Chapter 4: Selectors

CSS selectors identify specific HTML elements as targets for CSS styles. This topic covers how CSS selectors target HTML elements. Selectors use a wide range of over 50 selection methods offered by the CSS language, including elements, classes, IDs, pseudo-elements and pseudo-classes, and patterns.

Section 4.1: Basic selectors

Selector	Description
`*`	Universal selector (all elements)
`div`	Tag selector (all `<div>` elements)
`.blue`	Class selector (all elements with class `blue`)
`.blue.red`	All elements with class `blue` **and** `red` (a type of Compound selector)
`#headline`	ID selector (the element with "id" attribute set to `headline`)
`:pseudo-class`	All elements with pseudo-class
`::pseudo-element`	Element that matches pseudo-element
`:lang(en)`	Element that matches :lang declaration, for example `<span lang="en">`
`div > p`	child selector

> **Note:** The value of an ID must be unique in a web page. It is a violation of the HTML standard to use the value of an ID more than once in the same document tree.

A complete list of selectors can be found in the CSS Selectors Level 3 specification.

Section 4.2: Attribute Selectors

Overview

Attribute selectors can be used with various types of operators that change the selection criteria accordingly. They select an element using the presence of a given attribute or attribute value.

Selector(1)	Matched element	Selects elements...	CSS Version
`[attr]`	`<div attr>`	With attribute `attr`	2
`[attr='val']`	`<div attr="val">`	Where attribute `attr` has value `val`	2
`[attr~='val']`	`<div attr="val val2 val3">`	Where `val` appears in the whitespace-separated list of `attr`	2
`[attr^='val']`	`<div attr="val1 val2">`	Where `attr`'s value *begins* with `val`	3
`[attr$='val']`	`<div attr="sth aval">`	Where the `attr`'s value *ends* with `val`	3
`[attr*='val']`	`<div attr="somevalhere">`	Where `attr` contains `val` anywhere	3
`[attr|='val']`	`<div attr="val-sth etc">`	Where `attr`'s value is exactly `val`, or starts with `val` and immediately followed by `-` (U+002D)	2
`[attr='val' i]`	`<div attr="val">`	Where `attr` has value `val`, ignoring `val`'s letter casing.	4(2)

Notes:

1. The attribute value can be surrounded by either single-quotes or double-quotes. No quotes at all may also work, but it's not valid according to the CSS standard, and is discouraged.

2. There is no single, integrated CSS4 specification, because it is split into separate modules. However, there are "level 4" modules. See browser support.

Details
`[attribute]`

Selects elements with the given attribute.

```css
div[data-color] {
  color: red;
}
```

```html
<div data-color="red">This will be red</div>
<div data-color="green">This will be red</div>
<div data-background="red">This will NOT be red</div>
```

Live Demo on JSBin

`[attribute="value"]`

Selects elements with the given attribute and value.

```css
div[data-color="red"] {
  color: red;
}
```

```html
<div data-color="red">This will be red</div>
<div data-color="green">This will NOT be red</div>
<div data-color="blue">This will NOT be red</div>
```

Live Demo on JSBin

`[attribute*="value"]`

Selects elements with the given attribute and value where the given attribute contains the given value anywhere (as a substring).

```css
[class*="foo"] {
  color: red;
}
```

```html
<div class="foo-123">This will be red</div>
<div class="foo123">This will be red</div>
<div class="bar123foo">This will be red</div>
<div class="barfooo123">This will be red</div>
<div class="barfo0">This will NOT be red</div>
```

Live Demo on JSBin

`[attribute~="value"]`

Selects elements with the given attribute and value where the given value appears in a whitespace-separated list.

```css
[class~="color-red"] {
  color: red;
}
```

```html
<div class="color-red foo-bar the-div">This will be red</div>
<div class="color-blue foo-bar the-div">This will NOT be red</div>
```

Live Demo on JSBin

`[attribute^="value"]`

Selects elements with the given attribute and value where the given attribute begins with the value.

```css
[class^="foo-"] {
  color: red;
}
```

```html
<div class="foo-123">This will be red</div>
<div class="foo-234">This will be red</div>
<div class="bar-123">This will NOT be red</div>
```

Live Demo on JSBin

`[attribute$="value"]`

Selects elements with the given attribute and value where the given attribute ends with the given value.

```css
[class$="file"] {
  color: red;
}
```

```html
<div class="foobar-file">This will be red</div>
<div class="foobar-file">This will be red</div>
<div class="foobar-input">This will NOT be red</div>
```

