Introduction to Cascading Style Sheets

Giving your Web pages some style!

What exactly is CSS?

- CSS is an abbreviation for *Cascading Style Sheets*
- The W3C (The World Wide Web Consortium) defines CSS as:

  …a simple mechanism for adding style (e.g. fonts, colors, spacing) to Web documents.

- CSS is a separate language with its own *syntax* distinct from HTML
- CSS allows us to separate our marked-up content from the manner in which that content is presented
- Without CSS, Web pages would all look pretty dull

Why use CSS?

- *Better type and layout controls* – A designer can do things that are not possible with HTML alone
- *Less Work* – Edit one stylesheet to change the presentation of an entire site
- *Potentially smaller documents* – <table> code and multiple <font> tags are unnecessary
- *Improved accessibility* – for mobile and non-visual devices
- *Reliable browser support* – mostly

Remember the *layers* of Web design?

- We've already learned how to add structure and *meaning* to our document's content using HTML markup
- Now we shall change the *presentation* of this content using style rules
- You can think of this in another way:
  - HTML provides a structural and *semantic* context for our content
  - CSS will provide a *presentational* context for our content

A Very Important Note

- Every browser (Opera, Safari, Chrome, Firefox, even IE!) has a built-in default style sheet!
- When a page is displayed, this style sheet is applied *first* (but not actually rendered)
- After the default style sheet is applied, any styles provided by *the page* are applied
- Default style sheets cause much of the formatting you see in unstyled pages:
  - tags are rendered in bold
  - tags are rendered in italics
- If there was no default style sheet, and would render as plain text!

What are style rules?

- A style rule consists of three (3) parts:
  1. A selector that provides the *connection* between the rule and our semantic mark-up
  2. A set of *braces* ("{…}") that define the beginning and end of a rule's description
  3. The CSS *property (properties)* that define how elements using this selector should be displayed.

- The "big three" selectors:
  - *Type* selector – rules that are associated with HTML tags
  - *Class* selector – rules applied to a tag using the tag's class attribute
  - *ID* selector – rules applied to a tag using the tag's id attribute

Creating a Style Rule

- Imagine a situation where we want *all* of our level-2 headers to be red instead of the color specified in the browser's default style sheet
- Our first step would be to create the general structure of the rule – the tag followed by a set of braces

```
h2 { }
```

- The definition of our style will go between the braces

Creating a Style Rule Part 2

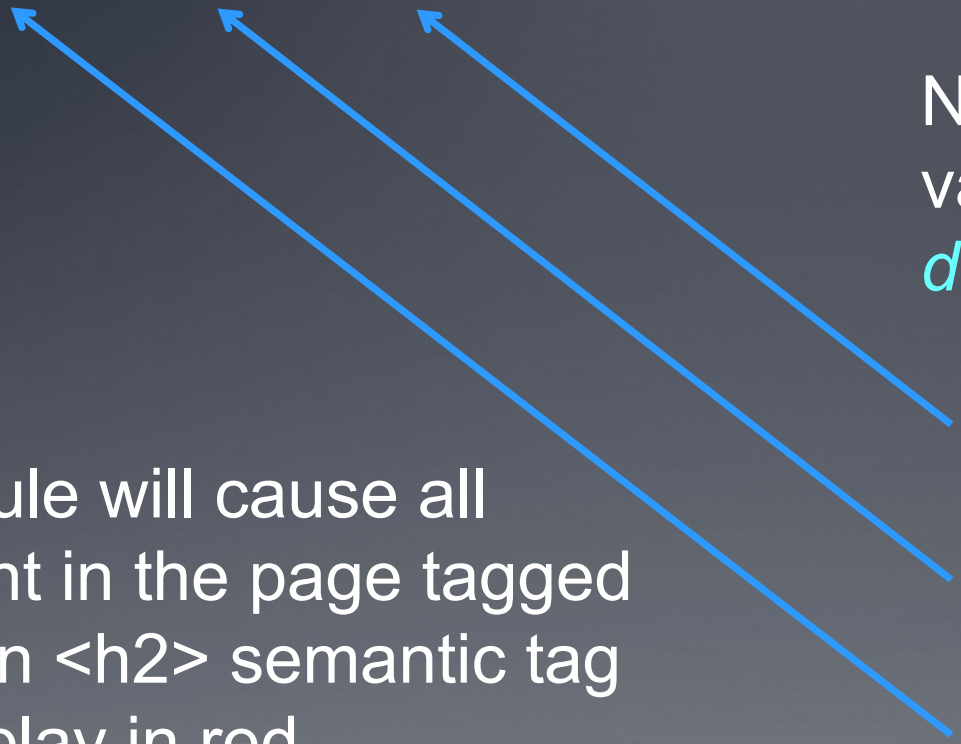
- Now we add the *properties* for this style
  - In this case, we want to set the font color to *red*
  - To do so, we will use the CSS property: *color*
- The completed style rule would look like this:

