tuts+

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.

1. *

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

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.

```
#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.

View Demo

Compatibility

- IE6+
- Firefox
- Chrome
- Safari
- Opera

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.

View Demo

Compatibility

- IE6+
- Firefox
- Chrome
- Safari
- Opera

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.

View Demo

Compatibility

- IE6+
- Firefox
- Chrome
- Safari

- Opera

4. X Y

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

The next most comment 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

```
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 {}`.

View Demo

Compatibility

- IE6+
- Firefox
- Chrome
- Safari
- Opera

6. X:visited and X:link

```css
a:link { color: red; }
a:visted { 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

```css
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.

View Demo

Compatibility

- IE7+
- Firefox
- Chrome
- Safari
- Opera

8. 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.

View Demo

Compatibility

- IE7+
- Firefox
- Chrome
- Safari
- Opera

9. X ~ Y

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

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]

```css
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? Well...

View Demo

Compatibility

- IE7+
- Firefox
- Chrome
- Safari
- Opera

11. X[href="foo"]

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

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"]

```css
a[href*="tuts"] {
  color: #1f6053; /* nettuts 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 *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"]

```css
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 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"]

```
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 certainly won't work for `gifs` and `pngs`.

View Demo

Compatibility

- IE7+
- Firefox

- Chrome
- Safari
- Opera

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

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

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:

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

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?

```
<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;
}
```

View Demo

Compatibility

- IE7+
- Firefox
- Chrome
- Safari

- Opera

16. 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 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.

```
"<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, ay?

View Demo

Compatibility

- IE7+

- Firefox
- Chrome
- Safari
- Opera

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.

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.

```
.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. 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

```
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 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.

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

> ***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)

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

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:

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

View Demo

Compatibility

- IE9+
- Firefox
- Chrome
- Safari
- Opera

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." - Source*

View Demo

Compatibility

- IE6+
- Firefox
- Chrome
- Safari
- Opera

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, 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)

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

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)

```
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 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.

View Demo

Compatibility

- IE9+
- Firefox 3.5+
- Chrome
- Safari

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

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

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

```css
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.

View Demo

Compatibility

- IE7+
- Firefox
- Chrome
- Safari
- Opera

27. X:last-child

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

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

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

Nothing special here; just a simple list.

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 shades lighter.*

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!

View Demo

Compatibility

- IE9+
- Firefox
- Chrome
- Safari
- Opera

Yep - IE8 supported `:first-child`, but not `:last-child`. Go figure.

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 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.

View Demo

Compatibility

- IE9+

- Firefox
- Chrome
- Safari
- Opera

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;
}
```

View Demo

Compatibility

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

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 mark-up into your code editor:

```html
<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`.

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

This snippet essentially says, "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.

```css
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! :)

View Demo

Compatibility

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

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 parsing, rather than its own.

It’s great that you’re spending time learning web design fundamentals, but if you need a quick solution, one of our ready-to-use CSS templates might be a good option. We also have a few premium CSS items for you to consider:

Premium CSS Items

Here are a handful of the ready-to-use CSS code items on Envato Market that you might find useful.

1. CSS3 Mega Drop Down Menu

This Mega Drop Down Menu is a flexible and easy to integrate solution to build your custom menus. The drop down relies only on CSS/XHTML and comes with a fully working contact form.

CSS3 Mega Drop Down Menu

2. CSS3 Responsive Web Pricing Tables Grids

CSS3 Responsive Web Pricing Tables Grids is a pack of pure CSS3, responsive and retina ready Pricing Tables. The template comes with 2 table styles, 20 predefined color skins, animated hover states, possibility to set one or many columns as active (popped-up) by default, CSS3 ribbons, CSS3 tooltips, 20 font based yes/no icons.

This version is dedicated for any custom CMS based site as well as for non-CMS site and will work with straight HTML or PHP-based websites.

CSS3 Responsive Web Pricing Tables Grids

3. The Tooltip

The Tooltip is a handsome, modern gentleman that appears when it's showtime. Its features include:

- 6 positions
- 12 color schemes
- gracefully degrades in older browsers
- LESS file included so you can create a custom color scheme, generate a compact production version of The Tooltip (less code), and mix it in your LESS stylesheet
- featured on 1stewebdesigner.com and tripwire magazine

The Tooltip

4. MegaNavbar (v 2.2.0). Advanced Mega Menu for Bootstrap 3.0+

MegaNavbar is a pure HTML5/CSS3 navigation component that uses the standard navbar markup and the fluid grid system classes from Bootstrap 3. It works for fixed and responsive layouts, and has the facility to include other Bootstrap components. MegaNavbar is compatible with mobile devices and modern web browsers.

MegaNavbar (v 2.2.0). Advanced Mega Menu for Bootstrap 3.0+

5. Responsive CSS3 Pricing Tables

This pack of CSS3 Pricing Tables is a complete solution for building awesome tables in minutes. It comes with 6 color variants as well as 2 table variants (light or dark).

Responsive CSS3 Pricing Tables

Jeffrey Way

Jeffrey Way

I used to be the editor of Nettuts+ and head of web development courses at Tuts+.

jeffrey_way

FEED LIKE FOLLOW

Weekly email summary

Subscribe below and we'll send you a weekly email summary of all new Code tutorials. Never miss out on learning about the next big thing.

