**Element Selectors**

Elements of a document serer as the most basic selector.

**Declarations and Keywords**

The declaration block contains one or more declarations. A declaration is always formatted as a property followed by a colon and then a value followed by a semicolon.

These space-separated words are referred to as keywords because, taken together, they form the value of the property in question.

consider the following fictional rule:

rainbow: red orange yellow green blue indigo violet;

There is no such property as rainbow, but the example is useful for illustrative purposes. The value of rainbow is red orange yellow green blue indigo violet, and the seven keywords add up to a single, unique value.

**What if you want to more than one style to an element or group of elements?**

**Grouping Selectors:**

h2, p {color: gray;}

By placing the h2 and p selectors on the left side of the rule and separating them with a comma, you’ve defined a rule where the style on the right (color: gray;) applies to the elements referenced by both selectors.

**THE UNIVERSAL SELECTOR**

CSS2 introduced a new simple selector called the *universal selector*, displayed as an asterisk (\*). This selector matches any element at all, much like a wildcard. For example, to make every single element in a document red, you would write:

\* {color: red;}

Note that using semicolons at the end of each declaration is crucial when you’re grouping them. Browsers ignore whitespace in stylesheets, so the user agent must rely on correct syntax to parse the stylesheet.

**Class and ID Selector**

Useful when you want to style specific set of elements (certain number of elements of the same type)

.warning {font-weight: *bold*;}

Here elements with class “warning” are applied with font-weight of “bold”. There is a universal selector which is applied implicitly.

Note: The universal selector, represented by \*, is implied when an ID, class, attribute selector, pseudo-class or pseudo-element selector is written without being attached to an element selector.

As you can see, the class selector works by directly referencing a value that will be found in the class attribute of an element. This reference is always preceded by a period (.), which marks it as a class selector.

p.warning {font-weight: *bold*;}

The above select only those ‘p’ elements with class of ‘warning’.

## **Multiple Classes**

Now let’s say you want all elements with a class of warning to be boldfaced, those with a class of urgent to be italic, and those elements with both values to have a silver background. This would be written as follows:

.warning {font-weight: *bold*;}

.urgent {font-style: *italic*;}

.warning.urgent {background: silver;}

By chaining multiple class selectors together, you can select only those elements that have all class names (mentioned in the selector), in any order.

**ID Selector**

**Deciding between class and id**

You may assign classes to any number of elements IDs, on the other hand, should be used once, and only once, within an HTML document. IDs carry more weight than classes when styles are applied.

Also note that class and ID selectors may be case-sensitive, depending on the document language. HTML defines class and ID values to be case-sensitive, so the capitalization of your class and ID values must match that found in your documents.

On a purely syntactical level, the dot-class notation (e.g., .warning) is not guaranteed to work for XML documents. The hash-ID notation (e.g., #lead) will work in any document language that has an attribute that enforces uniqueness within a document. Uniqueness can be enforced with an attribute called id, or indeed anything else, as long as the attribute’s contents are defined to be unique within the document.

**Attribute Selectors:**

Some languages permit while other markup languages may not have ‘class’ or ‘id’ attributes on the element. To address this situation, CSS2 introduced attribute selectors, which can be used to select elements based on their attributes and the values of those attributes. There are four general types of attribute selectors: simple attribute selectors, exact attribute value selectors, partial-match attribute value selectors, and leading-value attribute selectors.

If you want to select elements that have a certain attribute, regardless of that attribute’s value, you can use a simple attribute selector.

h1[class] {color: silver;}

Attribute selectors are highly useful for XML documents

**Chaining Attributes**

It is also possible to select based on the presence of more than one attribute. You do this by chaining the attribute selectors together. For example, to boldface the text of any HTML hyperlink that has both an href and a title attribute, you would write:

a[href][title] {font-weight: *bold*;}

**Selection Based on Exact Attribute Value**

Suppose you want to select only those planet elements that have a value of 1 for the attribute moons:

planet[moons**=**"1"] {font-weight: *bold*;}

As with attribute selection, you can chain together multiple attribute-value selectors to select a single document. For example, to double the size of the text of any HTML hyperlink that has both an href with a value of <http://www.w3.org/> and a title attribute with a value of W3C Home, you would write:

a[href**=**"http://www.w3.org/"][title**=**"W3C Home"] {font-size: 200**%**;}

**What if the there are multiple values (let’s say separated by whitespaces) for an attribute?**

Matching becomes an issue when the selector form encounters values that can in turn contain a space-separated list of values (e.g., the HTML attribute class). For example, consider the following markup fragment:

<planet type="barren rocky">Mercury</planet>

The only way to match this element based on its exact attribute value is to write:

planet[type**=**"barren rocky"] {font-weight: *bold*;}

If you were to write planet[type="barren"], the rule would not match the example markup and thus would fail. This is true even for the class attribute in HTML.