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

- Notice the colon (“:”) character separates the CSS property and its value
- Also note that a CSS property ends with a semicolon (“;”)

Your First Style Rule

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



Note: the property and value pair is called the *declaration*

Value

Property

Type Selector

This rule will cause all content in the page tagged with an <h2> semantic tag to display in red

Styles with Multiple Declarations

- What if we wanted to do more than just change the color of the level 2 headers? What if:
  - We want the color to be red
  - We want the font size to be 30 pixels
  - We want the font itself to be Trebuchet MS
  - If the user doesn't have Trebuchet MS, use Arial
- That style would look like this:

```
h2 {
    color: red;
    font-size: 30px;
    font-family: "Trebuchet MS", Arial;
}
```

Some General Rules For Style Declarations

- Note that the declarations (the property/value pairs) are separated by semicolons, and all of the declarations for the selector are enclosed in braces
- If the value has a *unit*, there cannot be a space between the value and the unit, e.g.:

```
font-size: 30px Not font-size: 30 px
```

- If there is a space in a value, such as the name of a font, you should put that value in quotes.

```
font-family:"Trebuchet MS", arial;
```

Applying One Rule To Multiple Type Selectors

- Imagine that you want all of your level-1 through level-4 headers to be colored red and use Trebuchet MS as the font with Arial as the alternate

```
h1, h2, h3, h4 {
                    color: red;
                    font-family: “Trebuchet MS”,
                                 Arial;

}
```

- This rule applies to all four header tags anywhere in the document
- Also, notice that (like HTML) CSS ignores unquoted white space! (Except for units as mentioned earlier)

Adding CSS Rules to Your HTML Document

- Style rules for a page can be added three different ways:
  1. *Inline* – the style is added as an *attribute* to an HTML element within the document:

```
<h2 style="color: green;">Welcome!</h2>
```

  2. *Embedded* – (Also known as *Document Level*) Style rules are placed within a set of <style> tags in the <head> section of the document
  3. *External* – the style rules are listed in a separate document. The CSS document "connects" to the HTML document through a *link* tag in the <head> section. (More about that in the next class)

Now that I know about Type Selectors…

What about class and ID selectors?

A new kind of selector, class

- *Class Selectors* let you designate a style that can be used with multiple, unrelated elements
- The name of the class selector always starts with a "." (dot or period)
- HTML elements are assigned to these classes using the *class* attribute

Example of declaring and using a *class* selector…

```
.assignment{
   color: green;
   font-weight: bolder;
   font-size: 125%;
}
…

<h3>Due next week</h3>
<ol>
   <li class=“assignment”>Project 1</li>
   <li class=“assignment”>Project 2</li>
   <li>Poetry Reading</li>
</ol>
```

…Renders As:

Due next week

1. **Project 1**
2. **Project 2**
3. Poetry Reading

What About the *id* Selector?

