ISTA 230

HTML 5

HTML 5

The many flavors of HTML

As discussed in earlier lectures, HTML has had a number of versions, each with their own improvements and drawbacks.

- HTML
- HTML+
- HTML 2.0
- HTML 3.2
- HTML 4.0 (Strict or Transitional)
- HTML 4.01 (Strict or Transitional)
- **XHTML 1.0 Strict**

Before 2008, the most common versions of HTML used were HTML 4.01 (Strict or Transitional) and XHTML 1.0 Strict. While both were well supported by browsers, they had significant differences from one another.

HTML 5

HTML 4.01 vs. XHTML 1.0

HTML 4.01 allowed some elements to be 'improperly' nested. Conversely, XHTML 1.0 Strict did not allow this, forcing all tags to be closed in the opposite order in which they were opened.

HTML - Valid HTML 4.01

```
<strong><em>Important!</strong></em>
```

HTML - Valid XHTML 1.0

```
<strong><em>Important!</em></strong>
```

HTML 5

HTML 4.01 vs. XHTML 1.0

HTML 4.01 didn't require that empty elements (e.g.,
, <img>) to be closed. XHTML 1.0 Strict, on the other hand, required that all elements be closed. There are no self-closing tags in HTML 4.01 Strict.

HTML - Valid HTML 4.01

```
<img src="image.jpg" alt="Some picture"><br>
```

HTML - Valid XHTML 1.0

```
<img src="image.jpg" alt="Some picture" /><br />
```

HTML 5

HTML 4.01 vs. XHTML 1.0

HTML 4.01 allowed designers to use both upper and lowercase for HTML tags. Additionally, it didn't require that the opening and closing tags use the same capitalization. Even attribute names could be capitalized. XHTML 1.0 Strict wasn't having any of that mess. All element names and attribute names must be lowercase. No exceptions.

HTML - Valid HTML 4.01

```
<BODY>
<P>My content goes here.</p>
</BODY>
```

HTML - Valid XHTML 1.0

```
<body>
<p>My content goes here.</p>
</body>
```

HTML 5

HTML 4.01 vs. XHTML 1.0

In HTML 4.01, attributes could be quoted if the designer so desired but it was not required in all cases. Additionally, designers could 'minimize' their attributes, specifying a single word for the attribute rather than the form

```
attribute_name='attribute_value'
```

The browser would automatically expand these minimized attributes into the above form.

XHTML 1.0 Strict required that designers be much more verbose with element attributes. No attribute minimization allowed and all attribute values must be quoted.

HTML - Valid HTML 4.01

```
<input type="text" class=required>
<input type="radio" checked>
<input type="checkbox" class=required checked>
```

HTML - Valid XHTML 1.0

```
<input type="checkbox" class="required" checked="checked" />
```

Lastly, XHTML 1.0 Strict required users to include a XHTML DOCTYPE, to specify an 'xmlns' attribute in the <html> element, and users were required to minimally include <html>, <head>, <title>, and <body> elements (which HTML 4.01 didn't require!). While some of these requirements may seem extraneous, there is a reason for them; the goal of XHTML 1.0 was to take the HTML 4.01 specification and convert it to a valid, "well-formed" XML specification. The claim was that this would make rendering said documents much easier and more reliable across different browsers and devices and, ultimately, would lead to better web page architecture.

HTML 5

Working towards the future...

XHTML 1.0 was published in the year 2000. In 2002, the W3C began working on XHTML 2.0.

XHTML 2.0 was touted as a clean break from HTML 4.01 and XHTML 1.0. Additionally, there would be **no backwards compatibility with prior versions of HTML!** For example, there was discussion of breaking the self-contained <img> tag into opening and closing tags, removing the 'alt' attribute, and placing alternative text between the opening and closing tags (similar to how the <object> element can be used).

HTML

```
<img src="titanic.jpg">RMS <i>Titanic</i></img>.
```

As you can imagine, there was a fair amount of concern from developers web designers and developers based on the lack of backwards compatibility.

When Tim Berners-Lee first created HTML, the web was a (relatively) small network of people, mostly sharing documents. These roots were clearly evident in XHTML 2.0, with it's primary goals on making web pages valid XML with a focus on published documents. However, by 2002, the web had evolved far beyond its initial state, with websites offering video/audio streaming and developers creating robust web-based applications in addition to document-based websites.

By 2004, developers outside of the W3C had grown concerned waiting for the HTML specification(s) to catch up with modern development. In addition to the slow pace of the work on XHTML 2.0 specification, developers were concerned with the lack of progress in addressing the need for specifications around web forms and web-based applications. In June, 2004, developers from Opera Software and the Mozilla Foundation submitted a joint paper, proposing that the W3C extend the existing HTML (4.01) specification to maintain backwards compatibility, to address the needs of the new web-application environment, and to provide browser makers with a standard specification rather than leaving it to vendors to come up with their own best practices (which was, alarmingly, becoming more common).

The W3C rejected this proposal, choosing to continue with their focus on XHTML 2.0. Two days later, the group of developers who had submitted the paper formed their own group known as the Web Hypertext Application Technology Working Group (WHATWG). They justified their group's existence by stating that it was a necessity based on the W3C's "lack of interest in HTML and apparent disregard for the needs of real-world authors."

The WHATWG took on the motto, "Maintaining and evolving HTML since 2004." The group was founded by developers from Opera Software, the Mozilla Foundation, with developers from Apple joining shortly thereafter. Their goal was to develop a specification that would be less document-focused and more application-friendly. Additionally, they wanted HTML specifications that would be backwards compatible with HTML 4.01 and would provide a standard that browser makers could all use to ensure relatively consistent user experiences from browser to browser.

HTML 5

WHATWG vs. W3C

Independent of W3C, the WHATWG released two specifications: Web Applications 1.0 and Web Forms 1.0. Both of these specifications were backwards compatible with HTML 4.01, as promised. However, developers who were used to relying on the W3C as the sole source for (X)HTML specifications were now stuck trying to figure out which specification(s) to refer to. While the W3C was seen as an authority, the developers that made up the WHATWG had the final say as they were the ones making the browsers themselves. Slowly but surely, developers started relying on the WHATWG to provide the guidance for how things should be done.

With the introduction of more robust web-applications like Gmail (2004) and Google Maps (2005), it became clear that web applications were here to stay and needed specifications like those offered by the WHATWG. By October, 2006, Tim Berners-Lee addressed some of the concerns that developers had with the W3C's work on XHTML and announced that the W3C would move forward with the HTML specification development while XHTML 2.0 would be a separate project.

In April, 2007, the WHATWG proposed that the W3C HTML working group adopt the work that they had done thus far. W3C agreed, taking the "Web Apps 1.0" & "Web Forms 2.0" specifications and merging them into what was collectively labeled as HTML 5 specification, released in January, 2008. While this was certainly a step in the right direction, there was a bigger issue to still be addressed: Who was responsible for the maintenance and development of the new specification?

From 2008 until 2012, both the W3C and the WHATWG would work together, each releasing their own regular drafts of the HTML 5 specification, either of which could "change at any time". This meant that that elements and/or features listed in the specification one day could be gone or modified the next. To add to the confusion, the HTML 5 specifications released by the W3C and the WHATWG weren't identical. As noted by the WHATWG, "there are numerous differences between [the two specifications]; some minor, some major. Unfortunately, these are not currently accurately documented anywhere, so there is no way to know which are intentional and which are not." As you can imagine, this made keeping up with the specification difficult for designers and developers.

By 2011, it became clear that the W3C and the WHATWG had different goals in mind for the specification. The W3C wanted to 'finalize' a standard, much as they had done for previous specifications. Conversely, the WHATWG wanted to maintain a 'living standard' for HTML 5, basically a specification that would never be finished and could always be added to and/or refined. The agreement they reached was that the two groups would continue to work together with a degree of separation. The WHATWG would maintain **HTML 5: The Living Standard**, "fixing bugs as we find them, adding new features as they become necessary and viable, and generally tracking implementations." Meanwhile, the W3C would produce 'snapshots' of the specification following the procedures long used to create previous HTML standards. In practice, the result of this separation was that the WHATWG would publish standards and then review how their implementation played out in the real world. The W3C would take the opposite approach of observing what parts of the specification were stable and implemented across multiple browsers and would then publish those as part of the HTML 5 specification.