Live Demo on JSBin

`[attribute|="value"]`

Selects elements with a given attribute and value where the attribute's value is exactly the given value or is exactly the given value followed by `-` (U+002D)

```css
[lang|="EN"] {
  color: red;
}
```

```html
<div lang="EN-us">This will be red</div>
<div lang="EN-gb">This will be red</div>
<div lang="PT-pt">This will NOT be red</div>
```

Live Demo on JSBin

`[attribute="value" i]`

Selects elements with a given attribute and value where the attribute's value can be represented as `Value`, `VALUE`, `vAlUe` or any other case-insensitive possibility.

```css
[lang="EN" i] {
  color: red;
}
```

```html
<div lang="EN">This will be red</div>
<div lang="en">This will be red</div>
<div lang="PT">This will NOT be red</div>
```

Live Demo on JSBin

Specificity of attribute selectors
`0-1-0`

Same as class selector and pseudoclass.

```
*[type=checkbox] // 0-1-0
```

Note that this means an attribute selector can be used to select an element by its ID at a lower level of specificity than if it was selected with an ID selector: `[id="my-ID"]` targets the same element as `#my-ID` but with lower specificity.

See the Syntax Section for more details.

Section 4.3: Combinators

Overview

Selector	Description
`div span`	Descendant selector (all **`<span>`**s that are descendants of a **`<div>`**)
`div > span`	Child selector (all **`<span>`**s that are a direct child of a **`<div>`**)
`a ~ span`	General Sibling selector (all **`<span>`**s that are siblings after an **`<a>`**)
`a + span`	Adjacent Sibling selector (all **`<span>`**s that are immediately after an **`<a>`**)

> **Note:** Sibling selectors target elements that come after them in the source document. CSS, by its nature (it cascades), cannot target *previous* or *parent* elements. However, using the flex `order` property, a previous sibling selector can be simulated on visual media.

Descendant Combinator: `selector selector`

A descendant combinator, represented by at least one space character (), selects elements that are a descendant of the defined element. This combinator selects **all** descendants of the element (from child elements on down).

```
div p {
  color:red;
}
```

```
<div>
  <p>My text is red</p>
  <section>
    <p>My text is red</p>
  </section>
</div>

<p>My text is not red</p>
```

Live Demo on JSBin

In the above example, the first two **`<p>`** elements are selected since they are both descendants of the **`<div>`**.

Child Combinator: `selector > selector`

The child (`>`) combinator is used to select elements that are **children**, or **direct descendants**, of the specified element.

```
div > p {
  color:red;
}
```

```
<div>
  <p>My text is red</p>
  <section>
    <p>My text is not red</p>
  </section>
</div>
```

Live Demo on JSBin

The above CSS selects only the first `<p>` element, as it is the only paragraph directly descended from a `<div>`.

The second `<p>` element is not selected because it is not a direct child of the `<div>`.

Adjacent Sibling Combinator: `selector + selector`

The adjacent sibling (+) combinator selects a sibling element that immediate follows a specified element.

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

```
<p>My text is not red</p>
<p>My text is red</p>
<p>My text is red</p>
<hr>
<p>My text is not red</p>
```

Live Demo on JSBin

The above example selects only those `<p>` elements which are *directly preceded* by another `<p>` element.

General Sibling Combinator: `selector ~ selector`

The general sibling (~) combinator selects *all* siblings that follow the specified element.

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

```
<p>My text is not red</p>
<p>My text is red</p>
<hr>
<h1>And now a title</h1>
<p>My text is red</p>
```

Live Demo on JSBin

The above example selects all `<p>` elements that are *preceded* by another `<p>` element, whether or not they are immediately adjacent.

Section 4.4: Pseudo-classes

Pseudo-classes are **keywords** which allow selection based on information that lies outside of the document tree or that cannot be expressed by other selectors or combinators. This information can be associated to a certain state (state and dynamic pseudo-classes), to locations (structural and target pseudo-classes), to negations of the former (negation pseudo-class) or to languages (lang pseudo-class). Examples include whether or not a link has been followed (`:visited`), the mouse is over an element (`:hover`), a checkbox is checked (`:checked`), etc.

