More CSS Selectors

Although using classes and IDs are very handy for CSS, when there are many classes and IDs and you are targeting each individual class and ID, it can quickly make your CSS bloated. If you are clever with your CSS selectors you can make your CSS more flexible (i.e. less dependent on class or ID names) and much cleaner looking.

For example, you want to avoid the following CSS:

```
div.post-author, div.post-date, div.details-of-post {
  font-size: 0.8em;
}
```

Is there was a way to slim down the above selector? You can do this via advanced CSS selectors.

We covered some advanced selectors previously:

- :nth-child
- :first-child
- :last-child
- > (is a child of)

There are more advanced selectors and clever combinations you can use.

Attributes

You can target elements based on their attributes. (Recall, in HTML, the attribute is the stuff that is like class="myclass". The attribute name here is "class" and it has the value "myclass".)

Attributes in CSS are enclosed within square brackets ("[ ]"). For example, if you want to target a link (<a>) with an href pointing to https://www.google.ca, you can use:

```
a[href="https://www.google.ca"] {
  ...
}
```

For exact values, use the above format with [attr="value"].

There are more options, however. Attribute selectors don't need to be exact matches only.

[attr*="value"]

The above syntax means that the attribute **contains** the value.

Taking a look at the CSS from the top of the page:

```
div.post-author, div.post-date, div.details-of-post {
  font-size: 0.8em;
}
```

This can be turned into:

```
div[class*="post"] {
  font-size: 0.8em;
}
```

Now, instead of targeting each class name, you can just specify that all paragraphs with classes containing the word "post" should have a font size of 0.8em.

[attr^="value"]

This is used to target elements with the attribute with **begins** with the stated value.

In an example, you can style all links to a Facebook page differently from other links by using:

```
a[href^="https://www.facebook.com"] {
  ...
}
```

[attr$="value"]

This is used to target elements with an attribute which **ends** with the stated value.

A good example of this would be to add via CSS icons to download links depending on file type. For example, you want a PDF icon for a download link leading to a PDF file. You can use:

```
a[href$=".pdf"] {
  background-image: url("images/pdf-icon.png");
  padding-left: 35px; /* take into account icon width */
}
a[href$=".xlsx"] {
  background-image: url("images/excel-icon.png");
  padding-left: 35px;
}
```

You may need to adjust height or bottom padding as well depending on your icon size and height.

(Note: In terms of accessibility, it is okay to add these icons via CSS because it is not important content. It is stylistic only.)

[attr~="value"]

This is used to target an attribute value which is **within a space separated list**.

For example, if you have multiple values separated by spaces in a `rel` attribute:

```
<a href="https://www.google.ca" rel="external search">Google</a>
```

You can use:

```
a[rel~="search"] { ... }
```

***rel** attribute*

The `rel` attribute allows you to specify the relationship between the link and the current document. For example:

```
<a href="https://www.google.ca" rel="external">Google</a>
```

specifies that the link is an external link in relation to the current document (the HTML page). This attribute is usually used with the `<link>` or `<a>` elements.

[attr|="value"]

You can use this to find an element with the value **first in a hyphen-separated list**.

For example, if you have multiple text tokens separated by dashes in a class attribute:

```
<div class="details-post">…</div>
<div class="detail-author">…</div>
```

You can use:

```
[class|="detail"] {
  ...
}
```

Note that the entire value must match for the hyphen-separated words. So the first `div` with `class="details-post"` will not get selected because "details" is not equal to "detail".

[attr]

You don't need to specify a value. You can just select elements for which the attribute **exists**.

For example, if you want to quickly view which images contain an `alt` attribute for easy accessibility testing you can use:

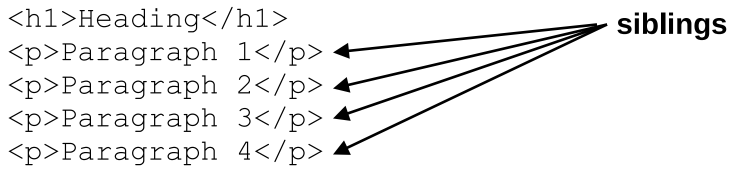
```
img[alt] {
  border: 2px solid red;
}
```

[attr1="value"][attr2^="value"]

You can combine multiple attributes in one selector. Note that there is no space between attributes because you are searching for an element with **both** attributes satisfying the values specified.

