HTML Fundamentals Introduction

Introduction

Hello and welcome to this class on HTML Fundamentals. My name is Matt Milner and I'm going to be guiding you through this course. In this first module we're going to talk about HTML and how it fits into the World Wide Web. Talk a little about the HTML history and then focus on what an HTML document is and some of the root elements and things that we need to be concerned about. And finally we'll talk a little bit about HTML and browsers and how these two interact. As we look at a little bit of history for HTML we know of course that the internet was created by Al Gore. I'm joking of course, Al Gore was a big supporter of the internet and gave a lot of support to different elements that helped that along. It's an off misquoted statement, a little bit of humor to get us started here. But the reality is we have the internet however we got there that allows different computers to be connected. And this includes all sorts of things like TCPIP, DNS, name resolution, an IP addresses, all that good low level stuff that allows the computers to be connected. And it was all generated back for defense so they could connect their computers and share information. Now on top of that we had the World Wide Web which was essentially created by this man, Tim Berners-Lee and the Web, the World Wide Web is actually made up of a few different things. One of course is our focus here is HTML. And HTML is the language that we use to create documents that we want to share across the World Wide Web or more appropriately across the internet. We want to share these documents using this underlying network topology. Another piece of that is the notion of a universal resource identifying or universal resource locator, the URI or URL. You can see here down by our document we have an address for a particular HTML document. That's a big piece of the Web is being able to go out and use an address to get a particular document that we're interested in. And finally if we watch our HTML document we have the HTTP protocol, hypertext transfer protocol which is all about taking those documents and exchanging them at layers on top of the internet protocols and provides us a language for exchanging these documents. So HTML hypertext markup language, http the hypertext transfer protocol. Now Pluralsight has excellent courses already on TCPIP. We've got courses on http so if you want to find out more about those lower level protocols you can do that. In this course we're going to focus on HTML or that language that we use to create these documents we share over the internet.

HTML documents

HTML documents or the whole idea of the World Wide Web was created with the intent of being able to share documents. Essentially research documents that people had created and they wanted to be able to have text, have data in them, images, charts, those sorts of things and importantly they wanted them to be linked together so that I could create a research document and if I were referencing someone else's work I could just have you follow a link to that person's document on the World Wide Web and see that information as well. So it's a big piece of it. I wanted the ability to create these rich documents but also they should all be linked together, it should be a web of information. And HTML really is what's known as a markup language. We're going to create our documents, mark them up with these tags or this meta information and then it's meant to be processed by a client application. Today we think of that almost exclusively as a browser, internet browser like Internet Explorer, Chrome, Firefox, the Safari browser. And so what we're going to focus on in this course is creating that mark up and how the various elements and tags that we use are interpreted by the browsers to give us the document that we want to share with people. A simple example, we look at this markup here, see there are a lot of things that are really instructions to the client saying this is an HTML document, here's the body and I have an H1 or first level heading called semantic web with the idea that you're not going to see those tags or those elements, that markup, that's going to be used by the client in order to display your data. So we have the semantic web here, obviously this page has a lot more markup behind it but the idea is similar that we're going to provide this markup and a client is going to interpret it. If we look at an HTML document, we've got a very simple example here, we have at the top a doctype. And the doctype is a declaration in your document of which version of the HTML standard you're following. So here you can see we're following the HTML4, we have the strict in there. There are a variety of different document standards that you can follow including XHTML which is HTML4 but all documents are valid XML documents as well. We'll look at that in a moment. So the doctype is there and tells any client which standard we're following. We follow that by an HTML root element which is our single top level element that's going to wrap our document. All of our other pieces are going to be contained inside of that. We have the head and the head or header is really where we're going to provide a lot of meta data almost exclusively used by the clients or the browsers. So we'll provide information that's here, we've got the title, we can provide meta data, we can provide links to related documents be those style sheets, script, if we're talking about a document with many pages we might be linking to a previous or next page as well. So all that information can be used by the browser but with the exception of title you almost never see that information represented in the page. And finally of course we have the body, this is where we put our document data. This is what the user's going to see, this is where the bulk of our markup happens and where we focus on the information that we want to present to the user.

Demo: using doctype

In order to show you the use of the doctype I'm going to create a simple HTML document here, about as simple as I can get and I can put a head, a body. So this is just simple, we'll call this our hello world, and this is about as basic as we can get and you'll notice I have the HTML, the head and the body and if I now go open that you'll notice that chrome, the Google browser is perfectly happy to display that hello world, it understands that. If I open that up with another browser, say Firefox, it's also perfectly happy to render that. Now the issue here is that those browsers are having to guess at what standard I'm following, which version of HTML or XHTML I'm using here and they're having to render based on that guess. I don't know about you but I don't like my tools in software, my computer tools to guess. I want them to know what they're doing. And so one of the things that we could do with a doctype is we can tell the browser or the client this is the standard that I'm following. So if we switch here over to Chrome we look at the w3.org for the valid DTD list you'll see that we've got templates here and we can also go though and see there are a variety of different standards that we can be following. Now the most common are going to be the HTML 4.01 or the XHTML, those are the most common ones today. You can see there's a variety of others. HTML 5 is out there, it's not yet a standard but they do have a doctype that you use for that. And then there's some other things like MathML and other SVG related declarations. But if we take this same HTML now, just copy that, we'll paste it in here and we can see we've got a number of different items in here. We've got the title that's in there as standard so we can say hello HTML is our title and then we've got the body and we'll change that to our hello world, we can save that. Again of course the browser's perfectly happy, it doesn't look any different then what we had before but if we go out to the markup validation service, so validator.w3.org what we can do is validate by direct input and we'll take our content here, paste that in and check it and see it was successfully checked as XHTML 1.0.strict. And it's going to tell me that it's assuming that the encoding here based on what we've put in the tags. So what happens if for example we come in and take out the p and we revalidate. Well you can see now because it understands that we're trying to follow this standard, this XHTML 1.0 strict, it's going to give us an error and tell us that we can't put just plain text inside the body. We need to wrap that with some sort of element, the div tag or p tag or paragraph tag. And so that document type declaration is your contract with the client that says I'm following this particular standard, these are the items that you can expect from me. Now again there are many different standards out there we can follow or that we can use in terms of document type declarations. Even for HTML 4 you can see we have a strict, a transitional and a frame set. Likewise for XHTML. The strict means we're going to follow only those items that are actually in the standard and have not been deprecated. So there are several items that have been deprecated in there that you'll want to avoid if you're following the strict whereas the transitional or loose means that you're following it but you also might be using some of those items that have been deprecated. And frameset is when you're not just going to include a body, inside your HTML you're going to have a frameset or a series of frames each one pointing to a separate HTML document or containing its own content. XHTML is a revision of the HTML standard which follows the guidelines of XML. So for example, one of the things you could do in HTML is you could leave the p off in the closing tag and simply have the hello world. If we switch over to the validator and we try that we can put just the start tag in, we'll revalidate and see that we still get a warning, we still get an error that the end tag is omitted. So when we're using the XHTML this also has to valid XML which means that we have to have open and closing tags for all those items. So now I can revalidate that. So it's important to put that doctype at the top of your file. It lets the client know which standard you're following. It's also important to follow that standard to make sure that your HTML that you're sending down is going to be rendered properly. And that's what it all comes down to with this doctype. The browsers will do their best as we saw. It will try to render things even if it doesn't have the doctype but it's better to be sure to tell the browser this is a standard I'm following and then follow that standard. Whatever HTML editing tool you choose, as long as it's not something like notepad here, you will most likely get that kind of guidance from a tool, a visual studio or another HTML editing tool's going to show you where you have errors, where you're not following the document type that you've declared at the top.

