Pseudo-classes and pseudo-elements

The next set of selectors we will look at are referred to as **pseudo-classes** and **pseudo-elements**. There are a large number of these, and they often serve quite specific purposes. Once you know how to use them, you can look at the list to see if there is something which works for the task you are trying to achieve. Once again the relevant MDN page for each selector is helpful in explaining browser support.

Prerequisites:	Basic computer literacy, basic software installed, basic knowledge of working with files, HTML basics (study Introduction to HTML), and an idea of how CSS works (study CSS first steps.)
Objective:	To learn about the pseudo-class and pseudo-element selectors.

What is a pseudo-class?

A pseudo-class is a selector that selects elements that are in a specific state, e.g. they are the first element of their type, or they are being hovered over by the mouse pointer. They tend to act as if you had applied a class to some part of your document, often helping you cut down on excess classes in your markup, and giving you more flexible, maintainable code.

Pseudo-classes are keywords that start with a colon:

```
:pseudo-class-name
```

Simple pseudo-class example

Let's look at a simple example. If we wanted to make the first paragraph in an article larger and bold, we could add a class to that paragraph and then add CSS to that class, as shown in the first example below:

Veggies es bonus vobis, proinde vos postulo essum magis kohlrabi welsh onion daikon amaranth tatsoi tomatillo melon azuki bean garlic.

Gumbo beet greens corn soko endive gumbo gourd. Parsley shallot courgette tatsoi pea sprouts fava bean collard greens dandelion okra wakame tomato. Dandelion cucumber earthnut pea peanut soko zucchini.

```css
.first {
    font-size: 120%;
    font-weight: bold;
}
```

```html
<article>
    <p class="first">Veggies es bonus vobis, proinde vos
postulo essum magis kohlrabi welsh onion daikon amaranth
tatsoi tomatillo
            melon azuki bean garlic.</p>

    <p>Gumbo beet greens corn soko endive gumbo gourd.
Parsley shallot courgette tatsoi pea sprouts fava bean
collard
            greens dandelion okra wakame tomato
```

Reset

However, this could be annoying to maintain — what if a new paragraph got added to the top of the document? We'd need to move the class over to the new paragraph. Instead of adding the class, we could use the `:first-child` pseudo-class selector — this will *always* target the first child element in the article, and we will no longer need to edit the HTML (this may not always be possible anyway, maybe due to it being generated by a CMS.)

Veggies es bonus vobis, proinde vos postulo essum magis kohlrabi welsh onion daikon amaranth tatsoi tomatillo melon azuki bean garlic.

Gumbo beet greens corn soko endive gumbo gourd. Parsley shallot courgette tatsoi pea sprouts fava bean collard greens dandelion okra wakame tomato. Dandelion cucumber earthnut pea peanut soko zucchini.

```
article p:first-child {
    font-size: 120%;
    font-weight: bold;
}
```

```
<article>
    <p>Veggies es bonus vobis, proinde vos postulo essum
magis kohlrabi welsh onion daikon amaranth tatsoi
tomatillo
            melon azuki bean garlic.</p>

    <p>Gumbo beet greens corn soko endive gumbo gourd.
Parsley shallot courgette tatsoi pea sprouts fava bean
collard
            greens dandelion okra wakame tomato
```

Reset

All pseudo-classes behave in this same kind of way. They target some bit of your document that is in a certain state, behaving as if you had added a class into your HTML. Take a look at some other examples on MDN:

- `:last-child`
- `:only-child`
- `:invalid`

User-action pseudo classes

Some pseudo-classes only apply when the user interacts with the document in some way. These **user-action** pseudo-classes, sometimes referred to as **dynamic pseudo-classes**, act as if a class had been added to the element when the user interacts with it. Examples include:

- `:hover` — mentioned above; this only applies if the user moves their pointer over an element, typically a link.
- `:focus` — only applies if the user focuses the element using keyboard controls.

Hover over me

```
a:link,
a:visited {
    color: rebeccapurple;
    font-weight: bold;
}

a:hover {
    color:hotpink;
}
```

```
<p><a href="">Hover over me</a></p>
```

Reset

What is a pseudo-element?