Email Address

Update me weekly

Translations

Envato Tuts+ tutorials are translated into other languages by our community members—you can be involved too!

Translate this post

Powered by

native

624 Comments **Nettuts+** 1 **Login**

Recommend 51 Tweet Share **Sort by Best**

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Craig • 8 years ago

Familiarize NOT memorize.

"Never memorize something you can look up" -- Albert Einstein

262 ^ | ⌄ 5 • Reply • Share ›

Rafał Borowski ➜ Craig • 7 years ago

For a front end developer that is not gonna happen. It's like a mechanic "looking up" every part of engine. You just need to know them.

47 ^ | ⌄ 5 • Reply • Share ›

Seph ➜ Rafał Borowski • 7 years ago

Except a Mechanic, much like a web developer, is not an encyclopedia and therefor must familiarize not memorize. This is actually a well adopted principle in the engineering world. When you advance on from HTML and CSS you'll realize that a part of being a developer is knowing how to properly use Google.

163 ^ | ⌄ 1 • Reply • Share ›

BP ➜ Seph • 6 years ago

Truth..!!

7 ^ | ⌄ • Reply • Share ›

BJ ➜ Seph • 5 years ago

Let's get real here, everybody. You need to memorize some stuff. Familiarize and look it up until it is committed to memory. Of course, over time, the knowledge will fade if you are not using it regularly. Sure, look it up. But if it's something you use on even a somewhat regular basis, you should be memorizing.
My point is, you need both. The issue is not black and white.

4 ^ | ⌄ • Reply • Share ›

Jonas Russel Green ➜ BJ
• 5 years ago • edited

kids should learn this right after the alphabet
http://www.memrise.com/cour... - short, unique, universal
why doesnt css support regex? was regex "abbreviated" in other contexts already than css and google syntax?
ok $= ^= *= seems handy since people know a bit of regex but dont like it? and know +=, -= if they programmed with any

know += -= if they programmed with any numbers - yet you may also used *= or ^= in other languages and double assignments in the head are inefficient? What about ~= and |= in css? Why do they look like combined assignment operators? (how css looks isn't rather for developers than designers?)

- Are there any guides how to learn/achieve the most complete toolset in programming or overall logic thinking - yet with the least contradictons - in other words is there ranking ordered by what allows most "density in brain storage" and what should you avoid and only learn when necessary?

^ | ˅ • Reply • Share ›

Pat ➦ Seph • 3 years ago

Depends on the context... if you are interviewing for a frontend dev job, high chance that they will ask you some kind of css selector question, and get pretty tricky with it... so memorization is necessary!

^ | ˅ • Reply • Share ›

HaakonKL ➦ Seph • 3 years ago

Ask any java developer to list some things that exist in the package "java.util".
Even though our tools look them up for us, I'm willing to bet that most everyone is going to be able to list List, LinkedList, ArrayList, Map, HashMap, and StringBuilder, because they're used every day all the time.

Similar for devs in other languages.

^ | ˅ • Reply • Share ›

mirek ol ➦ Seph • 6 years ago

True

^ | ˅ • Reply • Share ›

Gacek ➦ Rafał Borowski • 7 years ago

Memorizing everything is not the point. The more you use, the more common element is in your work you will memorize it. The less you use it you must only be fammiliar it exists. That's what all docs are for. You must know it exists, and you when you need to use it, you must know how to find it and find usage examples.

Your statement says that I have to learn every PHP function and class and their methods. + all css elements, selectors + all hex colors + all html tags (even deperecated ones?). My head would explode... but I know they exist. Good example are array sorting functions in PHP. You know there are few of them, you don't memorize them, but you know where to look for them :)

12 ^ | ˅ • Reply • Share ›

Dillon Sylvestre ➦ Rafał Borowski • 7 years ago

With that in mind, a front-end developer would never be reading an article like this in the first place.

25 ^ | ˅ 3 • Reply • Share ›

CreadevDotOrg ➦ Dillon Sylvestre • 7 years ago • edited

Yet you're all here....theres no way you memorized every obscure syntax like this. We spend our time where it counts. 70% of these calls are incredibly uncommon and un-needed till they're needed :) They exist as fixes and toolbag selectors for crappy SaaS templates and targeting done using JS (most of the time). Nice article regardless of what these guys think what "pros" should do hah.

20 ^ | ˅ 3 • Reply • Share ›

Dillon Sweeten ➦ CreadevDotOrg • 6 years ago

Unfortunately, this is not obscure syntax in a production environment. A true front-end developer would use these on a daily basis. Furthermore, javascript is not for targeting unless you have no other choice. You should not mislead people.

23 ^ | ˅ • Reply • Share ›

mrExpertCom ➦ Dillon Sweeten • 6 years ago

all of you are nubs, stop thinking you're a web dev using wordpress php and dreamweaver plus buying themes. That's copy/paste nubz!

