HTML and CSS | HTML5 Semantic Elements

sessionscollege
FOR PROFESSIONAL DESIGN

Lecture 4

HTML5 Semantic Elements

In this course, we've talked a lot about the importance of separating page structure (HTML) from presentation (CSS). We've also touched on some of the ways that HTML5 has been evolving to streamline and improve Web coding practices.

In this lesson, we'll delve into HTML semantic elements and find out how this new(ish) approach to structuring pages is helping developers and browsers make better sense of Web pages.

Another kind of box: HTML semantic elements

We'll also discuss how to use HTML5 to implement audio and video content.

Learning Objectives

In this lecture, you can expect to:

- Learn how HTML uses semantic elements to structure page content.
- Learn guidelines for using semantic elements.
- Learn how to add audio and video content using HTML5.
- Learn principles for testing on multiple platforms and browsers.

Semantic Versus Non-Semantic Elements

Semantics is the study of the meaning of words. So what is semantic HTML? Simply, **semantic elements** are HTML elements with inherent meaning.

As we've discovered in this course, many HTML tags have very specific meanings. Examples of semantic elements that you've learned about include:

- `<img>`
- `<em>`
- `<h1>` - `<h6>`

Each one of those tags tells the browser something specific about the information on the page: this is an image, this is emphasized, this is a header.

Others tags have no meaning. Div IDs and classes, for example, don't have any semantic meaning on their own. If you wanted to, you could wrap div tags around any part of a Web page and call that part of the page `<div id="nav">`.

note

Semantic tags such as <h1> and <img> tell you something about the meaning of the content.

While it wouldn't make sense to label your footer "nav", you could do it! The name of a div or class is a name of your choosing. It doesn't look or behave differently until you apply CSS styles to its selector in your style sheet. And most importantly, the Web browser does not look at your `<div id="nav">` and interpret it as navigation.

Examples of non-semantic elements that you've learned about include:

note

HTML5 semantic elements describe the meaning of a part of a Web page.

-
- <class>
-

Along Comes HTML5

In HTML5, new semantic elements were created to better describe the different parts of a Web page. Those elements include, among others:

- <header>
- <nav>
- <section>
- <article>
- <aside>
- <footer>

These HTML5 semantic elements, which are supported in all modern browsers, serve the function of helping to streamline the way we describe the structure of a Web page. Previously, a Web designer might have used any of the following to describe a site navigation bar:

```
<div id="nav">
<div id=navigation">
<div class="nav">
<div id="menu">
```

Now, a designer can simply use the HTML5 tag:

```
<nav>
  <a href="#">Link 1</a>
  <a href="#">Link 2</a>
  <a href="#">Link 3</a>
  <a href="#">Link 4</a>
</nav>
```

note

The header tag can specify the header for a Web page or section of a page.

HTML5 semantic elements are clean, simple, and help make the Web a more functional and accessible place.

Using HTML5 Semantic Elements

So, how do you go about using these semantic elements on your Web site? The <footer> element should just replace the footer div, right? That's correct, but their use isn't always straightforward. There are special rules for how each of these elements should be used, when each element should be used, and where these elements should appear in the structure of your document.

Let's go through them, one by one.

The <header> Element

The <header> element contains important introductory information about your page. The <header> element can be used to specify a single header area for your entire Web page, or it can be used to describe the header of an individual

important!

Each header must contain at least one heading tag.

page section. Whereas you were only allowed to use one `<div id="header">` per document, the `<header>` element can be used multiple times in one document, as long as it's being used correctly!

Example #1:

```
<header>
    <h1>My Fabulous Web Site</h1>
    <h2>About Me</h2>
    <p>Some introductory text about me.</p>
  </header>
```

Example #2:

```
<header id="banner">
  <h1>Website Title</h1>

  <nav><ul>
    <li><a href="#">Home</a></li>
    <li><a href="#">About</a></li>
    <li><a href="#">Portfolio</a></li>
    <li><a href="#">Contact</a></li>
  </ul></nav>

</header>
```

As shown in the examples above, the header element can be used alone `<header>` or combined with an ID attribute `<header id="banner">`, which can be useful if you have more multiple `<header>` elements that require different styles. CSS styles for the `<header>` element might look something like this:

```
header {
        width: 960px;
        height: 100px;
        background-color: #eee;
}

header#banner {
        width: 960px;
        height: 100px;
        background-color: #eee;
}
```

note

The nav element is designed to include the primary navigation on a site.

In order for the `<header>` element to properly validate, each `<header>` must include at least one heading tag (`<h1>`-`<h6>`). `<header>` elements can also include images (like a site logo), paragraph text, and site navigation links `<nav>`. A `<header>` element cannot be placed inside a `<footer>` or another header tag.

Here's an example in the wild:

note

HTML5 semantic elements are helpful for search engines and screen readers.

