What?

Okay, so we know a bit about CSS now; enough to use it for basic application to fonts, borders, margin and padding. We’re going to learn how to apply css to more than just HTML elements using class and ID selectors.

Last week we looked only at HTML selectors - those that represent an HTML tag. Classes and ID’s open up a whole new world of presentation for HTML selectors; they can be displayed differently depending on a unique name for that class or ID.

Application

In the CSS, a class selector is a name preceded by a period or dot ("."), and an ID selector is a name preceded by a pound sign ("#.").

The CSS itself might look something like this:

```css
.highlight {
  color: #a00000;
  font-weight: bold;
}
```

```css
#left-callout {
  background-color: #333;
  padding-left: 20px;
}
```

The HTML applies the CSS by using attributes (id and class). An Example:

```html
<div id="left-callout">
  <h2>Subheader</h2>
  <p class="highlight">This is an intro callout.</p>
  <p class="highlight">This is a second paragraph.</p>
</div>
```

An ID can be used to identify one element per page, so use it for structural items or elements that you don’t want to repeat. Classes can be used many times per page, and can be repeated for selectors.

You can directly apply a selector to a specific HTML element by stating the HTML selector first:

```css
p.first {color:#333; }
```

The above will be applied to paragraph elements that have the class "first."

Grouping and Nesting

You can give the same properties to a number of selectors without having to repeat them.

```css
h1 {
  color: #000099;
}
```

```css
.blue {
  color: #000099;
}
```

```css
.tertiary {
  color: #000099;
}
```

You can separate selectors with commas and apply the same properties to them all.

```css
h1, .blue, .tertiary {color: #000099;}
```

You can also specify properties to selectors within other selectors, opening up more flexibility:

```css
#left-callout {
  background-color: #333333;
  padding-left: 20px;
}
```

```css
#left-callout h1 {
  color: #999;
}
```

```css
#left-callout p {
  color: blue;
  font-weight: bold;
}
```
Pseudo Classes

Pseudo classes are applied to selectors to specify a relation to the selector. They take the form of `selector:pseudo_class { property: value; }`. Note the use of the colon separating them.

Links

Links are the most basic pseudo classes. "link" applies to "unvisited links" while "visited" applies to links that have been visited by the user.

```css
a:link { color: blue; }
a:visited { color: red; }
```

Dynamic Pseudo Classes

The dynamic pseudo classes can be used to apply styles when actions are performed.

- **active** is applied when something activated by the user, such as when a link is clicked on (and held).
- **hover** is applied when a cursor moves over a link.
- **focus** is applied when something gains focus, that is when it is selected by, or is ready for keyboard input. It can be accessed using the tab key.

```css
a:active { color: #0000ff; }
a:hover {
  text-decoration: none;
  color: #ff00ff;
  background-color: #ffff00;
}
```

First Children

This will target something only if it is the very first descendant of an element:

```html
<body>
<p>This is the first p child of the body element.</p>
<p>This is a second paragraph.</p>
</body>
```

CSS:

```css
p:first-child {
  font-weight: bold;
  font-size: 20px;
}
```

CSS3 contains even more pseudo classes: last-child, target, first-of-type, etc. We will explore those in-depth next week.

Shorthand Properties

Some CSS properties allow a string of values, replacing the need separate lines of CSS. These are represented by values, separated by spaces.

```css
p {
  margin-top: 5px;
  margin-right: 15px;
  margin-bottom: 10px;
  margin-left: 25px;
}
```

becomes ...

```css
p {
  margin: 5px 15px 10px 25px;
}
```

The values are specified in a clockwise manner: top, right, bottom, left.

Borders can be specified similarly. You can combine border-width, border-color, and border-style within the border property:

```css
p {
  border-width: 2px;
  border-color: #333;
  border-style: solid;
}
```

becomes ...

```css
p {
  border: 2px #333 solid;
}
```
Font-related properties can also be shortened with the `font` property:

```css
p {
  font: bold italic 12px/3 "Helvetica", sans-serif;
}
```

This combines `font-weight`, `font-style`, `font-size`, `line-height`, and `font-family`.

Let's combine the examples:

```css
p {
  font: bold italic 12px/3 "Helvetica", sans-serif;
  padding: 20px 15px;
  border: 1px solid;
  border-width: 2px 5px 10px 15px;
  border-color: blue green yellow red;
  margin: 15px 25px;
}
```

### Background Images

Besides using the `img` HTML element, employing CSS background images are a powerful way to add presentation to a page.

The shorthand property `background` can deal with all of the basic background image manipulation aspects.

```css
body {
  background: #ccc url(bkg.png) no-repeat top right;
}
```

*This utilizes these properties ...*

- **background-color**: explained previously.
- **background-image**: the location of the image itself.
- **background-repeat**: how the image repeats itself. values can include:
  - `repeat`: a “tiling” effect across the background,
  - `repeat-y`: repeating on the y-axis, vertically
  - `repeat-x`: repeating on the x-axis, horizontally
  - `no-repeat`: shows just one instance of the image.
- **background-position**: which can be top, center, bottom, left, right, a length, or a percentage, or any sensible combination, such as top right.

Background-images can be used in most HTML elements.

### Specificity

If you have conflicting CSS rules applying to the same element, there are some basic rules that a browser follows to determine which one is more specific.

If selectors are the same then the last one will always take precedence:

```css
p { color: #ccc; }
p { color: #000; }
```

The text within `p` elements would be colored black because that rule was specified later.

