

Cascading Style Sheets

Dr. Patrick Young, Stanford University

CS193C

Each style sheet consists of a set of style rules, with a selector which determines the elements on the webpage affected by the rule, and a set of declarations listing different properties set by the rule. Finding a website listing the different style properties is pretty easy. Finding a list of selectors is a bit harder, so I've summarized them here for you.

Selectors

The selector determines which elements are affected by a CSS rule. In this section we take a look at the selectors which are available.

Standard Selectors

The following selectors are in widespread usage.

Type Selectors

The most common selector is the type selector. Using a type selector, we specify the type of element affected by the rule—for example:

```
h1 {color: red}
```

tells the web browser that all `<h1>` elements should be displayed in red.

Descendant Selectors

We can limit the application of a rule in several ways. Using a descendant selector we tell the web browser to only apply the rule to tags which are descendants of another specific tag—for example:

```
ul li {font-weight: bold}
```

applies only to `` list item tags contained within `` unordered list tags. `` tags within `` ordered lists are not affected. We can test for as many layers of nesting as we want—for example:

```
ol ul li {font-weight: bold}
```

applies only to list items inside unordered lists which are contained within an ordered list.

Class Selectors

We can use the class attribute to place particular items into specific author-defined classes—for example:

```
<p class="important">CS193C is a great class</p>
<p>It covers a variety of technologies.</p>

<ul>
  <li>XHTML</li>
  <li class="important">JavaScript</li>
</ul>
```

identifies the initial paragraph and the JavaScript list item as being “important”. We can write a rule applying to all items in a class as follows:

```
.important {color: red}
```

This changes the first paragraph and the JavaScript list item to red. We can also combine a class selector with other selectors, such as the type or descendant selector like this:

```
p.important {color: red}
```

You can also define multiple classes on an element and test for them individually or separately. To define multiple classes, list them all in the same class attribute separated by spaces:

```
<li class="important fun">JavaScript</li>
```

JavaScript is now both important and fun! We can write a rule which only matches items with both classes as follows:

```
.important.fun {text-decoration: underline}
```

ID Selectors

You can associate style information with individual elements either by placing a style attribute directly on the item or by using an ID selector. Here’s a style attribute directly on an item:

```
<li style="text-decoration:underline">JavaScript</li>
```

Alternatively, if we place an ID on the item:

```
<li id="JavaScriptListing">JavaScript</li>
```

we can use an ID selector in our style sheet:

```
#JavaScriptListing {text-decoration: underline}
```

Remember, we can only have a single element in our entire webpage with a given ID.

Advanced Selectors

Here are some more advanced selectors.

Child Selectors

As its name suggests, the descendant selector works for any element which ultimately descends from the element listed. Using a child selector you can limit selection to direct descendants (i.e., children). The rule

```
body > table {border-color: red}
```

places a red border around tables which occur at the top level of the document *only*. In contrast:

```
body table {border-color: red}
```

places a red border around any table which descends from the body tag (which basically means all tables, since all webpage contents ultimately descend from the body tag).

Adjacent Selectors

You can select items adjacent to each other using the adjacent selector—for example:

```
li + li {text-decoration: underline}
```

underlines a list item, but only if it immediately follows another list item (this should affect all the items in the list, except the first one).

Attribute Selectors

There are additional selectors which check tag attribute values. This set of selectors, defined on CSS2, only affects tags which have particular attributes set—for example:

```
p[align] {color: red}
```

should turn all paragraphs with an align attribute set (to any value) to red.

```
p[align="center"] {color:blue}
```

should turn paragraphs with align attributes set to center to blue.

We may discuss these more when we get to XML. Check the CSS2 specification online for details.

Listing Multiple Selectors

If we want a rule to apply to more than one selector, we can list the selectors separated by commas. The rule

```
h1, h2, h3 {color: red}
```

sets all <h1>, <h2>, and <h3> tags to red. Similarly, the rule:

```
ul li, body table {font-weight: bold}
```

sets list items contained in unordered lists and tables contained within the <body> tag to bold.

Combining Selectors

In theory, all the selectors can be combined. Let's take a look at a few examples.

```
table h1.important {color: red}
```

sets all <h1> tags of important class which are inside tables to red.

```
table.important h1 {color: red}
```

sets all <h1> tags which are inside tables of class important to red.

```
body > table li + li {font-weight: bold}
```

sets all list items following another list item which are contained within a table which is at the top level of our document (i.e., which is an immediate child of the body tag) to bold.

Pseudo-Classes and Pseudo-Elements

Using the class attribute we can identify particular tags as associated with class. On its own, the web browser also classifies some tags. These browser-defined classifications are called pseudo-classes or pseudo-elements. I think this concept will make the most sense using the `:first-letter` class. According to CSS2 standard, all web browsers should identify the first letter in a tag's contents as a member of the `:first-letter` pseudo-element. If we write a rule as follows:

```
p:first-letter {color:red; font-weight:bold}
```

then the first letter of every paragraph will display in red and in bold. It's as if we'd gone in and added a `` `` around every first letter in every tag in our document.

The pseudo-classes which we will be using most are those associated with links on webpages. The standard defines several pseudo-classes related to links. These pseudo-classes are supported by all the latest web browsers.

`:link`—Linked text is usually displayed as if it were part of the `:link` pseudo-class.

`:visited`—Linked text which corresponds to a website which the user has visited acts as if it were part of the `:visited` pseudo-class.

`:active`—Linked text that the user is currently clicking on acts as if it were part of the `:active` pseudo-class.

`:hover`—If the user moves the mouse over a link, but doesn't click on it, the text acts as if it were part of the `:hover` pseudo-class.

We now define some rules using our pseudo-classes. While the `:active` and `:hover` pseudo-classes were originally associated with links, they now work on other element types as well, so if we want our rules to apply only to links, precede the pseudo-class with an `a` making it apply only to anchor elements.

```
:link {color: blue}
:visited {color: purple}
a:active {color: silver}
a:hover {color: red}
```

Using these rules, text and images that are linked, but not visited, will appear in blue. Text and images which are linked and have been visited will appear in purple. Linked text or images will turn to silver when the mouse is clicked on them. Linked text or images will turn to red when the mouse moves on top of them.