HTML 5

XHTML 1.0 vs. HTML 5

As previously discussed, XHTML 1.0 had the following requirements:

- Elements must be nested properly
- Elements must be closed (either as open/closing tags or as a self-closing tag)
- Element names must be lowercase
- Element attribute names must be lowercase
- Element attribute values must be enclosed in quotes (single or double)
- Element attributes may not be minimized
- Documents must contain additional attributes for XML validation

While maintaining backwards compatibility with HTML 4.01, HTML 5 recognized some of the benefits of XHTML 1.0 and adopted some of the specification as its own. Below are the requirements for HTML 5:

- Elements must be nested properly
- Elements can be closed, though this is not required
- Element names are case-insensitive
- Element attribute names are case-insensitive
- Element attribute values do not have to be enclosed in quotes (though its still a good idea)
- Element attributes can be minimized

It is worth noting that while HTML 5 documents are not inherently valid XML documents, they can be made so by adhering to the requirements of XHTML 1.0 (lowercase tags, quoted attributes, etc.) I, personally, recommend this approach as it tends to lead to more well-formed documents and leaves opportunities for utilization of XML-based tools.

HTML 5

HTML 5

One of the best changes with HTML 5 is a new, improved DOCTYPE statement.

HTML - XHTML 1.0 doctype

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

HTML - HTML 5 doctype

```
<!DOCTYPE html>
```

You'll notice that this new DOCTYPE doesn't include any information about what version of HTML it's using. This is in line with the WHATWG's approach of seeing HTML as a living standard rather than as a static specification.

HTML 5

HTML 5

HTML 5 also removed the requirement that <link> elements have a 'type' attribute. While you can still include the type attribute to specify the MIME type of the file, it is 'purely advisory'.

HTML - XHTML 1.0 link element

```
<link src="style.css" rel="stylesheet" type="text/css" />
```

HTML - HTML 5 link element

```
<link src="style.css" rel="stylesheet" />
```

HTML 5

Block-Level Elements

As mentioned multiple times in our lectures, the purpose of HTML is to provide **structure, organization, and meaning** for our content. Over the semester, we have introduced the following block-level elements to help us provide the above for our content:

- <p>
-
-
- <dl>
- <h1>...<h6>
- <form>
- <table>
-

While you've likely used each of these throughout your course work, the is by far the most used element in both your pages, as well as those of most developers. That said, is also the most generic of all the elements above, providing the least amount of semantic meaning.

HTML 5

Most-Common Class and ID Values

During the development of HTML 5, both Google and Opera Software performed a search of the web, noting what class names and id attributes were most commonly used in web pages. Their results are below:

Most-Common Class Names

- footer
- menu
- style1
- msonormal
- text
- content
- title
- style2
- header
- copyright
- button
- main
- style3

Most-Common ID Attributes

- footer
- content
- header
- logo
- container
- main
- table1
- menu
- layer1
- search
- nav
- wrapper

Some of the class names and id attributes are specific to a given use case or application (e.g., 'msonormal' is a class used by Microsoft Office (MSO) when documents are 'Saved as HTML'). Others, however, provide us with some additional information about what content they might contain (e.g., header, footer, menu).

HTML 5

HTML 5 - Semantic Elements

Based on the research done by Google and Opera, the following semantic elements were added to HTML 5:

- **<header>**
  Used for 'header' content (i.e., introductory content and/or navigational aids). This element can be used as a container for things like your page logo, a search form, page navigation, or introductory elements for the page content.
  This element can be used multiple times on a page (e.g., for the page heading as well as the heading for an article on the page).

- **<main>**
  Used for the main content of a page. This element can be used only once per document.
  The <main> element also cannot be a descendant of an <article>, <aside>, <footer>, <header>, or <nav> element.

- **<footer>**
  Used for 'footer' content. This element can be used as a container for things like a copyright statement, information about the author of an article, links to related documents. It can also be used to repeat the site navigation.
  Like the <header> element, this element can be used multiple times on a page.

- **<nav>**
  Used for navigational content. This element can be used as a container for a group of links. Note that this does not have to be used for all of the links on your page. Rather, it should be used to provide additional structure to your HTML (this is especially helpful for those accessing your site using assistive technologies such as screen readers).
  This element can be used multiple times on a page.

- **<section>**
  Used to group related content. This is not simply a generic element like Rather, it should be used to indicate a section of your website.
  This element can be used multiple times on a page.

- **<article>**
  Used to group a 'self-contained' set of related content. Again, this should not be used simply as a generic element (like).
  This element can be used multiple times on a page.

  > ***When should I use an <article> and when should I use a <section>?***
  >
  > Great question, hypothetical user! <article> elements are considered independent entities which could be redistributed on their own. For example, a news story could easily be displayed on its own (or on a separate site) without any additional context.
  > <section> elements, on the other hand, indicate a division of your content but not-necessarily a stand-alone entity. Typically, <section> elements are used to break up a page's content in the same way a table of contents or outline would. It's worth noting that <article> elements can contain <section> elements and vice-versa.

- **<aside>**
  Used to contain content that is tangentially related to the content around it. This element is typically used for things like pull-quotes, side bars, social media links, and possibly for related ads. The <aside> element **should not** be used for site-wide navigation!
  This element can be used multiple times on a page.

HTML 5

HTML 5 - Semantic Elements

- **<figure>**
  Used for 'self-contained' content (e.g., a picture with a caption, a code sample with instructional text, etc.). This element should be used for content that is part of the main content of the page (i.e., not an <aside>) but who's position within the content isn't fixed. For example, an image and its caption are part of the main content. However, if they're moved from before the first paragraph to after it, it doesn't change the meaning of the content.
  The <figure> element can serve as a container for just about every HTML element.

- **<figcaption>**
  Used as a caption or legend for a <figure> element. This is optional for <figure> elements. However, if you choose to use it, it must be either the first or last child of the <figure>.

- **<time>**
  Used for dates, times, or a combination of the two. The original goal of this element was to provide designers with a tag for **machine-readable** dates/times. However, this proved to be less useful than originally thought. The WHATWG decided to modify the element to represent dates/times in any format, with the added 'datetime' attribute which should be used for providing the date/time in a machine-readable format.

```
<time>15:30</time>
```

```
<time datetime="15:30">3:30PM</time>
```

```
<time datetime="2012-10-16">October 16</time>
```

```
<time datetime="2012-10-16T15:30-0600">3:30P on October 16</time>
```

It's worth noting that the <time> element shouldn't be used for dates prior to the introduction to the Gregorian calendar. See http://www.quirksmode.org/blog/archives/2009/04/making_time_saf.html for a long discussion of this caveat!

HTML 5

Internet Explorer

There are six browsers that make up most of the market today: Chrome, Firefox, Opera, Safari, Microsoft Edge, and Microsoft Internet Explorer. Not all of these browsers are created equal and some support HTML 5 features better than others.

In particular, Internet Explorer proves to be somewhat problematic, especially as we get into older versions. There are currently four versions of Internet Explorer that have significant market share:

- IE 8
- IE 9
- IE 10
- IE 11

While Internet Explorer 8, 9, and 10 are all past "end of life" as of January 12, 2016, many users continue to use them. Because they're no longer supported by Microsoft, these browsers will not receive any new updates (for bug fixes or feature updates). It is important to keep these older browsers in mind when implementing new approaches to web design and web application development.

To address these limitations, we rely on two principles: progressive enhancement and graceful degradation.

Progressive Enhancement	This approach involves building your website based on a basic level of user experience that all browsers will be able to provide when rendering your page. After you have built a fully-functional site that will provide users with a good experience on any browser, you add advanced functionality to enhance users' experience on browsers that support said functionality (but won't impact user experience on those that don't).
Graceful Degradation	This approach is like progressive enhancement but does things the other way around. You build your website for modern browsers that can support all of the features you'd like to use. For older browsers, your site may degrade (and users may not have as nice an experience) but the basic functionality of your site is still in tact. In other words, users on older browsers may not have a great user experience but they'll be able to use every aspect of your website without problem.