- *id selectors* let you designate a *single* element on a page as the target of your style rule
- The name of the *id* selector always starts with a # symbol
- HTML elements are assigned to these groups using the *id* attribute

Declaring and Using the *id* selector

```
#footer{
   color: yellow;
   font-size: 80%;
}
…

<p id="footer">&copy; Copyright 2010</p>
```

- Renders as:

  © Copyright 2010

- Remember: an *id* can only be used *once* on a page! If you need to use the same style in multiple locations, create a *class* selector

Bonus Selectors – *Pseudo Class Selectors*

- A couple examples:
  - :hover for rollover effects
  - :first-letter for a "drop cap" effect

```
a:hover{
    color:red
}

p:first-letter{
    font-weight:bold;
    font-size: 2em;
}
```

Properties, Values, and Units of Measure

More about CSS rules and their declarations

CSS Property Values

- The *values* of CSS properties can be specified using one of the following value types:
  - Keywords
  - Absolute Units
  - Relative Units

Keywords

- Keywords map to values that the browser can understand. For example:
  - *medium* sized text is text set at the user's preferred font-size
  - *large* text is a certain percentage larger
  - A *green* background will be rendered by the browser as the hexadecimal color `#00FF00`
  - *solid* or *dotted* can be used to specify how a border is rendered
- There are many other keywords available for use – check your text and on-line documentation for more information

Absolute Units

- *Absolute* units are most useful for printing or if we are ***absolutely*** sure of the device's resolution (for example, a page being viewed on a standardized display or public kiosk)
- The most common absolute unit is the *point* (pt). A point is 1/72 of an inch
- There are other absolute units available, but these are rarely used:
  - **in** (inches)
  - **cm** (centimeters)
  - **mm** (millimeters)
  - **pc** (picas, 1 pica is equal to 12 points)

Relative Units

- *Relative* units specify a size *relative* to the parent container's size property
  - The *em* unit when used with `font-size` is relative to the computed font-size of the parent element
    - For example,

```
h1 {font-size: 1.2em;}
```

      for an h1 element means that the font-size for that element will be 20% larger than the size inherited from the body (parent) tag
  - *Percentages*, such as font-size: 120%, would be comparable to font-size: 1.2em

Relative Units

- The *ex* unit, which is not used frequently, is *relative* to the x-height of that font (the height of a lower-case *x)*.
  - Note: A lower-case *x* is usually the shortest letter in any given font.
  - For example, border-width: 0.5ex would make a border ½ the height of a lower-case x of that font.

The Pixel Unit: A Special Case

- The *pixel* (px) unit specifies the size of an entity based on the number of pixels used to display that entity
- Many designers think of this as an absolute unit
- *It's not* – the size is *relative!*
  - It depends on the size of a pixel on the user's display
  - For example,

```
font-size:72px
```

might appear to be ½" tall on a mobile device, but 1" on a lower-resolution, desktop computer

TMI! I'm Overwhelmed! What Should I Use?

- Generally, your best choice is to stick with *relative* units
- Starting out with pixels (the *px* unit) for all of your measurement units means that your page layout has a better chance of being preserved even if the user is on a mobile device with a high-resolution screen
- Many designers use the *em* unit for the same reason
- There are subtle differences, so check out some CSS design sites and:

  http://www.w3.org/TR/CSS2/syndata.html#values

- For now, when in doubt, use *px* for the screen, and *pt* for printing

CSS Property and Value Examples

Property	Possible Values
color:	red (color keyword); #FFOOOO; #FOO; rgb(100%, 0,0); rgb(255,0,0)
margin-left:	5pt (absolute points); 5px (relative/pixels); 2em (relative/em); 10% (relative/percent)
border:	1px solid black (1 pixel high, solid black border); 5px dotted red (5 pixel high, dotted red border);
list-style-type:	circle (keyword); url(circle_pic.gif) (external file)
background-image:	url(me.jpg) (external file); none (keyword)

