Week_14

Quick Review: CSS

Styles

CSS stands for Cascading Style Sheets, and is the visual language of the web. With it, we can design how HTML pages look like. CSS allows us to separate between the data of the page, encoded within the HTML, and the styling of the page, encoded in the CSS of the page.

The reason that CSS is "Cascading" is because of the way it defines which styles to use. HTML elements inherit their styles in a "cascading" manner, according to a very specific set of priorities, from lowest to highest:

- The browser default
- Styles defined in the page
- Some styles will inherit a style to their childs, for example which font to use
- The last rule to be defined in the loading order will be the one that will kick into effect
- The element selector (for example, styling all `<footer>` elements)
- The class selector (for example, styling all `.main` elements)

- The ID selector, used to select a specific element in the page
- Media type (explained in the Media Queries section)
- Defining element specific CSS using an HTML "style" attribute
- The `!important` CSS override directive (Use this directive only as a last resort)
  In addition, there are relative "cascading" properties that can stack up relative to their parent, for example line height and font size, for example the `rem` (relative em) CSS metric which is defined relative to the parent, or the percentage unit.

Defining CSS

CSS can be defined using four methods:

1. Inline

To define a CSS style using the inline method, use the HTML `style` attribute:

```
<p>This is the default serif font. It is commonly used in
printed media for better readability, since letters
are more distinct in serif fonts.</p>

<p style="font-family: sans-serif">This is a sans-serif font.
It is commonly used in screens because it is hard
for screens to render letters with such great detail.</p>
```

2. Using a CSS tag

Defining CSS inline is not recommended at all, avoid it as much as you can. You should always define a CSS stylesheet and use selectors to apply the stylesheets. The Selectors tutorial will give a detailed tutorial on how to select HTML elements using CSS selectors.

Here is an example:

```
<!DOCTYPE html>
<head>
    <style>
        .nice {
            font-family: sans-serif;
        }
    </style>
</head>
<body>
    <p>This is the default serif font. It is commonly used in
    printed media for better readability, since letters
    are more distinct in serif fonts.</p>

    <p class="nice">This is a sans-serif font.
    It is commonly used in screens because it is hard
    for screens to render letters with such great detail.</p>

</body>
```

3. **Using a different stylesheet**

You can define a CSS stylesheet in an external file (usually noted with the .css extension) and load it.

Here is an example for defining the same CSS class defined in the previous example, but in a **file called "style.css".**

```
.nice {
    font-family: sans-serif;
}
```

To load the CSS file, we would need to use the `<link>` HTML tag in the following manner:

```
<link rel="stylesheet" href="nice.css">
```

Please note that the stylesheet should be positioned properly relative to the HTML page. In this example, both the HTML file and the CSS file would need to be in the same directory.

The link tag should be positioned inside the `<head>` section of the HTML page, like this:

```
<!DOCTYPE html>
<head>
    <link rel="stylesheet" href="nice.css">
</head>
<body>
```

```
    <p>This is the default serif font. It is
commonly used in
    printed media for better readability,
since letters
    are more distinct in serif fonts.</p>
<p class="nice">This is a sans-serif font.
    It is commonly used in screens because it
is hard
    for screens to render letters with such
great detail.</p>
</body>
```

4. Programmatic access

HTML elements have the `style` attribute which you can use to add styles programmatically.

For example:

```
<!DOCTYPE html>
<body>
    <p id="serif-text">This is the default
serif font. It is commonly used in
    printed media for better readability,
since letters
    are more distinct in serif fonts.</p>
<p class="nice" id="sans-serif-text">This
is a sans-serif font.
```

```
    It is commonly used in screens because
it is hard
    for screens to render letters with
such great detail.</p>
<script>
        var sansSerifText =
document.getElementById("sans-serif-
text");
        sansSerifText.style.fontFamily =
"sans-serif";
    </script>
</body>
```

CSS Units

CSS has several different units for expressing a length.
Many CSS properties take "length" values, such as width, margin, padding, font-size, border-width, etc.
Length is a number followed by a length unit, such as 10px, 2em, etc.
A whitespace cannot appear between the number and the unit. However, if the value is 0, the unit can be omitted.
For some CSS properties, negative lengths are allowed.

allowed.
There are two types of length units: absolute and relative.

Absolute Lengths

The absolute length units are fixed and a length expressed in any of these will appear as exactly that size.
Absolute length units are not recommended for use on screen, because screen sizes vary so much. However, they can be used if the output medium is known, such as for print layout.

Unit	Description
cm	centimeters
mm	millimeters
in	inches (1in = 96px = 2.54cm)
px *	pixels (1px = 1/96th of 1in)
pt	points (1pt = 1/72 of 1in)
pc	picas (1pc = 12 pt)

* Pixels (px) are relative to the viewing device. For low-dpi devices, 1px is one device pixel (dot) of the display. For printers and high resolution screens 1px implies multiple device pixels.

Relative Lengths

Relative length units specify a length relative to

another length property. Relative length units scales better between different rendering mediums.

Unit	Description	
em	Relative to the font-size of the element (2em means 2 times the size of the current font)	
ex	Relative to the x-height of the current font (rarely used)	
ch	Relative to width of the "0" (zero)	
rem	Relative to font-size of the root element	
vw	Relative to 1% of the width of the viewport*	
vh	Relative to 1% of the height of the viewport*	
vmin	Relative to 1% of viewport's* smaller dimension	
vmax	Relative to 1% of viewport's* larger dimension	
%	Relative to the parent element	

Tip: The em and rem units are practical in creating perfectly scalable layout!
* Viewport = the browser window size. If the viewport is 50cm wide, 1vw = 0.5cm.

Browser Support

The numbers in the table specify the first browser version that fully sup

ports the length unit.

Length Unit			
em, ex, %, px, cm, mm, in, pt, pc	1.0	3.0	1.0
ch	27.0	9.0	1.0
rem	4.0	9.0	3.6
vh, vw	20.0	9.0	19.0
vmin	20.0	9.0*	19.0
vmax	26.0	Not supported	19.0

Note: Internet Explorer 9 supports vmin with the non-standard name:

1.0	3.5
7.0	20.0
4.1	11.6
6.0	20.0
6.0	20.0
7.0	20.0

vm.