It is worth noting that while older versions of Chrome, Firefox, Safari, Opera, and Microsoft Edge do exist, they are not nearly as common as older versions of Internet Explorer. These browsers are known as 'evergreen browsers' because they have the ability to automatically update themselves, implementing new features and/or bug fixes shortly after they become available. Because of this, we will focus only on the most recent version of these browsers, and the most common versions of Internet Explorer (8, 9, 10, and 11).

HTML 5

HTML 5 Semantic Elements

- Provide additional elements
- Added structure, organization, and meaning

Browser Support:
Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: **>= IE 9 only**

HTML 5

HTML 5 - IE 8 Workaround

Case in point, Internet Explorer 8 doesn't support the new HTML 5 semantic elements. Instead, it sees these 'undefined' elements and assumes they should be displayed as inline elements. Additionally, it doesn't let us apply any CSS rules to these elements. This provides a level of degradation that falls far short of 'graceful'.

Fortunately, there's a solution to fix this. Using JavaScript and a little bit of magic/bugginess, we can force Internet Explorer, versions 6-8, to render HTML 5 elements correctly and make them 'styleable'. *Note: JavaScript is a programming language. As this is not a programming course, we won't be delving too deep into the details of how this works. That said, it is a good idea to understand how to include JavaScript files in your web pages.*

To include a JavaScript file, we insert a <script> element into the <head> of our webpage. The <script> element includes opening/closing tags and the following attributes:

- type - The MIME type for the script (typically 'text/javascript')
- src - The URL to the JavaScript file you're trying to include

The JavaScript file we're going to use to make IE 6-8 play nice with HTML 5 is called *HTML5 Shiv*. Details about script can be found at https://github.com/afarkas/html5shiv. For those not interested in the details, the main thing you need to know is the URL for the JavaScript file we'll be including in our webpages:

```
http://html5shiv.googlecode.com/svn/trunk/html5.js
```

We can use this to insert the script into our webpages as follows:

HTML

```
<head>
   <script type="text/javascript" src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
</head>
```

HTML 5

IE conditional comments

While using HTML5 Shiv does solve the problem of HTML 5 compatibility in Internet Explorer 8 and below, it adds an additional HTTP request to our website, increasing the amount of bandwidth used and the time it takes for our page to load.

Rather than forcing all of our users to incur this penalty for IE's transgressions, we can limit the inclusion of our JavaScript file to a specific set of users by using Internet Explorer's conditional comments feature. This feature (available only in Internet Explorer, versions 5-9) allows us to provide special instructions to Internet Explorer that are ignored by other browsers.

The basic form of a conditional comment is the same as that of a normal HTML comment (which is why other browsers ignore them). However, IE has been designed to detect these specific comments and, if certain conditions are met, including the HTML specified within them as though it was part of the document.

For example, the following conditional comment allows us to include our JavaScript file **only if the user is using Internet Explorer**.

HTML

```
<!--[if IE]>
   <script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
<![endif]-->
```

HTML 5

IE conditional comments

In the previous example, we targeted all Internet Explorer users. We can actually do a little bit better than that. If we wanted to show a message to users using IE 8, we'd do the following:

HTML

```
<!--[if IE 8]>
   <p>You are using IE 8.</p>
<![endif]-->
```

Below are some other examples of how to target specific IE versions.

HTML

```
<!--[if lt IE 9]>
   <p>You are using an IE browser less than version 9.</p>
<![endif]-->
```

HTML

```
<!--[if gte IE 8]>
   <p>You are using an IE browser greater than or equal to version 8.</p>
<![endif]-->
```

HTML

```
<!--[if !IE]> -->
   You are not using IE 5-9.
<!-- <![endif]-->
```

HTML 5

IE Document Modes

To further complicate matters, older versions of Internet Explorer included multiple rendering engines! Internet Explorer 8, for example, as three different rendering engines with for different modes!! Why would they do that?!

When Internet Explorer 6 was introduced, it had some major differences in the way it rendered things compared to IE 5. To handle this, Microsoft included both the IE 6 rendering engine and the IE 5 rendering engine and it would try to figure out which to use based on the DOCTYPE (or absence thereof) of the webpage. While IE 6 had made some improvements, it was still a long ways from adhering to the W3C HTML/CSS standards.

When Internet Explorer 7 was released, it fixed a number of bugs in IE 6. However, many websites had already implemented hacks to get around those bugs and now their sites didn't work in IE 7! Additionally, some sites (those with valid DOCTYPE statements) would render using IE 7's engine while others (those without DOCTYPE statements) would render using IE 5's quirky rendering engine! They called this 'Quirks Mode'.

Like IE 7 before it, IE 8 would revert to IE 5 Quirks Mode if a valid DOCTYPE statement wasn't found on a given page. Otherwise, it would **usually** use the IE 8 rendering engine. However, there was a third option. With IE 8, Microsoft had done a fair amount of work trying to get their browser to adhere more strictly to W3C standards. While this was a great thing for developers, it had the potential of breaking a number of web pages that had been designed with the old, non-adherent behavior of IE 7 in mind. Learning from their mistakes with the release of IE 7, Microsoft introduced 'Compatibility Mode'. With the click of a button, users could force IE 8 to act like IE 7, even if they didn't really know what that implied. Additionally, Microsoft maintained a list of website domains known to have issues when rendered with IE 8 and, if a domain was on this list, Internet Explorer would **automatically** switch to Compatibility Mode, using the IE 7 engine! Of course, most of this behavior wasn't well documented, leaving web designers banging their heads on their desks anytime they had to design for IE.

To further complicate matters, IE 8 introduced two modes when using the IE 8 rendering engine: standards mode and "almost standards" mode. "Almost Standards" mode was nearly identical to "Standards" mode except in how it handled images within a table cell (a common occurrence during the dark ages when designers would rely on table-based layouts). Using specific DOCTYPE statements, designers could force browsers (including Firefox, Safari, Chrome, and Opera) to render content using this "Almost Standards" mode. So, for those keeping count, that's four modes for IE 8: IE 5 Quirks Mode, IE 7 Mode, IE 8 "Almost Standards" Mode, and IE 8 Standards Mode.

Confused yet? Now imagine being a designer when all this was going on... But I digress. IE 9 was released and included all of the same modes as IE 8, bringing the number of possible viewing modes up to seven (IE 5, IE 7, IE 8 "Almost Standards" Mode, IE 8 Standards Mode, IE 9 "Almost Standards" Mode, IE 9 Standards Mode, and IE 9 XML Mode). With IE 10, added four more modes: IE 10 "Almost Standards" Mode, IE 10 Standards Mode, IE 10 XML Mode, and IE 10 Quirks Mode (an improved version over IE 5 Quirks mode that more closely matched how other browsers handled really old pages).

HTML 5

IE Document Modes

For those keeping count, that's 11 different viewing modes available in IE 10, determined by a combination of the DOCTYPE statement, the HTTP headers sent by the server, whether the user is on an intranet site, whether the domain is on Microsoft's Compatibility Mode list, and whether the user had clicked the 'Compatibility Mode' button. What a mess! Fortunately, Microsoft realized around the time that they released IE 8 that this issue was going to cause problems for designers. To help address that issue, Microsoft provided designers with a tool that allows us to specify which document mode our site should be displayed with:

Using the <meta> tag above, we can tell Internet Explorer exactly how our webpage should be rendered. Options for the 'content' attribute are as follows:

- IE=edge *(Forces IE to use the most modern rendering engine available in the given browser)*
- IE=IE9
- IE=EmulateIE9
- IE=IE8
- IE=EmulateIE8
- IE=IE7
- IE=EmulateIE7
- IE=IE5

HTML

```
<head>
   <meta http-equiv="X-UA-Compatible" content="IE=edge" >
</head>
```

HTML

```
<head>
   <meta http-equiv="X-UA-Compatible" content="IE=IE9" >
</head>
```

HTML

```
<head>
   <meta http-equiv="X-UA-Compatible" content="IE=IE8" >
</head>
```

HTML

```
<head>
   <meta http-equiv="X-UA-Compatible" content="IE=EmulateIE7" >
</head>
```

HTML

```
<head>
   <meta http-equiv="X-UA-Compatible" content="IE=IE5" >
</head>
```

While all versions of IE from version 8 on have multiple document modes, more recent versions (10+) have made the option of viewing a site in "Compatibility Mode" available only to developers (press F12 in IE to bring up the developer tools). IE 11 also deprecated the <meta> tag shown above, opting instead to always default to "edge" mode. Even better, Microsoft Edge (Microsoft's latest browser), will only have one rendering engine and will rely on a "living" document mode, much the way other modern engines do.

