CSS selectors

In CSS, selectors are used to target the HTML elements on our web pages that we want to style. There are a wide variety of CSS selectors available, allowing for fine-grained precision when selecting elements to style. In this article and its sub-articles we'll run through the different types in great detail, seeing how they work.

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 how CSS selectors work in detail.

What is a selector?

You have met selectors already. A CSS selector is the first part of a CSS Rule. It is a pattern of elements and other terms that tell the browser which HTML elements should be selected to have the CSS property values inside the rule applied to them. The element or elements which are selected by the selector are referred to as the *subject of the selector*.

```
h1 {
  color: blue;
  background-color: yellow;
}

p {
  color: red;
}
```

In earlier articles you met some different selectors, and learned that there are selectors that target the document in different ways — for example by selecting an element such as `h1`, or a class such as `.special`.

In CSS, selectors are defined in the CSS Selectors specification; like any other part of CSS they need to have support in browsers for them to work. The majority of selectors that you will come across are defined in the Level 3 Selectors specification, which is a mature specification, therefore you will find excellent browser support for these selectors.

Selector lists

If you have more than one thing which uses the same CSS then the individual selectors can be combined into a *selector list* so that the rule is applied to all of the individual selectors. For example, if I have the same CSS for an `h1` and also a class of `.special`, I could write this as two separate rules.

```
h1 {
  color: blue;
}

.special {
  color: blue;
}
```

I could also combine these into a selector list, by adding a comma between them.

```
h1, .special {
  color: blue;
}
```

White space is valid before or after the comma. You may also find the selectors more readable if each is on a new line.

```
h1,
.special {
  color: blue;
}
```

In the live example below try combining the two selectors which have identical declarations. The visual display should be the same after combining them.

Type selectors

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.

Turnip greens yarrow ricebean rutabaga *endive* *cauliflower* sea lettuce kohlrabi amaranth water spinach avocado daikon napa cabbage asparagus winter purslane kale. Celery potato scallion desert raisin horseradish spinach

```
span {
```

```
    background-color: yellow;
}

strong {
    color: rebeccapurple;
}

em {
    color: rebeccapurple;
}
```

```
<h1>Type selectors</h1>
<p>Veggies es bonus vobis, proinde vos postulo essum
magis <span>kohlrabi welsh onion</span> daikon amaranth
tatsoi tomatillo
    melon azuki bean garlic.</p>

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

<p>Turnip greens yarrow ricebean rutabaga <em>endive
```

When you group selectors in this way, if any selector is invalid the whole rule will be ignored.

In the following example, the invalid class selector rule will be ignored, whereas the `h1` would still be styled.

```
h1 {
  color: blue;
}

..special {
  color: blue;
}
```

When combined however, neither the `h1` nor the class will be styled as the entire rule is deemed invalid.

```
h1, ..special {
  color: blue;
```

```
}
```

Types of selectors

There are a few different groupings of selectors, and knowing which type of selector you might need will help you to find the right tool for the job. In this article's subarticles we will look at the different groups of selectors in more detail.

Type, class, and ID selectors

This group includes selectors that target an HTML element such as an `<h1>`.

```
1 h1 { }
```

It also includes selectors which target a class:

```
1 .box { }
```

or, an ID:

```
1 #unique { }
```

Attribute selectors

This group of selectors gives you different ways to select elements based on the presence of a certain attribute on an element:

```
1 a[title] { }
```

Or even make a selection based on the presence of an attribute with a particular value:

```
a[href="https://example.com"] { }
```

Pseudo-classes and pseudo-elements

This group of selectors includes pseudo-classes, which style certain states of an element. The `:hover` pseudo-class for example selects an element only when it is being hovered over by the mouse pointer:

```
a:hover { }
```

It also includes pseudo-elements, which select a certain part of an element rather than the element itself. For example, `::first-line` always selects the first line of text inside an element (a `<p>` in the below case), acting as if a `<span>` was wrapped around the first formatted line and then selected.

```
p::first-line { }
```

Combinators

The final group of selectors combine other selectors in order to target elements within our documents. The following for example selects paragraphs that are direct children of `<article>` elements using the child combinator (`>`):

```
article > p { }
```

Next steps

You can take a look at the reference table of selectors below for direct links to the various types of selectors in this Learn section or on MDN in general, or continue on to start your journey by finding out about type, class, and ID selectors.

Reference table of selectors

The below table gives you an overview of the selectors you have available to use, along with links to the pages in this guide which will show you how to use each type of selector. I have also included a link to the MDN page for each selector where you can check browser support information. You can use this as a reference to come back to when you need to look up selectors later in the material, or as you experiment with CSS generally.

Selector	Example	Learn CSS tutorial
Type selector	`h1 { }`	Type selectors
Universal selector	`* { }`	The universal selector
Class selector	`.box { }`	Class selectors
id selector	`#unique { }`	ID selectors
Attribute selector	`a[title] { }`	Attribute selectors
Pseudo-class selectors	`p:first-child { }`	Pseudo-classes
Pseudo-element selectors	`p::first-line { }`	Pseudo-elements
Descendant combinator	`article p`	Descendant combinator
Child combinator	`article > p`	Child combinator
Adjacent sibling combinator	`h1 + p`	Adjacent sibling
General sibling combinator	`h1 ~ p`	General sibling

In this module

1. Cascade and inheritance
2. CSS selectors
    - Type, class, and ID selectors
    - Attribute selectors
    - Pseudo-classes and pseudo-elements
    - Combinators
3. The box model
4. Backgrounds and borders
5. Handling different text directions
6. Overflowing content
7. Values and units
8. Sizing items in CSS
9. Images, media, and form elements
10. Styling tables
11. Debugging CSS
12. Organizing your CSS

Last modified: Dec 4, 2019, by MDN contributors

Related Topics

Complete beginners start here!

- Getting started with the Web

HTML — Structuring the Web

- ▶ Introduction to HTML
- ▶ Multimedia and embedding
- ▶ HTML tables
- ▶ HTML forms

CSS — Styling the Web

- ▶ CSS first steps
- ▼ CSS building blocks
  - CSS building blocks overview
  - Cascade and inheritance
  - CSS selectors
  - The box model
  - Backgrounds and borders
  - Handling different text directions
  - Overflowing content
  - Values and units
  - Sizing items in CSS
  - Images, media, and form elements
  - Styling tables
  - Debugging CSS
  - Organizing your CSS
- ▶ Styling text
- ▶ CSS layout

JavaScript — Dynamic client-side scripting

- ▶ JavaScript first steps

- JavaScript building blocks
- Introducing JavaScript objects
- Asynchronous JavaScript
- Client-side web APIs

Accessibility — Make the web usable by everyone

- 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

you@example.com

Sign up now