9 ^ | ˅ 3 • Reply • Share ›

GermanSoundPet ➦ mrExpertCom • 6 years ago

Gotta ask dude, what brought you here? :P

4 ^ | ˅ • Reply • Share ›

Doke ➦ GermanSoundPet • 6 years ago

Smart ASP ...

^ | ˅ • Reply • Share ›

CreadevDotOrg ➦ Dillon Sweeten • 5 years ago • edited

Late reply....CSS doesn't do ALOT. JS does ALOT. Therefore JS targeting counts. Those targets are based on CSS clauses and rules. Misleading? I think not. Have you ever used JS?

1 ^ | ˅ 3 • Reply • Share ›

Brad ➦ CreadevDotOrg • 5 years ago

CSS and JS are completely different. CSS is for styling, which is what this article is about, JS is for user interactions. How can you say that CSS doesn't do a lot? Every website has tons of CSS on it. Additionally, a lot of things that in the past were done through JS can now be achieved better (faster and with less CPU load) through CSS - by using the tips posted in this article. Using :hover and :checked in combination with CSS3 transitions can achieve a lot of really cool affects that were more traditionally handled through JS. Additionally - if I (or most developers) ever needed to target an obscure element without using IDs, it will most likely be done through jQuery, which... USES THE SAME SELECTORS AS CSS. So even if you're

see more

7 ^ | ˅ • Reply • Share ›

CreadevDotOrg ➦ Brad • 5 years ago

Actually this article is not about styles....its about "selectors" as stated in the title. +1 for agreeing with me that these selectors are used constantly with jQuery (or other construct layers on top of JS).

1 ^ | ˅ 2 • Reply • Share ›

John ➦ CreadevDotOrg • 5 years ago

Nothing does "ALOT" because "ALOT" is not a f'ing word.

3 ^ | ⌄ 1 • Reply • Share ›

CreadevDotOrg ➦ John
• 5 years ago • edited

Well arent you just so cute using the 'ol "word card" tactic! Just to let you know: f'ing isn't a word either, nor are the other thousands of combinations of letters we use on a daily basis....yet we all miraculously understand eachother. Using the "word card" to gain intellectual credibility died with web 2.0 my friend....nowadays it just makes you look like a fool with nothing better to contribute. We all know there should be a space, but really, no one cares.

PS: "eachother" isnt a word either but i'm willing to bet you understand it means "each" and "other" in "combo". Also just to clarify, when I used "ol" above it means "old" which is a shorter form of of the older

see more

3 ^ | ⌄ 2 • Reply • Share ›

ijuhyg ➦ CreadevDotOrg
• 5 years ago • edited

Actually, your misuse of English could hardly be more pertinent. Your subsistence method of communication perfectly illustrates what results when a person refuses to memorize key components of a language. In the same way, one need only visit your website to see that you've taken the same approach with CSS. Your rudimentary text-only site (i.e., it doesn't do "ALOT") could work perfectly with a single style sheet and no JavaScript at all, but instead it forces the browser to do several rounds of http requests to pick up a bunch of JavaScript libraries ("poorly crafted 3rd party tools for which you have no proper access"?) that it barely uses, without which it completely breaks down. Dillon Sweeten's words bear repeating: "You

see more

^ | ˅ • Reply • Share ›

CreadevDotOrg ➦ ijuhyg
• 5 years ago • edited

Arguing about engrish on the internet!!!

I make apps, often for SaaS. Regardless of what you "think", there are the real world use cases of these selectors. Perhaps you should try actually getting your feet wet before assuming all schemas are the same as your own (often the app is constructed purely in JS anyways....BOOOOM)

^ | ˅ • Reply • Share ›

ijuhyg ➦ CreadevDotOrg • 5 years ago

Listing facts does not constitute arguing. I won't bother explaining why. What I think doesn't come into this at all. The "schema" of your particular website is an objectively poor choice, indicating a lack of experience with CSS, which would be unremarkable except that you're trying to portray yourself as an expert . . . pew, pew.

^ | ˅ • Reply • Share ›

CreadevDotOrg ➦ Dillon Sweeten
• 5 years ago • edited

A daily basis if you were locked out of your theme and you have no concept of how to properly wrap + loop on the parse side? To me it sounds like your example "production environment" includes poorly crafted 3rd party tools for which you have no proper access, therefore you must rely on obscure targets. In a better environment you would find that classing, nesting, wrapping, and looping at parse level would invalidate 90% of these complex selectors.

^ | ˅ 1 • Reply • Share ›

Justin ➦ CreadevDotOrg • 5 years ago

I hate the person above you as well...

1 ^ | ˅ 1 • Reply • Share ›

Ian ➦ CreadevDotOrg • 6 months ago

IMHO these are not "complex selectors". They make full logical sense, and I'm all for reducing resource use (thus making the internet a better place) by using them.

QUICK LINKS - Explore popular categories

ENVATO TUTS+ ✚

JOIN OUR COMMUNITY ✚

HELP ✚

envato-tuts+

28,645 Tutorials 1,277 Courses 40,456 Translations

Envato.com Our products Careers Sitemap

© 2020 Envato Pty Ltd. Trademarks and brands are the property of their respective owners.

Follow Envato Tuts+ Facebook Twitter Pinterest