The Lost World's Fairs site was created to showcase the beauty of HTML5. It uses the <header> element and other HTML5 tags.

```
<header id="header">
        <div id="mtn">
                <div class="container">
                        <div class="clearfix title">
                        <h1>Lost World's Fairs</h1>
                        <div id="button">
                                <span id="txt_nowwith">now with</span>
                                <span id="txt_woff">WOFF</span>
                        </div>
                        </div>
                        <div id="banner">
                                <div id="txt_beauty">Beauty of the Web</div>
                        </div>
                </div>
        </div>
</header>
```

The header element

The <nav> Element

The `<nav>` element contains the site navigation menu, or other navigation functions that will take you to a different page within the site. Only the primary site navigation links should go inside the `<nav>` element—not all anchor links on the site.

For example, it's not uncommon for a site footer to contain a list of navigation links. Because those are secondary or tertiary navigation links, the `<nav>` element isn't needed.

note

The section element groups related content on a page.

In HTML5 markup, the `<nav>` element can often be found inside the `<header>` element, though it isn't required there. The `<nav>` element can be used in place of div tags like `<div id="nav">` or `<div id="menu">`, as in the example below:

```
<nav>
<ul>
<li><a href="#">Home</a></li>
<li><a href="#">About</a></li>
<li><a href="#">Contact</a></li>
</ul>
</nav>
```

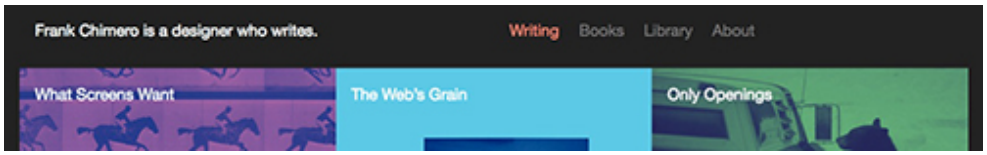
Because the `<nav>` tag has semantic meaning (where `<div id="nav">` does not), it communicates to the browser that this particular list of links is a primary navigation menu. This can be especially useful for SEO (Search Engine Optimization) and screen readers!

important!

The section element should not be used for styling purposes.

CSS styles applied to the <nav> element might look something like this:

```
nav {
display: block;
height: 100px;
background-color: #000;
}
```



Designer Frank Chimero uses the <nav> element inside of the <header> element for a clean, single line navigation bar.

```
<nav role="navigation">
    <ul>
        <li><a href="/writing/">Writing</a></li>
        <li><a href="/books/">Books</a></li>
        <li><a href="/library/">Library</a></li>
        <li><a href="/about/">About</a></li>
    </ul>
</nav>
```

The nav element

The <section> Element

The <section> element is used to group related sections of a document together. This one can be a little tricky to grasp, so I'll give a few different examples to illustrate how it's used.

Imagine that you're about to undertake a big Web design project. Before diving in, you'll probably want to outline your page content so you know where everything's going to. Your draft outline might look like this:

- My Portfolio
  - Animation Projects Section
    - Descriptions of animation projects with images
  - Audio Projects Section
    - Descriptions of audio projects with sound clips
  - Illustration Projects Section
    - Descriptions of illustration projects with images

note

The article element is designed to tag things like blog posts and news articles.

Resulting in the following HTML5 markup:

```
<h1>My Portfolio</h1>

<section>
<h2>Animation Projects Section</h2>
<p>Descriptions of animation projects with images.</p>
```

note

The article element should generally contain at least one heading.

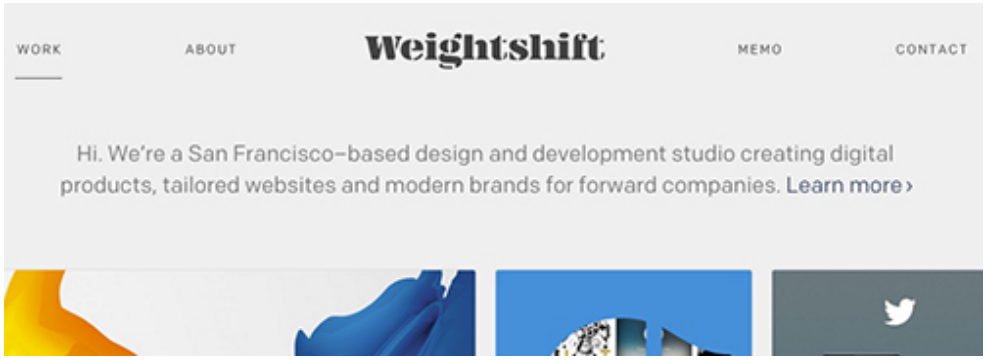
```
</section>

<section>
<h2>Audio Projects Section</h2>
<p>Descriptions of audio projects with sound clips.</p>
</section>

<section>
<h2>Illustration Projects Section</h2>
<p>Descriptions of illustration projects with images.</p>
</section>
```

As you can see, there's no limit to the number of times you can use the `<section>` tag per page. There are also no hard and fast rules about what defines a document section, so you'll have to use your best judgement when grouping thematic content together. A section can even contain a `<header>` or `<footer>` tag, as long as the content is all related.