OK, enough time discussing the intricacies of Internet Explorer. Back to HTML 5...

HTML 5

HTML 5 - Meter Element

As previously mentioned, the WHATWG wanted to make the focus of the HTML standard less about documents and more compatible with web application development. As part of that effort, they introduced the <meter> element. The <meter> element is used to show a visual representation of a measurement within a known range.

The <meter> element has opening and closing tags and three required elements:

- value - The current value of the meter
- min - The minimum value of the meter
- max - The maximum value of the meter

HTML

```
<meter value="75" min="0" max="100">75%</meter>
```

Browser

HTML 5

HTML 5 - Meter Element

In addition to the three required attributes, the <meter> element has three optional attributes that can affect its appearance:

- low
- high
- optimum

The 'low' attribute is used to set the maximum value in the low range. The 'high' attribute is used to set the minimum value in the high range. By using these two attributes, we can essentially break our meter into sections: low, mid, and high.

Lastly, we can use the 'optimum' attribute to specify where the optimum value lies within our meter. For example, if we had a grading scale where the range was from 0 to 100, we'd set 'optimum' to 100 to indicate that a higher value was preferred. Conversely, a meter showing hard drive usage would have an 'optimum' value of 0.

By default, the <meter> bar is green in most browsers. However, when we use these optional attributes, we can change the color of our meter for various values.

HTML

```
<meter value="10" min="0" max="100" low="40" high="85" optimum="0">10%</meter>
```

For the example above, the low end of the meter is optimum. Thus, a value of 10 would result in the meter being green (within the optimum, i.e., low section of the meter). Changing the value to 50 would result in it being yellow, while a value of 90 would result in the meter being red.

If we were to set the optimum to '100', the colors would flip, with 10 being a red meter, 50 being a yellow meter, and 90 being a green meter.

Lastly, we could set the optimum value to 50. This would result in 10 being a yellow meter, 50 being a green meter, and 90 being a yellow meter. There is no red meter possible in this scenario.

HTML 5

HTML 5 - Progress Element

Like the <meter> element, <progress> element is useful in web applications that need to show the progress of a specific task. When used, the browser displays a small progress bar, indicating the progress of a specific task. The <progress> element has opening and closing tags but does not have any required attributes. Without any attributes, the status of the progress bar is considered **indeterminate**, meaning that progress is being made but that it is not clear how much more work remains to be done before the task is complete. When in this state, most browsers will animate the progress bar, either showing a small colored block, bouncing back and forth between the left and right sides of the progress bar, or displaying a moving set of stripes, a barber pole-like pattern.

HTML

```
<progress>In progress</progress>
```

Conversely, we can add two optional attributes, 'max' and 'value' to convert the state to **determinate**:

- value - The current progress value for the task
- max - The maximum value of the progress bar, typically 100

In the 'determinate' state, we're telling the browser that we know the exact progress of a task and want to display it on the screen. The browser will take that value and the maximum value for the progress bar, determine a percentage of completion based on those values, and will then fill up the progress bar with a colored bar based on that percentage.

HTML

```
<progress value="75" max="100">75%</progress>
```

Browser

Example of an HTML 5 progress bar

For older browsers who don't support the new <progress> element, the browser will simply display it as an inline element. That said, you'll want to include the HTML5 Shiv JavaScript library discussed earlier in this lecture to ensure that you can style your <progress> elements using CSS. *It is worth noting that <progress> elements are notoriously difficult to style effectively across multiple browsers. For more information about how to do this, see the CSS Tricks article at https://css-tricks.com/html5-progress-element/.*

Browser Support:

Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: IE 10+

HTML 5

HTML 5 - Custom data-* Attributes

HTML 5 also introduced the idea of custom HTML element attributes. This allows us to specify additional data about our elements that may not be defined in the HTML 5 specification, giving us lots of flexibility. We can specify one or more of these custom attributes to a given element by using the following form:

HTML

```
<div data-custom='whatever you want here'>
   ...
</div>
```

Any attribute whose attribute name starts with 'data-' is allowed. Using CSS, we can make use of this in our CSS-generated content. This turns out to also be very helpful in building complex web applications.

CSS

```
#special:before
{
   content: attr(data-custom);
}
```

HTML 5

HTML 5 - Accessibility and ARIA

With the rise of complex web-based applications came the need for improved accessibility measures. Introduced by the W3C's Web Accessibility Initiative, the Accessible Rich Internet Applications (ARIA) specification provided a way to address the missing semantics needed by assistive technologies (e.g., screen readers, etc.) to describe the roles of various components of web applications and their state.

ARIA 'roles' allow us to set an attribute that indicates how a given element is being used on the page. This is useful for providing assistive technology with additional semantic information when a semantic element doesn't exist for the given case. For example, a <header> element could be uesd for the heading at the top of a web page or for the heading of an article (title, author name, etc.). To provide additional information, we could use a 'role' attribute with a valid ARIA role as follows:

HTML

```
<header role="banner">
   ...
</header>
```

Valid ARIA roles include things like 'menuitem', 'alert', 'math', and 'toolbar'. There are **many** other ARIA 'roles', a full list of which can be found at http://www.w3.org/TR/wai-aria/roles.

HTML 5

XHTML 1.0 Forms

One of the earliest focuses of WHATWG during their efforts to modernize the HTML standard was improving web forms. As previously discussed during our lecture on forms, we noted that the first child of a form must be a block element (typically a <fieldset> or a). We also discussed a number of options for the <input> element's "type" attribute:

HTML

```
<input type='   ' >
```

- **text**
- password
- hidden
- radio
- checkbox
- file
- reset
- submit

The most common type of input used on forms today is the "text" input. This field type can be used to contain a user's name, their email address, their telephone number, their social security number, a URL to their homepage, their date of birth, or any other piece of information that we might want to collect that is textual in nature. While this flexibility is good on the one hand, it provides a challenge when it comes to validating user input. To help address this issue, a number of new input "types" were introduced in HTML 5.

HTML 5

HTML 5 Forms - Email Input

The 'email' input type is intended to be used for email addresses. By using this input type, we gain a couple of things in browsers that support it. First, using semantically specific input type is always better than a generic input type. It tells designers exactly what the field was intended for, provides accessibility devices with more information about what type of field it is, and is used by mobile to provide an optimal 'email-friendly' keyboard for users. Additionally, browsers that support this field type will automatically provide a notification if the user enters a non-valid email address. While there are ways around this validation, it is a nice feature to users who do so accidentally.

Lastly, we can add a boolean 'multiple' attribute to specify that users can enter more than one email address into the field, using a comma to separate each address. The browser will then validate each email address before allowing users to submit the form. As a boolean attribute, we don't have to provide a value for it in HTML 5 unless we want to maintain XML compatibility.

HTML

```
<input type="email" multiple />
```

For older browsers who don't support the new 'email' input type, the browser will simply revert to rendering a 'text' input field without any of the validation features.

Browser Support:

Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: IE 10+

HTML 5

HTML 5 Forms - URL Input

Similarly, the 'url' input type is intended to be used for **absolute** URLs. In addition to being semantically more specific, this input type also provides us with validation in browsers that support it. Some browsers will add the text, "http://" to text entered into the field if needed, providing some additional formatting for users. Many mobile devices will also display a 'URL-friendly' keyboard for this input type.

HTML

```
<input type="url" />
```

For older browsers who don't support the new 'url' input type, the browser will simply revert to rendering a 'text' input field.

