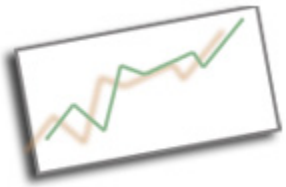

Advanced Online Media
Dr. Cindy Royal
Texas State University - San Marcos
School of Journalism and Mass Communication

HTML5
HTML5 is the most recent version of Hypertext Markup Language. It's evolution has been varied and meandering. Early versions of HTML presented a limited number of tags that allowed for development of basic Web pages. XHTML came along after HTML 4, and that added some much needed structure to the language. Now HTML5 provides new features that offer improved semantics, better multimedia support and improved interactivity and graphics. However, it will be some time before all browsers support HTML5. So for now, be aware of the impending features and integrate as applicable.

Semantics
HTML5 provides some great new structural tags that change the way you will use the generic in the future. The elements are called sectioning elements.

<header> - a container for a group of introductory or navigational aides.
<nav> - a container intended for major navigational elements.
<section> - a container to group together thematically related content.
<article> - much like <section>, this is a container for self-contained related content.
<aside> - a container for tangentially related content.
<footer> - a container that should hold information on the containing element.

Also, in html, tags used to be known as inline or block elements. Inline elements, or those that can be contained within another element without creating a new “block”, are now called text-level elements. Examples are <img> and <a>. Block elements forced a new block when they are closed. They are now referred to as grouping elements. Examples include <p>, all level headings (<h1> etc), <li>...

Tags are no longer considered deprecated, but many are now obsolete (font, frames, etc).

The new sectioning elements don't exactly correspond with the standard box model used in traditional websites: a header with a logo and intro material in it, a navigation sections, content areas and a footer. Instead, these elements support more complex page outlines for blogs, social media formats and RSS feeds.

The important thing is to attempt to use the elements properly based on the meaning of the content (with no regard to how it will look – that will be handled by CSS).

Traditionally, we may have used the following div model for a Web page:
<html>
<head>
...title and CSS links go in head of document
</head>
<body>

```
</div>
<div id=”nav”>
</div>
<div id=”section”>
</div>
<div id=”footer>
</div>
</body>
</html>
```

Of course, the stylesheet defines what the page looks like.

The ids for the divs could have been named anything we wanted, but I applied the names to correspond to the new structural tags. A sidebar could have also been added with

But HTML5 does not simply specify that you directly replace divs as above with the new tags. Each has a particular meaning as stated above. And, with divs and ids, each name could only be used once, to define the size and position of the box to which it corresponded. But in HTML5, each of the new sectioning elements can be used multiple times, with ids or classes to designate unique qualities.

```
<header>
</header>
<nav>
</nav>
<aside>
</aside>
<section>
<article>
</article>
</section>
<footer>
</footer>
```

You can do this, but it's not exactly the way the sectioning elements were meant to be used.

Some clarifications
You can use <header> for the top section of your page, but sections can also have headers. Plus, your “navigational aides” should also go in the <header>

```
<header>//include image or heading for top of page
<nav> //include nav bar items
</nav>
</header>
<section>
<header>
<h2>head of section, like the title of blog</h2>
</header>
<p>Content for the section</p>
```

<footer>closing material, author info, date, etc.
</footer>
</section>
<footer>
</footer>

May use classes or ids to style these appropriately.

You are seeking to develop an outline that can be interpreted by things like RSS feeds, so structure matters. Sometimes, you want items on the page to be eliminated from the outline. For this, you use <hgroup>. This only recognizes the 1st element in the tag and ignores the rest in regard to the page outline.

Rich Media
Audio
It can be as simple as this:
<audio src="music.mp3></audio>

which creates a player without having to use a plugin. There are attributes like autoplay and loop that can also be used (but don't). The attribute controls adds controls to the browser-generated player, and there is JavaScript that allows you to customize the player's controls.

The biggest catch with audio is the file formats supported by browsers.

Safari – mp3
Firefox - .ogg

.ogg is a Vorbus codec – open source. So, you need to include both versions.

<audio controls>
<source src="music.mp3" type="audio/mpeg" />
<source src="music.ogg" type="audio/ogg />
</audio>

And, browsers like IE still need Flash, so the embed code can be added. It's also a good idea to add the direct link to the audio file at the bottom of the stack, as a last resort.

Video
Video works much the same way, with the same type of attributes:

<video scr="movie.mp4"></video>
You can add autoplay, loop or controls.You can also add width and height to control the size of the player

You can also add a poster image to be in the player before the the video starts.
<video controls poster="placeholder.jpg" width="360" height="240">
<source src="movie.mp4" type="video/mpeg" />
<source src="movie.ogv" type="video/ogg />

</video>

And you would also add the Flash embed for IE and a direct link to the file, if you wanted to support the widest range of browsers.

The presence of the <video> tag allows you to customize the look with JavaScript and CSS.

You can use an application like Miro Video Converter to get open source, Theora .ogg or .ogv versions of files.

Canvas
<canvas id=”firstcanvas” width=”300” height=”200”></canvas>

Anything between those tags will be shown only to browsers that don't support canvas, so you can add a paragraph explaning that.

The rest is done with JavaScript. We'll do more with JavaScript soon, but this will give you a brief intro.

```
<script>
var canvas = document.getElementById(“firstcanvas);
var canvas = canvas.getContext(“2d”);
context.strokeStyle = #999999;
strokeRect(20,30,100,50);


</script>
```

There are different shapes and even text that can be rendered with canvas. You can see how Javascript could accept inputs from users or data and regenerate the shape on the fly (as in creating a chart or graph).

A few different examples.