One thing that's important to note is that the `<section>` element should not be used for styling purposes. Let section tags do their semantic work behind the scenes. Any sections of the page that need special styling should be styled using div tags. Remember, even if you can't see what your section tags are doing on the page, your Web browser can read and interpret their meaning.



Weightshift studio in San Francisco uses multiple <section> elements to divide their page content. There is a <section> tag wrapped around the intro message shown above.

```
115     <section id="intro">
116         <div class="container center">
117             <p>Hi. We're a San Francisco&ndash;based design and
development studio creating digital products, tailored websites and
modern brands for forward companies. <a href="/about/">Learn
more</a></p>
118         </div>
119 </section>
```

The section element

note

The aside element is designed for pullquotes and sidebars, like me!

The <article> element

The `<article>` element is for self-contained content that could be read independent of the Web site. Examples include things like blog posts, news articles, forum posts, and recipes.

It's easy to confuse `<article>` and `<section>` tags, in that both tags group related content together. It's especially confusing because article tags can contain multiple sections and section tags can contain multiple articles. So, when should you use which tag?

When trying to figure out whether an article or a section is more appropriate, ask yourself whether the content could easily be printed from the Web site as

a complete piece. If the answer is no, the `<section>` tag is probably more appropriate. If the answer is yes, the `<article>` tag might be a great choice!

Example #1:

```
<article>
  <h1>Rabbit Habitats</h1>
  <p>Rabbit habitats, or "rabbitats", include meadows, woods,
  forests, grasslands, deserts and wetlands. Rabbits live in
  groups, and the best known species, the European rabbit,
  lives in underground burrows, or rabbit holes.</p>
</article>
```

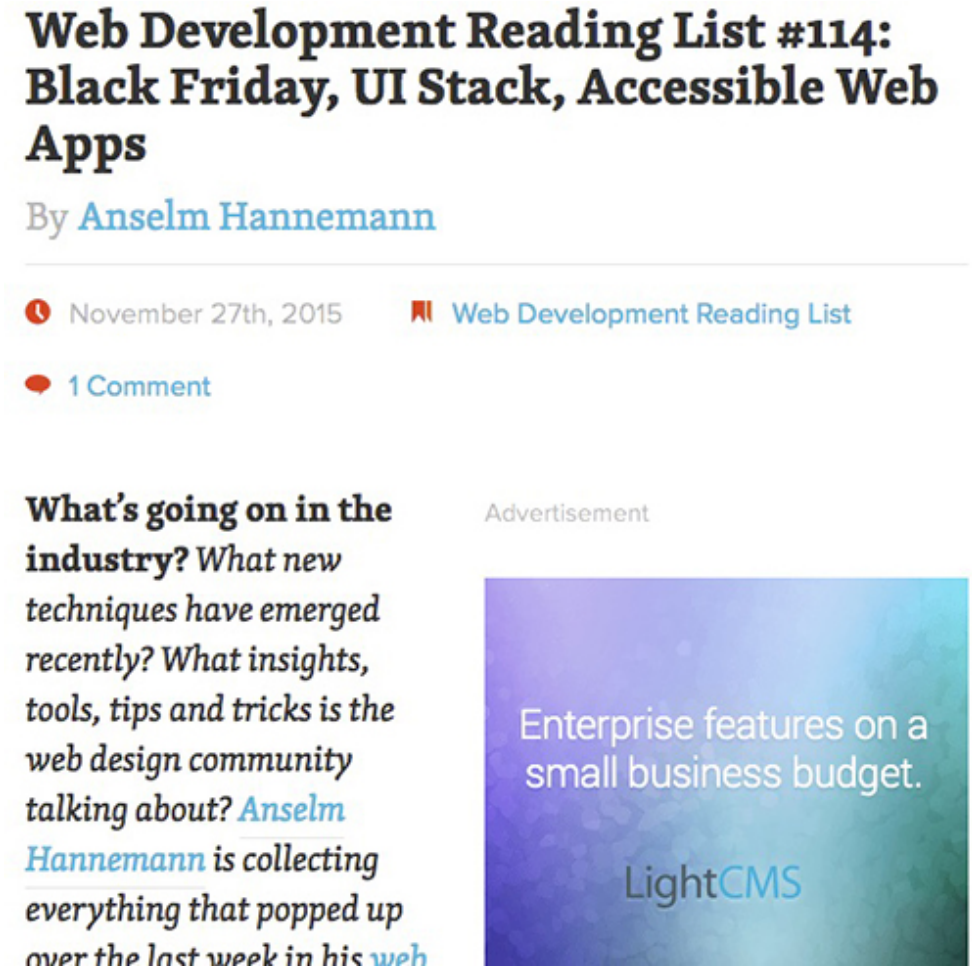
Example #2:

```
<article>
  <h1>Rabbit Habitats</h1>
  <p>Rabbits live in many different habitats...</p>

  <section>
    <h2>Meadows</h2>
    <p>Sometimes they live in meadows...</p>
  </section>

  <section>
    <h2>Woods</h2>
    <p>Often they live in woods...</p>
  </section>
</article>
```