<p class="urgent warning">When handling plutonium, care must be taken to

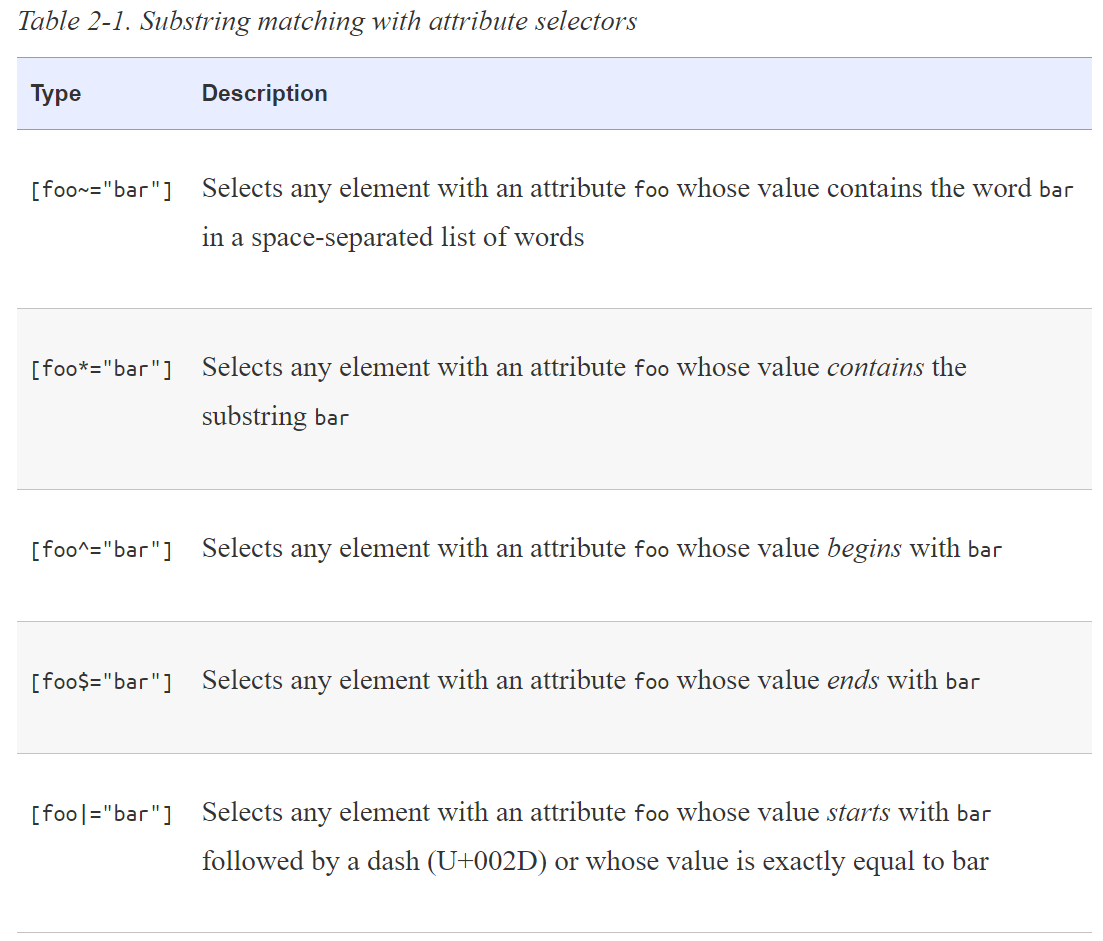
    avoid the formation of a critical mass.</p>

To select this element based on its exact attribute value, you would have to write:

p[class**=**"urgent warning"] {font-weight: *bold*;}

it selects any p element whose class attribute has exactly the value "urgent warning", with the words in that order and a single space between them.