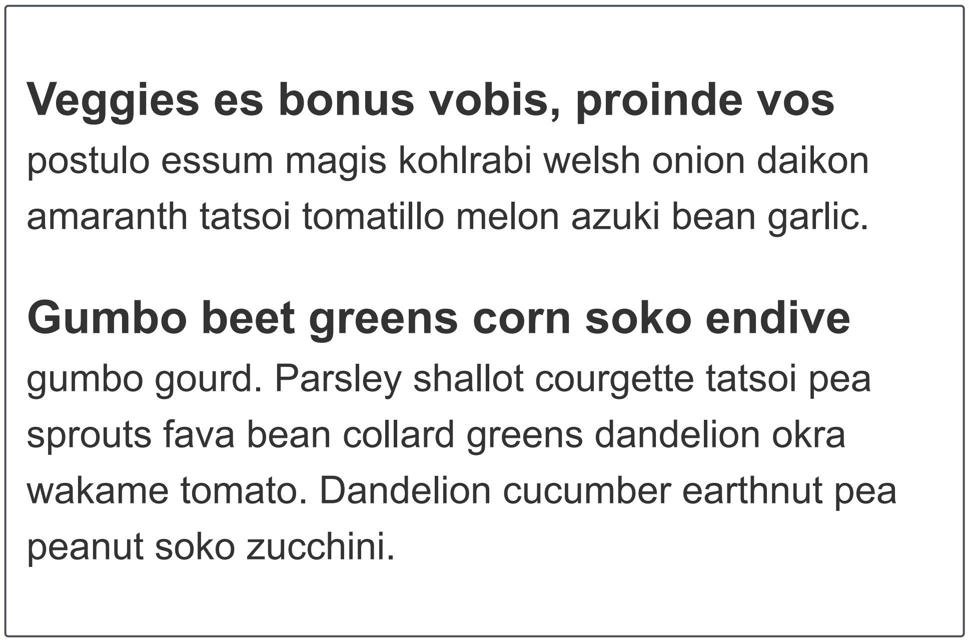
Pseudo-elements behave in a similar way, however they act as if you had added a whole new HTML element into the markup, rather than applying a class to existing elements. Pseudo-elements start with a double colon `::`.

```
::pseudo-element-name
```

Note: Some early pseudo-elements used the single colon syntax, so you may sometimes see this in code or examples. Modern browsers support the early pseudo-elements with single- or double-colon syntax for backwards compatibility.

For example, if you wanted to select the first line of a paragraph you could wrap it in a `<span>` element and use an element selector; however, that would fail if the number of words you had wrapped were longer or shorter than the parent element's width. As we tend not to know how many words will fit on a line — as that will change if the screen width or font-size changes — it is impossible to robustly do this by adding HTML.

The `::first-line` pseudo-element selector will do this for you reliably — if the number of words increases and decreases it will still only select the first line.



Veggies es bonus vobis, proinde vos
postulo essum magis kohlrabi welsh onion daikon
amaranth tatsoi tomatillo melon azuki bean garlic.

Gumbo beet greens corn soko endive
gumbo gourd. Parsley shallot courgette tatsoi pea
sprouts fava bean collard greens dandelion okra
wakame tomato. Dandelion cucumber earthnut pea
peanut soko zucchini.

```css
article p::first-line {
    font-size: 120%;
    font-weight: bold;
}
```

```html
<article>
    <p>Veggies es bonus vobis, proinde vos postulo essum
magis kohlrabi welsh onion daikon amaranth tatsoi
tomatillo
            melon azuki bean garlic.</p>

    <p>Gumbo beet greens corn soko endive gumbo gourd.
Parsley shallot courgette tatsoi pea sprouts fava bean
collard
```

Reset

It acts as if a `<span>` was magically wrapped around that first formatted line, and updated each time the line length changed.

You can see that this selects the first line of both paragraphs.

Combining pseudo-classes and pseudo-elements

If you wanted to make the first line of the first paragraph bold you could chain the `:first-child` and `::first-line` selectors together. Try editing the previous live example so it uses the following CSS. We are saying that we want to select the first line, of the first `<p>` element, which is inside an `<article>` element.

```css
article p:first-child::first-line {
  font-size: 120%;
  font-weight: bold;
}
```

Generating content with ::before and ::after

There are a couple of special pseudo-elements, which are used along with the `content` property to insert content into your document using CSS.