CSS Property and Value Examples

Property	Possible Values
background-color:	green (color keyword); #OOFFOO; #OFO; rgb(0, 100%, 0); rgb(0,255,0)
line-height:	14px; 1.4em; 140% (relative sizing) 14pt (absolute sizing)
text-align:	left; right; center; justify (alignment keywords)
font-weight:	normal, bold, bolder (weight keywords)
font-family:	“Times New Roman”, serif (serif-style font list) Arial, san-serif (san-serif-style font list)

CSS Property and Value Examples

Property	Possible Values
font-size:	Fixed Sizes: 14pt, .3in, 150mm, 15cm Relative Sizes: 14px, 1em, 2ex, 140% Keywords: xx-small, x-small, small, medium, large, x-large, xx-large
font-variant:	Keywords: smallcaps, normal
font-style:	Keywords: normal, italic, oblique
text-decoration:	Keywords: none, overline, underline, line-through, blink
text-transform:	Keywords: none, capitalize, lowercase, uppercase

How Do I Know What Tags Have Which Properties?

- Online resources:
  - W3C: http://www.w3.org/
  - W3 schools: http://www.w3schools.com/css/css_reference.asp
  - Mozilla Developer Network: https://developer.mozilla.org/en-US/
  - Books 24x7: http://wally.rit.edu/electronic/ebooks.html
- Books
  - Numerous resources at your bookstore. Browse before you buy.

Putting the "C" in CSS: The *Cascade* In Action

How elements *inherit* properties

Inheriting style rules

```
p{color: green;}
…
<p>This is <em>pointless</em>!</p>
```

- The markup above will render as:

This is *pointless*!

- The tag *inherits* the green color from the parent <p> tag, but then adds in its own default appearance (the *italicized* text)
- Remember, the default appearance for the tag is built into the browser's default style sheet

Overriding Inheritance

```
p{color: red;}
em{color: green;}
…
<p>This is <em>pointless</em>!</p>
```

- The markup above will render as:

  This is *pointless*!

- The tag *inherits* the red color, but then *overrides* it with a green color.

Not All Properties Will Inherit

```
p{
   color: red;
   border: 1px solid black;
   background-color: gray;
}

   em {color: green;
}
…    <p>This is <em>pointless!</em></p>
```

- Will render as: This is *pointless*!
- Note the tag *inherits* the background-color: gray, but *does not inherit* border: 1px solid black.

How do I know which properties inherit, and which do not?

- *Reason it out*. *Should* nested elements inherit such properties as *border*, *margin* and *padding*? See the previous example. What if the browser allowed the *em* tag to inherit *border*?
- *Test it* in a web browser (actually, test it in *multiple* browsers!)
- See the *online resources* we showed you earlier

When Styles Conflict, Who Wins?

- For example, what happens when an *external* style sheet says that <p> tags are red, and an *embedded* style sheet (in the <style> tag) says that <p> tags are green?
- When styles conflict, the more *specific* rule wins:
  - An *embedded* style beats an *external* style sheet
  - An *inline* style beats them both
  - Inherited declarations are beaten by more specific declarations – see the inheritance examples a few slides back
- How this final (or "used") value is computed is what the "Cascade" in CSS is referring to. See http://www.w3.org/TR/CSS2/cascade.html for the details.

Debugging CSS…

- Right click!
- Inspect!

CSS Applied – get inspired

- Sites:
  - CSS Zen Garden - http://www.csszengarden.com/
  - A List Apart - http://www.alistapart.com/
- CSS 3 (like HTML5, "evolving standard")
  - http://www.css3.info
  - http://www.w3.org/Style/CSS/current-work
  - "CSS Site of Awesomeness"
    - no images used
    - view in Safari for best results.
    - http://www.bobwei.net/cssEffectsPart1/index.html

That's enough for now.

- Demo and ICE!