Syntax

```
selector:pseudo-class {
    property: VALUE;
}
```

List of pseudo-classes:

Name	Description
`:active`	Applies to any element being activated (i.e. clicked) by the user.
`:any`	Allows you to build sets of related selectors by creating groups that the included items will match. This is an alternative to repeating an entire selector.
`:target`	Selects the current active #news element (clicked on a URL containing that anchor name)
`:checked`	Applies to radio, checkbox, or option elements that are checked or toggled into an "on" state.
`:default`	Represents any user interface element that is the default among a group of similar elements.
`:disabled`	Applies to any UI element which is in a disabled state.
`:empty`	Applies to any element which has no children.
`:enabled`	Applies to any UI element which is in an enabled state.
`:first`	Used in conjunction with the `@page` rule, this selects the first page in a printed document.
`:first-child`	Represents any element that is the first child element of its parent.
`:first-of-type`	Applies when an element is the first of the selected element type inside its parent. This may or may not be the first-child.
`:focus`	Applies to any element which has the user's focus. This can be given by the user's keyboard, mouse events, or other forms of input.
`:focus-within`	Can be used to highlight a whole section when one element inside it is focused. It matches any element that the :focus pseudo-class matches or that has a descendant focused.
`:full-screen`	Applies to any element displayed in full-screen mode. It selects the whole stack of elements and not just the top level element.
`:hover`	Applies to any element being hovered by the user's pointing device, but not activated.
`:indeterminate`	Applies radio or checkbox UI elements which are neither checked nor unchecked, but are in an indeterminate state. This can be due to an element's attribute or DOM manipulation.
`:in-range`	The `:in-range` CSS pseudo-class matches when an element has its value attribute inside the specified range limitations for this element. It allows the page to give a feedback that the value currently defined using the element is inside the range limits.
`:invalid`	Applies to **`<input>`** elements whose values are invalid according to the type specified in the `type=` attribute.
`:lang`	Applies to any element who's wrapping **`<body>`** element has a properly designated `lang=` attribute. For the pseudo-class to be valid, it must contain a valid two or three letter language code.
`:last-child`	Represents any element that is the last child element of its parent.
`:last-of-type`	Applies when an element is the last of the selected element type inside its parent. This may or may not be the last-child.

`:left`	Used in conjunction with the `@page` rule, this selects all the left pages in a printed document.
`:link`	Applies to any links which haven't been visited by the user.
`:not()`	Applies to all elements which **do not** match the value passed to (`:not(p)` or `:not(.class-name)` for example. It must have a value to be valid and it can only contain one selector. However, you can chain multiple `:not` selectors together.
`:nth-child`	Applies when an element is the `n`-th element of its parent, where `n` can be an integer, a mathematical expression (e.g `n+3`) or the keywords `odd` or `even`.
`:nth-of-type`	Applies when an element is the `n`-th element of its parent of the same element type, where `n` can be an integer, a mathematical expression (e.g `n+3`) or the keywords `odd` or `even`.
`:only-child`	The `:only-child` CSS pseudo-class represents any element which is the only child of its parent. This is the same as `:first-child:last-child` or `:nth-child(1):nth-last-child(1)`, but with a lower specificity.
`:optional`	The `:optional` CSS pseudo-class represents any element that does not have the required attribute set on it. This allows forms to easily indicate optional fields and to style them accordingly.
`:out-of-range`	The `:out-of-range` CSS pseudo-class matches when an element has its value attribute outside the specified range limitations for this element. It allows the page to give a feedback that the value currently defined using the element is outside the range limits. A value can be outside of a range if it is either smaller or larger than maximum and minimum set values.
`:placeholder-shown`	**Experimental.** Applies to any form element currently displaying placeholder text.
`:read-only`	Applies to any element which is not editable by the user.
`:read-write`	Applies to any element that is editable by a user, such as **`<input>`** elements.
`:right`	Used in conjunction with the `@page` rule, this selects all the right pages in a printed document.
`:root`	matches the root element of a tree representing the document.
`:scope`	CSS pseudo-class matches the elements that are a reference point for selectors to match against.
`:target`	Selects the current active #news element (clicked on a URL containing that anchor name)
`:visited`	Applies to any links which have has been visited by the user.

> The `:visited` pseudoclass can't be used for most styling in a lot of modern browsers anymore because it's a security hole. See this link for reference.

