**CSS Syntax**

Observe the syntax of a single style declaration in your stylesheet:

selector {

property: value; /\* This is a comment. Don't forget your semicolons! \*/

}

The **values** of the **properties** apply to any HTML elements that match the given **selector**.

Here's how to style all <p> elements with blue, bold, sans-serif font.

p {

font-family: sans-serif;

font-weight: bold;

color: blue;

}

You can have as many style declarations as you wish in a CSS document. In some situations several selectors match similar elements, hence the name **Cascading** Style Sheets.

**Cascading Styles**

p {

color: black;

}

p span {

font-weight: bold;

}

p span.highlight {

color: yellow;

}

<p>

This is some text in a paragraph element.

<span>

This text has been styled bold because it is in a span element

that's a child of a paragraph element.

</span>

<span class="highlight">

This text is yellow because the class selector overrides

the tag selector when multiple style declarations apply to the same

attribute (`color`) of the same element. It's still bold though!

</span>

</p>

The above example introduces another type of selector and hints at how we can maintain DRY CSS using cascading/overiding styles.

**CSS Basic Selectors**

**Element Selector**

You can directly select HTML elements by stating their name.

Example: Select all <h1> elements and make their font size 24px.

h1 {

font-size: 24px;

}

**Descendant Selector**

Select descendants of elements by using a space between the names of parents and children. A descendant of a <ul>, for example, is an element that is nested somewhere inside a <ul>.

Example: Select all <a> elements that are descendants of <ul> elements and remove all text-decoration.

<ul>

<li><a href="example.com">Style Me!</a></li>

</ul>

ul a {

text-decoration: none; /\* This assures the a elements are not underlined \*/

}

**Child Selector**

The child selector is exactly like the descendant selector except it can only select the *direct children* of an element. That means if the element is nested too deep then it will not be selected.

<ul>

<li><a href="example.com">Style Me!</a></li>

</ul>

ul > a {

text-decoration: none; /\* this does not work \*/

}

ul > li > a {

text-decoration: none; /\* this does work \*/

}

**Class Selector**

Classes group elements together for styling purposes. Classes are more specific than selecting by an element. Any time you want a group of elements to have the same styling properties, you should give them the same class. Classes are extremely popular in front-end development and are key to making our style code DRY. They are also helpful by making our stylesheets more readable.

Reference a class selector with a .

<section>This is a normal section</section>

<section class="bg-grey">This section has a greyish background</section>

<header class="bg-grey">This header also has a greyish background</header>

.bg-grey {

background-color: whitesmoke; /\* there are many color names \*/

}

**ID Selectors**

ID attributes should be unique to their element. Selecting elements by their id can be helpful if an element has unique styling that it won't share with other elements. However, this is often not the case and leads to repetitive styling.

Reference an id selector by using a #.

<span id="banana">This span has a banana color</span>

#banana {

color: yellow; /\* banana is not a color name unfortunately \*/

}

Class and Element selectors will get you very far in CSS styling, and you should avoid using id for styling. We'll discuss more advanced selectors later in the following readings.

**Universal Selectors**

Universal selectors are usually too broad and are rarely necessary. The following is an example of where it can be useful, in this case it can be used a testing tool to see a border around every element on your page.

\* {

border: 1px solid red; /\* puts a red border around EVERY element \*/

}

**Overspecificity**

If you want to override properties defined for a "specific" (high-specificity) selector, you must use an *even more* specific selector. Multiple overrides in your styles can become difficult to manage, so it's important to follow this simple rule: **Be only as specific as necessary.**

You may run into !important in the wild, and you should question anyone using it in a non-testing scenario. You can add !important to a css property to make it override any other styling. It is the only styling rule that overrides inline styling in specificity.

Here is how it can be used:

header {

min-width: 500px **!important**; /\* sets the min-width of the header to 500px regardless of any other styling rules \*/

}

**Precedence**

Precedence governs which styles are overridden. The more specific a selector is, the higher its precedence. Selectors with higher precedence override those with lower ones.

Both the type of selector and how it's applied to the HTML document determine specificity. Here's the hierarchy of specificity (from most to least):

* !important
* inline styles
* ID selectors
* class selectors
* element selectors
* universal selector

**Additional Resources:**

* [CSS Precedence](http://www.vanseodesign.com/css/css-specificity-inheritance-cascaade/)
* [CSS-Tricks Precedence](http://css-tricks.com/specifics-on-css-specificity/)
* [CSS Selectors](http://www.w3.org/TR/CSS21/selector.html#pattern-matching)
* [30 Selectors to Memorize](http://net.tutsplus.com/tutorials/html-css-techniques/the-30-css-selectors-you-must-memorize/)