JOHNS HOPKINS
UNIVERSITY

Zanvyl Krieger School of Arts and Sciences
Advanced Academic Programs

410.712.81
Advanced Practical Computer Concepts For Bioinformatics

Topic: Cascading Style Sheets (CSS3)

Instructor: Joshua Orvis

Overview

Cascading Style Sheets (CSS) are the primary way to apply visual styling on your pages. It isn't a programming language like Perl or Javascript, but rather a syntax that allows you to select elements on your page and list out style rules and attributes for them.

Learning CSS at first is an exercise in memorization, but then transitions into an artistic and elegant way to transform your pages with simple rules. First you learn the different *selectors* that allow you specify specific elements, then memorize the possible *attributes* and values that make up the *rules* applied to those selected elements.

CSS styling covers simple things like font selection and more advanced topics such as page layout (creating columns, etc.)

In this week's lecture we'll cover a bit of background and history of CSS, introduce the methods for applying style to your page, and then look into a few more advanced topics with CSS using examples.

History

Cascading Style Sheets (CSS) have actually been around for longer than most people think. Developed together by Håkon Wium Lie and Bert Bos, CSS were initially proposed in 1994 and finally recommended by the W3C in late 1996 (CSS1).

Only the especially brave used it until 1999 though, when the first browser finally implemented (nearly) all of the features given in the CSS1 specification. Naturally, by that time the CSS2 specification had already been defined by the W3C.

CSS3 has been under development since 1998, and in late 2010 the W3C CSS Working Group published the stable specification.

The "cascading" part of CSS refers to the fact that any given element can obtain its style rules from multiple sources. The specification provides a series of inheritance and weighting rules so the browser can choose the right rule. You can, for example, specify rules for the same element in multiple places and a series of precedence checks define which rule should apply.

Why use CSS?

The much older method of controlling site appearance and layout involved lots of embedded images and extensive use of table elements. This was both difficult to maintain but also involved creating a lot more code than necessary (aside from violating the content separation principles.)

Take, as an example, the (now) old use of <font> tags. They would be littered throughout a document, anywhere there was text visible to the user, specifying the same font to be used throughout a document. Using CSS, these can all be removed and in a few lines you can select all elements in the document and define the default font family, size, etc. This is both a dramatic savings in file size but also makes it much easier to update your site. Using the <font> tag method, if you wanted to change the font on your site you'd have to change the values of all of these elements on all pages. With CSS, you chould change a single line of code and your entire site will be updated.

One of the first major companies to completely refactor their site using CSS that also published information on the result of this transition serves as a great practical example of the benefits of CSS.

ESPN example

The ESPN.com CSS redesign is a great example of a large commercial site recognizing the benefits of CSS. By re-coding their pages to adhere to more strict markup rules and shifting the burden of styling to CSS they trimmed the average size of each page by 50 kilobytes. That may not sound like much, but it means less manual coding and code generation has to happen on the server side and less data needs to be pushed to the client's browser. This means pages load faster and ESPN.com saves a lot of bandwidth. When you serve as many pages as they do, this means saving millions per year.

As described in an interview with Mike Davidson, ESPN.com serves over a billion pages per month. By cutting their average page size they saved an estimated 730 terabytes of bandwidth per year! Of course, file size reduction was not the only reason ESPN.com performed the migration. The newer, modular design simplified future development and enabled display on new portable devices such as phones, PDAs and other portable viewers.

The interview:

http://www.mikeindustries.com/blog/archive/2003/06/espn-interview

Where does it go?

There are two primary ways you can add stylesheet information to your document - either directly within your HTML or contained in an external file. We'll first give examples of both, so you know where to put your CSS rules, then we'll learn the syntax of the rules themselves.

Embedding CSS within your document - An easy method for adding CSS information within your HTML document is to add a <style> element within the header and type it right there. The general form is like this:

```
<style type="text/css">
    CSS rules here
</style>
```

If you know that your style information is going to be minimal and/or it doesn't apply to multiple pages this is an acceptable way to define your style information. If, on the other hand, you are going to have a multi-page website and the pages will share common style rules, this probably isn't the best way to go.

By embedding style information directly in your page you are likely to copy the same rules onto each page. Then, if you want to change the way something looks across your entire site, you'll have to go and change each of those embedded stylesheets. Linking to an external stylesheet would be a better solution.