Section 4.5: Child Pseudo Class

> "The :nth-child(an+b) CSS pseudo-class matches an element that has an+b-1 siblings before it in the document tree, for a given positive **or zero value** for n" - MDN :nth-child

pseudo-selector	1	2	3	4	5	6	7	8	9	10
`:first-child`	✓									
`:nth-child(3)`			✓							
`:nth-child(n+3)`			✓	✓	✓	✓	✓	✓	✓	✓
`:nth-child(3n)`			✓			✓			✓	

`:nth-child(3n+1)`	✓			✓			✓			✓
`:nth-child(-n+3)`	✓	✓	✓							
`:nth-child(odd)`	✓		✓		✓		✓		✓	
`:nth-child(even)`		✓		✓		✓		✓		✓
`:last-child`										✓
`:nth-last-child(3)`								✓		

Section 4.6: Class Name Selectors

The class name selector select all elements with the targeted class name. For example, the class name `.warning` would select the following **`<div>`** element:

```
<div class="warning">
    <p>This would be some warning copy.</p>
</div>
```

You can also combine class names to target elements more specifically. Let's build on the example above to showcase a more complicated class selection.

CSS

```
.important {
    color: orange;
}
.warning {
    color: blue;
}
.warning.important {
    color: red;
}
```

HTML

```
<div class="warning">
    <p>This would be some warning copy.</p>
</div>

<div class="important warning">
    <p class="important">This is some really important warning copy.</p>
</div>
```

In this example, all elements with the `.warning` class will have a blue text color, elements with the `.important` class with have an orange text color, and all elements that have *both* the `.important` and `.warning` class name will have a red text color.

Notice that within the CSS, the `.warning.important` declaration did not have any spaces between the two class names. This means it will only find elements which contain both class names `warning` and `important` in their `class` attribute. Those class names could be in any order on the element.

If a space was included between the two classes in the CSS declaration, it would only select elements that have parent elements with a `.warning` class names and child elements with `.important` class names.

Section 4.7: Select element using its ID without the high specificity of the ID selector

This trick helps you select an element using the ID as a value for an attribute selector to avoid the high specificity of the ID selector.

HTML:

```
<div id="element">...</div>
```

CSS

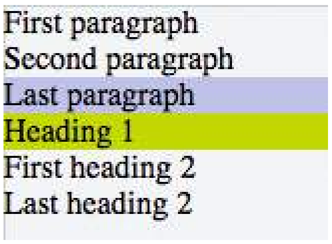
```
#element { ... } /* High specificity will override many selectors */

[id="element"] { ... } /* Low specificity, can be overridden easily */
```

Section 4.8: The :last-of-type selector

The `:last-of-type` selects the element that is the last child, of a particular type, of its parent. In the example below, the css selects the last paragraph and the last heading `h1`.

```
p:last-of-type {
  background: #C5CAE9;
}
h1:last-of-type {
  background: #CDDC39;
}
```

```
<div class="container">
    <p>First paragraph</p>
    <p>Second paragraph</p>
    <p>Last paragraph</p>
    <h1>Heading 1</h1>
    <h2>First heading 2</h2>
    <h2>Last heading 2</h2>
</div>
```



```
First paragraph
Second paragraph
Last paragraph
Heading 1
First heading 2
Last heading 2
```

jsFiddle

Section 4.9: CSS3 :in-range selector example

```
<style>
input:in-range {
    border: 1px solid blue;
}
</style>



<input type="number" min="10" max="20" value="15">
```

```
<p>The border for this value will be blue</p>
```

The `:in-range` CSS pseudo-class matches when an element has its value attribute inside the specified range limitations for this element. It allows the page to give a feedback that the value currently defined using the element is inside the range limits.[1]

Section 4.10: A. The :not pseudo-class example & B. :focus-within CSS pseudo-class

A. The syntax is presented above.

The following selector matches all **`<input>`** elements in an HTML document that are not disabled and don't have the class `.example`:

HTML:

```
<form>
    Phone: <input type="tel" class="example">
    E-mail: <input type="email" disabled="disabled">
    Password: <input type="password">
</form>
```

CSS:

```
input:not([disabled]):not(.example){
   background-color: #ccc;
}
```

The `:not()` pseudo-class will also support comma-separated selectors in Selectors Level 4:

CSS:

```
input:not([disabled], .example){
   background-color: #ccc;
}
```

Live Demo on JSBin

See background syntax here.