The `<article>` element should typically contain at least one heading. Similar to the section element, we must let article elements do semantic work but not style work. If styling is needed, you'll want to use a div.



Smashing Magazine, a blog for Web professionals, wraps every blog post in an <article> tag.

```
154 <article class="post-230553 post type-post status-publish format-standard has-post-
    thumbnail hentry category-general tag-web-development-reading-list"
    vocab="http://schema.org/" typeof="TechArticle">
155
156                 <h2>
157                 <a property="url"
    href="http://www.smashingmagazine.com/2015/11/black-friday-ui-stack-accessibility-
    performance/" rel="bookmark" title="Read 'Web Development Reading List #114: Black
    Friday, UI Stack, Accessible Web Apps'"><span property="name">Web Development Reading
    List #114: Black Friday, UI Stack, Accessible Web Apps</span></a>
    ...
    original_referer=http%3A%2F%2Fwww.smashingmagazine.com%2F2015%2F11%2Fblack-friday-ui-
    stack-accessibility-
    performance%2F&source=tweetbutton&text=Web%20Development%20Reading%20List%20%2311%3A
    %20Black%20Friday%2C%20UI%20Stack%2C%20Accessible%20Web%20Apps&url=http%3A%2F%2Fwww.s
    mashingmagazine.com%2F2015%2F11%2Fblack-friday-ui-stack-accessibility-
    performance%2F&via=smashingmag">Tweet it</a><a class="sot single"
    href="http://www.facebook.com/sharer/sharer.php?
    u=http%3A%2F%2Fwww.smashingmagazine.com%2F2015%2F11%2Fblack-friday-ui-stack-
    accessibility-performance%2F">Share on Facebook</a></p></p>
266
267 </article>
```

The article tag

The <aside> Element

Still with me? Good! The `<aside>` element is a cousin of the `<article>` or `<section>` element. Use the `<aside>` tag when indicating content that's related to, but separate from, the content it's inside of. A good use of the aside element might include a pull-quote, a bit of sidebar information, or a special factoid.

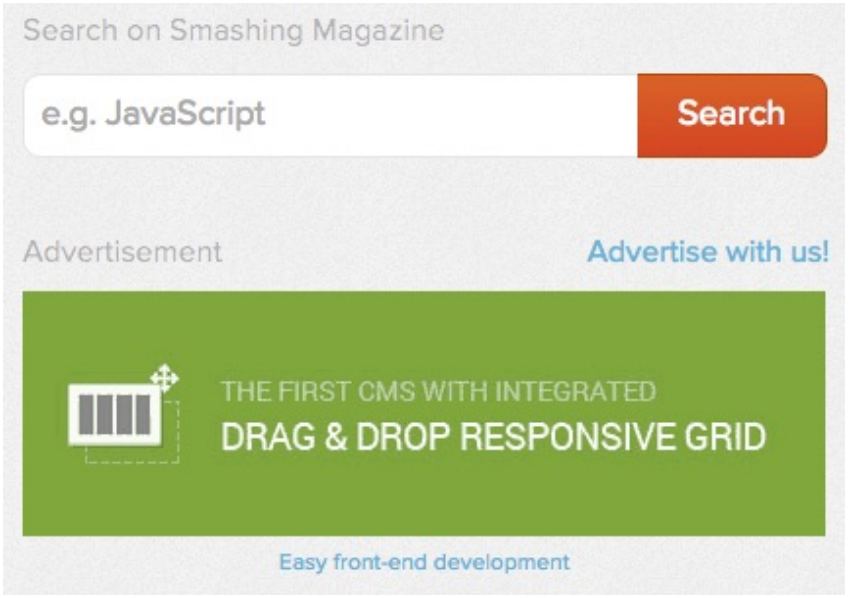
Example:

```
<p>Rabbits often live in burrows.</p>

<aside>
  <h4>Did you know?</h4>
  <p>A burrow is another name for a rabbit hole.</p>
</aside>
```

If you're wondering whether the <aside> tag is appropriate, ask yourself whether the information you'd like to pull aside is needed to understand the main content. If removing the content will negatively impact a person's understanding of the page content, <aside> probably isn't appropriate. If removing the content won't impact someone's understanding, then give it a whirl!