Where does it go?

Linking to an external stylesheet - Still within the page's <head> element, you can put a pointer to a CSS file rather than embedding it directly in the document. This way many different HTML files can just reference a particular stylesheet, and any changes made to that stylesheet will be reflected immediately upon any HTML pages that reference it. The general syntax for referencing an external stylesheet is:

```
<link rel='stylesheet' type='text/css' href='/path/to/some_file.css'>
```

Styling an element directly - I said there were two primary ways for adding style to your document, which is true, but there is one other way that is discouraged. You can actually embed the style for a given element directly as that element's attribute, like this:

```
<p style=”font-weight: bold;”>This text will be bold</p>
```

While you should be aware this is possible, there is little justification for actually ever doing it. Instead, focus on describing your content correctly using HTML markup and then styling it with separately using CSS; don't style elements directly in this manner.

Anatomy of a CSS definition

CSS isn't a programming language like those that have loops, conditionals, and repetition structures. The syntax is relatively simple, and learning it is mostly a matter of memorizing attributes, values and learning strategies for using them to modify your page.

generic structure

```
selector {
    attribute: value;
    attribute: value;
}
```

example

```
p {
    margin-bottom: 10px;
    font-weight: bold;
}
```

Your CSS will be a series of definitions like those above. Each *selector* is used to choose which elements you want to apply one or more *rules* to. Each rule is made up of an *attribute* and *value*.

Selectors – Markup matters

Learning how to use selectors to apply styles to the specific elements on your page is critical. It also shows the importance of doing descriptive markup within your HTML. If you never use classes and ids, for example, it will be difficult to select the precise components to which you want to apply styles.

On the left, the markup is very basic. If we want to style the first paragraph it's not as straight-forward since nothing differentiates it from the others. On the right, the author has added classes to the appropriate elements to describe them better. Now we can learn to do things like “increase the font size of the intro paragraph, if any, within Op Ed articles.”

```
<article>
    <h2>PB: crunchy or smooth?</h2>
    <p>
        blah blah blah
    </p>
    <p>
        more blah blah blah
    </p>
    <p>
        a little more blah
    </p>
</article>
```

```
<article class='op_ed'>
    <h2>PB: crunchy or smooth?</h2>
    <p class='intro'>
        blah blah blah
    </p>
    <p>
        more blah blah blah
    </p>
    <p class='acknowledgements'>
        a little more blah
    </p>
</article>
```

CSS comments

If you want to annotate your CSS with comments (you should) the style might be familiar to those who have done other programming languages, such as C.

```
/*
    You can do multi-line comments
    like this
*/

/* single line comments, like this */

p {
    margin-bottom: 10px;  /* comments don't have to start a line */
    font-weight: bold;
}
```

Selectors – Survey of types


```
/* selects ALL elements on a page */
* {
    … rules here …
}

/* all paragraphs on the page */
p {
    … rules here …
}


/* paragraphs of class 'intro' */
p.intro {
    … rules here …
}


/* paragraph with id 'directions' */
p#directions {
    … rules here …
}

/* all spans within a paragraph */
p span {
    … rules here …
}
```

```
/* p directly descendent of ul */
ul > p {
    … rules here …
}

/* p after a ul (siblings) */
ul + p {
    … rules here …
}

/* any anchor with a title attr */
a[title] {
    … rules here …
}

/* a with specific attr value */
a[href=”foo”] {
    … rules here …
}

/* a with 'foo' anywhere within a
specific attribute */
a[href*=”foo”] {

}
```

Selectors – Survey (pseudoclasses)


```
/* any unvisited links */
a:link {
    … rules here …
}

/* any visited links */
a:visited {
    … rules here …
}

/* when an element is hovered */
p:hover {
    … rules here …
}

/* The first child of an element,
regardless of its type */
section:first-child {
    … rules here …
}

/* The last child of an element
*/
section:last-child {
    … rules here …
}
```

```
/* A specific numbered child of
an element (numbered from 1) */
section:nth-child(4) {
    … rules here …
}

/* A specific child, numbered
from the end */
article:nth-last-child(3) {
    … rules here …
}
```

There are even more selectors than this, but if you learn these you'll be able to do almost anything you need to.

See the last slide for further reading about selectors types.