The head element

The head element as I described is primarily about metadata, information that the browser will use but not necessarily display to the user. And one exception to that is the title. This is where you put the title of your document. And this element often gets displayed in the tab or the window bar of the browser. And also is used a bit in search engine optimization. So search engines as they're looking at documents will look at the title to see if it's relevant and to harvest that information. The meta tag which you can have many of allows you to specify metadata about the document. This might be key words or descriptions and things of that nature can also be the equivalent of http headers. So you can have information in there as well that might normally be at the protocol level. But typically you'll use this for those key words for information about the author and the description and those sorts of things such that the browser can use them but often they won't be displayed. Also have the script element where we can specify inline script, usually JavaScript to allow you to create interactive pages and manipulate your markup after it's been rendered. I have the style element which allows you to create inline style definitions using cascading style sheets. This allows you to apply to the body elements different styles such as positioning, colors, alignment, those sorts of elements. And we have a whole course on styles, cascading style sheets where you can get more information on that and we have courses, several different courses on JavaScript is you want to learn more about creating interactive pages. We also have the link element. This is where we can tell the client that there are other documents related to this one. And so the link will have both a relationship that might say it's the previous document or page or it's the next page or the contents. Often this gets used for linking in those style sheets and scripts as they've been created in external files. This allows those styles and those scripts to be reused across pages and gives you a nice separation of your markup file that has your data and information in it and your styling and interaction and scripting in separate files. So the link will often pull those in. We also have a base element. And this really provides for the browser an indication of what the base URI or URL is for any links in the page that are relative. So a link might say I'm a link to first.html. Well what's the base address for that particular item? If it's not specified here, the browser will make some assumptions based on what the current address of the document is but you can use the base tag to give it a specific base address you want it to use for all those relative links.

Demo: using <head>

In my document the head element is going to be important for me to provide metadata to browsers and search engines that are looking at this document. We have the title in there now, it says hello HTML, we can see that gets displayed in the browser tab. And that's about the only piece of information you're going to see out of heading that actually gets displayed but as I said the browsers and search engines are going to be looking to this information as they review your page. So one common thing to put in here is this meta tag where you often have a name, I could say author, and some content. We'll put in my name there. And that meta information is something that the browser could use but more likely you're going to find that search engines are going to use that for defining or determining what's on your page when they search and they provide their search results. So we could put an author and description in there for example. And you'll notice if we come down and we refresh in the browser nothing really happens, it doesn't change the representation in any way, it's just metadata that can be used in there. That one key piece of metadata that you might want to give is keywords. You put keywords here and for the content you can do some comma separated list of keywords that you want to include. Again, this is something that a search engine then could find. So rather then having to look over all of the content in your document to figure out what it's about, you can pull out those keywords to indicate hey, this is about Pluralsight, it's about the sample, it's HTML. Now there's many things that search engines use including the URL itself, the name of the file, some of the content primarily the H1 or the primary heading in your page. But the keywords can help there as well. Now we had before in the previous example an issue with the validation of our document because the validating engine didn't know what the content type was or what the encoding was. We'll just paste in here. Another thing you can do with the meta tags is instead of a name you can use this http equiv or http equivalent which says this particular content is the equivalent of this http header. So the http protocol level you would likely get a content type back. We're going to put it right in the document and say this is a text HTML document, that's the content type, and we're using the UTF 8 character set. So the meta tag provides those directions to the browser or to the search engines. We also have the ability to add in script, so we can put in script and we can say well this is JavaScript. So it's text/javascript is the type of this script and script allows us to come in then and do things like put a function in so I'll say there's a function for when the window loads and I can simply do an alert. That's just a simple example of script. We can do a variety of things. We can reference the document, change the document but if we save that and come run it we'll see now we get a JavaScript alert or message box that pops up. So script is one thing you can add in here. Like I said best practice is that often that script will be in an external file so you'll reference it with the script tags. So you could do something such as this and you could put the type in. You could also put in styles so you can do inline styles. Again we do the type, say it's a cascading style sheet and in here I might say I want to apply a style to the body and I could say that the color is red. So if we refresh that and we our script runs and our style is applied and so the color of our content is now showing up as red. As I said we've got courses on style sheets and script that will take you a lot deeper into those things but you'll see in those courses that you can reference these things either in the header or you can use the link. So I can use a link element here and I can provide a relationship. So let's say for the style sheets for example I might want to link to that and I'll say the href is index.css. If we look over here I've got a css file, open that up with notepad, we'll see that is has a body, the background color's set to black. So by linking that in I'm telling the browser, hey here's a related document, it's a style sheet, that's the relationship, here's the href or the address of that, go find that and use that and so now if I refresh the page our script runs and now the styles from within the document as well as those that were linked in with that link tag are applied. So we see the black background as well. The final thing we want to touch on then is the base element. And what the base allows you to do is provide a base address for all relative links or relative URIs within the page. So here I've provided a base address of http://www.Pluralsight.com and that now is going to be used as the base address for any relative references that I use in the page. As an example the css file we were just referencing used a relative address. And said relative to this page there's a css file. So here's our page, our HTML markup, here's a file right in that same directory with the index.css. That worked before but now we've changed that base address. We said well whenever you're going to use a relative address it's not going to be relative to the page, it will be relative to this base URI. So now it resolve out to the Pluralsight.com server/index.css. And if we refresh the page, we see our script in the page runs, our styles in the page still get applied, the body color of red, but because it went out and tried to find this file out on the Pluralsight servers and it doesn't exist there behind the scenes did a request for it, didn't find it and therefore none of those styles got applied. And we can fix that pretty easily simply by putting a comment around our base so we can get back to a working state. And this is how we apply a comment in HTML markup, use the less than exclamation and two dashes to start the comment and then two dashes and a greater than symbol to close out the comment. And if we save and refresh now you can see we're back to where we started. So the base can be helpful if you want to provide a common root or base for your relative addresses in the page so they all resolve based off of that as opposed to resolving based on the page that those links or those hrefs are currently on.

The body element

The body is where we're going to have the bulk of our markup and the bulk of our information that we want to display to a user using a client. And when we think about the body and the various elements, again remember that HTML was created to share research documents essentially or documents that you would find at say university. So we have things such as headings where we're going to call out the various sections of our document. We have various ways of representing text, both block and inline representations, quotes, preformatted text, superscripts and subscripts, a variety of different things that we need if we're going to actually create a document. We also have the ability to create a variety of different lists for listing off maybe the steps we've taken in a research experiment. All right, or items that we have in terms of details on a product if we think of how we use this today. We have links of course to allow us to connect one particular HTML document to another HTML document or somewhere within the document. So we think about a table of contents, we may want to link from an entry there to a particular spot within that same page or we may want to link off to additional HTML pages or documents on the same server or somewhere else on the internet. We have tables which of course were originally intended for displaying information and data. Think about again a research document where you have a table of information that you would display. Now these have been used and some would say misused to use or to create rather formatting in our documents. I'm going to steer you away from that and have you go to the CSS course we have and learn more about how you can use divs or different element for doing that. But tables are important. And we also have the ability to embed images or various objects within the page. So if you think about again research documents images might include pictures of the research that was being done, we might need audio or video embedded in there. Today often we see this becoming one of the largest parts of pages unfortunately as get advertisements on the side and various images popping up, various objects such as Flash, Silverlight video and audio things coming up within the documents that we view. But the idea is that the body is where we're going to put all this information over the course of this Pluralsight course. We're going to go into each of these different areas and look at headings and text, we're going to look at lists and links and tables and images and objects. We'll talk about the various elements within HTML, how you can use them, how you shouldn't misuse them and pull all these things together to create this HTML document.

Using id and class attributes

We're not going to spend too much time in this module talking about the various elements in the body because that's really what the rest of the course is about. But I do want to focus on two attributes that you'll apply to many body elements in your documents and that's the id and the class attribute. The id attribute allows you to identify a particular item within the page. Now you might use this for scripting or manipulating that object with JavaScript or another type of script. And you might also use it for styling. So using style sheets you can reference a particular item by its id. That id needs to be unique for the page. So as you go through and create ids for elements on the page you'd want to create unique ids for those items. The class is something that also allows you to identify items but with the class you could have many items on the page that share the same class. Often the class is used to apply styles but it can also be used in script to go and find all the elements on the page with a certain class. So a couple quick examples for the class for example, I might indicate in the style sheet that anything with a class of message, so dot message here, I want to change that to green. And with the script rather then popping up an alert when the document loads we could do something like this where we do document get element by id and we'll use that id attribute then and I set the inner HTML. So instead of hello world it would say hello JavaScript since we're manipulating this from script. So if we save that, open it up in the browser you'll see now that the text is hello JavaScript because it was written over by that text and it's green. So rather then the body color of red because that particular text is also in this class of message, its color's been changed to green. So you'll find the use of id and class critical as you start building out more involved documents where you want to apply styles and work with those things in script. And it's a good idea to apply those things to give your documents and those elements within there's some semantic meaning and unique identifiers with the id.