You'll often find the aside element nested inside the article tag, but it can be used elsewhere as well. When used inside the article element, the content in the aside tag should relate to the article. When used elsewhere, the content inside the aside tag should relate to the Web page or the site as whole (like a group of sidebar links or advertisements).



Smashing Magazine uses the <aside> element in this complementary sidebar, which includes a search bar and advertisements.

```
477 <aside id="wpsidebar" class="sb" role="complementary">
478         <div class="col side">
479                 <div class="mise sf clearfix" >
480         <form role="search" id="search_2" method="get"
action="http://www.smashingmagazine.com/search-results/"
target="_top">
481     <label class="sl" for="searching_2">Search on Smashing
Magazine</label>
...
567                 <div class="clearfix">
568   <div class="oa_zone--ad sidebar_sky" id="sidebar_sky-3"
data-ad-name="Sidebar 3" data-ad-zone="40"></div>
569 </div>
570
571         </div>
572 </aside>
```

The aside element

The <footer> Element

The `<footer>` element contains footer information for a Web page or section. It can be used in place of `<div id="footer">` and generally contains copyright information, information about the site creator or authors, contact information, or secondary site navigation links.

Example:

```
<footer>
        <p>Copyright 2015 Rabbit Burrow Studios.</p>
        <p>Contact: <a href="mailto:rabbit@website.com"</a>.</p>
</footer>
```

Just like you're allowed multiple `<header>` elements in one document, you can include multiple `<footer>` elements, as long as they're used correctly. For example, each `<article>` or `<section>` on a page could include its own `<footer>` tag.

Lost World's Fairs takes advantage of a large <footer> area, which includes author names and bios. The information in the <footer> element can be simple or complex, as long as it comes at the end of a section or at the bottom of the site.

```
78          <footer id="footer">
79                  <div class="container clearfix"><img class="fom"
src="img/friendsofmighty.png" alt="friends of mighty"/>
80                  <div class="grid_6 mighty_box">
81
82                          <div class="bio_row clearfix">
83                                  <div class="grid_2 alpha">
84                                  <img class="biopic" src="img/jasonsantamaria.png"
alt="Jason Santa Maria"/>
...
116                 </div><p id="copyright">&copy; 2010 <a
href="http://madebymighty.com">Friends Of Mighty</a><br/><a
href="http://typekit.com/colophons/ops5zjr" class="footer_tk_link">View the fonts used
on this page</a></p>
117
118         </footer>
```

The footer element

Those are just some of most important semantic tags introduced in HTML5. For a complete list of HTML5 elements, check out the helpful Element Index at HTML5 Doctor.

Adding Audio and Video Using HTML5

important!

Only the newer versions of the main browsers support HTML5 audio and video.

An exciting new development in HTML5 is the ability to play audio and video without using third-party plugins such as Flash or QuickTime. HTML5 has defined `<audio>` and `<video>` tags that enable developers to easily encode players that play these media files natively within Web documents.

Despite this new potential promise for headache-free media deployment on the Web, there are a few snags.

First, there is currently limited browser support for these elements. Only the newest versions of Firefox, Safari, Chrome, and Internet Explorer (as well as the iOS and Android) support `<audio>` and `<video>`. And while we would like to imagine a world where everyone is using the newest browser, browser usage statistics show us that simply isn't true. For example, Internet Explorer users are known for using a version one or two generations old. Second, even for browsers that support `<audio>` and `<video>`, different browsers support different audio and video formats.

A third factor to bear mind is that the advent of video-sharing Web services such as YouTube and Vimeo has changed the dynamic in video and audio publishing on the Web. Both Youtube and Vimeo present compelling advantages to self-publishing media content:

- It's easy to upload/publish.
- Your movies will play on almost any system.
- Your movie can reach a wide search audience and be found through searches for related content.
- Movies published on YouTube or Vimeo can *also* be published on your site, by grabbing provided code.

Keeping this mind, it's important to learn how to make use of these nifty HTML5 features, as they may become more important over time, and certain clients/projects may require it.

The <audio> Element

note

The audio element has a controls attribute that enables you to create a simple interface.

Adding audio to a page couldn't be simpler with the HTML5 `<audio>` element. Download the Lesson 4 folder. Create a new basic HTML file inside it called audio.html and place it inside the audio folder that you downloaded. Remember to add your HTML5 doctype and charset. Good habits!

There is an MP3 file in that folder that you will place in your new HTML page. Write the following code in the body of your new page.

```
<audio controls src="iwannago.mp3">
    Browser not supported.
</audio>
```

Let's break down the simple code you just added. We have the `<audio>` tag here along with some essential attributes, the most important of which is the src attribute. Similar to its use in an img tag, this defines the URL of the audio file to be played, in this case the MP3 file in your download folder.

important!