**What if I want to select an element based on Partial Attribute Values?**

****

**Consider the following rule:**

\*[lang**|=**"en"] {color: white;}

This rule will select any element whose lang attribute is equal to en or begins with en-. Therefore, the first three elements in the following example markup would be selected, but the last two would not:

<h1 lang="en">Hello!</h1>

<p lang="en-us">Greetings!</p>

<div lang="en-au">G'day!</div>

<p lang="fr">Bonjour!</p>

<h4 lang="cy-en">Jrooana!</h4>

For any attribute that accepts a space-separated list of words, it is possible to select elements based on the presence of any one of those words. The classic example in HTML is the class attribute, which can accept one or more words as its value. Consider our usual example text:

<p class="urgent warning">When handling plutonium, care must be taken to

    avoid the formation of a critical mass.</p>

Let’s say you want to select elements whose class attribute contains the word warning. You can do this with an attribute selector:

p[class**~=**"warning"] {font-weight: *bold*;}

 The attribute names and values must be case-sensitive only if the underlying document language requires case sensitivity. Class names, titles, URLs, and ID values are all case-sensitive, but HTML attribute keyterm values, such as input types, are not:

<input type="checkbox" name="rightmargin" value="10px">

input[type**=**"CHeckBoX"] {margin-right: 10**px**;}

## The Case Insensitivity Identifier

CSS Selectors level 4 introduces a case-insensitivity option to attribute selectors. Including an i before the closing bracket will allow the selector to match attribute values case-insensitively, regardless of document language rules.

For example, suppose you want to select all links to PDF documents, but you don’t know if they’ll end in .pdf, .PDF, or even .Pdf. Here’s how:

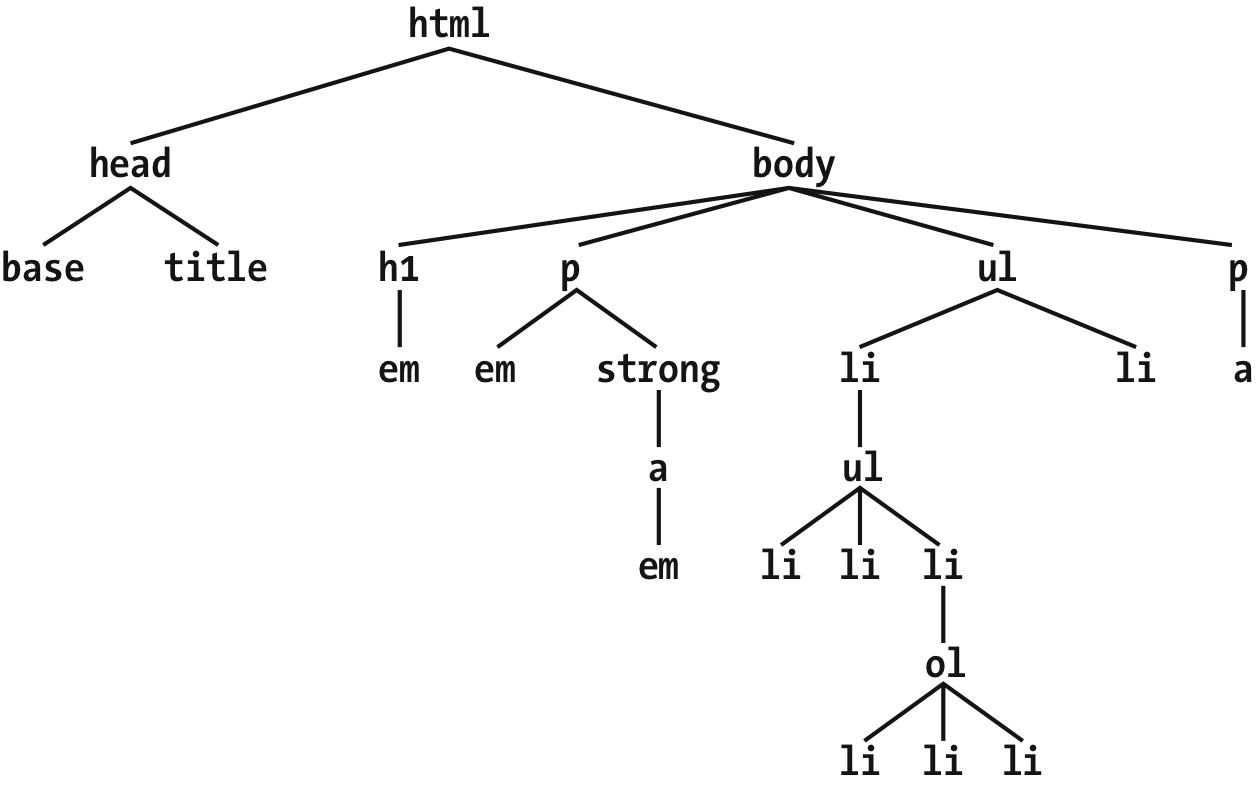
a[href**$=**'.PDF' i]