Summary

In this module we've looked at how HTML or the hypertext markup language allows us to create documents that we can share over the Web using the http protocol. We've seen that they have unique URIs or URLs that we can use to address a particular document and we'll talk more about those when we get into linking. And I've provided a few links here for you for tools that we've used in terms of the validator so you can take an HTML document either as a file or paste in the text or point to URL and validate that it matches with the DTD you've provided. And also a list of the valid DTDs that you can choose from and easy way for you to copy and paste those if your HTML editing tool doesn't provide that. We looked at the doctype, the head and the body, those core elements of an HTML document. And as we move through the course we'll see all the various things now that we can add to the body to create our content, to provide visual cues to people about headings and lists, tables, charts, graphs and link these things together.

HTML Text

Introduction

Hello and welcome to this module in the HTML Fundamentals course, focusing on text elements in HTML. My name is Matt Milner. And I'm going to guide you through this module. Well the module is about text elements. When you think about HTML documents, primarily they were focused on text when originally intended. Now, today when you go to a website you end up seeing a lot of images and flash videos and all sorts of things that aren't textural. But the focus of HTML was originally documents, and document text. So we'll be talking about things such as headings, but also things like block and inline elements, or this notion of "As I'm adding things to the document, how are they going to be handled and rendered within the document?" Some of those things will contain text, but they may also be generic containers for other items. We'll talk about text breaking and white space a little bit, how you can control where things break or where they don't and formatting. And then we'll talk about a variety of elements you can use to call out specific text or maybe you're using abbreviations, quotations, you're referencing other things, or you're including things like code snippets or keyboard input that you want somebody to type.

Headings

We'll start our discussion of text by looking at headings. Heading are the way that you're going to break up your document and identify the various sections within that document, to call your readers' attention to the various chunks of information that are present. Typically, you'll start by using a single H1 element, somewhere near the top of your page often. That's the primary heading for your document. This calls out what the intent is, what the primary focus is. It's also used by search engines. So, if you think about it, if you're searching for something, and that search term happens to show up as the primary heading of a document, chances are good it might contain information that you're interested in. But we also have headings that go all the way from H2 to H6, so we have the various subheadings for those other sections of your document. And you want to consider the semantics as you do this, because you don't necessarily want to have a Heading 2 and then underneath that have a Heading 1. All right; you want to -- going to look at the structure of the document and use those headings to break it up appropriately. As I said, you probably want a single H1 on the page. That's a good practice, and it's also something that will help with the search engines.

Demo: Headings

What I have here is a simple HTML document, and in the body just a representative example of these headings. So you can see the H1 element here identifies the first level heading, and I close that out with the closing tag. And if you look in the browser on the right, you'll notice that they get rendered out in various sizes based on the heading level. This is pretty typical for all browsers. And what I'll caution against is that you don't want to choose a heading tag or an H tag here in order to get the right size of font. You want to use styles for that, you want to apply some stylistic font sizes and font styles to your text to get it to look the way you want. Use the headings really as an organizational piece and use it for the semantic meaning in which it was intended. Well, let's clean this up a little bit. So from a H1, I'm going to say this is my HTML fundamentals. So I'm going to create a page all about the course here. And then as H2 I might have an overview. And underneath that we might have something like objectives for our learning objectives. And maybe we want to have a section here on the outline. So we'll get rid of a couple of these. I don't necessarily need those. Now, I think the outline in the objectives are really all part of the overview section. So I'm going to change the outline back to an H3, and have that come in as the same level as the objectives here. So we'll save that. And now I have an outline of the structure of my document. So I'm going to have an overview, underneath that I have some objectives and an outline that I can create it, plus the use of the headings to really layout those various sections within your document.

Block vs. Inline Elements

As you begin working with your document and the various elements you can use to encapsulate content, it's important to understand how the browser is going to interpret based on the specification those various items. Some items are going to be considered block elements. Those are containers for grouping things together, maybe just text, or a variety of items. You could have a group that contains a whole section of your document. Those block elements can contain other block elements and inline elements. So as we get into the examples you'll see use of things like the div tag, which allows you to create a division of your page or a section of your page that you want to wrap in a container and treat as a single unit for some purposes. You also have the P element, or the paragraph element, which allows you to specify a chunk of text or other elements that you consider a paragraph. So with the example the two top level block elements are paragraphs of text, but they might also contain links to other documents, or a variety of other inline elements. The bottom example is a quote, block quote, which is rendered here as a block of text as well. Now, there are also inline elements. These are containers for text and other inline elements. So in the examples you can see highlighted here we may have things like acronyms, we may have a span of text within a paragraph that we want to do something with or treat as an overall element. So where you see, "HTML drives the web" highlighted, that span of text we want to be able to treat as a single unit and do things with it, maybe it's apply styles or script, or something else. The key with these is that block elements can contain other blocks as well as inline. Inline elements really are just containers for text and other inline elements. So you can't have, for example, a span tag, which is an inline element, that contains a div, which is a block element. Let's look at some examples of these various tags that are considered block and inline elements for laying out your document.

Demo: Block and Inline Elements

Let's look at some examples of using block elements versus inline elements. The div is a pretty common item that we'll use, so I might put this in and have this be my overview container, and move it in a minute to have that make more sense. And I want to close that tag as well. We can put in here, "This is my overview." And if I put another div in here, we'll just call this, "second div" for now. We're going to put it in and we could say, "This is another div." And we'll save that and re-render, and you can see here's this is my overview and this is another div rendered on different lines. Now, you might think that's because I put them on different lines in the source, but in fact as we'll see shortly when it comes to white space, the browsers don't really care. I'll put the divs on the same line and refresh, and I have the same output. These are block level elements. This particular element, the div, by default is going to be treated as a block level container and it's going to render, and then whatever fall it is, it's going to render on its own next kind of logical line or space in the document. It's not going to show up in line with that. Now, by contrast if I put a span, which is an inline element in here for my overview -- we'll this up a little bit because the white space doesn't matter. Then we re-render, noticing no changes here. This is an inline element, so this is going to cause overview to be logically grouped under this span, but because it's an inline element that overview renders right inside of the div. We switch that now and try and make that its own div within there. And we'll even put another item here to spice it up a little bit; save. Notice now what happens. "This is my" goes on one line because that's the first part of our div, but now we have a new block element here, this div, and so that then, the overview, is going to show up as a new block. Likewise, the content finishes out there as a block element. So keep that in mind as you're developing that these elements, like the div, are by nature block elements and now they're going to render in that particular way. Another example of a block element is the P, or paragraph element. And we'll put some text in here, just a long string across there. And we'll hit "save" and "reload." And you'll notice now that that renders out as a paragraph of text. We copy that. We get two of those and as we refresh, notice we see the same kind of rendering where we have these things blocked in their own blocks. It doesn't continue on the same way, but also since it's a paragraph, notice that the spacing is a little different. When we have the div, we're getting this kind of single type spacing. When we have the paragraph it's going to render it as a paragraph. Most browser will do this, a paragraph of text with familiar spacing for the rendering and the reading of those paragraph elements. And one of the things that you might think about doing is saying, "Well, I have this big chunk of text to this paragraph and I want to change that up." Maybe I want to wrap something in here. I want to wrap this piece in a div because I want to call it out in some way. I want to wrap it. And we'll save that. Now, the browsers that we've shown previously will do their best here to try and help you out. So you could see because it's a div it got rendered as a block on its own line. And we see a break there. It's just a little weird. But if we take that content, save that, and we go to the validater, the w3.org and we paste that in and check it, we're going to get an error based on the specification. And if we look down here, we'll see that it doesn't really like our div inside of a p-tag. Now, the paragraph element is technically rendered as a block or similar to the blocks. But this is a case where the paragraph or other -- or inline elements such as the span can't contain block level elements like div. So while the browser may in fact render it, may even get it to do what you want, you shouldn't do that. You want to follow the specification, you want to make sure that you're creating valid documents that's going to help you make sure that those things get rendered correctly in all of the browsers that come and do it. And you don't -- you don't go on the whims of the browser vendor and how they decide to render those strange things you're doing that don't follow the spec.