Not all browsers support all audio file formats, also called codecs.

In the above example, we've also included the **controls** attribute, which displays a simple user interface for your media. The exact appearance of the interface varies according the browser.

The player on the left appears in the Safari browser when the controls attribute is used. The player on the right is from Firefox.

If you didn't want to include a set of controls, you would most likely include the **autoplay** attribute instead, which automatically plays the audio file in the browser when it is loaded.

How about that "Browser not supported" line in the code example? This is an error message for the user in case he or she is using a browser that doesn't support the `<audio>` element.

Codec Support

Unfortunately, not all browsers support all audio file formats (or codecs). The MP3 file in the above code, for example, won't play in older version of Firefox. The chart below shows the browser support for popular audio file formats at the time of this writing.

	Firefox	Safari	Opera	Chrome	IE	Microsoft Edge
MP4	No Native Support	Yes	No Native Support	Yes	Yes	Yes
Ogg Vorbis	Yes	No Native Support	Yes	Yes	No	Yes
WAV	Yes	Yes	Yes	Yes	No	Yes

As you'll notice, there isn't one audio codec that all browsers support. So, in order to ensure cross-browser support, we have to encode our audio in more than one codec, and reference both audio files within the `<audio>` tag. A good choice is to include both an MP3 (.mp3) and an **Ogg Vorbis** (.ogg) file (which is an open-source audio codec similar in size and quality to MP3). WAV files are not as practical, as this format can be much larger than the other two.

important!

Users should always have some control over the playback of your media.

Your download folder contains an Ogg Vorbis file of the audio. Adjust the code on your page to reference both the .mp3 and the .ogg files. The code should look something like what you see below. Note that I have also added the `autoplay` attribute:

Important: Users should always have some level of control over the playback of your media.

```
<audio autoplay controls>
        <source src="iwannago.ogg" />
        <source src="iwannago.mp3" />
        Browser not supported.
</audio>
```

(Did you notice the self-closing `<source>` tags above?)

important!

Make sure you test any audio or video you add to a page in multiple browsers.

The browser will try to play the first audio file it comes to. If it can't play the first file, it will skip it and move on to the next file.

Adding Your Own Controls with JavaScript

If you don't like the default controls for your audio player (or, if you just want the functionality to be more consistent across browsers), you can create your own quite easily using JavaScript. Don't worry! It may sound advanced, but we're going to work with just some very straightforward controls.

First, add the following button below your closing `<audio>` tag (on the next line):

```
<input type=button value="Play/Pause" tabindex="0" />
```

This line of code displays a simple "Play/Pause" button below the audio player.

The Play/Pause button in Firefox, under the controls

Now, we have to attach some actions to this button so something happens when the user clicks it.

Include the following attribute in the existing `input` tag (in bold below):

```
<input type=button onclick="playPause()" value="Play/Pause"
tabindex="0" />
```

This tells the browser to run a piece of JavaScript called "playPause" when the button is clicked. So, now, we'll need to include that script inside the `<head>` tags of our HTML document. Remember, you don't actually need to understand JavaScript right now to use it. This JavaScript can be used and reused as often as you like for your audio pages without any modifications.

```
<script>
        function playPause() {
      var myAudio = document.getElementsByTagName('audio')[0];
      if(myAudio.paused)
      myAudio.play();
      else
      myAudio.pause();
        }
</script>
```

tip!

You can include backup options for users whose browser doesn't support video.

This script plays the audio if it is paused, and pauses the audio if it is already playing. How does it do this? It assigns a variable to the `<audio>` tag (called "myAudio"). Then there is a simple if/else statement that essentially says, "If myAudio is paused, play it; if myAudio is playing, pause it."

Once we've created the button and added the JavaScript, we really don't need the default browser controllers anymore. So we can just get rid of the controls attribute in the `<audio>` tag, leaving only the autoplay attribute:

```
<audio autoplay>
```

You can add an opening frame to a page with the poster attribute.

Try out your HTML page in at least two browsers now. If you have trouble, check out the finished version of the page in your download folder, which also includes some CSS styling. When you gain more scripting experience, you can create more elaborate JavaScript controls than these. But this example shows how easy it is to add your own customized controls.

Important: You do not need to know how to write your own JavaScript to use the audio and video controllers shown in this lecture.

The <video> Element

Before we start with a page that incorporates video, let me give you some best practice tips for using video so that you don't scare your users away. (These apply to audio as well.)

- Don't autoplay media files unless you want to annoy people. You know, old people like myself.
- Video files are rather large, so keep their use to a minimum unless they are necessary to your site. If you are serious about putting videos online, look into movie compression and video codecs to squeeze more movie out of your files.
- Provide a backup option in case a visitor's browser doesn't support native video support.

