CSCE 120: Learning To Code

Module 6: Presenting Data II

Introduction

This module introduces you to working with Cascading Style Sheets and defining stylistic elements in a web page.

Cascading Style Sheets

One of the main purposes of using HTML is to separate *content* from *presentation*. That is, HTML includes the actual content (text) along with markup (tags) that define the semantics of the content. However, it does not (or at least *should* not) specify how that content is ultimately presented to the user. Stylistic element such as whether or not paragraphs are indented or separated by some vertical space, or if some word is displayed italicized or in bold font, or the size, color, and typeface of fonts in general are all defined using Cascading Style Sheets instead. There are several ways to define CSS *rules* for the styling of HTML elements.

Inline Styling

Any HTML element can have the `style` attribute. Using this attribute, you can specify as many style elements as you want, each of which is separated by a semicolon. To specify a style element, you need to specify a key-value pair called a CSS *property*. For example, we could add a `style` attribute to a paragraph element:

```
<p style="color: red; font-weight: bold; font-family: calibri">Hello world!</p>
```

In this example, we've specified three elements of the paragraph: its font color (set to red), its "weight" (bold), and its typeface (Calibri).

Inline styles are applied to the HTML element in which the `style` attribute is placed *as well as* all of its child elements. This is known as *inheritance*; which defines a hierarchy of styles. The browser itself has some predefined style defaults. Inline styles override those defaults while inline styles specified in child elements *override* those settings. An example:

```
<div style="color: red">
  <p>This is a red paragraph, inherited style</p>
  <p style="color: green">Now its green</p>
  <p style="color: yellow">And now yellow</p>
  <p>Now back to red</p>
</div>
```

There are hundreds of different CSS properties that are part of the official CSS specification and many more hundreds of non-standard properties supported by particular browsers and rendering engines. A comprehensive list would span multiple CSS specification documents. However, one good resource listing many of these properties as well as valid values can be found here: http://meiert.com/en/indices/css-properties/.

The `<span>` element

Sometimes you may want to specify styling of a particular piece of text that doesn't necessarily have a separate HTML element. For example, perhaps you want to highlight a particular word in a paragraph. To do this you may use the `<span>` HTML tag which does not have any particular semantic meaning other than enabling you to define an inline CSS style that is to be applied to (or *span*) the content inside the `<span>` tag. An example:

```
<p><span style="font-weight: bold">Congratulations!</span> You have won!</p>
```

In this example, only "Congratulations!" would be rendered in bold-font with the remaining sentence rendered in the style inherited from the paragraph's parent.

Global Styling

Using inline styling somewhat defeats the purpose of using CSS: it embeds styling elements inside the document and makes maintenance or changes difficult. Instead, CSS styling rules can be placed in an external style sheet and included in your HTML document by using the `<link>` tag. In the `<head>` of your document you would include a line similar to the following.

```
<link href="myStyles.css" rel="stylesheet" type="text/css">
```

Within your style file (`myStyles.css`), you would use the following syntax:

```css
/*
  myStyles.css
  Comments in CSS files are similar to JavaScript
  however no single line comments using // are not allowed
*/

body {
  font-size: 80%;
}

/* this applies the style to all p (paragraph) elements: */
p {
  color: red;
  text-align: center;
}

h1 {
  font-family: Helvetica, Verdana, sans-serif;
}
```

If you want multiple elements to have the same styling you can specify multiple *selectors* delimited by a comma to have the same style applied to all of them. Moreover, you can repeat selectors with different styles:

```css
h1, h2 {
  font-family: Helvetica;
  font-weight: bold;
}

h1 {
  color: red;
}

h2 {
  margin-left: 2em;
}
```

In this example, `h1` *and* `h2` elements would be rendered as bold Helvetica. However, `h1` elements would be red and `h2` tags would be indented by 2em (an em is a unit in typography equal to 16-point).

Common Style Elements

There are several hundred CSS properties with dozens of possible values each. New properties have been added with each new version of CSS (CSS3 is the current version

with various modules and recommendations currently under draft and consideration for CSS4). We will only focus on a few of the most common ones here. For a complete list of properties and documentation, the following is a good resource: http://meiert.com/en/indices/css-properties/.

Fonts

There are several CSS properties that control how fonts are displayed. Each property has several possible values, but we'll only focus on a small number of them.

`font-style`

This property can be used to italicize a font using the value `italic`. By default, the value is `normal`.

`font-weight`

Various values can be used to provide a bolder (or lighter) font using the values `normal` (default), `bold` `bolder`, `lighter`, etc.