Multiple selectors

You can also use the same block of rules for multiple selectors at the same time simply by separating the selectors with commas. Here is an example of the same set of rules applied to four different selectors:

```
p.intro,
article#news,
div ul span,
span.species {
    font-size: 80%
    font-family: verdana;
}
```

Doing this saves you from repeating the same rules for each selector and is a common practice.

Example: Styling a table

```
<!doctype html>
<html>
<body>
    <table>
        <thead>
            <tr>
                <td>gene</td>
                <td>fmin</td>
                <td>fmax</td>
                <td>strand</td>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td class='gene'>recA</td>
                <td>1145</td>
                <td>2087</td>
                <td>1</td>
            </tr>
            <tr>
                <td class='gene'>tonB</td>
                <td>7771</td>
                <td>8500</td>
                <td>1</td>
            </tr>
            <tr>
                <td class='gene'>mutR</td>
                <td>43314</td>
                <td>45119</td>
                <td>-1</td>
            </tr>
        </tbody>
    </table>
</body>
</html>
```

Here is a simple HTML table with relatively minimal markup. Note the use of thead and tbody elements to separate the header row from the rest of the table, and the addtion of a class to the first column of each row.

By default, this renders like this in a browser:

gene	fmin	fmax	strand
recA	1145	2087	1
tonB	7771	8500	1
mutR	43314	45119	-1

All the cells are pretty packed together, and nothing differentiates header rows from the data rows.

Let's learn some CSS attributes we can use to improve this.

Example: Styling a table


```
<!doctype html>
<html>
<head>

<style type=”text/css”>
    table {
        width: 400px;
        border-collapse: collapse;
    }
    td {
        border: 1px solid rgb(150,150,150);
        padding: 3px;
    }
    td.gene {
        color: rgb(0,0,150);
        font-style: italic;
    }
    thead td {
        font-weight: bold;
    }
</style>

</head>
<body>
     <!--
   See previous slide for body content
     -->
</body>
</html>
```

I've now added a style element to the head and populated with a few CSS definitions. This transforms our table to look like this:

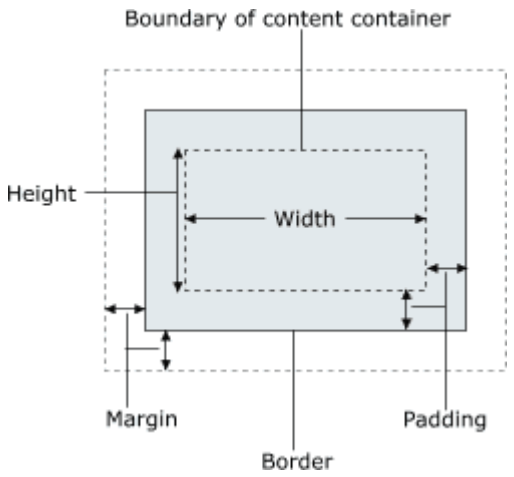
gene	fmin	fmax	strand
recA	1145	2087	1
tonB	7771	8500	1
mutR	43314	45119	-1

We've added 4 selectors, and here are the new attributes to learn:

width: Most often has values like 100px or 40%
border: Defines the width, style and color of a border at once.
border-collapse: Merges borders for adjacent cells in the table.
padding: Provides spacing between the border of the table cells and the content.
color: defines the color of the text
font-style: values 'normal', 'italic' and 'oblique'
font-weight: values like 'normal', 'bold', 'bolder', 'lighter'

Basic CSS box model

It's best to think of elements in HTML as little blocks (even in-line elements). The content area has a height and width and is then surrounded by padding, then a border, then margin outside of that border. All of this can be customized with CSS. It's very important though that you first learn this, known as the 'box model'.



It should be noted that Internet Explorer got this wrong for a very long time (up to and including v7 in quirks mode), including padding and borders in the content height calculation, requiring developers to apply hacks and work-arounds for different browsers.

On the previous slide we added padding to the td elements to space them apart. Now, looking at this box model and picturing any table as a grid of td elements, you should be able to see that padding is the space between each cell's content and their borders. This makes the table seem less compact.

You will often use the padding and margin attributes to do things like control spacing between paragraphs, pad text away from images, etc.