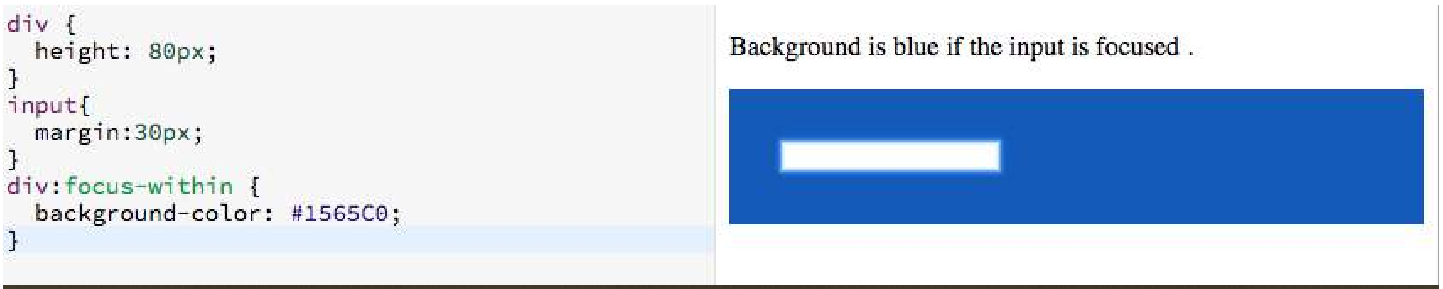
B. The :focus-within CSS pseudo-class

HTML:

```
  <h3>Background is blue if the input is focused .</p>
  <div>
    <input type="text">
  </div>
```

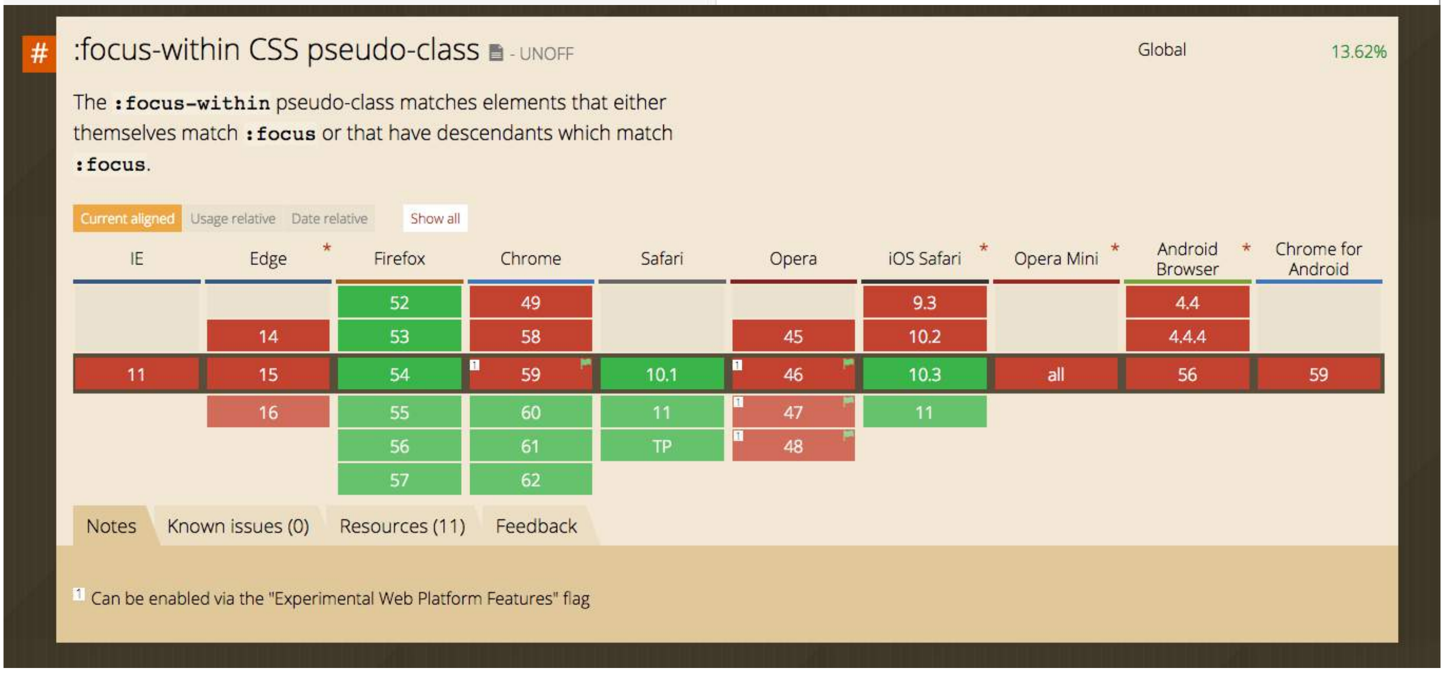
CSS:

```
div {
  height: 80px;
}
input{
  margin:30px;
}
div:focus-within {
  background-color: #1565C0;
}
```



```
div {
  height: 80px;
}
input{
  margin:30px;
}
div:focus-within {
  background-color: #1565C0;
}
```

Background is blue if the input is focused .



