Classnotes: 19 September 2016

More CSS … with a focus on colors, text, the “box” model, div and span, and pseudo-classes and pseudo-elements.

Colors

Important things to bear in mind:

- We have three ways of identifying a color in HTML:
  - `rgb( red, green, blue )`, where each color takes on a value 0 through 255 (decimal)
  - Hexadecimal notation: #XXXXXX, where each pair of XX (digits, 0 through the letter “f”) corresponds to the decimal equivalent, 0 through 255, and by name—but only 147 actual names are used. People generally avoid using names.
- Doing the simple math, we have about 16 million possible colors, but actually only about 216 are these are “color safe,” meaning that they are the same between all browser implementations.
  - Why is this so? Because your hardware most likely is only capable of rendering 256 colors, and 40 colors are “reserved” by various operating systems. For a handy explanation and helpful ideas about these colors, navigate to http://searchsoa.techtarget.com/definition/216-color-browser-safe-palette.
- The “raw color” is only part of the story. Colors have components, “hue,” “saturation,” and “brightness.” Hue is basically the color; saturation is the amount of gray in the color, and brightness is the amount of black in the color.
- CSS3 adds some new operators that provide more flexible ways of specifying colors.
  - `rgba( red, blue, green, alpha )`, adds “opacity/transparency” or alpha values-a number between 0.0 and 1.0, which expresses the precent of transparency or opacity for this color (see the class slides on Color).
  - `hsl( hue, saturation, lightness )`, where “hue” is based upon the color’s position on a continuous disk of color whose top is red. Proceeding clockwise, we go red, orange, yellow, green, blue, indigo, violet (roughly). This means that the value specified for the hue is a number that reflects the color’s position on a circle, 0 through 360. Saturation is expressed as a percent, 0 to 100, and lightness is a percentage.

- `hsla( hue, saturation, lightness, alpha )`, where the first three parameters have the same interpretation as for the "hsl" operator, above, and the alpha values can be decimal numbers between 0.0 and 1.0. 0 is white and 1.0 is black—this refers to the color's opacity/transparency when combined with other colors.
- In addition to hue, saturation, lightness and opacity, a chosen color is related to any other color by the "contrast" that may result when one is placed within or alongside the other.
  - Contrast is particular relevant when colored text is displayed against any background (see the class slides).

Additional Design Considerations (Color)

The addition of these new CSS3 color properties opens up new possibilities, and should likely be used in place of the older properties that they replace:

`rgba()` should be used in place of `rgb()`.
`hsla()` should be used in place of `hsl()`.

Problems might arise when you assume that every browser will support these "newer" ways of expressing color. Wise designers anticipate that clients may NOT have the latest and greatest browser available to them, so they design their rules with "failsafe" provisions. For example:

```
/* Inform the browser to use a red color with about 78% lightness.
 * alternatively, provide a "fall back" or a "failsafe" rule that
 * will be used if the browser does NOT support the "hsl" operator.
 */
body {
   background-color: #c8c8c8; /* 78% gray */
   background-color: hsl( 0, 0%, 78% );
}
```

Make sure that you understand what this rule says and why it works!
(Note: this rule is fine, but when we define colors for fonts (below), we need to also define background-colors to satisfy the requirements for CSS3 validation—more on this later).

For those who wish to become experts, use the tutorial found at http://www.w3schools.com/colors/default.asp

Notes on Text

Originally browsers were almost entirely text-based, meaning that most data was textual.[1] Text is comprised of letters, and letters appear on computer screens as characters within a particular "font." (The font that you are reading at this moment is Helvetica 11pt.)

- The systematic study of "fonts" (text and how it's represented) and the associated design properties entails the study of typography, graphics art and, in extreme applications, psychology and linguistics.

We can manipulate (control)

- The appearance of text (the actual structure of the fonts used and how they are painted). This includes such properties as typeface, bold, italics, and point-size. And,
- Attributes that are independent of a particular font, such as spacing between characters, lines of text, the color of the text, etc.
- We also control the area around the text by specifying the background color(s) of elements as well as the "space" between fonts.

Some additional design considerations relating to text ...

Fonts with distinguished serifs, such as Times Roman, are used in long tracts of text because they are believed to be easier to read.

Fonts without serifs, such as the font you're reading here, is often used in smaller text sizes and when text is read on a computer screen.

Fonts with "fixed width" (equal spacing between the occurrence of each character) is often used in formatting programs—for example, the fonts used to format programming examples in my class notes is Courier New. This also preferred by editors because it's easier to have program fragments "line up" correctly with fixed-width fonts.