`font-size`

Font size can be controlled using a variety of values that can be set with a variety of units. Values can be expressed using a series keywords: `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, `xx-large`. Alternatively, a font size can be expressed by using pixels `px`, points `pt`, or ems `em` (a unit used in typography; 1em = 16pt). A numerical value is associated with each unit with no space between them. For example:

```
1 font-size: 10pt;
2 font-size: 20px;
3 font-size: 2em;
```

You can also use a percentage (`%` which is relative to the default font size. For example, `font-size: 50%` would be one half the size of the default font size, `font-size: 200%` would be twice as big.

Layouts

Each HTML element has surrounding margins, borders, and padding which, for most elements, do not effect how they are displayed. You can, however, change them using the

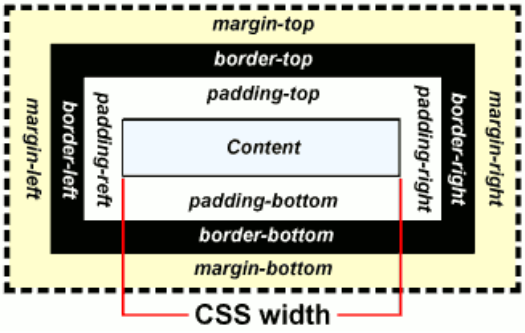

Figure 1: Relation between margins, borders, and padding in HTML elements

`margin`, `border`, and `padding` properties. Moreover, you can fine-tune each of these by specifying different values for the top, right, bottom, and left margin/border/padding.

Figure 1 depicts the relationship between margins, borders, and padding in the layout of an HTML element.

For margin and padding, you can use `margin-top`, `margin-right`, `margin-bottom`, `margin-left` and `padding-top`, `padding-right`, `padding-bottom`, `padding-left` to fine-tune properties of each of the four sides. Values can be specified using similar units to fonts, including pixels, points, ems, and percentage.

For convenience, all four sides can be specified with one rule using `margin` and `padding` and providing 1, 2, 3, or 4 separate values delimited with a space. An example:

```
/* all four sides are 10px */
margin: 10px;

/* top and bottom are 10px, both sides are 20px */
margin: 10px 20px;

/* top is 10px, sides are 20px, bottom is 30px */
margin: 10px 20px 30px;

/* top, right, bottom left */
margin: 10px 20px 30px 40px;
```

Borders have different properties including width, style, and color that can each be specified using `border-width`, `border-style`, and `border-color`. Width can be specified using pixels or other similar units. There are several styles to choose from including `none` (default), `dotted`, `dashed`, and `solid`. Color can be specified in several different ways that we explore in the next section. Finally, all three properties can be set by the `border` property.

```
border-width: 3px;
border-style: solid;
border-color: black;

/* equivalently: */
border: 3px solid black;
```

The layout relationship between different elements can be defined using various properties including `width`, `height`, `float` and `clear`. Elements can either float `left`, `right` or `none` (default). A floating element allows the surrounding content to "flow" around it. Often, you don't want elements *after* a float to flow around it, in which case you can use `clear` with values `left`, `right` or `both` to prevent content flowing around the respective sides.

Colors

Each element has a foreground color (usually the text color) and a background color which can be controlled using the `color` and `background-color` respectively.

There are many *named* colors such as `red` `green`, `black`, etc. (see http://www.w3schools.com/cssref/css_colorsfull.asp). More fine-tuned colors can be defined using the RGB (Red-Green-Blue) color model. In this model, you specify an additive amount of each color using an integer value from 0 (no contribution) through 255 (full contribution). These three values are defined using `rgb(255, 255, 255)`. Since it is additive, full contribution of all colors corresponds to white. Black therefore would be `rgb(0, 0, 0)`; `rgb(255, 0, 0)` would be red; `rgb(255, 255, 0)` would be full red, full green and no blue, giving yellow.

Alternatively, these three numbers are often expressed in hexadecimal (a base-16 number system that uses 0, 1, 2, ..., 8, 9, a, b, c, d, e, f). Two hexadecimals are used for each value with a hash symbol. Some examples:

```
/* all of the following are equivalent */
color: red;
color: rgb(255, 0, 0);
color: #ff0000;

color: yellow;
color: rgb(255, 255, 0);
color: #ffff00;

/* sort of a light green */
color: rgb(168, 247, 178);
color: #a8f7b2;
```

Elements can also be made transparent using the `opacity` property and a value from

0.0 (completely transparent) and 1.0 (completely opaque). For example, `opacity: 0.5;` would be 50% visible Any elements in the background will "bleed through". Opacity can also be included with a color using RGBA ("alpha channel") and specifying the opacity as a fourth component. For example, `color: rgba(255, 0, 0, 0.5);` corresponds to red, but 50

Visibility

Another way that you can control the visibility of elements is to use the `visibility` and `display` properties. With visibility, you can make an element either `visible` (default) or `hidden`. With `display`, you can use `inline`, `block` or `none`. An inline element does not start on a new line and takes only as much width as necessary (like a `<span>` element). A block element starts a new line and takes up the full width of parent element (like a `<div>` element).