Create a new page called video.html and place it in the video folder you downloaded for this lecture.

The syntax and available attributes of the `<video>` element are very similar to those of the `<audio>` element. So let's dive right in and try a simple example.

```
<video controls>
   <source src="Kissing.mp4" type="video/mp4" />
        Browser not supported.
</video>
```

We've included the default control set that the browser provides (the `controls` attribute), and have sourced an MPEG-4 video file (Apple's popular video format). Try playing it now, by opening your video.html page in the Safari browser.

The video playing in the Safari browser with basic browser controls.

Codec Support

As with audio, different browsers support different video formats.

Every video file you play is actually a "container" that encases both an audio and video codec. For example, an MP4 video file is a container that usually includes an H.264 video codec and an AAC audio codec.

While different containers can encase various combinations of audio and video codecs, for our purposes we'll narrow our consideration to the two widely supported video containers: MP4 and Ogg. As I just noted, MP4 contains the H.264 video file format (which is also used in Blu-ray) and the AAC audio file format. Ogg is an open-source solution that contains a Theora video file and a Vorbis audio file.

Here's what browser support for the two major video codecs looks like at the time of writing.

	Firefox	Safari	Opera	Chrome	IE	Microsoft Edge
MP4	Yes	Yes	Yes	Yes	Yes	Yes
Ogg	Yes	Yes	Yes	Yes	No	No

While there isn't any one format that works on all browsers, including both file formats should provide a working video for all of these browsers:

```
<video controls>
        <source src="Kissing.ogv" type="video/ogg">
        <source src="Kissing.mp4" type="video/mp4">
        Video Not Supported
</video>
```

Setting Additional Attributes

A great feature of adding videos in HTML5 is that you can add your own opening frame for the video. This is called a **poster frame**, which displays if the browser can't play the video, or while the video is loading.

In your video folder is a file called poster.jpg. This is simply a graphic created in Photoshop at the same dimension as the video itself. To add this to your video, simply include the poster attribute in the `<video>` tag, and have it point to the poster image file:

```
<video controls poster="poster.jpg">
```

Open the page in a browser to see the results.

The poster frame in Firefox

In addition to adding the poster frame, you can further customize your video. First, you can adjust the size of your video by setting the height and width in the `<video>` tag:

```
<video controls poster="poster.jpg" width="320" height="240">
```

And if you just can't get enough of this kissing clip, you can loop the video so that it plays over and over again:

```
<video controls poster="poster.jpg" width="320"
height="240" loop>
```

Controlling Your Video with JavaScript

We can create custom controls for our video, much like we did for our audio clip. In fact, the code is nearly identical and can be used as-is without any fancy JavaScript expert modifications.

As you did for audio, start off by adding a play/pause `input` button just under your closing `video` tag.

```
<input type=button onclick="playPause()" value="Play/Pause"
tabindex="0" />
```

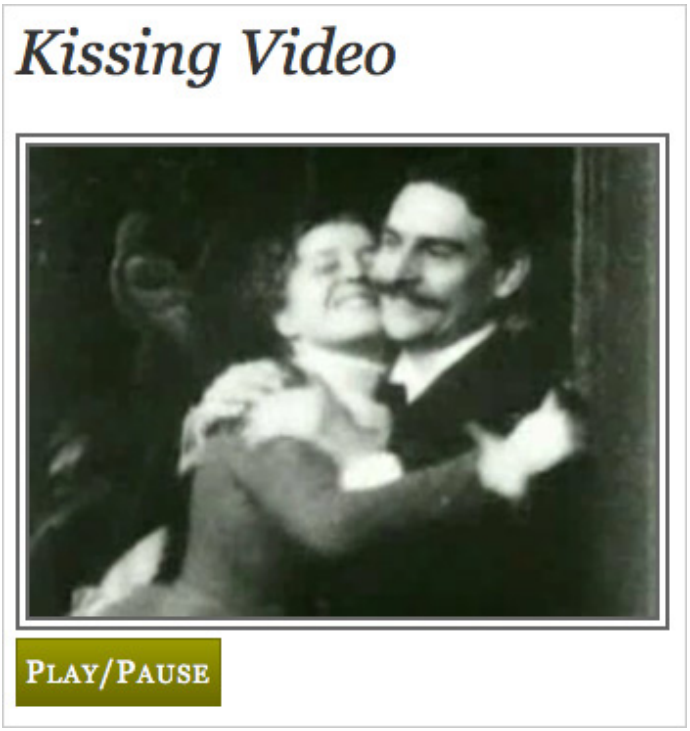
Then, add the following JavaScript in the head of the HTML document:

```
<script>
    function playPause() {
      var myVideo = document.getElementsByTagName('video')[0];
      if(myVideo.paused)
      myVideo.play();
      else
      myVideo.pause();
    }
</script>
```

The only crucial difference between this script and the one we had for the audio player is that we have to make sure to assign the variable to the `<video>` (not the `<audio>`) element, which you can see in the fourth line. Then you can remove your controls attribute from the `<video>` tag, because you now have your own JavaScript controller.