For example: design a CSS class named `ProgramText` that uses a fixed-width font, uses single-spacing between lines of text, and perhaps provides other kinds of formatting information that is relevant to the presentation of computer programs—maybe the background-color and the font color are chosen to enhance readability, etc.

1 Pictures and other kinds of data were not rendered within the browser but instead were "downloaded" by a link and rendered by the user.

Some practical, nuts-and-bolts issues

When designing CSS rules for typefaces, provide the client with several font-faces, from most specialized to the least specialized in the event that the most specialized font is NOT available on their browser. For example:

```
/* CSS rule to set up some fonts */

body {
   font-family: Georgia, Times, serif;
}

h1, h2, h3 {
   font-family: Arial, Verdana, sans-serif;
}

.credits {
   font-family: “Courier New”, Courier, monospace;
}
```

Notice what’s happening here (and notice the explicit use of double-quotes). Compare this style of CSS programming with the “failsafe” rule(s) that we provided in the section above, when we discussed Colors.

Understanding font “sizes”

Traditional unit was “points,” 1/72 of an inch. On most screens resolution is 72pixels per inch.

Defaults fonts sizes in browsers is 16px; roughly 75% of this is 12points. So,

```
body {
  font-family: Georgia, Times, serif;
  font-size: 75%; /* or better 12px; */
}
```

Why is 12 pixels better that 75% of 16 pixels? Only because it’s more precise.Be careful with font-sizes if you use percents: what happens if we specify a “default” font-size for the whole document as a percentage and then specify the font-size for one of its contained elements as a percent? What is 75% of 75% of 16 pixels?

Storing fonts requires space

Browsers do not generally store (provide natively) lots of fonts. Be careful using fancy fonts because the browser may have to “download” these which will slow-down the rendering of your pages.

A browser is NOT a Kindle or a Nook … ask how these differ.

Typesetting is expensive and difficult in a browser

Browsers are best-suited to producing relatively short, general purpose documents that briefly describe or discuss a topic.

Programs, such as Word or Pages, are text “processing” programs. They are designed for producing relatively well-designed documents for moderately long, complex pieces of text.

Typesetting programs, on the other hand, are more specialized and require a steeper learning curve; they are designed for producing high-quality, long complex documents, such as books, medical, legal compendia, mathematical and scientific texts, etc.

Many websites offer downloadable files usually in PDF format for long, complex pieces of text. These text files were prepared by professional typesetting programs (and professional authors and professional text designers); keep this in mind when designing and writing web-content.

Elevating the Discussion …

Words are comprised of fonts and are recognized with the assistance of syntax: spaces, punctuation marks, etc. Words compose sentences, sentences paragraphs, and so on. CSS provides a rich variety of properties that address "text" at this level of description. You should examine a few.[2]

Besides fonts and their "micro" properties, we have "macro properties:" width, height, color, space between "lines" of text, space between "words" of text, how text is "broken" and "filled"within various elements, such as paragraphs, the direction that text is read, and several others.

This should lead you to think that terms such as "contrast" are "relative," meaning that each of these elements, colors, fonts, etc., bear a relationship one to the other that is not completely known at "design time." For example: we do not know which browser will view the site, nor do we know anything about the "environment" in which that browser will be found: is it in bright daylight or a dark room etc. Will the client experience the web-page(s) in order, or in some other order, perhaps based upon the results of a search engine, etc.

Sometimes it's fine to define a few CSS rules that are collected into one or two documents which are then "shared" (via the <link> mechanism) across all of the pages that are contained within a web-site. For very large, complex presentations, however, HTML provides additional elements and CSS provides an additional features that, if used in a responsible manner (i.e., well-documented and consistent) greatly simplifies the CSS code and makes it easier for others to understand (and perhaps modify) our design.

We begin by looking at an important CSS feature that takes advantage of the containment relations that HTML creates, and then talk about some new and helpful HTML elements.

2 In your tutorials, the page http://www.w3schools.com/cssref/default.asp#text presents an extensive list of the properties provided by CSS that control both the local and global experience of text.

Inheritance within CSS

If you have spent any time examining the web-pages contained within the CSS tutorial, you may have noticed that along with the descriptions and examples of the term in question, you are told whether or not a particular property is “inherited.”
Recall that HTML associates data, such as text, with semantic elements, such as paragraphs, headings, etc. In addition to these associations, however, HTML also implicitly describes a “parent-child” relationship between elements, and “containment” is the nature of that relationship.

> The term “inheritance” informally means that the value attached to a particular property that appears in the “parent” is implicitly found in the “child.”