More CSS properties

Now that you know how to use selectors to choose elements on page for styling, the next thing you need to do is learn what CSS properties/attributes there are so you can use them within your rule definitions. I'll list and discuss some common ones here, and the best way to play with them is to create a simple page with a few HTML elements and add these in the CSS to see how it changes your page. For a full reference and examples, see the W3C site here:

http://www.w3schools.com/cssref/default.asp

background-color – Set the color of the background using a name or RGB value
background-image – Provide the URL for an image to use as an element's background
border – Using this you can set the size, location, color, etc. of a border all at once.
border-top, border-bottom, border-left, border-right: Set the border on each side individually.
height and width: Define the dimensions of the content area of an element.

font – Sets all properties of the font at once; this is most commonly applied to the body
font-size – Change just the size of the font.
list-style – Set all properties of a list. Set to 'none' to turn off the default list styling.
text-align – Sets the alignment of text horizontally.

These are just some of the more commonly-used properties we'll cover a more topic-specific ones next but be sure to spend some time on the link above and read over all those that are available. You may not remember them all immediately, but by reading over them you might remember what general categories are available to you.

Page layouts

Doing page layouts in CSS used to be a delicate art involving a property called 'float', which removed an element from the normal flow of the document and sent it to the left or right side, causing other elements to wrap around it. By exploiting this and the 'clear' property, one could do things like create multi-column layouts relatively quickly, though there were still problems and the learning curve was steep.

If you went through this rite of passage and doggedly refuse to do layouts another way you are certainly free to continue. CSS3 provides a host of other methods now that make this a lot easier than it was before.

For some of these, you'll have to remember that you're using the latest versions of the specifications available, and that invariably means doing some custom code to make up for browsers that don't full support it yet (usually Internet Explorer.) I'll try to make note of these as you go along, but you should always look up a new property on a site like w3schools.com and see the current browser support.

Yet another note is to always try your code from within one of the html5 templates such as the HTML5 Biolerplate we used earlier in this class. They will often have included libraries that correct for browser deficiencies.

Newspaper-style columns

One area where web developers previously struggled was in generating multi-column layouts where the content would flow from the bottom of one column into the next seamlessly, like you see in newspaper pages. Normally, sequential paragraphs would take up the full width of the screen and you'd have to manually partition the lines into columns, depending on the height you wanted to use, all of which needed to be calculated. Now, with just a few lines of CSS, that same set of paragraphs can easily look like this:

Text, images, and anything that fits within the columns flows nicely from one column to the next.

This is very configurable, and uses the CSS3 Multi-Column Layout Module. Unfortunately, support for this hasn't converged among browsers yet, so it takes a few more lines of code than it should (see next slide.)

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam nec nisi quis turpis aliquam cursus. Pellentesque suscipit ullamcorper felis nec rutrum. Mauris consectetur consequat ipsum, nec blandit orci molestie vitae. Aliquam augue augue, viverra in pellentesque et, venenatis non sem. Curabitur dictum ante vitae ipsum consectetur sit amet malesuada ante hendrerit. Sed et ornare massa. Vivamus tellus turpis, pretium eleifend interdum quis, convallis a diam. Duis turpis metus, molestie sed accumsan sit amet, porttitor ac arcu. Donec urna risus, egestas id lobortis sagittis, convallis non magna. Aenean id mauris arcu, vel dignissim risus.

Nulla cursus consectetur sem, nec pellentesque nisl convallis in. Vivamus vehicula malesuada ligula, sed tempus orci rhoncus non. Ut id nisl justo, in scelerisque mauris. Curabitur feugiat pharetra lacus non malesuada. Donec dapibus ligula a lacus fringilla vel venenatis augue tristique. Donec egestas ligula sagittis risus congue dapibus. In nec arcu sit amet sem tempor sollicitudin. Maecenas adipiscing congue nisi eget bibendum. Sed vitae est ipsum. Phasellus non massa quis nisl pretium ultricies eu non augue. Suspendisse potenti. Pellentesque posuere risus imperdiet eros porttitor at consequat risus auctor. Quisque tempor molestie sapien eget rhoncus. Donec rhoncus pretium augue, ac porttitor diam pharetra id.

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos.