Browser Support:

Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: IE 10+

HTML 5

HTML 5 Forms - Telephone Input

The 'tel' input type is intended to be used for telephones. Browsers don't enforce a specific format for entering telephone numbers so users can enter any combination of numbers, letters, and symbols into this input field. While this may seem pointless, the semantically more-specific type attribute provides us as designers with a better understanding of what the field should be used for, provides accessibility technology with more information about what type of field it is, and is used by mobile to provide an optimal 'telephone-like' keyboard for users. If you're asking for a telephone number, you should use this input type!

HTML

```
<input type="tel" />
```

For older browsers who don't support the new 'tel' input type, the browser will simply revert to rendering a 'text' input field.

Browser Support:

Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: IE 10+

HTML 5

HTML 5 Forms - Search Input

The 'search' input type doesn't gain us any additional validation like the 'email' or 'url' fields. However, some browsers will add a small magnifying glass icon to the input field, indicating that it is a search box. Others will add a small 'x' icon, allowing users to clear their entries with a single mouse click.

HTML

```
<input type="search" />
```

For older browsers who don't support the new 'search' input type, the browser will simply revert to rendering a 'text' input field.

Browser Support:

Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: IE 10+

HTML 5

HTML 5 Forms - Number Input

The 'number' input type is for integers and floating point numbers. Some desktop browsers will limit the type of input users can type into this field to numbers, periods, and dashes. Similarly, most mobile devices will display a numeric keyboard for this input type. Some desktop browsers will also display 'spinner' controls, allowing users to increment/decrement the value of the field with a mouse click or by scrolling the mouse wheel.

Because users can easily increment/decrement the value of this field inadvertently, it is not a good idea to use this input type for things like zip codes, credit card numbers, or social security numbers.

HTML

```
<input type="number" />
```

For older browsers who don't support the new 'number' input type, the browser will simply revert to rendering a 'text' input field.

Browser Support:

Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: IE 10+

HTML 5

HTML 5 Forms - Number Input

The 'number' input type also introduces three new optional attributes:

- min - Allows us to specify the minimum value allowed
- max - Allows us to specify the maximum value allowed
- step - Allows us to specify the interval by which the value can increment/decrement

When submitted, most browsers will validate 'number' input fields to ensure that the submitted value is between the min and max values (if specified). Additionally, if the value submitted is a step mismatch, then the browser will notify the user that the value is invalid.

HTML

```
<input type="number" min="1" max="10" step="3" />
```

- Three new optional attributes
  - min - minimum value allowed
  - max - maximum value allowed
  - step - Specifies legal number intervals from 0

HTML 5

HTML 5 Forms - Data Lists

HTML 5 also introduced the <datalist> element. This element allows designers to specify a list of options that are available for users to select for a given input field. For example, if we wanted to know what a user's favorite flavor of ice cream is, we could provide them with a list of options using the following HTML:

HTML

```
<input type="text" name="flavors" list="flavorList" />
<datalist id="flavorList">
   <option value="Vanilla">
   <option value="Chocolate">
   <option value="Strawberry">
</datalist>
```

You'll note that we added a 'list' attribute to our input field, the value of which matches the 'id' attribute on our <datalist> element. When a user clicks on the field, a dropdown menu (similar to that of a <select> element) will appear, allowing the user to select one of the options listed in the datalist. **Note that while users will be shown the list of options available to them, they can still opt to type in their own value.**

Browser Support:

Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: No
MS Edge: Yes
IE: IE 10+

It is worth noting that, while supported, there are some bugs with <datalist> elements in IE 10 and 11. Additionally, long lists of options will not be scrollable in some browsers, resulting in unselectable options.

HTML 5

HTML 5 Forms - Range Input

HTML 5 also offers us the 'range' input type. This input type is good for "imprecise" controls. For example, if you're asking someone to rate an app on a scale of 1 - 100, the difference between 87 and 88 seems negligible. A 'range' input might be a good choice for this case.

The 'range' input displays a 'slider' for users, allowing them to click and drag their mouse along a track to select a value.

HTML

```
<input type="range" />
```

For older browsers who don't support the new 'range' input type, the browser will simply revert to rendering a 'text' input field.

Browser Support:

Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: IE 10+

HTML 5

HTML 5 Forms - Range Input

On their own, 'range' input types are somewhat limited in their application. By default, most browsers don't display any tick marks, giving the user a limited idea of what value they're selecting. To actually display the current value of the range input in most browsers, you would need to utilize JavaScript (which is beyond the scope of this course).

That said, most browsers will add tick marks to the slider if you provide a 'list' attribute that points to a <datalist> element. Browsers that support it will add a tick mark for each valid <option> present (but won't display the values themselves).

By default, 'range' inputs have a minimum value of 0, a maximum value of 100, and increment by 1 when dragged. However, we can modify these properties using the same new optional attributes introduced by the 'number' input type:

- min - Allows us to specify the minimum value allowed
- max - Allows us to specify the maximum value allowed
- step - Allows us to specify the interval by which the value can increment/decrement

HTML

```
<input type="range" min="2" max="10" step="2" />
```

According to the official HTML standard, sliders can be vertical or horizontal based on the ratio of the width to the height. However, at this point, most browsers do not support this. While not based on the HTML standard, modern browsers each implement this functionality using their own proprietary methods:

HTML

```
<input type="range" orient="vertical" style="width:20px; height:200px; -webkit-appearance:slider-vertical; writing-
mode:bt-lr;" />
```

Not great. In the above example, we use the "orient" attribute for Firefox, the CSS property "-webkit-appearance" for Chrome, Safari, and Opera, and the CSS property "writing-mode" for Internet Explorer and MS Edge. We also include "width" and "height" in the CSS properties to ensure that future compliance with the HTML standard will work as well. While the example given will display in most browsers, there is some bugginess in the implementation and you may see weird side effects. Use with caution!

HTML 5

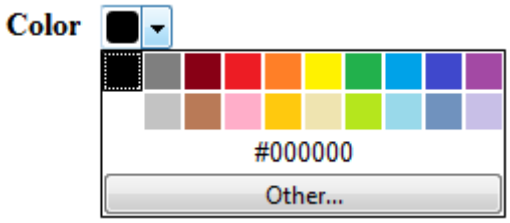
HTML 5 Forms - Color Input

In the past, providing users with an easy way to pick a color was not a trivial task. It required a combination of some pretty fancy HTML, CSS, and JavaScript. HTML 5 attempted to reduce the complicated nature of this task by providing us with a 'color' input type. In browsers that support it, a color picker is provided, allowing the user to visually pick a color without knowing the exact six-digit hexidecimal* value for said color.

While this is supported by a handful of browsers, each of them use their own proprietary color picker so users may have a different experience when using multiple browsers. For browsers that don't support the 'color' input type, a 'text' input field will be displayed instead.

HTML

```
<input type="color" />
```



- Displays a color-picker

Browser Support:
Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: No
MS Edge: No
IE: No

It's worth noting that in some browsers, you can provide a <datalist> element to provide the users with a predefined list of colors to pick from. For browsers that don't support the 'color' input type but support <datalist> elements, users would still see the predefined list of options.

* 'color' input types only accept six-digit hexadecimal values. In other words, CSS color keywords, RGB, RGBa, and other color representations are not allowed.

HTML 5

HTML 5 Forms - Date Inputs

One of the most common uses of input elements is to gather some sort of date/time information (birthday, appointment time, etc.). In the past, we'd use a 'text' input for this sort of thing. HTML 5 provides us with a number of new input types for capturing this sort of information, some more useful than others.

The 'date' input type provides us with a means to capture dates in the format "yyyy-mm-dd". When users interact with this input type, browsers that support it will display a small calendar for the user to pick a date from. Some browsers will also present the date in a more user-friendly format than 'yyyy-mm-dd', though the data will ultimately be submitted as 'yyyy-mm-dd'.

HTML

```
<input type="date" min='2001-01-01' max='2001-12-31' />
```

Like 'number' and 'range' input types, we can specify 'min' and 'max' attributes for 'date' input types. Desktop browsers that support it will use these attributes to prevent users from submitting a value outside of the given date range. However, some mobile devices are known to have bugs enforcing these ranges.