You could use these to insert a string of text, such as in the live example below. Try changing the text value of the `content` property and see it change in the output. You could also change the `::before` pseudo-element to `::after` and see the text inserted at the end of the element instead of the beginning.

This should show before the other content.Content in the box in my HTML page.

```
.box::before {
    content: "This should show before the other content."
}
```

```
<p class="box">Content in the box in my HTML page.</p>
```

Reset

Inserting strings of text from CSS isn't really something we do very often on the web however, as that text is inaccessible to some screen readers and might be hard for someone to find and edit in the future.

A more valid use of these pseudo-elements is to insert an icon, for example the little arrow added in the example below, which is a visual indicator that we wouldn't want read out by a screenreader:

Content in the box in my HTML page. ➥

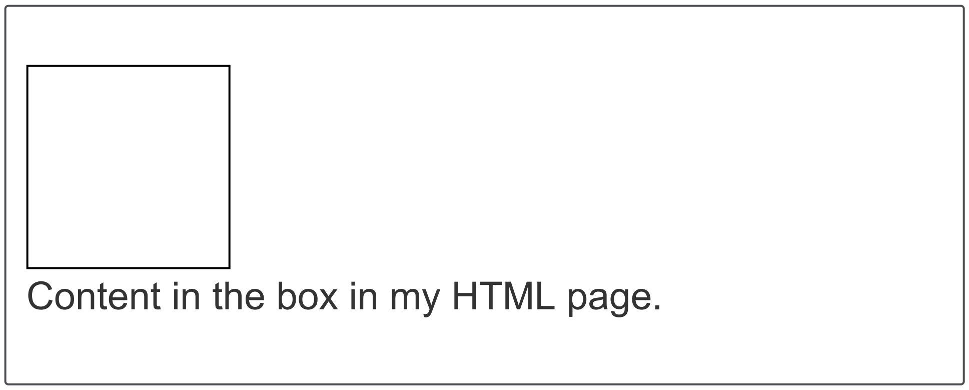
```
.box::after {
    content: " ➥"
}
```

```
<p class="box">Content in the box in my HTML page.</p>
```

Reset

These pseudo-elements are also frequently used to insert an empty string, which can then be styled just like any element on the page.

In this next example, we have added an empty string using the `::before` pseudo-element. We have set this to `display: block` in order that we can style it with a width and height. We then use CSS to style it just like any element. You can play around with the CSS and change how it looks and behaves.



Content in the box in my HTML page.

```css
.box::before {
    content: "";
    display: block;
    width: 100px;
    height: 100px;
    background-color: rebeccapurple;
    border: 1px solid black;
}
```

```html
<p class="box">Content in the box in my HTML page.</p>
```

Reset

The use of the `::before` and `::after` pseudo-elements along with the `content` property is referred to as "Generated Content" in CSS, and you will often see this technique being used for various tasks. A great example is the site CSS Arrow Please, which helps you to generate an arrow with CSS. Look at the CSS as you create your arrow and you will see the `::before` and `::after` pseudo-elements in use. Whenever you see these selectors, look at the `content` property to see what is being added to the document.

Reference section

There are a large number of pseudo-classes and pseudo-elements, and it is useful to have a list to refer to. Below are tables listing them, with links to their reference pages on MDN. Use this as a reference to see the kind of things that are available for you to target.

Pseudo-classes

Selector	Description
`:active`	Matches when the user activates (for example clicks on) an element.
`:any-link`	Matches both the `:link` and `:visited` states of a link.
`:blank`	Matches an `<input>` element whose input value is empty.
`:checked`	Matches a radio button or checkbox in the selected state.
`:current`	Matches the element, or an ancestor of the element, that is currently being displayed.
`:default`	Matches the one or more UI elements that are the default among a set of similar elements.
`:dir`	Select an element based on its directionality (value of the HTML `dir` attribute or CSS `direction` property).
`:disabled`	Matches user interface elements that are in an disabled state.
`:empty`	Matches an element that has no children except optionally white space.
`:enabled`	Matches user interface elements that are in an enabled state.
`:first`	In Paged Media, matches the first page.
`:first-child`	Matches an element that is first among its siblings.
`:first-of-type`	Matches an element which is first of a certain type among its siblings.
`:focus`	Matches when an element has focus.
`:focus-visible`	Matches when an element has focus and the focus should be visible to the user.
`:focus-within`	Matches an element with focus plus an element with a descendent that has focus.
`:future`	Matches the elements after the current element.
`:hover`	Matches when the user hovers over an element.