Pellentesque in velit risus. Sed fringilla volutpat tincidunt. Suspendisse tristique commodo velit, vel feugiat turpis tristique eu. Curabitur eget diam non felis molestie suscipit. Proin faucibus sapien nec turpis vestibulum quis rutrum libero facilisis. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Quisque varius sagittis risus, sit amet aliquam nisi ullamcorper vitae. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean tortor tellus, faucibus in auctor id, fermentum ut leo.

In hac habitasse platea dictumst. Duis orci arcu, lacinia vitae placerat eu, euismod vel purus. Aenean sed felis quis tortor ullamcorper imperdiet. Donec at sem ipsum. In luctus sapien nec velit ultrices vehicula. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Aliquam erat volutpat. Nunc sit amet auctor nisl. Nullam scelerisque, arcu sit amet sagittis malesuada, erat sapien posuere nibh, ut feugiat nibh quam sit amet dolor. Duis elementum erat eu dui volutpat convallis. Aenean pharetra, nunc quis varius euismod, erat nisi adipiscing leo, eleifend suscipit tellus justo eget turpis. Proin imperdiet purus non libero scelerisque eget auctor lectus ornare. Nam interdum, odio et lacinia

porttitor, elit mi aliquet diam, vitae mollis turpis eros eget eros. Etiam tincidunt lobortis aliquet. Etiam quis fermentum enim.

Nullam lobortis purus eu lectus dapibus eu tincidunt quam volutpat. Proin sapien augue, placerat eu suscipit vel, sagittis eu libero. Pellentesque nec sapien ac sapien rutrum rutrum. Sed feugiat tincidunt nisi, eget rutrum diam accumsan a. Suspendisse posuere laoreet elit, vitae scelerisque mauris consectetur quis. Maecenas vestibulum tempor tellus, ac dignissim lorem sodales pharetra. Donec enim ipsum, iaculis ac blandit nec, pretium a est. Vestibulum consequat, felis non pulvinar cursus, leo metus feugiat magna, eget lobortis mi metus non orci. Pellentesque luctus luctus sem. Donec vel tellus ipsum. Proin et consequat erat. Maecenas iaculis faucibus erat. Nunc sem leo, commodo vestibulum egestas in, pretium ut nulla.

Curabitur bibendum rutrum elementum. Cras lobortis metus et turpis varius tempor. In interdum iaculis libero in sodales. Vivamus urna orci, bibendum volutpat volutpat vel, mattis et nunc. Ut vestibulum, augue in viverra adipiscing, nisi ipsum porttitor nisl, mollis ultricies libero augue vitae orci. Curabitur tincidunt tempor libero at vestibulum. Morbi nunc massa, tincidunt sed faucibus eu, venenatis in eros. Quisque at velit quis lectus

CSS3 Multi-Column Layout Module

A demo for the example on the previous slide is available on my personal server here:

http://sivro.net/AS410.712.81.FA11/demos/multi_column_module.html

The demo contains a single <article> with an id of “coldemo” that's loaded with <p> elements. Ideally (and eventually) all you should have to do is the code below in bold. For now you'll need to add the -webkit and -moz versions due to different browser implementations. It won't work at all until IE 10.

For more notes and options, see this demo:

http://webdesignernotebook.com/css/remembering-the-css3-multi-column-layout-module/

```
#coldemo {
    -webkit-column-width: 290px;
    -webkit-column-gap: 20px;
    -webkit-column-rule: 1px solid rgb(150,150,150);

    -moz-column-width: 290px;
    -moz-column-gap: 20px;
    -moz-column-rule: 1px solid rgb(150,150,150);

    column-width: 290px;
    column-gap: 20px;
    column-rule: 1px solid rgb(150,150,150);
}
```

Other layout models

While all the intricacies of page layout and design is outside of the scope of this class, you should know that CSS3 provides several different methods for controlling the layouts on your page. If you want to learn more, I highly recommend the article below. It has short descriptions and links to the various methods available.

http://www.netmagazine.com/features/future-css-layouts

Further reading

"30 CSS selectors you must memorize"

http://net.tutsplus.com/tutorials/html-css-techniques/the-30-css-selectors-you-must-memorize/

Printable CSS and HTML "cheat sheets"

http://devcheatsheet.com/tag/css3/

"Quick hits with the flexible box model"

http://www.html5rocks.com/en/tutorials/flexbox/quick/

CSS3 property reference

http://www.w3schools.com/cssref/default.asp