APPLIED COMPUTING 1P01
 Fluency with Technology
 Cascading Style Sheets (CSS)

APCO/IASC 1P01

Brock University
HTML

Remember web pages?

- A document is composed of plaintext that’s then parsed and used to render a web page to the user.
- The document’s source contains markup for describing layout, roles, etc., but we consciously avoided using it to describe pure decoration and stylistic choices.
- In modern web pages, it’s advised to separate content from styling.
What Are Styles?

Actually, just what they sound like...

- Elements within a web document have formatting *rules* applied to them, so the browser knows to render them differently
- Styles control everything from font type and formatting to layout and positioning of elements
How To Set Styles

There are three basic ways:

- Per element
- Specified in the head of the document
- Specified in a separate document, and then linked from within the page

But first, let's look at a simpler approach...
Tables

Remember HTML tables? Do you recall how we enabled the border on it?

- We set the table’s `border` attribute to "1"
- This approach is frowned upon
  - It doesn’t separate the styling from the content
  - It doesn’t give us as much flexibility as newer CSS-based approaches

E.g.

```html
<table border="1">
<tr><th>A</th><th>B</th></tr>
<tr><td>1</td><td>2</td></tr>
</table>
```
Table Borders

How else could we set that border?

```html
<table style="border: 1px solid black;">
  <tr style="border: 1px solid black;">
    <th style="border: 1px solid black;">A</th>
    <th style="border: 1px solid black;">B</th>
  </tr>
  <tr style="border: 1px solid black;">
    <td style="border: 1px solid black;">1</td>
    <td style="border: 1px solid black;">2</td>
  </tr>
</table>
```

- It’s probably worth pointing out here that the reason we’re learning how to use CSS correctly is so we *don’t* write nonsense like this!
CSS
Cascading Style Sheets

- CSS is based on the principle of setting up general rules that selectively apply to elements, depending on criteria (*selectors*).
- Any single element might have multiple properties set to its style:
  - An element might qualify under multiple selectors
  - An element might inherit from an encapsulating element
    - This makes sense: if you have a block of *blue* text, and three words of it are bolded, wouldn’t you expect those bolded words to also be blue?
- A single style property could potentially be applied to multiple qualifying elements:
  - Elements might be selected based on id, tag, *class*, or through more complicated rules.
When a browser renders a document, it has a basic hierarchy for applying styles. When a later-encountered style conflicts with an existing style, the new one replaces it. The sequence is pretty simple:

1. Browser’s Internal Style Sheet
2. User-Defined Styles
3. Webpage’s External Style Sheets
4. Webpage’s Embedded Style Sheets*
5. Element’s Inline Styles

*Actually, External and Embedded Style Sheets have the same priority, but this is assuming the document was written according to good conventions.
Style Rules

A *Style Rule* requires:

- A *selector*
- Pairs of *properties* and *values*
  - Each rule is terminated with a semicolon (;
  - The rules for a selector are encapsulated within curly braces (\{ and \})
Embedded CSS

Let’s revisit that table!

```html
<html>
  <head>
    <style type="text/css">
      table { border: 1px solid black; }
      th { border: 1px solid black; }
      td { border: 1px solid black; }
    </style>
  </head>
  <body>
    <table>
      <tr>
        <th>A</th><th>B</th>
      </tr>
      <tr>
        <td>1</td><td>2</td>
      </tr>
    </table>
  </body>
</html>
```
Embedded CSS
(cont)

We’ve actually introduced a few new things here:

- We have a style tag, used for defining a style
  - Since we’re doing this separately from the individual tags, but still within the document, it’s an embedded style
  - Our previous example (that was a mess) was an example of an inline style — it explicitly defined the styling for individual elements

- Note that this is typically placed within the head tag because we’re really describing aspects of the document’s formatting, rather than its actual content

- We’ve also introduced the idea of selectors
  - e.g. table matches up with with the table html tag
  - If we had three such tables within the document, they would all have been styled by that single entry
    - This makes it substantially easier to keep the visual style consistent within the document
External Stylesheets

Suppose we were to move that style information into a separate document (say, mystyle.css):

```
  table {border: 1px solid black;}
th {border: 1px solid black;}
td {border: 1px solid black;}
```

In our document, we’d have:

```
<head>
  <link rel=”stylesheet” type=”text/css” href=”mystyle.css” />
</head>
```

- `rel` indicates the relation of the link
- `type` indicates the document format
- `href` says where to find the style sheet
Why Are External Style Sheets a Thing?

Actually, a few reasons:

- They’re easier to edit separately
- They’re easier to drop-in a replacement
  - e.g. For sites that have a desktop version and a mobile version
  - e.g. For those sites that let you swap out the style
- They can be separately cached by your browser
- **They let you reuse them across multiple documents!**
Then, Why Are Embedded Style Sheets a Thing?

- Some actually advise against embedded sheets
  - Then again, some advise against external sheets!
- Embedded sheets can be good for a set of styles that really only applies to a single page (and is expected to stay that way)
- Unless the styling gets very long and elaborate, an embedded sheet might make a document easier to read overall
Okay, But What About Inline Styles?