Note, however, that this only applies to the values in the attribute selectors. It does not enforce case insensitivity on the attribute names themselves.

Again, that’s in languages that enforce case sensitivity in the element and attribute syntax. XHTML was one such. In languages that are case-insensitive, like HTML5, this isn’t an issue.

**Understanding Parent Child Relationship**

Much of the power of CSS is based on the parent-child relationship of elements. HTML documents (actually, most structured documents of any kind) are based on a hierarchy of elements, which is visible in the “tree” view of the document



An element is said to be the parent of another element if it appears directly above that element in the document hierarchy.

 Conversely, an element is the child of another element if it is directly beneath the other element.

The terms “parent” and “child” are specific applications of the terms ancestor and descendant. There is a difference between them: in the tree view, if an element is exactly one level above or below another, then they have a parent-child relationship. If the path from one element to another is traced through two or more levels, the elements have an ancestor-descendant relationship, but not a parent-child relationship. (A child is also a descendant, and a parent is also an ancestor.)

**Descendant Selectors (Contextual Selectors)**

As an example, let’s say you want to style only those em elements that are descended from h1 elements.

h1 em {color: gray;}

In a descendant selector, the selector side of a rule is composed of two or more space-separated selectors. The space between the selectors is an example of a combinator. Each space combinator can be translated as “found within,” “which is part of,” or “that is a descendant of,” but only if you read the selector right to left. Thus, h1 em can be translated as, “Any em element that is a descendant of an h1 element.”

Here’s another example: let’s say that you want gray to be the text color of any b (boldface) element that is part of a blockquote and for any bold text that is found in a normal paragraph:

blockquote b, p b {color: gray;}

One overlooked aspect of descendant selectors is that the degree of separation between two elements can be practically infinite.

A more subtle aspect of descendant selectors is that they have no notion of element proximity. In other words, the closeness of two elements within the document tree has no bearing on whether a rule applies or not.

**Selecting Children**

What if you want to select a children instead of descendant?

To do this, you use the child combinator, which is the greater-than symbol (>):

h1 **>** strong {color: red;}

You can also combine descendant and child combinations in the same selector. Thus, table.summary td > p will select any p element that is a child of a td element that is itself descended from a table element that has a class attribute containing the word summary.

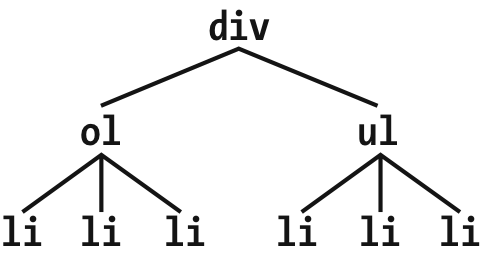
**Selecting Adjacent Sibling Elements**

 To select an element that immediately follows another element with the same parent, you use the *adjacent-sibling combinator*, represented as a plus symbol (+). As with the child combinator, the symbol can be surrounded by whitespace, or not, at the author’s discretion.

To remove the top margin from a paragraph immediately following an h1 element, write:

h1 **+** p {margin-top: 0;}

The selector is read as, “Selects any p element that immediately follows an h1 element that shares a parent with the p element.”



In this fragment, a pair of lists descends from a div element, one ordered and the other not, each containing three list items. Each list is an adjacent sibling, and the list items themselves are also adjacent siblings. However, the list items from the first list are *not* siblings of the second, since the two sets of list items do not share the same parent element. (At best, they’re cousins, and CSS has no cousin selector.)

