

# 98-130

## TECHNIQUES FOR WEB INTERFACES

### LESSON 2: "CSS IN A LECTURE"

#### LECTURE OVERVIEW

- Using CSS
  - CSS Rules & Selectors
  - Pseudo-Classes (:link, :visited, :hover, :active) and Pseudo-Elements (:first-line, :first-letter)
  - Inheritance
  - Units (Measurement & Colors)
  - Specificity
- 

#### RELATED READING

- List of Hex Values & Color Keywords  
[http://www.w3schools.com/HTML/html\\_colornames.asp](http://www.w3schools.com/HTML/html_colornames.asp)
- 

#### USING CSS

First, it's important to know that there is a crucial middle step between your CSS and the visuals rendered on the screen. Your CSS will be interpreted by some client program; usually this client is a browser like Internet Explorer or Safari, which, based on how the W3C standards say your CSS and HTML should act, will render the appropriate visuals to the screen. However, this isn't a perfect process because browsers aren't perfect. Internet Explorer, for example, has a majority of market share yet is known for its occasional lack of conformance to W3C standards and buggy rendering issues (though it is rapidly improving). So you should always remember to look at your page in all major browsers before calling it a job well done.

With that said and done, there are a few different ways to embed or link to CSS from your HTML. The best way is generally to define your CSS declarations in an external `.css` file and link to it using the `LINK` tag inside the `HEAD` of your HTML:

```
...  
<HEAD>  
<LINK rel="stylesheet" type="text/css"  
      href="filename.css" media="screen" />
```

...

The `media` attribute of the `LINK` tag is pretty interesting; it allows us to specify exactly what medium that stylesheet is served for. Common values are `screen` (traditional computer screens), `print` (printers), `projection` (projectors), `tv` (televisions), or `all`.

---

## CSS RULES & SELECTORS

Let's take a look at an example of a CSS *rule*:

```
p { color: red; padding: 10px; }
```

Before we examine what this does, let's break it apart. Rules are made up of three elements: *selectors*, *properties*, and *values*. The selector indicates which HTML elements to apply your style to. A selector is always followed by a set of property/values in the form of "property: value;" – these are enclosed in curly braces. So, in this example, the selector would be `p`, and there are two property/value pairs (one that says that the text color should have a value of red, and the other that says that the value of padding is 10px). Thus, all elements indicated by the `p` selector should have a text color of red and a padding of 10 pixels. We'll get into the details of padding, units, and so forth soon, but first let's examine what different kinds of selectors are available and what they do.

**TYPE SELECTORS** This is the kind of selector used above. Basically, just name the HTML element or "type" that you want the rule to affect; in our case, all `P` tags on the page will be affected.

**DESCENDANT SELECTORS** This kind of selector allows you to target the *descendants* of a particular HTML element (in other words, the elements that occur inside of the specified ancestor element). The syntax for a descendant selector simply places a space between the ancestor and its descendant. For instance:

```
p strong { color: red; }
```

will affect any `STRONG` tags that are inside of a `P` tag. Likewise,

```
div p strong { color: red; }
```

will affect any `STRONG` tags that are inside of a `P` tag that are inside of a `DIV`.

**ID AND CLASS SELECTORS** Remember the `id` and `class` attributes in HTML? We can target those with selectors, too. The syntax for an ID selector is a hash followed by the ID name, and the syntax for a class selector is a period followed by the class name. For example:

```
#footer { color: red; }  
.secondaryNav { color: red; }
```

The former will target the element with `id="footer"` and the latter targets all elements with `class="secondaryNav"`. We can also prepend type selectors, as follows:

```
div.secondaryNav { color: red; }
```

This will limit the selector to only DIVs with `class="secondaryNav"`.

**UNIVERSAL SELECTOR** The `*` is the universal selector, and will match everything. Thus:

```
p * { color: red; }
```

will affect *all* descendants of P tags.

**OTHER SELECTORS** There are some cool other selectors, too. Unfortunately, they're mostly unsupported by Internet Explorer 6 and below so we won't get into them. Most selectors you will ever use will be just be type, descendant, ID, and class selectors (or combinations thereof).

As a quick sidenote, we can also separate multiple selectors by commas. For instance, if we wanted to target all first-level and second-level headers and make them red, we could say:

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

instead of breaking this into two rules.

## PSEUDO-CLASSES & PSEUDO-ELEMENTS

CSS also includes selectors for things called "pseudo-classes" and "pseudo-elements" which give us even more power over what we select. Both have the same syntax; they are preceded by a colon and must come after some other selector (for example, `a:link` or `p:first-line`).

**PSEUDO-CLASSES** Pseudo-classes give us a way of selecting a part of an element based on some dynamic or invisible criteria. These are some of the major pseudo-classes in CSS:

- `:link` will select any link that has not been visited before. For example, `a:link { color: blue; }` will turn all unvisited links blue.
- `:visited` does the opposite; it selects any link that *has* been visited before.
- `:hover` affects any element that is currently being *hovered* over; in other words, the element that the user has his mouse over at the time. A common mistake that people make is in thinking that this element applies only to links; it actually doesn't, which means you can do some really cool stuff using only CSS. The reason for this misunderstanding is that, for all versions before IE7, Internet Explorer only supported `:hover` when applied to links.