Whitespace

Another important aspect of working with text in your HTML documents is understanding how the browsers are going to deal with white space in the markup, and breaking your text across lines. Now, generally browsers are going to ignore white space in your documents, in your blocked elements and in your inline elements. So even if you have ten space between a word, it's going to show up as a space. If you have ten line breaks in your markup between your H1 and your H2, they're still going to show up one right after the other. You're not going to see those carriage returns in the source marked up. What if you do care about the white space, though? What if you have a poem that you wanted to use, or you've got some other kind of text that you want to actually format with white space, whether that's things like indentations or extra space between items, you can use the pre-element, or preformatted text. When browsers see the pre element they will respect the white space that's inside that content of that text that you contain within the pre. We also have the notion of the break tag, the BR, which gives you an explicit line break. So if you have some text or some elements that are in line and would normally flow across, and you want those things to break apart, you actually want a line break, you can explicitly use the break element to ask the browser to put a line break in there. You can also use the HR or horizontal rule if you actually want a visible line to show up. Now, this was all trendy back when HTML first came about. These days most people will use images and style sheets to render any kind of visual breaks that you might see. But it is one option for identifying that horizontal rule, and you could certainly style it to look different ways. And the final item are character entities. So these are representations that you put in your markup. They use this syntax of the ampersand, some characters, and a semicolon, and these character entities tell the browser to render a particular character or to consider this entity as a particular character. A popular one is the non-breaking space. This allows you to put a space in-between two inline elements, for example, and instruct the browser that it shouldn't break at that point. I don't want you to break this line apart here. So if you're wrapping text, if you're wrapping elements around because the window is smaller or larger, then don't do that breaking here. Keep these two elements together. Also, if you want to represent anything that would normally be interpreted by the browser as HTML markup, so in our markup we have the less than and greater than symbols all over the place around our elements. So if you actually want to show a less than sign, then you need to use the ampersand LT semicolon or the less than character entity, or for greater than we have a similar item there. So you use these character entities in order to render characters that would normally be interpreted as markup.

Demo: Whitespace

In my document I've fixed things up a little bit to use my block items correctly. So I now have a div called, "overview container," that contains all of the content in my overview. I might use that later in the course in terms of styling or scripting. Maybe I want to make that section collapsible through some script. So I've identified the whole section to allow me to treat it as one element and collapse the whole thing visually. And I have my paragraph here, and my objective headers. And then I've also got a new div down here that I'm going to use for some examples. So let's look at the white space. Let's take our paragraph from up top here and we'll put this down in an example. Let's go in here and we'll put an H2, and we'll call it examples. And now in here we'll come in and this will be H3, so our example is white space. And we'll call out that particular part of the document. So you'll notice all of this paragraph is on one line. But what happens if I do this, if after every sentence or section here I put some carriage returns in the source and save it? When we come down and look at the rendered content you'll notice it's still all rendered as a paragraph. Those carriage returns are completely ignored by the browser, and it's really rendering based on these elements. It sees that this is a paragraph. It's going to take all the content within there and render it as a paragraph unless it sees some other HTML element that instructs it to do something differently. So even if I put a ton of spaces between these browser names, you'll notice, again, I get a single space. So the white space is considered something that is not going to be preserved in these cases. Now, let's go down a step further and let's take an example for this where we actually want to have some content that is preformatted. So I want to put in here some content that I want the browser to display, and I want it to display it a particular way. So I'm going to stick a little haiku in here. See if I can remember my haiku, and notice that I have spacing here so that the first and last lines are indented and the second one is not. Then you'll notice that when we do the pre, that information in the browser normally gets rendered with this type, this particular typeface, and it now does respect that white space. So if I put a bunch of spaces in here, those get rendered out now as visible white space within the document. So if you want to -- the white space to show up, if you care about that, use the pre-element in order to handle that for you. So we've got that. Now let's go in and look, if I do a span and do another one here, oops, we know that these are inline elements, and so if we look at them, they should appear right next to each other. They're inline elements. Even if I go in and put those on a second line, they're still going to show up. I do now the carriage return does end up adding a space in there in this particular browser. But I have now all of this information. Now, I want to add some breaks. I'm going to add some breaks because I want to actually have some white space at the bottom of this document so every time I refresh it, it's not at the bottom of the page. But what if I actually don't want those on the same line? I can use the break for that. I put a line break in, I save, and I refresh, and now we're going to see an actual line break in the document. So even though these are two inline elements that would normally render on the same line, we can tell the browser, tell the client, "Please add a line break here at this particular point." Now, what if I want to separate these examples? Well, that's where the horizontal rule comes in, where I could put that in, and refresh. And you'll notice now I have a nice line above this first span. Second span I could do the same thing here between my white space items here, and get that line to separate those items out. So, again, kind of old school, but you could style that or make it appear different ways or use images to get the same effect. Finally, I want to use some character references. So let me go back in here and we'll follow our pattern here. And I want to identify now some content. I want to say maybe Div. This is how you write a div. And what I want is actually to show this content in the page. I actually want you to see the div element here. And so let's do it like this. I want to do div. But you can already tell from the syntax highlighting in here that the client is going to expect it to be a div, and it doesn't show up. So I need to come in and use my character references. And now those items actually get rendered out. I can actually see the less than, the greater than sign around the element. The browser knows that I want to render these character entities, not that I want to actually use a div within the markup. And finally we'll look at the non-breaking space. So if we look at this paragraph, notice that it shows up in my browser, and as I shrink this, if you watch, as the words get close to the right-hand side, they begin to wrap and go around to the other side. So let's say I don't want it to break in some particular spot. So maybe I don't want it to break between this and course. So I'm going to put a non-breaking space in there. And we'll refresh this. So now as we go, you'll see word as I expand goes off the page. But notice here course doesn't go off the page, this end course goes off and onto the next line. So that non-breaking space tells the browser, Do not break at this particular point. You could break at any other logical point, all right, left or right of this in the stream of text, but do not break at this particular point.

Additional Text Elements

I'm going to wrap up this section on text by talking about some availables and even some common elements that you can use on the text in your documents. We have the super and subscript tags that you can use. Think about HTML being built for documents, research papers. This is pretty critical to be able to have superscripts and subscripts, whether it's for creating notation of equations and having powers, or for calling out footnotes or endnotes that you want to allow somebody to go reference. I also have the cite element, which is simply a wrapper around a person's name or some text, where you want to cite work by another person. And so typically this will get rendered out in the browser as some italicized text. We have the abbreviation and acronym element. And these allow you to use these items, these acronyms and abbreviations in your document, but also to provide the expanded text that goes along with that as an attribute. This allows somebody usually to hover over the acronym or abbreviation you're using, and then we able to see the expanded out text. So it's very helpful. You know, I remember from my English classes it was always recommended that the first time you use that acronym or that abbreviation somewhere that you also pair with it the full name. So these allow you to do that. We have the emphasis and the strong elements. And the emphasis is typically rendered as italics and strong as bold. And these are some tags that were originally there and typically today you would use a style sheet if you actually want to have text get rendered with a particular font style in your document. So you might make something a span and then use style sheets to apply that there. Got a few others for code and samples, as well as keyboard and variables. Again, this was built by techies, so documents that you're creating that might have code or might have sample output, or if you have keyboard input or variables that you're specifying, you can use these elements to call those out. Typically the browser will render those often like it does the preformatted text in terms of the font face it uses. But generally won't pre-format or do other things with it. Then we have the quotes, so block quote, we saw an example earlier where you'd have a chunk of text. And you want to have it rendered out as a quote. Typically, that becomes an indented block level element with all the text in it. And the cue element is for inline quotes; so more of an inline element, much like a span, if you're in the middle of a sentence or you just want to call out that somebody is being quoted and put that text inside the cue element. Now, with these the block quote and the quote, you can also specify a URL to tie into where that citation is coming from. So you can indicate that this particular quote is coming from a particular document out on the web and give somebody the ability to reference that information. Depending on the browser, that may or may not render out in any discernible way, but it is good to provide that reference in there as you're quoting someone else's work.