Sibling selectors

You have learned of the child selector (>) and descendant selector (this is simply a space like for `ul li`), but what if you want to style an element which is a sibling (i.e. on the same level)?



```
<h1>Heading</h1>
<p>Paragraph 1</p>
<p>Paragraph 2</p>
<p>Paragraph 3</p>
<p>Paragraph 4</p>
```

~ (select all siblings of specified element)

To select all paragraphs which are ***siblings*** of <h1> in the example above:

```
h1 ~ p {
  ...
}
```

This will style paragraphs 1 through 4.

+ (select the adjacent sibling)

This will select the **adjacent** sibling only. The adjacent sibling is the sibling which immediately follows the given element.

For example:

```
h1 + p {
  ...
}
```

will style only paragraph 1 because the first paragraph is immediately adjacent to <h1>.

Text pseudo-elements

You have already learned some pseudo-elements (::before, ::after). There are also pseudo-elements for text. The following only work on **block-level elements**:

- ::first-letter
- ::first-line

::first-letter

You can use this to stylize the first letter of text for an element.

For example, you can make the first letter in the first paragraph of an article (of class "post") be a large, fancy letter.

```
.post p:first-of-type::first-letter {
  font-size: 3em;
  font-family: Georgia, "Times New Roman", Times, serif;
  float: left; /* this makes the letter a drop cap */
}
```

Notice that you can chain a pseudo-class with a pseudo-element selector. Also notice that the first <p> child was specified so that only the first letter of the first paragraph is styled. If you don't specify `:first-of-type`, all first letters for every paragraph will be styled.

Notice that the pseudo-class :first-of-type is used instead of :first-child. In order to use :first-child the paragraph **must be the first child within an element** (regardless of element type).

```
<article class="post">
  <h1>Heading</h1>
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
  <p>Paragraph 3</p>
  <p>Paragraph 4</p>
</article>
```

In the above HTML, `.post p:first-child` won't be called because the first child of `.post` is <h1>. What `.post p:first-child` is selecting is a paragraph which happens to be the first child *element* of .post.

To grab the first paragraph which is a child of `.post` (even if it is not the first child element), use `:first-of-type` because `:first-of-type` looks at the element type then looks at whether or not it is the first instance of that type.

::first-line

Use this to style the first line of text for an element.

For example, sometimes the first line of text in a paragraph is bolded. To do that, you would use `::first-line`.

Other pseudo-elements

::before

You can think of this as an invisible element which is located before the specified element. You generally use this to add **stylistic** content to an element using the content property. Do **NOT** use this to add meaningful or important content to your page because screen readers may not be able to read the content since it is added via CSS. Important or meaningful content should be in your HTML.
For example:

You can add a curly bracket before the page title with

```
h1::before {
  content: '{';
}
```

::after

Again, think of this as an invisible element located **after** the specified element. Remember that meaningful content should not be added via CSS.

To add a curly bracket after the page title, use:

```
h1::after {
  content: '}';
}
```

More useful pseudo-classes

Support for the following pseudo-classes is only available for newer browsers (2021+) so be careful about its usage. It's good to know what you will be able to use more in the future, however.

:is(<selector_list>)

https://developer.mozilla.org/en-US/docs/Web/CSS/:is

:is() allows you to select elements which match the selectors in the :is() parentheses. It's useful for shortening a long selector list. There's a good example in the link above.

:where(<selector_list>)

https://developer.mozilla.org/en-US/docs/Web/CSS/:where

:has(<selector_list>)

https://developer.mozilla.org/en-US/docs/Web/CSS/:has

This effectively works as a parent selector because you're selecting an element which has some other element.