Similarly, you can set a 'step' attribute to indicate the number of days the date input should increment. If the user enters a value that causes a step mismatch, the browser will alert them that they must enter a valid date. Additionally, users will not be able to pick a date from the calendar that doesn't match the step value. Again, it is important to note that some mobile devices don't support the 'step' attribute for 'date' input types.

Browser Support:

Chrome: Yes
Firefox: No
Opera: Yes
Safari: No
MS Edge: Yes
IE: No

As with all of our HTML 5 input types, browsers who don't support the new 'date' input type will simply revert to rendering a 'text' input field.

HTML 5

HTML 5 Forms - Date Inputs

Like the 'date' input type, HTML 5 offers us some additional options for capturing date/time information in a specific format:

HTML

```
<input type="datetime-local" />
<input type="time" />
<input type="week" />
<input type="month" />
```

The 'datetime-local' input type allows us to capture both a date and a time value. The date and time values are concatenated using a 'T' in the format "yyyy-mm-ddTHH:MM" or "yyyy-mm-ddTHH:MM:SS.S", depending on the browser. As with the 'date' input type, browsers may try to present the user in a more readable form, though the submitted value will match one of the above formats. The 'min' and 'max' attributes can be used but should be specified using the format "yyyy-mm-ddTHH:MM". The 'step' attribute can also be used to indicate the number of seconds between steps.

The 'time' input type allows us to capture a time value using the format 'HH:MM' (again, the user may be shown something more readable than the format submitted). When using a 'time' input field, the 'min' and 'max' attributes should be in 'HH:MM' format and the 'step' attribute should be the number of seconds between steps. For example, if we wanted to limit users to 15 increments between 8am and 5pm, we would use the following:

HTML

```
<input type="time" min="08:00" max="17:00" step="900" />
```

The 'week' input type is not likely to be used as often as the other options that we've discussed but it still worth a look. The 'week' input allows us to capture the week number for a given date. Like 'date', users will have the option of picking a date from the calendar. However, the data submitted will be in the format 'yyyy-Www' (the four-digit year, followed by a dash, followed by a 'W' character, followed by the two-digit week number). For this input type, a week begins on Monday and ends on Sunday. The weeks are numbered from 1 to either 52 or 53*.

The 'month' input type allows us to capture a specific year/month date value. As with our other date inputs, users will be presented a calendar from which they can pick a date. The value submitted is in the format 'yyyy-mm', though that's not usually what's shown to the user. Months are numbered from 01 to 12.

Browser Support:

Chrome: Yes
Firefox: No
Opera: Yes
Safari: No
MS Edge: Yes
IE: No

All of the above will be displayed as 'text' input fields in browsers that don't support the given input type.

* According to the HTML standard, "A week-year with a number year has 53 weeks if it corresponds to either a year year in the proleptic Gregorian calendar that has a Thursday as its first day (January 1st), or a year year in the proleptic Gregorian calendar that has a Wednesday as its first day (January 1st) and where year is a number divisible by 400, or a number divisible by 4 but not by 100. All other week-years have 52 weeks." So there's that to consider...

HTML 5

HTML 5 Forms - Placeholder Attribute

In addition to new input types, HTML 5 also introduced new input attributes to solve some of the issues designers had been dealing with for years.

The 'placeholder' attribute allows us to specify text that should be visible before the user has entered any text into a form field but that immediately disappears when the user enters a value into said field. This should be used to provide additional context for users, **NOT** as a replacement for the <label> element.

HTML

```
<input type="text" name="favClass" placeholder="ISTA 230" />
```

This attribute works on the following input types:

- text
- search
- url
- tel
- email
- number
- password

It also works with <textarea> elements.

Browser Support:

Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: IE 10+

It is worth noting that while IE 10 and 11 support the 'placeholder' attribute, the placeholder disappears as soon as the user focuses on a field (even if they haven't typed anything yet).

HTML 5

HTML 5 Forms - Required Attribute

The 'required' attribute allows us to specify which fields require a value before the form can be submitted. While this doesn't prevent malicious users from submitting incomplete data to your form, it is a nice feature for users who accidentally forget to fill out a required field in your form.

HTML

```
<input type="email" required="required" />
```

If a user tries to submit your form without providing a value for a required field, then the browser will automatically bring that field into focus and will display a notification that the field is required. If multiple required fields are missing, then the first one will be brought into focus.

The 'required' attribute can be used with the following input types:

- text
- search
- url
- tel
- email
- radio
- checkbox
- password
- number
- file

When used with checkboxes, the 'required' attribute will force the user to check the checkbox before submitting the form. To use the required attribute with a group of radio buttons, you simply need to apply the 'required' attribute to one of the buttons. The browser will ensure that one of the buttons in the group is selected (though not necessarily the one with the 'required' attribute).

Browser Support:

Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: No
MS Edge: Yes
IE: IE 10+

HTML 5

HTML 5 Forms - Form Attribute

HTML

```
<form id="userForm"/>
<input type="text" name="fname" />
</form>
<input type="text" name="lname" form="userForm" />
```

- Specifies what form the input field belongs to
- Can have multiple values

HTML 5

HTML 5 Forms - Form Attribute

HTML

```
<form id="userForm"/>
<input type="text" name="fname" />
</form>
<input type="text" name="lname" form="userForm" />
```

- Specifies what form the input field belongs to
- Can have multiple values

Browser Support:

Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
IE: No

HTML 5

HTML 5 Forms - Autocomplete

Often times, browsers will keep a record of what values have been entered into forms and will use that to optionally fill out forms for users. The 'autocomplete' attribute allows us to enable/disable this feature.

HTML

```
<form autocomplete="off" ... />
<input type="text" name="fname" />
</form>
```

By default, this attribute has a value of 'on' but we can disable it by setting the value to 'off'. This is useful when you have form fields that might share a name/label with a similar field on another form but in a different context. For example, if we had a form asking for information about a user's pet, we might have an input field asking for their 'Name'. We wouldn't want the browser to fill this out with the user's name automatically so we might set 'autocomplete' to 'off' for that form.

Browser Support:

Chrome: Yes
Firefox: Yes
Opera: No
Safari: Yes
MS Edge: Yes
IE: IE 11

This attribute can be applied to the <form> element to enable/disable autocomplete for all fields or can be applied to individual <input> elements to apply it to only that specific field.

While this is well supported, it's worth noting that a number of browsers have slightly modified their implementations to address specific issues. For example, Internet Explorer ignores this attributes for 'password' input types. Similarly, Firefox and Safari have taken measures to ignore 'autocomplete' for login forms. This is largely to ensure that password managers will work as expected, regardless of whether the designer has disabled 'autocomplete'.

HTML 5

HTML 5 Forms - Autofocus

Often times, designers might want to immediately focus the cursor within a specific input field. For example, login forms often focus the cursor on the first field of the form. In the past, this was done using JavaScript. However, HTML 5 provided us with the 'autofocus' attribute which allows us to specify a field that should automatically gain focus as soon as the page has loaded.

HTML

```
<input type="text" name="username" autofocus />
```

If multiple form fields have this attribute set, most browsers will give focus to the first element having the attribute. However, Safari gives focus to the last element having this attribute. For everyone's sake, never apply this attribute to more than a single form field on any given page.

Some would go further and argue that you shouldn't use this attribute at all as it can cause the page to scroll without the user understanding why. The argument is that the user should be the driving force behind what elements have focus, not the designer. I, personally, don't have strong opinions either way. Think of your target user and use your best judgement to determine if this attribute is right for you.