Remember that you can select the second of two adjacent siblings only with a single combinator. Thus, if you write li + li {font-weight: bold;}, only the second and third items in each list will be boldfaced.

To work properly, CSS requires that the two elements appear in “source order.” In our example, an ol element is followed by a ul element. This allows us to select the second element with ol + ul, but we cannot select the first using the same syntax. For ul + ol to match, an ordered list must immediately follow an unordered list.

**What if there is text between siblings?**

It doesn’t matter since the text is part of the parent element and not meant for siblings. However, if you wrap the text in an element, than that element becomes the adjacent sibling.

## Selecting Following Siblings

Selectors Level 3 introduced a new sibling combinator called the general sibling combinator. This lets you select any element that follows another element when both elements share the same parent, represented using the tilde (~) combinator.

As an example, to italicize any ol that follows an h2 and also shares a parent with the h2, you’d write

h2 **~**ol {font-style: *italic*;}

**Psuedo Class Selectors**

## Combining Pseudo-Classes

Before we start, a word about chaining. CSS makes it possible to combine (“chain”) pseudo-classes together. For example, you can make unvisited links red when they’re hovered and visited links maroon when they’re hovered:

a:link:hover {color: red;}

a:visited:hover {color: maroon;}

The order you specify doesn’t actually matter; you could also write a:hover:link to the same effect as a:link:hover.

Be careful not to combine mutually exclusive pseudo-classes. For example, a link cannot be both visited and unvisited, so a:link:visited doesn’t make any sense and will never match anything.

The majority of pseudo-classes are structural in nature; that is, they refer to the markup structure of the document. Most of them depend on patterns within the markup, such as choosing every third paragraph, but others allow you to address specific types of elements. All pseudo-classes, without exception, are a word preceded by a single colon (:), and they can appear anywhere in a selector.

Before we get started, there’s an aspect of pseudo-classes that needs to be made explicit here: pseudo-classes always refer to the element to which they’re attached, and no other.

**Selecting the Root Element**

This is the quintessence of structural simplicity: the pseudo-class :root selects the root element of the document. In HTML, this is always the html element.

In HTML documents, you can always select the html element directly, without having to use the :root pseudo-class. There is a difference between the two selectors in terms of specificity, which we’ll cover in [Chapter 3](https://learning.oreilly.com/library/view/css-the-definitive/9781449325053/ch03.html#selectors_comma_specificity_comma).

**Selecting empty Elements**

With the pseudo-class :empty, you can select any element that has no children of any kind, including text nodes, which covers both text and whitespace.

Of the following elements, only the first and last would be matched by p:empty: The second and third paragraphs are not matched by :empty because they are not empty: they contain, respectively, a single space and a single newline character. Both are considered text nodes, and thus prevent a state of emptiness.

<p></p>

<p> </p>

<p>

</p>

<p><!—-a comment--></p>

### Selecting unique children

If you’ve ever wanted to select all the images that are wrapped by a hyperlink element, the :only-child pseudo-class is for you.

It selects elements when they are the only child element of another element.

**There are two things to remember about :only-child. The first is that you always apply it to the element you want to be an only child, not to the parent element, as explained earlier. And that brings up the second thing to remember, which is that when you use :only-child in a descendant selector, you aren’t restricting the elements listed to a parent-child relationship.**

To go back to the hyperlinked-image example, a[href] img:only-child matches any image that is an only child and is descended from an a element, not is a child of an a element. To match, the element image must be the only child of its direct parent, and a descendant of a link, but that parent can itself be a descendant of that link. Therefore, all three of the images here would be matched,

a[href] img:only-child {border: 5**px** *solid* black;}

<a href="http://w3.org/"><img src="w3.png" alt="W3C"></a>

<a href="http://w3.org/"><span><img src="w3.png" alt="W3C"></span></a>

<a href="http://w3.org/">A link to <span>the <img src="w3.png" alt="">

   web</span> site</a>

what if you want to match images that are the only images inside hyperlinks, but there are other elements in there with them? Consider the following:

<a href="http://w3.org/"**>**<b**>**•</b**>**<img src="w3.png" alt="W3C"**>**</a**>**

