor. Lorem ipsum

Resources 1× 0.5× 0.25× Rerun

Conclusion

If you're compensating for older browsers, like Internet Explorer 6, you still need to be careful when using these newer selectors. But please don't let that deter you from learning these. You'd be doing a huge disservice to yourself. Be sure to refer here for a browser-compatibility list. Alternatively, you can use Dean Edward's excellent IE9.js script to bring support for these selectors to older browsers.

Secondly, when working with JavaScript libraries, like the popular jQuery, always try to use these native CSS3 selectors over the library's custom methods/selectors, when possible. It'll make your code faster, as the selector engine can use the browser's native