Browser Support:

Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: IE 10+

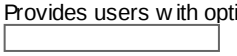
HTML 5

HTML 5 Forms - Data Lists

HTML

```
<input type="text" name="flavors" list="flavorList" />
<datalist id="flavorList">
   <option value="Vanilla">
   <option value="Chocolate">
   <option value="Strawberry">
</datalist>
```

- Provides users with options to select **or** the ability to type in their own



HTML 5

HTML 5 Forms - Data Lists

HTML

```
<input type="text" name="flavors" list="flavorList" />
<datalist id="flavorList">
   <option value="Vanilla">
   <option value="Chocolate">
   <option value="Strawberry">
</datalist>
```

- Provides users with options to select **or** the ability to type in their own

Browser Support:
Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: No
IE: Partial

HTML 5

HTML 5 Forms - Form Action Override

HTML 5 also provided us with an easy way to override where form data should be sent. Consider the following scenario: You're building a website for users to submit feedback to your company. The sales department has asked to you create a form that asks users for their name, their email address, and their feedback. Similarly, the marketing department has asked you to create a form with the exact same fields. Rather than creating two separate forms, you can use HTML 5 to create a single form and direct it to one of two URLs, depending on what submit button users click.

HTML

```
<form action="sales.php" ... />
  Name: <input type="text" ... />
  Email: <input type="text" ... />
  Message: <textarea...></textarea>
  <button type="submit">Send to Sales</button>
  <button type="submit" formaction="marketing.php">Send to Marketing</button>
</form>
```

Browser Support:

Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: IE 10+

HTML 5

HTML 5 Forms - Form Validation Override

In the spirit of giving the designer full control, HTML 5 introduced the "formnovalidation" attribute, allowing designers to turn off form validation for a given form. Like the "formaction" attribute, the "formnovalidation" attribute can only be applied to 'submit' buttons. When present, the form will be submitted without any validation being done by the browser.

HTML

```
<form action="taxes.php" ... />
Annual Income: <input type="number" />
<button type="submit" formnovalidation>Save For Later</button/>
<button type="submit">Submit to IRS</button/>
</form>
```

In the example above, we've created a form for users to submit their annual income. If they need to look up this information, they might want to save the form until later or add a note like "Need to get W2 from work". Rather than force them to complete the form in one sitting, we could allow them to save the form without validating the fields, only validating when they submit the final version.

All that said, you should never rely on browser validation as your sole source of validation as it has a number of weaknesses and can easily be sidestepped.

Browser Support:

Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: No
MS Edge: Yes
IE: IE 10+

HTML 5

HTML 5 Forms - The Reality

While HTML 5 offers us a number of new options for our forms, we must be sure to think of users who are using older browsers or browsers that may not support all of the new features/elements. Because most of the new HTML 5 form elements will revert to 'text' input fields in older browsers, we can use them knowing that users will still have a functional form. However, be aware of the fact that they may have a slightly different experience than other users and plan accordingly.

Similarly, it is important to remember that browser validation is there for the users' convenience, helping to inform them when they make a simple mistake. It is **NOT** there to provide security for your website! Similarly, if a browser doesn't support form validation (e.g., Safari), then the form will be submitted regardless of whether the user filled out all required fields, whether they submitted letters in a number field, etc.

All that said, the development of web browsers continues to move at a quick pace. The HTML standard is constantly evolving and browser support is moving just as quickly to keep up. It is your job as a designer to keep up with things as well. Here are a few websites to help you keep track of things:

- http://whatwg.org/html
- http://www.w3.org/TR/html5/
- http://caniuse.com/

HTML 5

HTML 5 - Video

One of the most significant changes from the early days of the web is the proliferation of online videos. Prior to HTML 5, there were very few options for embedding videos in a way that was easy, functional, and accessible. HTML 5 introduced the <video> element to address this need.

HTML

```
<!-- Simplest implementation of HTML 5 video -->
<video src="video.mp4"></video>
```

The <video> element, like the <object> element, can have nested elements for accessibility purposes. It can also be customized, often just by using the optional attributes provided by HTML 5:

autoplay	Should the video start playing as soon as it's ready? Because this is a boolean attribute, including it sets the value to true while leaving it out sets this to false.
controls	Should the video player controls be displayed to the user? It's a good idea to include this attribute, as users typically like to have the option of pausing, etc. Again, this is a boolean attribute.
loop	Should the video loop automatically, starting over after the end of the video is played? This is a boolean attribute.
muted	Should the audio for the video be muted by default? This is a boolean attribute.
src	The URL for the video file. In an ideal world, this would be sufficient for specifying our video source. However, as we'll see below, that is not usually the case.
preload	This is used to provide the browser with a hint about how the video data should be downloaded. If set to 'none', this indicates that the video should not be downloaded until the user clicks play or seeks (i.e., uses the video controls to move to a specific point in the video). If set to 'metadata', information about the video (e.g., length, etc.) will be downloaded immediately but the video itself will not be. Lastly, 'auto' indicates that the entire video file should be downloaded, regardless of whether the user views it. *Note that this is only a hint to browsers. Some browsers may ignore this attribute.*
poster	The URL of an image that can be used as a placeholder until the user plays the video or seeks. If not specified, then the first frame of the video will be used.
width/height	As with images, you can explicitly set the width and height of a video file. However, adhering to our philosophy that HTML is for structure, organization, and meaning while CSS is for guiding the display of our content, I'd recommend using CSS to specify the width/height of your <video> elements.

Browser Support:
Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: IE 9+

While most modern browsers support the <video> element, they don't all support the same video formats...

HTML 5

HTML 5 - Video

To address the issues of video format incompatibilities, the WHATWG introduced the <source> element which can be used by designers to specify multiple sources for a <video> element. This element has two attributes:

- src - The URL for the video file
- type - The MIME type for the video file (i.e., the format)

A browser will play video using the first <source> element that specifies a format that it supports. If no supported format is found, then the browser will display the content between the opening and closing <video> tags. We can use the <source> element with our <video> element to specify sources for the following video formats:

MPEG 4 / H.264 *(usually with a .mp4 or .m4v file extension)*
Based on Apple's older QuickTime video format, MPEG 4 (also known as H.264) is one of the most used video formats and is the preferred format for Safari and Apple mobile devices. Because H.264 is not royalty-free, some browser makers have argued that this format is not good for innovation and the future of video on the web. That said, these browser makers still support the format (while promoting WebM as their preferred format). In addition to Apple browsers, MPEG 4 / H.264 is supported in Chrome, Firefox, Opera, MS Edge, and Internet Explorer 9+. *This should be the first type of source you provide due to issues with older iPads ignoring everything but the first <source> element.*
MIME-Type: video/mp4

WebM *(usually with a .webm file extension)*
One of the most widely supported video formats today, WebM was originally based on the proprietary Matroska format. After some legal hurdles, Google was able to purchase a "perpetual, transferable, royalty free license" for the related VP8 codec, opening the door for WebM to be used by everyone for free. Because of this, it was quickly adopted by others and is now supported in Chrome, Firefox, and Opera. Internet Explorer and Safari (on desktop computers but not on mobile) can support the format through browser add-ons. Microsoft is also working to add support for WebM to their Edge browser.
MIME-Type: video/webm

OGG *(usually with a .ogg or .ogv file extension)*
OGG is an open-source video format, unencumbered by any licensing or patent issues. Well supported by older browsers, OGG has more recently been overshadowed by WebM, which provides a better compression-to-quality ratio. OGG is supported by Chrome, Firefox, and Opera. Safari (on desktop computers but not on mobile) can support the format through browser add-ons. Internet Explorer does not support this format and it is not currently being considered for support by MS Edge.
MIME-Type: video/ogg

In addition to providing multiple <source> elements, it is also a good idea to provide a Flash-based video player for older browsers (typically using the MPEG-4 format). Lastly, you can provide a link for users to download your video file if their browser simply can't play it. This approach will provide the greatest amount of cross-browser compatibility for your users.

HTML

```
<video controls="controls">
   <source src="video.mp4" type="video/mp4" />
   <source src="video.webm" type="video/webm" />
   <source src="video.ogv" type="video/ogg" />
   <object type="application/x-shockwave-flash" data="player.swf?file=video.mp4">
       <param name="movie" value="player.swf?file=video.mp4" />
   </object>
   <a href="video.ogv">Download the video</a>
</video>
```

If your server supports it, you can also specify the exact portion of a video that should be played by using a media fragment. For example, the <source> element below is requesting the video from seconds 30 to 45.