- `:active` will affect an *active* element; this usually refers to a link that you've clicked. So, for example, `a:active { color: red; }` will turn a link red for the time that you've clicked on it.

**PSEUDO-ELEMENTS** Pseudo-elements are slightly different in definition from pseudo-classes. They select some sub-part of an existing element. Here are the two most commonly used pseudo-elements in CSS:

- `:first-letter` will select the first letter of the text in an element. This is commonly used to do drop caps.
- `:first-line` will select the first line of the text in an element.

---

## INHERITANCE

Certain properties in CSS, including `color` (which defines the text color) and `font-size` (self-explanatory), are *inherited* by the descendants of the elements those styles are given to. In other words, let's take the following example HTML:

```
...
<body>
<h1>This is a header!</h1>
<p>Lorem ipsum dolor sit amet.</p>
</body>
...
```

Note that the `H1` and `P` elements are descendants of the `BODY` tag (aka, they are contained within the `BODY`). If we apply the style `body { color: red; }`, the `color` property will be inherited by those two elements; thus, both the `H1` and `P` text will be red. This can simplify your CSS a great deal.

---

## UNITS (LENGTH & COLOR)

We've seen that many CSS properties want values expressed in a unit of length or color. For example, the `color` property obviously demands a color value while `font-size` needs a unit of length.

**MEASUREMENT** Units of measurement can be divided into two subcategories: relative and absolute. *Absolute units* include in (inches), cm (centimeters), mm (millimeters), pt (points), and pc (picas). These units are mostly used for stylesheets dedicated towards printers. Only points are commonly used, and 1pt is classically defined as 1/72 of an inch. (A pica is 12 points.) Thus, we could define a CSS rule that says `h1 { font-size: 32pt; }` and all `H1` elements (first-level headers) will be in a 32 point font.

*Relative units* include px (pixels), em (ems), and % (percentages). Pixels are self-explanatory, but why are they considered relative? Mostly, because the size of a pixel differs from display to display depending on screen resolution, etc.; for absolute units like points, on the other hand, it is assumed that the display format is known (print media) and that the pt is always 1/72".

Ems are tricky, though; 1em equals the font size of the specified element. If the font size hasn't yet been defined (ie, if we're using ems to size the font itself), 1em refers to the font size of its parent. For example, in a paragraph with 16-pixel text, 1em is equivalent to 16px and 0.5em is equivalent to 8px. This is handy for creating a web site whose proportions are based around the text size; we'll discuss this technique ("elastic layouts") in our fourth lecture.

Percentages are similar to ems, but aren't necessarily based on font size. Instead, they're based on whatever the parent's value is for the property in question. So if we have an element (that has a width of 30%) inside of a DIV (that has a width of 300px), then that element will have an effective width of 100px. Note that percentage is basically the same as em if you use it for font-size.

**COLOR** We can use a *keyword* (like `red` or `aqua`) to define a color. More commonly, however, colors are expressed in a *hexadecimal* (16 digits) format like `#xyyzz`, where `xx` defines the amount of red, `yy` defines the amount of green, and `zz` defines the amount of blue. For example, `color: #ffffff;` is the same thing as `color: white;`. Both will render text with a white color.

A list of common colors (keywords and hex values) is available at [http://www.w3schools.com/HTML/html\\_colornames.asp](http://www.w3schools.com/HTML/html_colornames.asp).

---

## SPECIFICITY

So what if two CSS rules conflict with each other (for example, one says that the color of an element is blue, but the other says that the color of that same element is red)? Each rule is given a *specificity* defined by four numbers in the form of a,b,c,d.

- If the element is an inline style, a = 1. Since we're not really going to be using inline styles, we won't worry about this.
- b = the number of IDs in the rule.
- c = the number of classes and pseudo-classes in the rule.

- $d$  = the number of pseudo-elements and type selectors in the rule.

The rule with a higher value of  $a$  will take precedence; if they're equal, then the rule with a higher value of  $b$  takes precedence, and so forth. Check out the following table of examples, with increasing precedence:

Selector	Specificity	Notes
<code>div p</code>	0, 0, 0, 2	Two type selectors.
<code>div .comment</code>	0, 0, 1, 1	One type, one class.
<code>div p.comment</code>	0, 0, 1, 2	Two type, one class.
<code>#navigation a</code>	0, 1, 0, 1	One ID, one type.
<code>#navigation a:visited</code>	0, 1, 1, 1	One ID, one type, one pseudo-class.

`#navigation a:visited` would take precedence over `#navigation a`, and `div p.comment` would take precedence over `div p`.

If the two rules that are in conflict have identical specificity, then the rule that was defined last in your CSS wins.

## NEXT LECTURE

Now we're armed with a pretty good overview of the basic blocks of CSS. Next lecture we'll conclude our crash course in CSS by explaining the *visual flow* of a web page and its elements. This will be the last basic piece of the puzzle, and then we'll start to do some cool hands-on stuff.