You won’t typically have identical selectors with conflicting declarations on purpose. Conflicts can present themselves when you have nested selectors.

```css
div p { color: #ccc; }
p { color: #000; }
```

In this example it might seem that a `p` within a `div` would be colored black, seeing as a rule specific to `p` is shown afterwards.

However, they would actually be colored gray due to the specificity of the first selector. Specificity overrides the cascade.

### Calculating

The actual specificity of a group of nested selectors takes some calculating. In a nutshell, you give every ID selector ("#name") a value of 100, every class selector (".name") a value of 10 and every HTML selector ("element") a value of 1. Add them up to calculate the specificity value.

- `p` has a specificity of 1 (1 HTML selector)
- `div p` has a specificity of 2 (2 HTML selectors)
- `test` has a specificity of 10 (1 class selector)
- `div p.test` has a specificity of 12 (2 HTML selectors + 1 class selector)
- `#container` has a specificity of 100 (1 id selector)
- `body #container .test p` has a specificity of 112 (HTML selector + id selector + class selector + HTML selector)
**Display**

The browser’s default visual representation of most HTML elements consist of varying font styles, margins, padding and, essentially, display types.

The most fundamental types of display are **inline**, **block** and **none**. They can be manipulated with the display property.

**Inline**

Elements that are displayed inline follow the flow of a line. `<a>` tags and `<em>` tags are examples of elements that are displayed inline by default.

Inline display would be indicated by the following CSS (if necessary)

```
li { display: inline }
```

**Block**

Block display makes use of line break before and after the element. Block display allows greater manipulation of height, margins, and padding. Heading and paragraph elements are examples of elements that are displayed this way by browsers (in a default manner).

```
#navigation a {
    display: block;
    padding: 15px 10px;
}
```

**None**

`none` doesn’t seem to have much functional use, but can be used for dynamic effects and print stylesheets.

```
#navigation, #links { display: none }
```

**Pseudo Elements**

Pseudo elements latch onto selectors much like pseudo classes, taking the form of selector:pseudoelement { property: value; }.

**First Letters and First Lines**

The first-letter pseudo element applies to the first letter inside an element and first-line applies to the top line in an element.

You could create drop caps and a bold first-line with something similar to this:

```
p { font-size: 14px; }
p:first-letter { font-size: 24px; float: left; }
p:first-line { font-weight: bold; }
```

**Before and After Content**

The before and after pseudo elements are used in conjunction with the content property to place information on either side of an element.

The value of the content property can be *open-quote*, *close-quote*, any string enclosed in quotation marks, or any image, using `url(imagename)`.

```
blockquote:before {
    content: open-quote;
}
```

```
blockquote:after {
    content: close-quote;
}
```

```
li:before {
    content: "TEST! ";
}
```

```
p:before {
    content: url(images/test.jpg);
}
```

You can also add styles to the “presentational content”:

```
li:before {
    content: "TEST! ";
    background: #ccc;
    color: #f0c;
}
```
Example of the content property:

```css
.email-address:before {
  content: "Email address: ";
}
```

HTML::

```html
<ul>
  <li class="email-address">david.hardy2@csn.edu</li>
</ul>
```

"Email Address" won’t show up as part of your content; it’s only there for presentational purposes.

## Layout Design

Layout with CSS is relatively easy when the position property comes into play.

The position property is used to define whether an element is **absolute, relative, static** or **fixed**:

- Static is the default value and renders an element as they appear normally in the HTML.
- Relative can be offset from its original position with the properties top, right, bottom and left.
- Absolute positioning can be placed anywhere on the page using top, right, bottom and left values.
- Fixed is similar to absolute, but it will absolutely position a box in reference to the browser window, so fixed elements will stay exactly where they are on the screen even when the page is scrolled.

### Layout using absolute positioning

You can create the appearance of two-column layout with absolute positioning:

```html
<nav>
  <ul>
    <li><a href="#">test</a></li>
    <li><a href="#">test</a></li>
    <li><a href="#">test</a></li>
  </ul>
</nav>
<article>
  <section>
    <p>Maecenas pharetra sem id malesuada tempor. Duis non orci scelerisque, convallis orci a, luctus leo.</p>
    <p>Donec porta ultricies blandit. Nam sit amet nunc mollis, tempor est at, facilisis nibh.</p>
  </section>
</article>
```

Apply the following CSS:

```css
nav {
  position: absolute;
  top: 15px;
  left: 30px;
  width: 200px;
}
section {
  margin-left: 200px;
}
```

You will see that this will set the navigation bar to the left and set the width to 200 pixels. Because the navigation is absolutely positioned, it has nothing to do with the rest of the page. The margin of the `<section>` is separate and unaffected.

A downside to absolutely positioned boxes is that they don’t scale well or adapt to screen sizes.

### Floating

Floating an element will shift it to the next "column."

You can float: left or float: right.

Working with the same HTML, you could apply the following CSS:

```css
nav {
float: left
}
section {
float: right;
}
```
If want the next element to break from the floating objects, you can apply the clear property:

- **clear: left** will clear left floated boxes
- **clear: right** will clear right floated boxes
- **clear: both** will clear both left and right floated boxes.

So if, for example, you wanted a footer in your page, you could add a chunk of HTML...

```html
<footer>
  <p>test footer</p>
</footer>
```

...and then add the following CSS:

```css
footer {
  clear: both;
}
```

A footer will appear underneath the columns, regardless of the length of either of them.

**Next**

In the final CSS lesson, we will apply rounded corners, gradients, advanced colors, transitions, and use media queries to display content on different sized devices.