Demo: Additional Elements

I've expanded my overview document here for the HTML Fundamentals course. And I have a paragraph that describes the course and provides the overview. Now, in there I use things like HTML and IE for Internet Explorer. On another paragraph I'm talking about CSS and HTTP. These are all things where I want to call out that these are acronyms or abbreviations. So I use the acronym element here to wrap up HTML. You could see I'm able to apply the title attribute and call out that what that stands for is hypertext markup language, or the abbreviation tag similar concept, I call out that IE is an abbreviation for Internet Explorer. So we look at that, what that looks like. We render here; you'll notice that HTML just shows up as part of the text, that it doesn't render any differently. But if I mouse over I see the expanded title for hypertext markup language, likewise for the abbreviation. I see that says "Internet Explorer." CSS, (inaudible) style sheets, and HTTP as hypertext transfer protocol. So it gives me an easy way to use those abbreviations and acronyms, but also provide for people the expanded title of that particular piece of text. Now, another item I have down here is in this particular paragraph I talk about the fact that we don't cover certain topics because they're covered in other courses, but I'm going to later want to create these little links through a footnote or citation or a note citation here to those courses so I have a little one comma two. And I put it in the superscript tag here. So as we go look, that gets rendered out. You can see raised up and shrunk a little bit above that as a superscript for those items. And I could within that superscript then add links and other items within that text if I need to. And I have some quotes and some citations here. So I have a little blurb here where I'm saying this is some history about HTML, and I want to say that Tim Berners Lee is credited with creating the world wide web and HTML. And I'm going to cite Tim Berners Lee; so I just wrap that up, and typically that renders out in italics, where then I'm going to also provide a quote from him. So I use the blog quote. So here's a chunk of text and I want you to do block quote formatting on it. Then again the white space we can clean this up a little bit, and you'll see that the white space is going to get ignored as it does the block quoting. But for the citation really you're just calling out a person. This is primarily just informational that you're citing a particular person. And then the block quote actually has the cite attribute where I'm then going to point to the document where I got this quote. And this does in fact point to the document. It was authored by him, but draft around the world wide web initiative. And this quote was pulled from there. So I provide that as the link. So if we render that, again, you can see that the block quote is just going to render based on the wrapping. But it does get indented differently than a paragraph. So it is called out and you kind of notice that it is a quote. The citation simply gets noted here by changing the font face or the font style to an italicized version. And you'll notice that if I look at these things there isn't really any indication of the URI. So I wanted to be, I don't know, transparent, I would probably put a footnote link or some sort of a way for people to actually link to that document. But it's good to put those citation attributes on there so that in your HTML you have tied to that citation or that block quote a particular URI so that somebody who's maybe not using a browser but that's following this, or a search engine might be able to see that block quote and see that you have a citation to another document and help you and help other users navigate the web. Now, some of these items, as we've seen, are useful in providing us this information. So the acronym is there. One of the things you may have noticed, though, is there's no visual indication that mousing over this might actually be helpful. There's nothing that tells me that that's an acronym until I mouse over it. And so this is where we get into kind of leveraging other things outside of HTML proper into maybe using style sheets. So I'm not going to use styles extensively in this course, except to show some particular examples. And so what I have in the header here is a style for these things. And I've commented out. And this is another important piece of using text within your markup, that if you have a block and you do the less than, exclamation mark and then two dashes, that starts a comment. And then two dashes and the greater than symbol completes it. You can say everything is grayed out now because it considers it part of the comment. And that closes my comment block. So I'm going to lose those. And I'm going to add the style in; and what this says is, "Anywhere you see an acronym element, color it green." So I'm going to save that. We'll change that; and now you can see that those acronyms are called out. They're green so now I think, "Oh, I wonder what that is," and I'll mouse over it and get the example. Notice that because IE was an abbreviation, it's not called out, but I could do similar things with the abbreviation as well, or apply the same or similar style. Likewise, I have this highlighted applied here so we can just see as we drop back to the initial where we talked about block versus inline, my span of HTML drives the web has a class of highlight. And I've said anything that's set up as highlight should be underlined. So we can see that getting applied here as well where we're using those text elements to within a paragraph call out a particular block of text with a span, and do something with it. In this case I applied a simple style. You might change the font or the color and a variety of different things you might use that span for, but it allows me to specify that little chunk of text and do something with it without changing the flow of the document and continuing to allow that paragraph to be rendered as it was intended.

Summary

In this module we've seen that we can use headings to break up our document into logical and visual sections. We should use that H1 as our primary content identifier. We learned about block versus inline elements and how the browsers treat those, and how we can nest them or not nest them. And we also saw that in a lot of cases you have to be explicit about your white space; whether you use the pre-element to get preformatted text, you're using line breaks or horizontal rules to get those breaks. You want to be explicit about that to make sure that all browsers render your content correctly. And we also saw a variety of tags for rendering your text correctly or correctly marking up your document to identify citations and quotes, to use super and subscripts, or otherwise call out text that fits into your document in a way that's not simply a paragraph of plain old text.

HTML Lists

Introduction

Hello and welcome to this module on the HTML Fundamentals course. In this module, we're going to focus on the various types of lists that you can use within your HTML documents. A pretty short module, but we're going to look at the three different list types that are available and also how to use those lists appropriately, how to nest lists, how to format them in your documents as you list out various items.

List Types

There are three basic types of lists in HTML. The first is an Unordered List, or you can think of it as a Bulleted List. We're going to have a number of items and I simply want to list them off, have a little bullet or some sort of indicator off to the side. And next is the Ordered List, very similar to the Unordered List, except that the items are generally rendered with some sort of numeric or Alpha label giving them a sense of order as they go. And finally, we have Definition Lists. This is a notion that I have a series of terms and related definitions that I want to render out. I want to tell you what HTML means. I want to provide you a definition of HTTP. Pretty common in a research document, for example, where you want to provide that list of data or words or concepts and then provide a definition along with it. Now with each of these, there's different rendering in the browsers as to how to handle these, and we'll also look at how to apply styles to the list, which is the recommended approach now in order to handle the look and feel.

Demo: Creating Lists

In my course overview document here, I have this section called, objectives, where I want to create a notion of the objectives for the course. So let's go into our document and we'll create an Unordered List or use the UL tag here, make sure to close that. And in Unordered and Ordered Lists, the items are called out with an LI or List Item tag. So we'll go in here and say we want to explain the fundamental concepts of HTML. We'll close the List Item, and we'll add some others just so we actually have a list. ( Demonstration ) And if we save that list and refresh, you'll see that, that gets rendered out as a Bulleted List with those various items on there as block-level items. So even if I put these things on the same line -- we think about the rendering from previous modules -- you'll see that as I refresh, they still end up as a list on their own separate lines. It's really looking at that markup not the white space in the flow that we have there. So it's a pretty simple Unordered List. Let's go back now. And under our outline, let's create an Ordered List. So here we're going to use OL for the Ordered List. And here we want to create a list of items that have a sense of order, which means they're going to have tags in the front that are going to provide numeric or Alpha sorts of order. So we still use the List Item. So we might put an introduction. We think about the course. We talked about text elements. And now we're talking about lists. And then we'll just put one more in there and, for that, we'll say we're going to talk about links. So we save that. And now we say we have a notion of a list and the default is that it's rendered out based on numeric prefixes here. Now with both of these types of lists, we can create sublists under here. So we might, under the have fun, we might put another list item underneath there and we could say, creating documents, close that out, and oh, we'll get cheesy and creating memories. Now right now I just have these as List Items, but really what I want this to be is a sublist. So I'm going to put another UL tag there for an Unordered List, and we'll nest that in there. So underneath, have fun, as a List Item, notice we don't close that List Item. We consider this other Unordered List as nested underneath there, and then we close the have fun. So we'll save that. And now you can see we have a sublist that's indented a little bit more for the rendering. It's changed the -- bullets still looks slightly different. So we can see that, that is a sublist underneath there. And we could do the same thing for our items here, if we might come into lists. We'll open up that List Item, and let's do another Ordered List in there. And so we can do a List Item; Unordered Lists, Ordered Lists and Definition Lists. So now we have, under here, the list element. Again, we don't close it. And then we have inside of that an Ordered List with its own List Items and then we close the list element. So now we get another sublist in order using those numeric characters. Finally, if we want to look at the notion of Definition Lists, then we can create a new list out here. I won't make you watch me type this one. We'll paste that in. So we have a DL or a Definition List. And in here, I'm going to have a definition term with a DT tag and then a definition, definition, which I realize is kind of redundant, but that's the definition of that particular term. Let's put a little header over here. And we'll call it terminology--save that. And I will go back out to our browser and, if we look down at the bottom, you can see there's our header. And then the way the Definition List is rendered is the term is rendered kind of left justified, and the definition then gets indented underneath that in a block-quote style. So if I take this and shrink it up a little bit, you'll see that it wraps and keeps that justification of that block style of the definition under there. So we have Definition Lists. We have Unordered Lists and we have the Ordered List, and we can nest those Ordered and Unordered Lists to give us those sublists as well.