In this case, we have an a element that has two children: b and img. That image, no longer being the only child of its parent (the hyperlink), can never be matched using :only-child. However, it can be matched using :only-of-type.

a[href] img:only-of-type {border: 5**px** *solid* black;}

<a href="http://w3.org/"><b>•</b><img src="w3.png" alt="W3C"></a>

<a href="http://w3.org/"><span><b>•</b><img src="w3.png" alt="W3C"></span></a>

The difference is that :only-of-type will match any element that is the only of its type among all its siblings, whereas :only-child will only match if an element has no siblings at all.

section **>** h2 {margin: 1**em** 0 0.33**em**; font-size: 1.8**rem**; border-bottom: 1**px** *solid*

    gray;}

 section **>** h2:only-of-type {font-size: 2.4**rem**;}

Given those rules, any section that has only one child h2 will have it appear larger than usual. If there are two or more h2 children to a section, neither of them will be larger than the other. The presence of other children—whether they are other heading levels, paragraphs, tables, paragraphs, lists, and so on—will not interfere with matching.

There’s one more thing to make clear, which is that :only-of-type refers to elements and nothing else. Consider the following:

p.unique:only-of-type {color: red;}

<div>

    <p class="unique">This paragraph has a 'unique' class.</p>

    <p>This paragraph doesn't have a class at all.</p>

</div>

The class name is irrelevant here. We’re fooled into thinking that “type” is a generic description, because of how we parse language. Type, in the way :only-of-type means it, refers only to the element type. Thus, p.unique:only-of-type means “select any p element whose class attribute contains the word unique when the p element is the only p element among its siblings.” It does not mean “select any p element whose class attribute contains the word unique when it’s the only sibling paragraph to meet that criterion.”

**Selecting First and Last Children**

This can be accomplished by :first-child

p:first-child {font-weight: *bold*;}

li:first-child {text-transform: *uppercase*;}

The first rule boldfaces any p element that is the first child of another element. The second rule uppercases any li element that is the first child of another element (which, in HTML, must be either an ol or ul element).

As has been mentioned, the most common error is assuming that a selector like p:first-child will select the first child of a p element. Remember the nature of pseudo-classes, which is to attach a sort of phantom class to the element associated with the pseudo-class. If you were to add actual classes to the markup, it would look like this:

The mirror image of :first-child is :last-child.

p:last-child {font-weight: *bold*;}

li:last-child {text-transform: *uppercase*;}

Interestingly, you can combine these two pseudo-classes to create a version of :only-child. The following two rules will select the same elements:

p:only-child {color: red;}

p:first-child:last-child {background-color: red;}

**Selecting First and Last of Type**

table:first-of-type {border-top: 2**px** *solid* gray;}

Note that this does not apply to the entire document; that is, the rule shown will not select the first table in the document and skip all the others. It will instead select the first table element within each element that contains one, and skip any sibling table elements that come after the first.

Similarly “:last-of-type” selects the last instance of a given type amongst its sibling elements.

Similar to what was noted in the previous section, you can combine these two pseudo-classes to create a version of :only-of-type. The following two rules will select the same elements:

table:only-of-type{color: red;}

table:first-of-type:last-of-type {background: red;}

### Selecting every nth child

p:nth-child(1) {font-weight: *bold*;}

li:nth-child(1) {text-transform: *uppercase*;}

**nth-last-child**

As you might expect, there is a corresponding pseudo-class in :nth-last-child(). This lets you do the same thing as :nth-child(), except with :nth-last-child() you start from the last element in a list of siblings and count backward toward the beginning. If you’re intent on highlighting every other table row and making sure the very last row is one of the rows in the highlighting pattern, either one of these will work for you:

tr:nth-last-child(odd) {background: silver;}

tr:nth-last-child(2n+1) {background: silver;} */\* equivalent \*/*

### Selecting every nth of a type

You can, for example, select every other hyperlink that’s a child of any given paragraph, starting with the second, using p > a:nth-of-type(even).