If you'd like to put your knowledge of CSS to use, you'll see that further customizations can be made to the overall style of the input button and the page itself. To customize my controller even further, I added my own button

graphic, subtle border, and CSS-styled text. If you're feeling adventurous, check out my code in the video_finished.html file contained in your video download folder.



A highly customized controller

On Browser Support

It's true: Concerns about browser support can sometimes throw a wet blanket over our eagerness to embrace new features.

So just to be safe, and to accommodate the widest possible variety of users, you will want to provide alternative paths to any audio and video content that you include on your pages. For example, include a fallback video or audio player such as a Flash player, or at least a direct link to the file itself within the `<audio>` and `<video>` tags. While this may be a bit of a hassle—especially considering how wonderfully simple the HTML5 code is—this ensures true cross-browser compatibility for your media players.

And on that note, let us now turn our attention to some other things we can do to ensure that pages look as good as possible, for as many users as possible.

Troubleshooting Web Pages

Test on Multiple Browsers

While looking at your page in a browser does not necessarily alert you to coding errors, getting your page to look right in a browser is certainly an essential step in Web page design. Unfortunately, completing this step is often easier said than done.

The most frustrating thing about designing Web pages is the varying ways in which different browsers render the same code. Even the same browser in a different operating system can interpret a page differently. Despite the existence of Web standards that state how browsers should interpret and

render code, different browsers comply with these standards to varying degrees.

Internet Explorer, the browser that until a few years ago made up the majority of the computer user base, is, ironically, notorious for its non-compliance with Web standards, though each version is typically is a major step up from its predecessor, and its successor Microsoft Edge is improved.

What this means, practically speaking, is that a page that looks great in one browser may appear "broken," or look just awful, in another, even if the code is technically valid. The Web designer is forced to test a site in various browsers (on various platforms) and sometimes employ various hacks and workarounds to get the site to look relatively consistent across browsers, as you saw in our audio and video examples. Hopefully, as Web standards compliance improves, the need for such rigorous testing and use of hacks will lessen.

In the meantime, it is still imperative that you test your HTML documents on various browsers, various browser versions, and, ideally, on different operating systems even if the page validates (conforms to Web standards).

It is best to start with browsers that more strictly adhere to Web standards, fix what is necessary to get the page working, then move on to less compliant browsers. If you start off by accommodating your code to browsers that don't interpret the code correctly to begin with, you tend to end up with code that is not clean and not compliant.

Start coding Web designs for browsers like Chrome, Firefox, and Safari, which are very standards-compliant. Internet Explorer is known to flout Web standards, so you'll want to test your pages on a couple of versions of IE once you get them working in Chrome, Firefox, and Safari.

Test your work in Chrome, Firefox, Safari... then Internet Explorer.

Getting Access to Different Browsers (on the Cheap)

The bigger your Web design projects get, the more you will want to add cross-platform and cross-browser testing into the mix. For example, a client may be able to give you Google Analytics data on the main operating systems and browsers, as well as mobile devices, used by site visitors.

Being able to test different browsers on different platforms can be a challenge. Internet Explorer, for example, is not available for Mac or Linux, and you generally can't install more than one version of Internet Explorer on Windows. Seeing how pages look on different operating systems can be problematic for the majority of us who only use one computer.

There are, however, workarounds to see your pages in various browsers:

- **Install a virtual machine or emulator.** Using Parallels for OS X or Wine for Linux allows you to run Windows programs while running your native operating system. (And if you use Linux, check out the IEs4Linux project, which installs several versions of Internet Explorer on systems that run Wine.)

- **Use a Web-based service.** The media site Mashable maintains a great list of Web based services for cross-platform and multiple browser testing. Many of them have free versions you can use, and as you get bigger projects or work in teams you might consider purchasing a product.
  - Browsershots.org provides you with screenshots of your page running in various browsers, on various operating systems.
  - NetRenderer.com provides browser views of your pages for Internet Explorer 5.5 through 11, Useful for Mac users!
  - Spoon.net provides a simple looking browser sandbox for PC users.

In the next lesson

- Learn how to create and style horizontal navigation using HTML and CSS.
- Learn how to use CSS height and width properties to standardize layout proportions.
- Learn how to add Web fonts to your site's HTML and CSS.

Coming Up Next:

Discussion
Share your thoughts and opinions on semantic HTML in the Discussion area.

Exercise
Apply semantic markup and coding as you style an informational Web page.