Demo: List Rendering

One of the things that you may have noticed with our lists is that they have some default bullet and numbering styles. So we have a solid circle here for the Bulleted List, and then, when it indents, it uses a disc or an empty circle. And then under the numeric, we're just getting numeric characters here and they're restarting as we go to the inner list. So you might want to change that. If we look at the Unordered List, I'll show you the original way to do this, and this is deprecated. I'm going to show you the new way to do this. But the type might be that you would put a attribute on there and say type equals square, and then for the inner one, we could even do type equals circle, and there's also type equals disc. And essentially that lets you control these. So you can see now this one is square and this one is circle. If I change that to disc, now we get the filled-in disc. So you can control those with that element and likewise with the numeric lists. If we come in here to the Ordered List, for example, and do type, if we do a capital A, that means it's going to do capital Alpha indicators. So we can see we've got a capital A here and we could go to the inner list and we could do a type equals a lower case A. Now you can see that, that one has the lowercase letters, as the indicator's off to the side there. Now as I've said, this is the way that you originally did this. It's since been deprecated. So I'm going to take out the type attribute, because when you think about it, this type information really is stylistic. It's all about the display and the style of your items. So the preferred way now is to use style sheets. So I might say the class for this one is with square. If we look up at my styles, we could see it's actually square; so an Unordered List with a class of square. And we use this list style type now to call that information out. So a lot of the same information. You'll see I have squares, circle and disc. And then for the Ordered Lists, I have this with Alpha, with Roman, and there the list style type matches up when I have, for example, lower Alpha or upper Alpha. So if we go back to our lists -- let's come back here. So it wasn't actually with square. It's just square. And I could do on my Ordered List then class equals with, and we'll do upper Roman on that one. And then for this one, we'll do class equals with Roman to get the lower Roman case. So now you can see we've got these items going in here. So the outer ones are upper Roman, the inner ones are lower. It looks like I probably had a mismatch on my casing there. So now if we get that right and it matches the style, now we get squares off to the side. So that's your model for indicating the styles for the various lists, assign them a particular class and then apply the styles through a style sheet to go in and indicate what kind of list style you want. Now there's one other item with our lists. As we look at the outline, and we have the Ordered List, we may want to have some control over the numbering of these items. So for example, we have the Ordered List, we have introduction as one, text elements lists. Let's say, for example, that we didn't want this Unordered List to start with the first character. We could do start equals five and indicate that we want to have it start with the fifth character. So if we look right now, it starts with I. If we refresh, now it starts with V, for the Roman numeral for five. Now this is where things get interesting, because the start tag is available and there was also a value attribute that you could set. So you could change things around. Both of these elements are technically deprecated in HTML 4 and X HTML in the strict document type definitions. I happen to be using the loose or the transitional items here; and so I am allowed, by the specification, to use those items in here because I've said I'm using the loose or the transitional and that means -- because we know from our first module -- that it's okay for me to use deprecated items. The browsers are going to know that I might use those things. HTML 5, now brings these things back in. So these aren't necessarily stylistic. They're more about the order itself. And so you'll see those being available to start the list or to give a List Item a particular value.

Summary

In this module, we've seen that we can use lists to display organized information, either in an Unordered or Bulleted fashion or using the Ordered List using Alphas, numerics or Roman numerals to organize our elements. We also saw the Definition List for laying out different terminology for definition terms and the definition data. And we looked at the styling of lists to more appropriately apply the numerics or the characters used for bulleting rather than using the deprecated type information.

HTML Links

Introduction

Hello and welcome to this module on HTML links and anchors, it's part of the HTML fundamentals course. My name is Matt Milner and we're going to walk through connecting these HTML documents together. When we start to connect documents together we're going to use the notion of anchors. Anchors provide both a jumping off point or starting point for link as well as a target for links. So we'll see how to link one document to another but we'll also see how to link within a document or link from one document to a particular point in another document not just the start of the page.

Link concepts

The whole promise of the worldwide web is to have this web of documents that we're all connected together. So we need to be able to link from our document to other documents and we need to be able to link to our document from other documents and ideally we need to be able to link within documents as well. Especially as they get lengthy. Anchors are the key way that enables us to do that. Now an anchor can either be a source or a target for linking. You think about your experience on the web a source is one of those clickable links something you can engage with and have it then take you to some other resource or some other item on the web. A target is a thing you're actually trying to get to. The way we represent this in our HTML is we use the A or the anchor tag and we can use that with a name to specify a target. This is a location within a document where you can link to. If we use the anchor tag or the A tag with an H ref attribute where we provide a URI to a particular document or a document plus another anchor name then we use that as the source that's where you'd see that clickable link by default in your documents that you're viewing. When we create these links so we're looking at our HTML document in our example that's our large orange document here in the middle we want to be able to link to other documents maybe within our own website or somewhere else on the web. Those links then when we create them can either be absolute or relative. Now an absolute link is where we provide the absolute full path to a particular HTML document. You can see in this example we're providing the protocol with HTTP and the server name and the actual path to the document here for page one HTM that's going to allow us to go reference that page on some other server. But if we assume that we have these other files, file one and file two here on the left in a content folder then we can use a relative path such as this which is relative to the current document I'm in. The reference is I'm starting in my directory I'm going to look in the content folder and then find this file one dot HTM. So we have absolute which you would use if you're going outside of your site and relative paths which you'll often want to use in your own site which allows you to pick up your pages move them to different servers and have them located in different places they're not then dependent on where they're hosted. Now when we create those anchor tags we certainly have the URI but we also have the representation of the link. Is it some text that I can clink on, is it an image, whatever we want it to be that's what we include inside of the anchor tag. Its content becomes the source, where we can clink on that and follow the path. The note about relative URI's remember from the initial talk when we looked at the head element we do have a base element that can go in the head of your document that provides the base path for any relative links. So if you didn't specify a base, everything would be relative to the document. If you specify the base element any relative paths are going to be relevant to that or relative rather to that base URI.

Demo: Linking Documents

In my document here my outline you'll recall we've got the outline of items and what I want to be able to do is for the outline in here I want to be able to create links to additional content, to pages that go over some specifics. So for the list and the text elements I want to point to specific pages for those. Those specific pages happen to be in this content folder under lists dot HTML and text dot HTML. So let's go in and create our link. So my anchor tag I'm going to provide an H ref and that's going to be in content, lists dot HTML and I'll close the anchor tag. So now I have this going to the content folder lists dot HTML and the content of the anchor this text now becomes the thing that is the link, that's the thing that I can now click on in the browser and follow that particular link. Likewise we'll put one up here, so that's going to be content and that was text dot HTML and we'll close that out, save that, and then come back in and refresh. You can see now each of those is link, so I go to the text elements, now I get a whole series of text filling out the pages there. If I navigate back and go to lists now that takes me to the content list HTML and I get the information that's in that particular document. So that's pretty straight forward we've now linked to those additional pages within our own site using that relative path. If we switch back over and look just a couple things to note here the dot in the path as you probably know indicates that we want to start in the current directly. So then we would go into content text dot HTML. That works for relative but I also want to, if we look in the background here if you remember my super scripts that I put on the sources I want to be able to use those as links as well. So let's go in here I'm going to bring this down on another line because white space doesn't matter and let's create an anchor. And we're going to put an H ref in there and then we're going to have the one and I'm just doing this quite simply so we can see if before I put my long H refs in here. Now these are going to refer to other courses out on the Pluralsight site so they're not here local on my machine they're going to be on a different server and I therefore need to provide an absolute URI so let me copy one and bring it in here. So here we've got the URI we can see that it's going to have the full address so now we give it the server name Pluralsight training dot net go into the courses and we have the course name and the title there. So this URI notice don't have a dot HTM or any extension on it, that's okay, still a valid URI to that particular course info. Now we can paste in here and that's our CSS course. Now we'll paste in here and we'll go into our HTTP fundamentals and we'll save that, switch back over here to the browser and notice now that those items are links I get the nice little glove that indicates that I can click them. If I follow the first one look at all the information on our introduction to CSS course and if I follow the second one of course we get the information on the HTTP fundamentals course. So now I have links from my document that point out to other servers and other content using absolute URI's and I have links that point to other content on my site using relative URI's.