Inline styles are often frowned upon

- They should only be used when you need to style individual elements uniquely, with no reuse at all
  - e.g. if you want to make one table cell stand out in a whole document, etc.
Basic Selectors

The simplest selectors are simply the HTML tag name.

We can, however, combine selectors into groups. For example, did you notice that the table example used the same style for `table`, `th`, and `td`?

```
table, th, td { border: 1px solid black; }
```

We can also use a special selector `*` to apply a style to any element (and by extension, all elements) within the document.
Contextual Selectors

We can also use selectors that acknowledge the document’s hierarchy (parent, child, sibling, descendent, etc.)

- $e_1 e_2$ – Selects all $e_2$s inside $e_1$s
- $e_1 > e_2$ – Selects all $e_2$s that have $e_1$ as an immediate parent
- $e_1 + e_2$ – Selects all $e_2$s that are immediately after $e_1$
- $e_1 ~ e_2$ – Selects all $e_2$s that are preceded by a $e_1$

All of these can help with things like, “make the links (a) inside a table green”, “make the first paragraph after a heading a different colour”, etc.

We’ll have a few more selectors eventually, but first let’s look at some more basic styling.
The Box Model

Elements in HTML follow a simple model: padding, border, margin.

- The padding is the space left between the contents of an element and its border
  - For a table cell, it’s the space between the text and the border of the cell
- The border is just that: a border
  - Borders also have additional attributes. e.g. style
- The margin is the space reserved outside the element
  - For example, suppose you wanted all images to stay separate from the surrounding text. You could set a large margin, and it would require a gap around all sides of the images.

- These spacings may be set for all four sides, or individually

See more information here:

- http://www.w3.org/TR/CSS2/box.html
- http://www.w3schools.com/cssref/default.asp
Basic Formatting

- color
  - http://www.w3schools.com/cssref/css_colornames.asp
- opacity
- background-color
- text-decoration
  - overline, line-through, or underline
- text-align
  - left, right, center, justify, initial, or inherit
- text-justify
  - e.g. inter-word
Fonts

The font properties of an element may be set in one rule, or piecemeal

- **font-size** – May be set directly, or relative to parent
- **font-style** – e.g. *normal*, *italic*, or *oblique*
- **font-weight** – e.g. *normal*, *bold*, *bolder*, *lighter*, or a number
- **font-variant** – e.g. *normal*, *small-caps*
- **font-family** – *More Specific, Less Specific, Even Less Specific*
  - e.g. font-family: "Times New Roman", Times, Serif;
  - More generalized options are only used if first choices aren’t available
- **font** – Lets you set multiple properties together
  - e.g. font: italic 2em/3em Times, Serif;
  - Note: You probably actually don't want to use *em*

Neat Reading:

- [http://www.w3schools.com/css/css_font.asp](http://www.w3schools.com/css/css_font.asp)
- [http://www.w3schools.com/cssref/css3_pr_font-face_rule.asp](http://www.w3schools.com/cssref/css3_pr_font-face_rule.asp)
Fancier Colours

We’ve already seen the list of predefined (named) colours we can use, but what if we want to define our own?

- \( \text{rgb}(r,g,b) \) can be used
- \( \text{rgba}(r,g,b,a) \) adds transparency as well
- \( \text{hsl} \) and \( \text{hsla} \) are even options!

Note that not all browsers will support (for example) \( \text{hsla} \).

- An easy fallback is to include properties that will definitely work, and those that \emph{might} work
  - But, in what sequence? Does it even matter?

Note: \textit{Gradients} are also allowed, but support and implementation are erratic, so read here only if you’re interested: http://www.w3schools.com/css/css3_gradients.asp
Sizing

Elements may have their sizes explicitly set

- You can set width, height, or both
- This could cause issues with overflow
- Sizes may be set as absolute values, or percentages
  - Both can be good or bad, depending on the situation
  - It’s often good to use a mix
  - The most flexible content should be the percentage (good for elastic layouts)
  - This is one of the uses of \texttt{em}
Floating elements

All elements have a `float` property. The default is `none`.

- Floating an element breaks it from the normal text flow by attaching it to the left or right side of the enclosing element
  - e.g. Float an image to the right to let text wrap around/past it on its left
  - Note that you can also use `clear` to force an element to come after the floated element
Positioning

Every element has a position. The default is *static*.

- *static* – Normal rendering; elements appear in the document when they appear in the source
- *absolute* – Element is positioned relative the first non-static ancestor element
- *fixed* – Element is positioned relative browser window
- *relative* – Element is positioned relative where it would normally appear

Changing the position doesn’t really do much until you also set at least one of top, bottom, left, or right.

- By default, they’re set to *auto*, but you can specify either a distance or a percentage.
### Z-index

Suppose you’ve started fiddling around with positioning. That increases the chances that one element will overlap another, right? That may be what you intended, but what if the element that’s on top is the one you really want in the back? (For example, if you want to place an image behind some text)

- The `z-index` property lets you assign a layer priority
- Basically, think of it as being like a *depth* (elements with a higher `z-index` are on top of those with a lower `z-index`)
More Advanced Selectors

Remember `span` and `div`? We said they were generic tags, with no semantic meaning, and that we’d be adding significance later.