HTML

```
<source src="video.mp4#t=30,45" type="video/mp4" />
```

HTML 5

HTML 5 - Audio

Like the <video> element, the <audio> element allows us to include an audio file for our users to listen to. The <audio> element can have nested elements for accessibility purposes and can be customized using many of the same optional attributes used by the <video> element:

autoplay Should the audio start playing as soon as it's ready? Because this is a boolean attribute, including it sets the value to true while leaving it out sets this to false.

controls Should the audio player controls be displayed to the user? It's a good idea to include this attribute, as users typically like to have the option of pausing, etc. Again, this is a boolean attribute.

loop Should the audio loop automatically, starting over after the end of the audio file is played? This is a boolean attribute.

muted Should the audio be muted by default? This is a boolean attribute.

preload This is used to provide the browser with a hint about how the audio data should be downloaded. If set to 'none', this indicates that the audio should not be downloaded until the user clicks play or seeks. If set to 'metadata', information about the audio (e.g., length, etc.) will be downloaded immediately but the audio itself will not be. Lastly, 'auto' indicates that the entire audio file should be downloaded, regardless of whether the user listens to it. *Note that this is only a hint to browsers. Some browsers may ignore this attribute.*

src The URL for the audio file. Like with <video> elements, it is better to use <source> elements to specify the source of the audio in multiple formats to provide the best user experience across different browsers.

If you thought <audio> elements might not deal with multiple formats like <video>, then I have some bad news for you... There are three main formats for audio, each which have varying support depending on the browser you're using.

MP3 (usually with a .mp3 file extension) The MP3 format has a long history of legal entanglements and is considered "patent encumbered". That said, it is supported in all major browsers and is the most widely used of the audio formats. That said, some operating systems may require additional libraries to be able to play MP3 files.
MIME-Type: audio/mp3

AAC (usually with a .webm file extension) AAC provides better quality than MP3 while maintaining similar bit rates. While AAC doesn't suffer from all of the same licensing/patent limitations of the MP3 format, it, too, is "patent encumbered". Apple **loves** AAC. It's no surprise, then, that it's supported in Safari, as well as Chrome, Internet Explorer 9+, MS Edge, and Opera. Firefox only supports AAC when it's used in videos in an MP4 format.
MIME-Type: audio/aac

OGG (usually with a .ogg or .oga file extension) While free from any sort of licensing or patent issues, OGG is still not supported on Internet Explorer, MS Edge, or Safari. Chrome, Firefox, and Opera all support this open format without any known issues. Microsoft has placed OGG audio support "under consideration", meaning that it will likely be included in future versions of IE and/or Edge.
MIME-Type: audio/ogg

HTML 5

HTML 5 - Audio

HTML

```
<audio controls="controls">
   <source src="audio.mp3" type="audio/mp3" />
   <source src="audio.aac" type="audio/aac" />
   <source src="audio.oga" type="audio/ogg" />
   <a href="audio.mp3">Download the sound clip</a>
</audio>
```

Browser Support:

Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: IE 9+

HTML 5

HTML 5 - Video/Audio

To make matters more complicated, you can explicitly tell the browser exactly how your video/audio files were encoded by providing a codec alont with the type information.

HTML

```
<source src="video.mp4" type='video/mp4;codecs="vp8, vorbis"' />
```

While necessary in some cases, video codecs are beyond the scope of this course. More information can be found in the official HTML 5 standard (https://html.spec.whatwg.org/multipage/embedded-content.html#the-source-element:attr-media-src. HTML5 Rocks also offers a great tutorial on using the HTML 5 <video> element. Dive Into HTML also has a detailed discussion about video, audio, codecs, and HTML 5.

The HTML 5 <video> and <audio> elements also provide us with improved accessibility options by allowing us to include timed text tracks (i.e., captions, subtitles, descriptions, chapters). To include captions for a video, we would use the <track> element as follows:

HTML

```
<video...>
   ...
   <track kind='captions' src='captionFile.vtt' srclang="en" />
</video...>
```

The <track> element has a number of attributes available for us to use:

default Boolean attribute indicating that the given track should be enabled by default.

kind Allows us to specify how the track should be used

- **subtitles** Provides a translation of the audio from the original language to another. You must set the 'srclang' attribute as well in order to use subtitles.
- **captions** Provides a transcription (and possibly a translation) of the audio.
- **chapters** Provides chapter titles that are intended to be used when users navigate the media resource.
- **description** Provides a full textual description of the video, typically used by users who are blind or otherwise unable to see the video. Also useful for search engine web crawlers.

label A user-readable title for the track (e.g., "English Subtitles"). This is used by the browser when presenting a list of available tracks to the user.

src URL for the track file (should be .vtt format*).

srclang The language used in the track file. Values must be valid BCP 47 language tags which can be looked up at http://r12a.github.io/apps/subtags/.

Browser Support:

Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: IE 10+

* WebVTT is a text-based format used to create timed text tracks for use with the <track> element. For the sake of time, we will not be covering these in depth in our course. However, a good introduction can be found at http://html5doctor.com/video-subtitling-and-webvtt/.

HTML 5

HTML 5 - Inline SVG

Scalar Vector Graphics (SVG) use an XML-based format for displaying 2-D images. Originally developed by the W3C staring in 1999, this format relies on vector shape information rather than on a fixed set of pixels. The result is that SVG images can be scaled without the pixelation found in other raster-based formats (i.e., PNG, JPEG, GIF, etc.). Additionally, the XML-based files can be more efficiently compressed, resulting in smaller file sizes.

As part of the HTML 5 standard, the WHATWG introduced support for SVG graphics, both as the source of a standard <img> element as well as an inline <svg> element for embedding the full SVG data inside of the HTML file.

HTML

```
<img src="logo.svg" alt="My Logo" />
```

HTML

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
   <circle cx="100" cy="50" r="40" stroke="black" stroke-width="2" fill="red" />
</svg>
```

Browser Support:

Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: IE 9+

In addition to being scalable, SVG images can be easily edited (their just XML, after all) and can be transformed on-the-fly using JavaScript.

While SVG won't serve as a replacement for the JPEG format (SVG doesn't handle photographic images well), it will serve as a good substitute for PNG files and non-animated GIF files in many cases, particularly for logos and icons.

Here are a few options for creating/finding SVG images:

- Create them Adobe Illustrator
- Create them with Inkscape (free)
- Open Clip Art
- Google Image Search

HTML 5

HTML 5 - A Catch-All

While the term "HTML 5" has been used quite a bit, it's actually a misnomer for a number of reasons. First, the WHATWG has argued that there are no longer any versions of HTML. There is simply a living HTML standard that represents where HTML is at in a specific moment in time. Secondly, people have attached a number of non-HTML technologies to the term. Often times, people use HTML5 to describe the new semantic HTML elements, HTML form enhancements, <video> and <audio> elements, inline SVG images, or the implementation of ARIA roles. They might also use it to refer to the following features, most of which are more closely linked with web application development and usually require JavaScript:

<canvas>	While technically an HTML element, <canvas> allows designers to "draw" things using JavaScript. While that sounds pretty simple, this is a **very** powerful tool which can be used to render complex graphics, create interactive data visualizations, and even create web-based video games.
GeoLocation Services	Browsers now have the ability to provide your location to designers. This is helpful in allowing websites to do location-aware things (e.g., find the nearest store location, show my location on a map, etc.).
Web Sockets	For much of the history of the web, communication between the browsers and web servers relied on the HTTP request/response. This meant that the browser had to initiate the communication, asking the server for a file or some bit of information. With the development of robust web applications, this proved to be a bit of undesirable overhead. To address this issue, browsers now have the ability to connect to the server using "web sockets". This is a connection between the server and the client that remains open, allowing either to send data to the other at any time without requiring a new HTTP request/response.
Local Storage/Offline Applications	Browsers now have the ability to store data locally, preventing unnecessary calls to the server and allowing developers to create web applications whose state persists (even if the browser is closed and opened again!). Along with this, the browser has evolved to the point that applications can run when your device isn't even online, storing data locally and then uploading it to the server when an online connection is re-established.

And there are other technologies and specifications that have all been grouped under the HTML 5 umbrella, each providing a new step in the evolution of the web and the browser environment.