Combinators

The final selectors we will look at are called combinators, because they combine other selectors in a way that gives them a useful relationship to each other and the location of content in the document.

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 different combinator selectors that can be used in CSS.

Descendant selector

You have already come across descendant selectors in previous lessons — selectors with spaces in between them:

```
body article p
```

These selectors select elements that are descendants of others selector. They do not need to be direct children to match.

In the example below we are matching only the `<p>` element which is inside an element with a class of `.box`.

Text in .box

Text not in .box

```
.box p {
    color: red;
}
```

```
<div class="box"><p>Text in .box</p></div>
<p>Text not in .box</p>
```

Reset

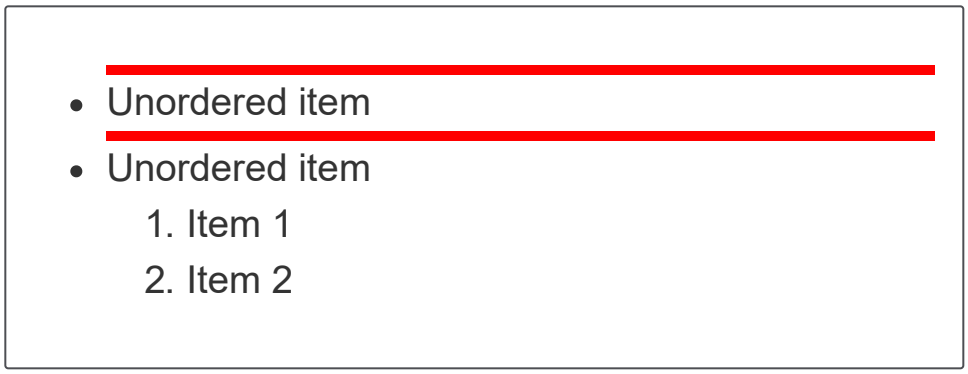
Child combinator

The child combinator is a greater-than symbol (`>`), which matches only when the selectors select elements that are direct children. Descendants further down the hierarchy don't match. For example, to select only `<p>` elements that are direct children of `<article>` elements:

```
1 article > p
```

In this next example we have an ordered list, nested inside of which is another unordered list. I am using the child combinator to select only the `<li>` elements which are a direct child of a `<ul>`, and have given them a top border.

If you remove the `>` that designates this as a child combinator, you end up with a descendant selector and all `<li>` elements will get a red border.




```
ul > li {
    border-top: 5px solid red;
}
```

```
<ul>
    <li>Unordered item</li>
    <li>Unordered item
        <ol>
            <li>Item 1</li>
            <li>Item 2</li>
        </ol>
    </li>
</ul>
```

Reset

Adjacent sibling

The adjacent sibling selector (`+`) is used to select something if it is right next to another element at the same level of the hierarchy. For example, to select all `<img>` elements that come right after `<p>` elements:

```
p + img
```

A common use case is to do something with a paragraph that follows a heading, as in my example below. Here we are looking for a paragraph which is directly adjacent to an `<h1>`, and styling it.

If you insert some other element such as a `<h2>` in between the `<h1>` and the `<p>`, you will find that the paragraph is no longer matched by the selector and so does not get the background and foreground color applied when the element is adjacent.

A heading

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.

```
h1 + p {
    font-weight: bold;
    background-color: #333;
    color: #fff;
    padding: .5em;
}
```

```
<article>
    <h1>A heading</h1>
    <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.
Dandelion cucumber earthnut pea peanut soko zucchini.
```

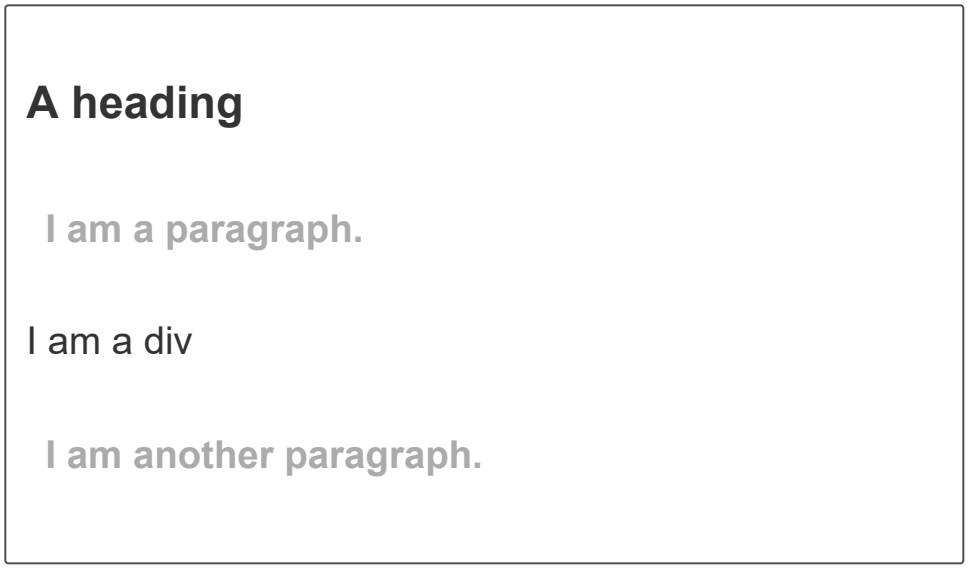
Reset

General sibling

If you want to select siblings of an element even if they are not directly adjacent, then you can use the general sibling combinator (`~`). To select all `<img>` elements that come *anywhere* after `<p>` elements, we'd do this:

```
1 | p ~ img
```

In the example below we are selecting all `<p>` elements that come after the `<h1>`, and even though there is a `<div>` in the document as well, the `<p>` that comes after it is selected.




```
h1 ~ p {
    font-weight: bold;
    background-color: #333;
    color: #fff;
    padding: .5em;
}
```

```
<article>
    <h1>A heading</h1>
    <p>I am a paragraph.</p>
    <div>I am a div</div>
    <p>I am another paragraph.</p>
</article>
```

Reset

Using combinators

You can combine any of the selectors that we discovered in previous lessons with combinators in order to pick out part of your document. For example if we want to select list items with a class of "a", which are direct children of a `<ul>`, I could use the following.

```
ul > li[class="a"]  {  }
```

Take care however when creating big lists of selectors that select very specific parts of your document. It will be hard to reuse the CSS rules as you have made the selector very specific to the location of that element in the markup.

It is often better to create a simple class and apply that to the element in question. That said, your knowledge of combinators will come in very useful if you need to get to something in your document and are unable to access the HTML, perhaps due to it being generated by a CMS.

Moving on

This is the last section in our lessons on selectors. Next we will move on to another important part of CSS — the CSS Box Model.

Previous | Overview: Building blocks | Next

In this module

Last modified: Nov 28, 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

Sign up now