Both `visibility: hidden` and `display: none` hide an element. The difference, however, is that using `display: none` does not affect the layout of the surrounding elements. The layout is rendered as if the element does not exist. Using `visibility: hidden`, however hides the element, but takes up as much "empty" space as the element would if it were visible.

Classes, Identifiers, Selectors, etc.

Classes

Another attribute that can be applied to any HTML element is the `class` attribute. This attribute is commonly used in conjunction with CSS to apply styles to elements that *belong* to a certain class rather than to all elements of a certain type (tag). When defining an element, you can give it a class using the following syntax:

```
<p class="greeting">Greetings!</p>
```

You can apply more than one class to an element by separating class names with a space:

```
<p class="greeting strong">Hello!</p>
```

When defining a style for classes in an external style sheet, you use a period:

```css
.greeting {
  color: green;
}

.strong {
  font-weight: bold;
}
```

You can further specify a styling to apply only to certain types of elements which belong to a certain class:

```css
div.greeting {
  margin-left: 2em;
  background-color: LightGray;
}
```

This CSS rule would apply to all `div` elements that had a `greeting` class added to it. For example, the rule would apply to `<div class="greeting">` but would *not* apply to a normal `<div>` element.

Identifiers

Another common attribute that can be given to HTML elements is the identifier attribute, `id`. The ID *should* be unique to each HTML element. If more than one HTML elements have the same ID, odd and unexpected behavior can result. An example:

```html
<p id="leadParagraph">Greetings...</p>
```

When applying a style to an element with a particular ID, you use a hash:

```css
#leadParagraph {
  font-size: 120%;
}
```

Combinators

Combinators give a way to combine selectors to form more complex rules.

- The *universal selector* (`*`) will apply a rule to *every* element in the document.
- The *descendent selector* allows you to combine two or more elements and applies the rule to nested elements. For example, `div p` applies the rule to any paragraph that is a descendent of a `<div>` element
- The *child selector* applies a rule to the *immediate child(ren)* of an element. For example, `div > p` applies the rule to any paragraph that is an *immediate child* of a `<div>` element but not to any descendants deeper in the tree.

- The *adjacent sibling selector* (`div + p`) and *general sibling selector* (`div ~ p`) apply rules to the sibling *immediately following* and to *all siblings*. The examples apply the rule to paragraphs that are immediately following and to all paragraphs that are siblings of a `<div>` element respectively.

Attribute Selectors

Rules can be created to apply to elements with certain attributes. For example, you may want a rule to apply to all links (anchors) that link to a PDF file or all images with an `alt` (alternative) attribute.

- To apply a rule to any element that simply *has* an attribute, use the following syntax: `tag[attr]`. For example, `a[href]` applies to all anchors that have *any* `href` attribute
- To apply a rule to an element that has a particular attribute with a *particular value*, you can use the following syntax: `tag[attr="value"]`. For example, `a[href="syllabus.pdf"]` would apply to any anchor that linked to the file `syllabus.pdf`
- You can also apply a rule to an element whose attribute *begins* with a particular string value. For example, `a[href^="http"]` would apply to all anchors that link to an http address.
- Likewise, you can match attribute values that end with a certain string using `a[href$="jpg"]` which applies to all anchors that link to a JPEG image file.
- Finally, you can match attribute values that contain a certain string value not just at the start/end, but anywhere within the attribute's value using `a[href*="cse"]`

Tools, Frameworks, Libraries

You can do very complex styling rules by combining elements, classes, and IDs as well as other selectors (see Advanced CSS Selectors), "combinators", and other special syntax to achieve a myriad of behavior. This can get complicated rather quickly. For that reason, many frameworks and tools have been created to allow developers to organize style sheets. For example, LESS (http://lesscss.org/) and Sass (Syntactically Awesome Stylesheets, http://sass-lang.com/) are CSS pre-processors that allows you to write CSS in a more programming language-like syntax (with variables and functions) which is then converted to plain CSS.

Moreover, many frameworks have been created that have defined a lot of nice stylistic elements that you can use. For example, Twitter's Bootstrap (http://getbootstrap.com/), or HTML5 Boilerplate (http://html5boilerplate.com/) can be used so that you

don't have to start from scratch. Many other CSS templates and resources can be found in the wild as well, allowing you to start with a nearly complete page template. Finally, many *Content Management Systems* (CMS) such as Drupal (https://www.drupal.org/) or WordPress (http://wordpress.com/) allow users to edit entire webpages in a user interface without having to directly modify the CSS or HTML.