In most (if not all) modern programming languages, “inheritance” is used to allow objects to “share” or “reuse” code, based upon the assumption that things are more similar than they are different, and so we need only provide the code that describes where they differ. But, an example is worth many words, so consider:

```
/* These rules take advantage of “inheritance”
 * by virtue of the containment relations between
 * these elements. Note the use of the values
 * “inherit”.
*/

p { color: red; }

h1 {color: green; }
ul {color: yellow; }

a {color: inherit; }
```

The “effect” of these rules is that

- Hyperlinks that appear within paragraphs will be red
- Hyperlinks that appear with h1 elements will be green; and,
- Hyperlinks that appear with unordered-lists will be yellow.

Think about how much “typing” this has saved us. Now, think like a computer scientist and ask: what are the “trade-offs?” Nothing is “free.” (Hint: what happens when the results of an action are far-removed from the cause of that action?)

When HTML classes and elements are not enough

Pseudo-Classes

Consider the problem of specifying how "hyperlinks" should behave on your webpage. You would like to have the HTML anchor element change based upon some "context," such as whether the link is presently under my mouse, or that the link has been visited, etc. CSS and HTML cooperate by providing you with "pseudo-classes," which is not classes in the "traditional" sense of the word … Your slides give an excellent example of changing how anchor elements appear based upon their "state," which includes `:hover`, `:visited`, etc. Note, these states may never apply to this particular element … hence, they differ from the common understanding of classes.

Pseudo-Elements

Consider now a different problem: We'd like to have the first line of text in any paragraph use a bold face font. (I didn't say that these examples were "desirable.") Well, CSS provides a pseudo-element, :first-line, that acts as a modifier for just this purpose.

```
p:first_line { font-weight: bold; }
```

You can, and should, look under the tutorial on pseudo-classes and pseudo-elements, found at

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

for a thorough explanation with examples of each of these—note, you'll likely never use many of these, but it's good to know, I suppose.

And, now we introduce two HTML elements that solve many problems … some that you didn't realize you had.

Carving up the HTML document space with div and span

Consider the following task:

> Create a "space" on a web-page, possibly intermingled with other elements/spaces, that is reserved for "business cards." This space needs to announce itself by formatting changes, but is not restricted as to exactly what it may contain. If it does contain particular kinds (classes) of text, however, these should be identifiable as different.

We could define a bunch of CSS rules for the class “businessCard,” and also a set of rules for things that likely belong to a Business card, such as Contact Information, Resume, etc. But, where does the Business Card reside, where does it begin and end? What are its HTML elements?

> The `<div>` element reserves an anonymous “block” of space in the document that may contain any other kind of HTML elements, including other `<div>` elements. It’s a place to attach class or id information. Similarly, the `<span>` element reserves an anonymous linear space that may contain any HTML elements; it, too, is usually a container labeled with class or id selectors that specify CSS rules.

Let’s use these to spec-out the problem:

```
<div class=”businessCard”>
   <span class=”contactInfo”><p> blah, blah, blah</p> </span>
   <div class=”resume”>
     <!-- resume goes here -->
   </div>
<!-- possibly other HTML elements go here …-->
</div>
```

Now, you should be able to visualize some CSS rules that use these classes to do the right thing.

Some Hands-On Suggestions for Designing with CSS

Keep in mind that we have two languages here:

- HTML which gives us the semantics (by elements) as well as the “inheritance” relations between elements.
- CSS which gives us a way of expression “rules” which select elements from the HTML document and apply “styles” to those elements.

These languages must be made to work together.

Komodo can help

Komodo provides some useful observation tools through the View menu. From View, choose View Minimap; a small overview of your HTML buffer should appear somewhere (depending upon your preferences) in your Editor.

Next, from the View Menu, pull down the various options under Fold: I suggest trying to Collapse All and then study what happens in the editor's code window and in the Minimap display.

Now, you should choose "parent" markers along the left hand side of the editor buffer (or within the Minimap's view of the buffer) and "drill down" by opening that section. Observe what happens when you do this.

What's happening here?

> You're navigating the structure of the page without attending to the atomic details. This is the beginning of "design."

Pay attention to the relative "flatness" of the document as it has been given to you. Every element seems to live one or maybe two levels "down" from the `<body>`. This might be fine, unless, of course, you wish to "distinguish" these elements in some way.

Ask: how would I add some structure here? Think about the discussion of `<div>` and `<span>` earlier this week.

Ask: do I need to treat the elements that appear at the same nesting level (siblings) the same, or should I somehow treat one or some differently from the others.

These kinds of patterns of mind commonly occur in many areas of human knowledge. In computer science, we use the term "factoring" for the kind of mental process we are describing here and in class. We find the set of common properties and use the structuring tools (classes, elements, etc.) to group elements to facilitate this "sharing" from a common "parent."

Naturally, we will practice factoring in greater detail upon our return to class on Monday.