Understanding Targets

We've seen how to link from one document to another we can also link within documents, using those anchor tags. When we have a named anchor we can simply reference that as our H ref in the source anchor. We use the hash or pound symbol plus the name as our H ref and that allows us to navigate to that particular item. We could also append a named anchor on to the H ref for another page and navigate into that page to the named anchor. Now typically these are named anchor elements but you can also link to other elements in the page based on their ID and I'll show you that in a demonstration.

Demo: Linking to Targets

Now we've seen how to link between documents but we want to see using named anchors to link within documents. Now I've got a bunch of terminology up here in the overview using HTML and HTTP all over the place and unfortunately my definition list for terminology is down here lower and you can't actually see it when we're on the top of the page. So I've got some text here that says see the section on terminology for definitions used here and I want to make a link that allows you to jump down further into the document to see that. So let's go over here to our outline and I'll go find that section the terminology section and I'm simply going to wrap an anchor tag without an H ref I'm going to say the name is terminology and I'm also going to give it a title, I'm going to call that technical terms. And then I'll close that after the H 2. So we've got our anchor now that's named and the thing we're targeting is this heading that's where we want to go. So if we follow a link to this named anchor then we should come to this terminology section. So back up here at the top where we have that text now I want to make this word my source. So I create an anchor tag I'm going to give it an H ref, and now it looks a little different then what we saw before because we're using this hash symbol and the name of that tag so hash or pound symbol terminology is the path that's the name of the anchor we want to go and find. Now this is a pretty common nomenclature or identifier for items within an HTML document. You see it used a lot in different scripting languages for identifying things by their name or by their ID. We'll save that, come back out and now we see that terminology is a hyperlink, if we click it now we get dropped down, it took me down as far as it could because my scroll bar is all the way down but it got me to the point where we can get to that terminology piece. We also notice if I mouse over that we see technical terms. That was the title that I put on the anchor tag and that renders out there to give us some indication of what that anchor is representing. Otherwise you don't really have any indication visually that that is an anchor that's a target you could go to. We did see visualization of this, right, when we're a source it usually shows up as some sort of hyperlink by default that we can clink on but a normal named anchor it doesn't show up so the title allows you to make it have some particular meaning might also be used if someone whose visually impaired is working with your page and a reader can identify that. So that's within the particular page, but what about going to another page. So we had our link down here to the text elements what I'm going to do is go to text HTML and I'm going to do deep link. So notice I'm using the syntax the relative syntax to get to the file and then I'm going to use the naming here for that deep link. So if we switch over to the text page you'll see right here I have an anchor named deep link that is on the second section of text. So it should take me down to the middle of the page and that's the end of the anchor. So what I'm doing is saying take me to that page and further take me down into the named anchor within that particular page. So we'll go in here, we'll go on the text elements now and click that and notice we're right in the middle of the page we're at that header called second section where that named anchor is. It dropped us right down there and you can see that we can scroll up and get to that. Now notice that that's right in the URI that was the address that it gave to the browser that the browser went to. So if we come in and just go to HTML we get to the top of the page and if we add that piece in that's a link down into a named anchor within the document and it allows us to go right to that section. So this is really useful if you're referencing these documents with lots of content in them to be able to jump to a specific point in that document that's relevant for you. I mentioned that we could also link based on the ID's. So let's go into this text file and we'll give this H 2 and ID here. Call it second section, and then I'm going to copy that and simply save the file and go back now to my outline and now I'm going to replace deep link with second section. So from the source perspective it's the same idea I'm going to a particular file and I'm going to a named element. In this case I just happen to be going to the ID of a standard HTML element that isn't an anchor. So we'll save that as well, switch back into our browser, and refresh, and now if we go down to the text elements, you'll see we're going to second section in here instead of the named anchor and that still gets us to the same place. So it didn't give us any different result it's just another way of referring to your target is by using the ID of those particular elements. So either work the named anchor is a bit more explicit in that you're creating that anchor explicitly for use, but you can also go to those named elements by their ID's.

Additional Link Attributes

Now there are another variety of other attributes that you can apply to your links that provide hints to the client about what you're linking to. Things such as the language that the document you're pointing to is in, whether its English or Spanish or German, the content type is it text HTML, is it XML, the relationship, is it previous or next document in your series of documents, is it the table of contents, then you can also provide an access key. I don't often find these used in practice because they're dependent on that thing you're linking to which you may not control and may change and it's often hard to track those sorts of things in terms of the relationships, the language, and the content type. But I did want to make you aware there are other attributes available to you especially as you look at considering things such as globalization or internationalization of a site.

Summary

As we've seen in this module links provide us the ability to connect our documents together or to help navigate within a particular document or to a particular location in a document. We can use relative or absolute addressing, typically we use relative when we're referring to our own documents within the website and we use absolute addresses when we're going outside of the website. We use those anchors and we can additionally wrap other items such as images or other text elements to make those our links or our sources to go to another document.

HTML Tables

Introduction

Welcome to this module in the HTML Fundamentals course. In this module we're going to focus on tables in our html documents. We'll talk about table structure, the various components that make up a table. We're going to look at working with data in the table, how you represent it, different ways that you can modify your columns and your rows to span across and also table formatting and with this I mean formatting your table not using tables for formatting. We'll get into a little bit more of that as we go and I'll caution you against that over and over again, but here we're focused on using tables within our documents to represent data. You think about representing may be programmers in their favorite languages or how many programmers use a particular programming language or research data that you want to represent in terms of temperatures and time. Those sorts of pieces of information are well suited for representation within tables.

Table Elements

Tables are actually made up of several different components and we'll see here as we go a table being created, so we have a table and on that we might have a caption that provides a title or that key concept that the table conveys. So we see the table element, inside of that we have the caption. You can see it rendered here above the table itself. We also have the notion of a header, so we add the T head element and you can see here highlighted that that includes the headers for the various columns that you're representing in your table. We also have the body and the body is where we're going to represent the data itself. This is the various cells, rows and columns where our data is going to be visible. And finally we have the footer represented with the T foot and that's usually a single row, this is where we can call out totals, averages, those sorts of bits of information that we want to appear at the bottom of the table and one thing you'll note in the markup is, is that the head and the foot actually come before the body. The head's not so surprising, but the foot actually comes before the body for a reason. In the specifications laid out that if you have lots and lots of rows, it's possible for clients, browsers in our case to take that table and render it at a particular size showing the header and footer and allowing you to do things like scroll through or page through the data, so it's important that the footer comes before the body so that the client knows how to render the head and foot without having to process all of the elements that are in the body.

Demo: Structuring a Table

What I have here is my page that talked about the html lists module and I want to add a table that's going to lay out the various types of lists, the uses, the tags that go along with that. So I add a table and to that I allowed my caption that's going to be my header or my title really for my table here and then I can add a header and a footer and don't really need the footer, but I want to show you what we can do with it, so I'll add one in here and then we'll add in the body. So table body comes in last and as I mentioned the table footer does need to come before the table body because it's possible for client browsers that have all this information about how to start t render the table before they start processing the body. So if you have a really long body full of, lots of data elements they can start to render that out and show it to the user before it's processed all of those inner elements there. Now we can save that, come down and look at it and it's pretty boring. We have a caption and nothing else showing up under there. Now the reason is we haven't put anything in the header, the footer or the body yet, we need some rows inside of our table and some columns full of data. Before we do that I want to show you one of the attribute of the table and that is the summary. So I might put something in here as this, this table lists the various elements related to creating lists, something along those lines. This is a summary of our table and what it shows and while this whole show up necessarily in the user interface in the browser, it is a summary of what this table shows and can be used for things like assisted readers that for someone who's visually impaired so it's a really helpful thing to have out there to provide a summary of that table information. Again as we update we don't really see any indication of that but it could be shown or read to someone so they can get a better sense of what's in that particular table. So we need to go back to one of the slides and we'll figure out how to fill in the rest of these portions of our table.

