Classnotes: 12 September

Cascading Style Sheets

HTML elements specify the “semantics” of the data items that appear in a browser.

- Data items can be text, pictures, sounds, video recordings, and other kinds of objects (which we will discuss later).
- Some “elements” do change how items appear in the browser. For example: headings, paragraphs, tables, iframes, code, pre, a, address, and emphasis elements change the appearance of textual items, but these changes are somewhat dependent upon the particular Browser that is executing/interpreting the HTML stream.
- What an entity is is not how it necessarily how it appears.

CSS was designed to provide designers with a consistent, extensible way to control the “syntax” of these data items, i.e., “how” they appear in a browser.

- CSS associates “rules” with HTML elements. (Draw the diagram):

  <rule> ::= <selector> { [<property>: <value>;],* }

  <selector> ::= <HTML element> | <user-defined-class> | <user-defined-id> |

  <compound-selector> | *

  <HTML element> :: = <p>, <h1>, … etc.

  <user-defined-class> ::= .<class-name>

  <user-defined-id> ::= #<id-name>

  <property> ::= CSS property name

  <value> ::= CSS value

CSS allows designers to control properties of fonts, colors, placement, display properties of tables, boxes, iframes, div, span. In short: CSS rules may apply to every HTML element plus user-defined “classes,” “ids,” and complex combinations of these!

- “Classes” look like: <html-element class=”<className>” ...>
- “Ids” look like: <html-element-name id=”<idName>” … >

12 September 2016 Classnotes—Introducing CSS …

- <html-element-name> .<className> { … }
- #idName { … }, html-element-name > html-element-name, { … }, etc.

CSS relies upon "inheritance" --uses containment (and other) relations to make it easier to control a great number of elements with a few "rules."

Review how HTML elements form a containment hierarchy. (Draw the following as a containment diagram, and/or as a graph.)

```
<html> <!--  Top level container (super set) -->
   <head> <!-- sibling of HTML-->
       <meta charset="utf-8"> <!-- sibling of <head> -->
       <title>Title text</title> <!-- sibling of <head> -->
  </head>
  <body>
    <h1>...</h1>
    <p>...</p> <!-- siblings -->
    <ol>
      <li> … <li> <!-- label these yourself -->
      <li> ...<a href="...">text</a></li>
   </ol>
</body>
</html>
```

We must distinguish children, from descendant and adjacent siblings.

```
td  a { … } <!-- any <a> children of a td element -->
td > a { … } <!-- any <a> elements inside of a td element. -->
h1+p <!-- applies to the first <p> element after any h1 element.-->
```

Discuss the difference between "descendant" and "child":

- Descendant: `td a { … }` any `<a>` downstream of a `<td>` element.
- Child `td > a { … }` only `<a>` elements within one level downstream of a `<td>` element.

12 September 2016 Classnotes—Introducing CSS …

Using CSS definitions within an HTML document.

Although we prefer to have you separate CSS from HTML by creating two documents, situations may arise where creating CSS within the current HTML file is easier: (1) You do not have to switch buffers to see the names of classes, ids, etc., and (2) You do not have to worry about creating and managing additional files (again: having separate files is often preferred, however).

In place of the `<link>` tag that appears in this week's example (which you should still have in your Komodo buffers), you can do something like:

```
<head>
   <style type="text/css">
      * { font-family "Courier New" }
      …. etc …
   </style>
…
</head>
```

Note, the syntax for comments differ for HTML and CSS so you need to be careful here. Generally, this is a fine thing to do when you just beginning your design work. As you progress and approach a final product, you can then remove the text that appears inside of the `<style>` element, create a new CSS file, store that text in that file, re-format comments to CSS-style, and finally replace the `<style>` element with a `<link>`--just as things appear in so many examples that you will see in this class.

Other items for review or discussion

Project 1 is due Sunday evening … with that in mind, here's some discussion of lingering questions/concerns.

On "Nested Tables"

Students should use the HTML Tutorial dedicated to nested tables, which is found at this link. (This link is the same that was provided in the relevant Elms Announcement., which also provides some additional details. In summary: you have three categories, Plants, Animals, and Microbes. Conceptually, we associate each of these with a row consisting of the information that corresponds to the headings, namely Name and Description (notice that Category is already taken-up so the rows of our embedded tables contain only two data-items (cells)).

How to Submit Your Projects

- Make sure to name your files exactly as they are specified in the Instructions for the project. This is particularly true for the early projects.
  - For Project 1, your uploaded file should be named as `Last-First-P1.zip`, for example, `Reinhardt-Tom-p1.zip`.
  - For Project 1, we expect three files:
    - `index.html`
    - `contacts.html`
    - `UMDGlobe.gif` (which was given to you).

Why this is important?

We will attempt to automate validation, but this requires that every submission satisfy the requirements that provide in the Projects' Instruction Sheets. If your file names, etc., don't conform to these specifications, then the workflow breaks down. Rather than hold up the grading process, we will just omit those points from your grade, leaving you two options: either accept the grade, or

1. Create a document that shows the result of validation with evidence that these results were generated by your code; do this by setting the “show source” option on the Validation page, and then take a snapshot of the results;
2. Submit this document to the grading Teaching Assistant.

Recurring questions about Zip files

Zip is a program that “compresses” the contents of your files (this includes directories, and files contained within these, and so on) and places them in a single “file” whose suffix is “zip.” This means that you cannot do much with the contents of a “zip” file until you reverse the process that was used to create the zip file in the first place, i.e., you must “decompress,” or “inflate,” or “restore” the contents of the zip file to their original states. This is called unzipping, extracting, or inflating, etc.

In order to “test” the contents of a zip file, we will uncompress it, which results in a temporary location being allocated on a machine and the contents restored, in their original order, into that location. Once there, we can treat these files as real files and specify them as URLs to browsers, validators, etc.

12 September 2016 Classnotes—Introducing CSS …

Friday's task

1. Open the Instructions for week3-inclass-work (found in the Module for week 3).
2. Construct a “bare-bones” table that contains the kinds of information that goes into a resume. Because you don't have lots of work experience, etc., you should create an “ideal” resume, i.e., what would YOUR resume look like upon graduation.
3. Validate your document.
4. Chances are that your present document is hard to read—maybe the experience doesn't easily line-up with the dates, or even the topic (left-column). You may wish to use “boxes” to separate text blocks that belong to particular dates and categories, for example, use a box to set-off educational achievements for a particular span of time. You might also consider the effect of colors (blocking) and fonts, italics versus bold, etc. All of these kinds of considerations should drive your CSS code.
5. Open a new file in Komodo, call it myCSS.css. Using the CSS code from Wednesday's in-class example, you can now start modeling your rules.
6. Insert in the <head> of your basic resume file a “link” to this CSS file. Again: you can do this by closely copying the example provided by Wednesday's in-class example.
7. Test, revise, test.

Next week …

- More CSS constructions and additional HTML elements that work hand-in-hand with these CSS rules.
    - Colors, fonts, iframes, div and span, additional treatment of meta tags.
    - Pseudo-classes and pseudo-elements (CSS).
- The mechanics of CSS Validation.
- A new Discussion where we will explore and discuss some additional journal articles that are relavent to your research paper.