:focus-within CSS pseudo-class - UNOFF

Global 13.62%

The **:focus-within** pseudo-class matches elements that either themselves match **:focus** or that have descendants which match **:focus**.

Current aligned | Usage relative | Date relative | Show all

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
		52	49			9.3		4.4	
	14	53	58		45	10.2		4.4.4	
11	15	54	[1] 59	10.1	[1] 46	10.3	all	56	59
	16	55	60	11	[1] 47	11			
		56	61	TP	[1] 48				
		57	62						

Notes | Known issues (0) | Resources (11) | Feedback

[1] Can be enabled via the "Experimental Web Platform Features" flag

Section 4.11: Global boolean with checkbox:checked and ~ (general sibling combinator)

With the ~ selector, you can easily implement a global accessible boolean without using JavaScript.

Add boolean as a checkbox

To the very beginning of your document, add as much booleans as you want with a unique `id` and the `hidden` attribute set:

```
<input type="checkbox" id="sidebarShown" hidden />
<input type="checkbox" id="darkThemeUsed" hidden />

<!-- here begins actual content, for example: -->
<div id="container">
    <div id="sidebar">
        <!-- Menu, Search, ... -->
    </div>

    <!-- Some more content ... -->
</div>

<div id="footer">
    <!-- ... -->
</div>
```

Change the boolean's value

You can toggle the boolean by adding a `label` with the `for` attribute set:

```
<label for="sidebarShown">Show/Hide the sidebar!</label>
```

Accessing boolean value with CSS

The normal selector (like `.color-red`) specifies the default properties. They can be overridden by following `true` / `false` selectors:

```
/* true: */
<checkbox>:checked ~ [sibling of checkbox & parent of target] <target>

/* false: */
<checkbox>:not(:checked) ~ [sibling of checkbox & parent of target] <target>
```

Note that **`<checkbox>`**, `[sibling ...]` and **`<target>`** should be replaced by the proper selectors. `[sibling ...]` can be a specific selector, (often if you're lazy) simply `*` or nothing if the target is already a sibling of the checkbox.

Examples for the above HTML structure would be:

```
#sidebarShown:checked ~ #container #sidebar {
    margin-left: 300px;
}

#darkThemeUsed:checked ~ #container,
#darkThemeUsed:checked ~ #footer {
    background: #333;
}
```

In action

See this fiddle for a implementation of these global booleans.

Section 4.12: ID selectors

ID selectors select DOM elements with the targeted ID. To select an element by a specific ID in CSS, the # prefix is used.

For example, the following HTML `div` element…

```
<div id="exampleID">
    <p>Example</p>
</div>
```

…can be selected by `#exampleID` in CSS as shown below:

```
#exampleID {
    width: 20px;
}
```

> **Note**: The HTML specs do not allow multiple elements with the same ID

Section 4.13: How to style a Range input

HTML

```
<input type="range"></input>
```

CSS

Effect	Pseudo Selector
Thumb	`input[type=range]::-webkit-slider-thumb, input[type=range]::-moz-range-thumb, input[type=range]::-ms-thumb`
Track	`input[type=range]::-webkit-slider-runnable-track, input[type=range]::-moz-range-track, input[type=range]::-ms-track`
OnFocus	`input[type=range]:focus`
Lower part of the track	`input[type=range]::-moz-range-progress, input[type=range]::-ms-fill-lower` (not possible in WebKit browsers currently - JS needed)

Section 4.14: The :only-child pseudo-class selector example

The `:only-child` CSS pseudo-class represents any element which is the only child of its parent.

HTML:

```
<div>
  <p>This paragraph is the only child of the div, it will have the color blue</p>
</div>

<div>
  <p>This paragraph is one of the two children of the div</p>
  <p>This paragraph is one of the two children of its parent</p>
</div>
```

CSS:

```
p:only-child {
  color: blue;
}
```

The above example selects the `<p>` element that is the unique child from its parent, in this case a `<div>`.

Live Demo on JSBin