Selector	Description
`:indeterminate`	Matches UI elements whose value is in an indeterminate state, usually checkboxes.
`:in-range`	Matches an element with a range when its value is in-range.
`:invalid`	Matches an element, such as an `<input>`, in an invalid state.
`:lang`	Matches an element based on language (value of the HTML lang attribute).
`:last-child`	Matches an element which is last among its siblings.
`:last-of-type`	Matches an element of a certain type that is last among its siblings.
`:left`	In Paged Media, matches left-hand pages.
`:link`	Matches unvisited links.
`:local-link`	Matches links pointing to pages that are in the same site as the current document.
`:is()`	Matches any of the selectors in the selector list that is passed in.
`:not`	Matches things not matched by selectors that are passed in as a value to this selector.
`:nth-child`	Matches elements from a list of siblings — the siblings are matched by a formula of the form *an+b* (e.g. 2n + 1 would match elements 1, 3, 5, 7, etc. All the odd ones.)
`:nth-of-type`	Matches elements from a list of siblings that are of a certain type (e.g. `<p>` elements) — the siblings are matched by a formula of the form *an+b* (e.g. 2n + 1 would match that type of element, numbers 1, 3, 5, 7, etc. All the odd ones.)
`:nth-last-child`	Matches elements from a list of siblings, counting backwards from the end. The siblings are matched by a formula of the form *an+b* (e.g. 2n + 1 would match the last element in the sequence, then two elements before that, then two elements before that, etc. All the odd ones, counting from the end.)

Selector	Description
`:nth-last-of-type`	Matches elements from a list of siblings that are of a certain type (e.g. `<p>` elements), counting backwards from the end. The siblings are matched by a formula of the form *an+b* (e.g. 2n + 1 would match the last element of that type in the sequence, then two elements before that, then two elements before that, etc. All the odd ones, counting from the end.)
`:only-child`	Matches an element that has no siblings.
`:only-of-type`	Matches an element that is the only one of its type among its siblings.
`:optional`	Matches form elements that are not required.
`:out-of-range`	Matches an element with a range when its value is out of range.
`:past`	Matches the elements before the current element.
`:placeholder-shown`	Matches an input element that is showing placeholder text.
`:playing`	Matches an element representing an audio, video, or similar resource that is capable of being "played" or "paused", when that element is "playing".
`:paused`	Matches an element representing an audio, video, or similar resource that is capable of being "played" or "paused", when that element is "paused".
`:read-only`	Matches an element if it is not user-alterable.
`:read-write`	Matches an element if it is user-alterable.
`:required`	Matches form elements that are required.
`:right`	In Paged Media, matches right-hand pages.
`:root`	Matches an element that is the root of the document.
`:scope`	Matches any element that is a scope element.
`:valid`	Matches an element such as an `<input>` element, in a valid state.
`:target`	Matches an element if it is the target of the current URL (i.e. if it has an ID matching the current URL fragment).
`:visited`	Matches visited links.

Pseudo-elements

Selector	Description
`::after`	Matches a stylable element appearing after the originating element's actual content.
`::before`	Matches a stylable element appearing before the originating element's actual content.
`::first-letter`	Matches the first letter of the element.
`::first-line`	Matches the first line of the containing element.
`::grammar-error`	Matches a portion of the document containing a grammar error as flagged by the browser.
`::selection`	Matches the portion of the document that has been selected.
`::spelling-error`	Matches a portion of the document containing a spelling error as flagged by the browser.

Previous Overview: Building blocks Next

In this module

Last modified: Dec 16, 2019, by MDN contributors

Related Topics

▸ Accessibility guides

▸ Accessibility assessment

Tools and testing

▸ Cross browser testing

Server-side website programming

▸ First steps

▸ Django web framework (Python)

▸ Express Web Framework (node.js/JavaScript)

Further resources

▸ Common questions

How to contribute