p **>** a:nth-of-type(even) {background: blue; color: white;}

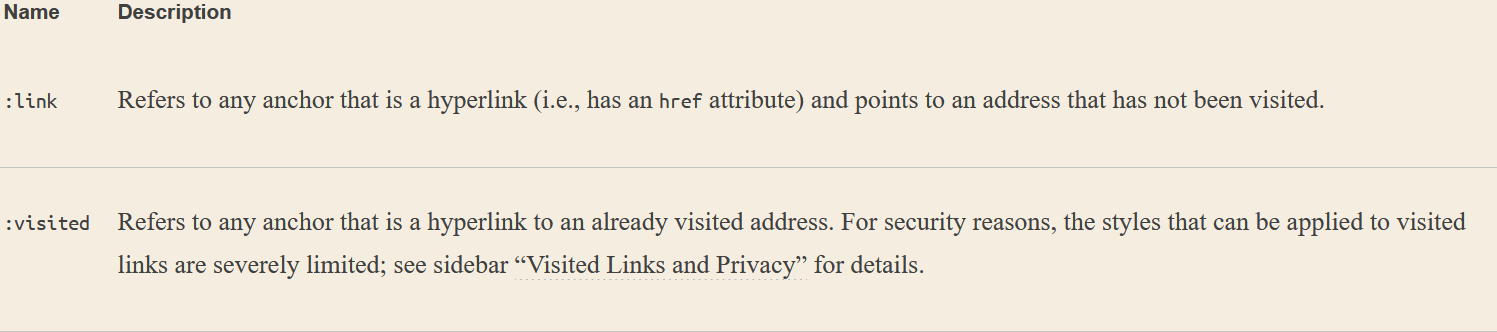
If you want to work from the last hyperlink backward, then you’d use p > a:nth-last-of-type(even).

As before, these select elements of a type from among their sibling elements, *not* from among all the elements of a type within the entire document as a single group. Each element has its own list of siblings, and selections happen within each group.

As you might expect, you can string these two together as :nth-of-type(1):nth-last-of-type(1) to restate :only-of-type, only with higher specificity. (We *will* explain specificity in [Chapter 3](https://learning.oreilly.com/library/view/css-the-definitive/9781449325053/ch03.html#selectors_comma_specificity_comma), I promise.)