- It’s now officially ‘later’

Though individual `span`s could be formatted with inline styles, this is a perfect time to bring up the idea of a `class`.

- A class is a label that you define
- It may be used to define your own semantic meaning
- It may be applied to any element you wish
- You can set up CSS rules for your class
  - It simply follows the pattern of `.class`
  - You can combine with the other tricks. e.g. `p .class`

E.g.

```
.neat background-color: green;
...
<span class="neat">This is neat</span>
```
Attribute Selectors

Similar to classes, you can create a rule unique to a specific element (identified by its id)

- An id should be unique in a document
- Set the CSS rule via: `#id property: value;`
- So, why not just use an inline style then?
  - Even for single-use styles, it’s better to keep all style information together, so you don’t have to manually find it within the document

You can also select elements based on whether or not they’ve set a particular attribute (e.g. `href`), or even the precise value of that attribute.

- e.g. `img[src$='.gif'] {border: 1px solid green;}` would put a green border around all `.gif` images
CSS also lets you set properties based on temporary conditions or hierarchical positioning. These are not true classes, but are included for convenience. For example:

- Being able to modify the first character of a paragraph
- Specifying the formatting of an element while the mouse is hovering over it
- etc.

It's outside the scope of this course of cover these, but you can find more info here: http://www.w3schools.com/cssref/css_selectors.asp
Other Border Styles

We’ve already looked at how to set the thickness of a border, and how to set it to *solid*, but obviously there’d be no point in specifying solid unless there were other options, right?

- A detailed explanation is here:  
  http://www.w3schools.com/css/css_border.asp
- Samples are here:  
  http://www.w3schools.com/css/tryit.asp?filename=trycss_border-style

But we can also even change the shape of the border!

- Fair warning, there’s typically no actual *need* to do this. If you have a background image with a shape, the vast majority of people won’t realize that isn’t the element’s real shape (and even fewer will care)
Rounded Corners

Suppose you want to have some text in a nice-looking box, but not so... boxy. Rounded corners might do the trick!
Example here:
http://www.w3schools.com/css/tryit.asp?filename=trycss3_border-radius

However, we don’t need to set all corners (non-corners?) the same!

```
.single { border: 1em solid indigo;
background-color: rgb(200,150,255);
padding: 10px 20px; width: 300px;
border-top-left-radius: 25px; }
<div class="single">Note that only the upper-left corner is rounded!</div>
```

```
.spiffy { border: 1em solid indigo; background-color: rgb(200,150,255);
padding: 10px 20px; width: 300px;
border-top-left-radius: 50px;
border-top-right-radius: 15px;
border-bottom-left-radius: 15px;
border-bottom-right-radius: 35px;
}
```
One Last Border Example...

```css
.nifty {
border: 1em solid;
border-top-color: red;
border-right-color: green;
border-bottom-color: blue;
border-left-color: yellow;
background: radial-gradient(rgb(225,180,255),rgb(150,70,255));
padding: 10px 20px; width: 300px;
border-top-left-radius: 50px;
border-top-right-radius: 15px;
border-bottom-left-radius: 15px;
border-bottom-right-radius: 35px;
}

Note that gradients for borders are far trickier. A bit of reading here if you’re interested:
http://css-tricks.com/examples/GradientBorder/
```
Final Thought on Borders...

Additional border decorations are a bit outside the scope of this course, but you can also find:

- *Border images here:*
  http://www.w3schools.com/cssref/css3_pr_border-image.asp

- A general guide to borders here:
  http://www.w3schools.com/css/css3_borders.asp
The display Property

The two most well-known display properties are:

- **block**
  - Like div, p, etc.

and

- **inline**
  - Like span, b, etc.

These default behaviours can be changed. But, also, there are additional options available (e.g. *inline-block*). Anything other than *inline*, *block*, and *none* is outside the scope of this course, though. (We’ll get to *none* in a bit…)
Fun With Backgrounds

Beyond just colours (and gradients), we can also change other aspects of backgrounds, including:

- Background image
- Repetition (in both axes) of such images
- And a couple others

It might be worth looking at:
http://www.w3schools.com/css/css_background.asp and
http://www.w3schools.com/css/css3_backgrounds.asp
Visibility

The visibility of any element can be set via the `visibility` property (`visible`, `hidden`, or `collapse`). Two obvious questions:

- Why?
- How is this different from `display: none`?
Commenting

/* stuff */
- Yup. I’d say that covers it
Additional Note: !important

When setting any property,

- `property: value !important;`

prevents later selectors/style rules from replacing this rule. It is especially vital for those with visual impairments.
Take another look at the Chrome developer tools (*Inspect Element*). You’ll see that the right side of the view is for dealing with CSS, and is actually both powerful and quite helpful.
Additional Link

In addition to the previous links:
http://www.w3schools.com/css/css_howto.asp