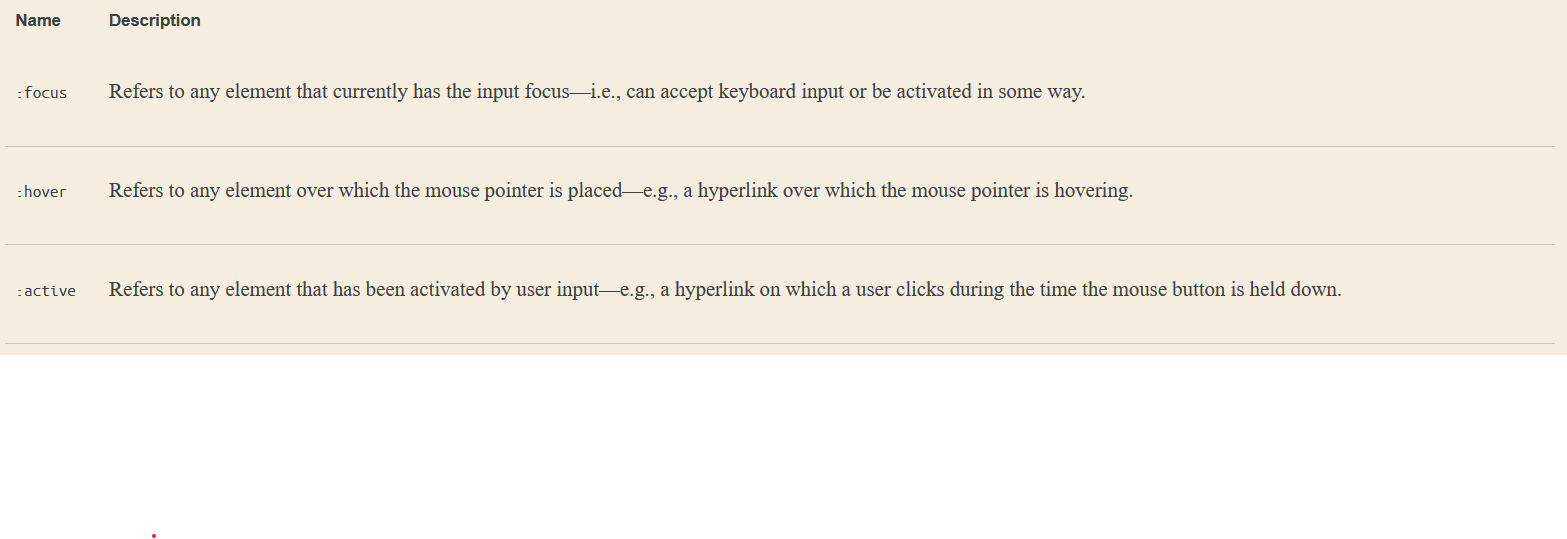
**Dynamic Pseudo Classes**

Beyond the structural pseudo-classes, there are a set of pseudo-classes that relate to structure but can change based on changes made to the page after it’s been rendered. In other words, the styles are applied to pieces of a document based on something in addition to the structure of the document, and in a way that cannot be precisely deduced simply by studying the document’s markup.

****

The above pseudo classes are valid for hyperlink (‘a’) elements only.

### User action pseudo-classes

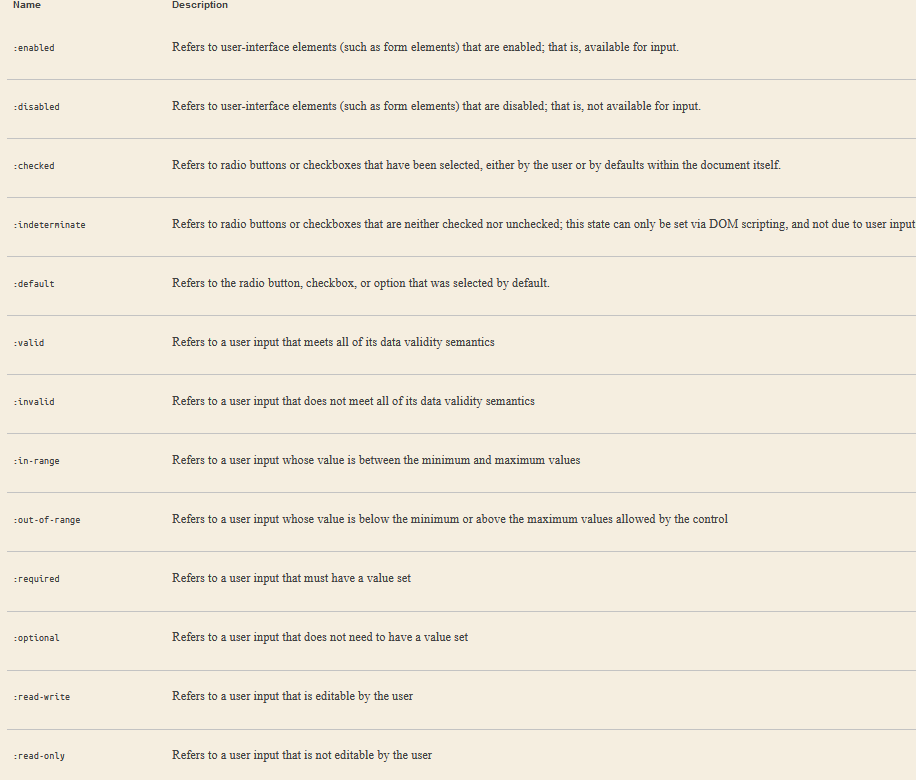


The order of the pseudo-classes is more important than it might seem at first. The usual recommendation is “link-visited-hover-active,” although this has been modified to “link-visited-focus-hover-active.”

Dynamic pseudo classes are applied to any element as oppose to structural pseudo classes which are applied on the element to which it is attached.

While you can style elements with :focus any way you like, do not remove all styling from focused elements. Differentiating which element currently has focus is vital for accessibility, especially for those navigating your site or application with a keyboard.

## UI-State Pseudo-Classes



There is a third state, “indeterminate.” As of late 2017, this state can only be set through DOM scripting or by the user agent itself; there is no markup-level method to set elements to an indeterminate state. The purpose of allowing an indeterminate state is to visually indicate that the element needs to be checked (or unchecked) by the user. However, note that this is purely a visual effect: it does not affect the underlying state of the UI element, which is either checked or unchecked, depending on document markup and the effects of any DOM scripting.