Table Data

We've seen how to create the basic structure of our table and now we need to start adding some rows and column to get some data in there that we can see. We use the table row or the TR element to represent a row in the table and we'll use this inside the header to represent a row of headings in the footer to represent the various elements we might show in the footer and in the body to represent the various rows of data. Then with the header we use a table header data element or TH to define the individual cells within the row where the individual columns you can think of in that row and that will be where we'll actually put the data that we want to show up in the header. In the body, in the footer, we use the table data or the TD element and that's how we define the cell data for the body or the footer. So the TD and the TH are essentially the same thing, but we use the TH and the header to represent those header items. Let's go into our document here now. We've got our T head and we'll just add a row to that, a table row and right now that doesn't do anything. It's when we put this table heading here that we're actually starting to add some data to it. So we'll put in or indicate that we want to do the various list types and tags. If we save that and refresh now we can start to see the table taking shape. We've got two column headers now that are showing up there. In the footer I'll put a row in the footer and here what I'll do the TD or the table data and I'll just put for our example just a, a sentence here rather than some summaries and we'll see how that will render out. So we've got our header, we've got our footer. You can see that's going to show up now and you'll notice the way these things are lining up. So we have a row in the heading and we have two different table headers, so we've got kind of two columns and then in the footer we have a row, we just have the one column and so you'll notice down here that adds flowing underneath the first column of the headers and then this second column is standing out here all in its own. So let's go to the body and we'll add in couple of table rows and each one of these is going to have two cells in it, so we'll have that row and then copy that and paste it down here. So we've got our list types so we might have our unordered list and our ordered list and then for the tags we want to use our escape characters here so we'll do and less than UL, and we'll do a greater than then we'll do the less than again and we'll do the list item. We'll put that in there as well. Now I'll kind of cheat a little bit, for the ordered list we'll copy that over and we'll just chase this to, ol. We'll saved that and refresh and now we're starting to get a little more of a table, still got our caption, we've got our headers and now we're starting to see these columns coming in and if I highlight it, you can kind of see that we've got the headers and that left column, that is the first header and the first cell, the first table data from each item because our footer only had one TD, one table data, it's falling into that first column along with all of the others. You'll also note that the headers because we use the TH, those headers are coming out as bolded that's something that the client is doing here. If I go back and change this to a TD in here, you'll see it just shows up left justified, slightly changes the, they type phase so that browsers, the clients treat those table headers a little bit differently when they're rendering not out. So we have a basic structure, we can see now that we've got these items in there, we've got our rows and our columns and our caption for the table.

Spanning Columns and Rows

We've got our table with some rows in it, but not everything's going to fit nicely. We've seen that our table footer for example had that one simple column and it was always going to be left justified and, and always lined up with those first columns. But we're going to see that we have needs for columns and rows to span or rather to have a table data element span columns and rows. So on a particular table cell we can use the call span or row span attributes to get those things to span across the table elements to render correctly. We go back to our table here and let's add one more header, we'll call that uses, so this is where I want to come in now and add in some uses for these various lists. Now I'll come down now to the unordered list. The first one I want to put a table data item in here and identify the use and I'll say it's useful for displaying a simple list of items. Now if we save that and refresh. We can see that that makes sense but I really want conceptually we'd highlight this again, you can kind of see that. I really want that not just to be in the unordered list but it really applies to the ordered list as well. We're going to do two things here. One, I'm going to come back up to my table and use something we'll talk about in a moment, just the border because that's going to make it a little bit easier for you to visualize how these cells are lying out. You can see now each of the cells has a, a border around them and you can see how they're flowing. And then I'm going to come down to that table data again and I'm going to add the row span. Let's see, this particular item spans two rows, so the default obviously is that I would just span one, we save that and refresh and you can see now from the box and also the alignment of the text that particular table cell now spans these two rows, so the content that's in there is going to flow across both of those rows. Likewise, I could go to my footer on the TD and I could say call span equals three and now if I refresh we can see that that's no longer stuck in the first column, everything re-rendered a little bit here because this column is now spanning or rather this cell is spanning three columns and the other columns now can be rendered smaller because they don't have to, that first one doesn't have to fit all of this content. So this now spans all the way across those items. The call span and the row span allow you to flow those table cells to really layout the table the way you need to when you have data that perhaps applies to multiple rows or that applies to multiple columns. I'm showing it here with the column, the table data on the far right spanning the rows and the bottoms spanning the columns, but it could really be anywhere within the table that you do this row span and call span.

Formatting Tables

As you create these tables, you may want to control to some extent how they get rendered or how they're formatted and this isn't to be confused with formatting your document by using tables. For a long time back in the day that was a common way to layout your table because tables or layout your document because tables provide a nice grid concept. Today, it's not the recommended approach, you're going to want to use DIVs or other block elements and style sheets to layout your pages. So we're going to focus on tables and how we can format them not how to use them for formatting. Now I have our table here and one of the things we may want to do is control the width or the alignment. The width is going to be by default based on how much content you have in that table. We've seen in our examples that the table is as wide as it needs to be to display the content. You may want to have a little more control over that. Likewise you may not want the table to be left aligned on the page, you want it to be centered or right aligned and we can do that as well. You'll notice the asterisk here on the alignment, that is deprecated, so while it's available in the xhtml and html for lose or transitional specs, it will fail validation in the strict and you're expected to use style sheets for that. You may want to control the boarder, the lines around the edge of the table, the lines between the cells whether those are horizontal lines or vertical lines also known as rules, you may want to control those as well in terms of how thick they are, what they're rendered or how they're rendered, which ones are rendered. The cell padding enables you on the table to indicate how much space you want to appear around the content within those table cells, those TDs and those THs. So padding is how much space is inside those. So spacing indicates how much space or how wide the space should be between those cells. You can control both of those with attributes on the table itself. And finally we have the notion of no wrap. This doesn't get used as often and, and can certainly mess with the layout of your table, but one of the things you can't do on a particular table cell is add a no wrap attribute indicating that you don't want the content within that cell to wrap based on trying to size the table, based on the, the width you've applied up top or based on the width of the visible page that the client has to render with. Again, this is deprecated, but obviously available in the loose or transitional document type definitions, but you do want to watch out for this because this can as I said really mess up the layout of your table. You may want your table to be 50 pixels wide, but if you put a no wrap on a particular table data element and that element then can't wrap, your table is going to go beyond what you're fixed width was and it's going to throw off your page and all the other items that are around it, so use that sparingly or not at all.

Demo: Table Formatting

We have our table. The first thing I want to do is set the width. I don't want it to be driven by the content, everything looks a little bit cramped and I can do this two ways. I can put in a percentage of the available screen width, I'll say 50%, save that and we'll see now that that takes up roughly half my screen. Now if I start to shrink this you notice the table is resizing, it's still trying to fill 50% of the screen. The good part of this is its responsive, it takes up only that amount of space that I've asked it to and it allows for the rest of the content to fill the other half of the space. The bad piece of this is as my table starts to shrink, stuff gets too cramped up and they'd may or may not look right for me. So percentage is one way, but I can also use pixels, so I can say 250 pixels here, save that and now I have a fixed size and as I start to change you'll notice now it's a fixed width, it's not changing as the screen changes, so have a concept of how, why that thing is going to be. So you get 350 and we can also use the ends and look t that and again we have a fixed width now for our table s that it's going to stay the same size no matter how big the window is. So you can choose whether you want to use percentages or these fixed widths as well. Also I don't like the fact that its' left aligned, so I'm going to use the align equal centered and remember I'm using the transitional xhtml doc type here. So while this is deprecated because I'm using the transitional or loose it's valid within my document to use that and so now the client understands that though I may be using that, it's going to render that centered, left or right within the page. Again you can no cheat the same thing with stiles especially want to do that if you're using the strict document type definitions. Now let's look at the border. I had this on here just to kind of highlight things as it the border equals 1. We could set the border equal to 5 and we'll see that if we re-render, the outside border is much thicker now so we see that it's, it's 5 pixels wide whereas the inside still has the small borders. This is kind of a, a little bit corky, this is how the browsers do it. I'm really defining this outer border, but as soon as I do browsers almost in every case we're going to render these inner borders as well around the individual table cells and table headers. So borders kind of the simple way to do this. Notice that I don't get a whole lot of control over those things, so I do 10, I can control the outside, but I'm not really controlling the inner borders here. So, the other thing I can do is get a little more granule and I can use something called a frame and the rules. I'm going to do frame equals border and rules equal all and you'll see now that I have a border around this and I also have these rules or these lines in between the various cells. They're very narrow, it looks different than the, than the border did and if I want to then I can just say I want rules just on the columns. You'll see now I just get the vertical rules here in between the columns, but I'm also still getting the border around the outside and if I want to I can set rules equal to none and then I just have the border around the outside. So the frame and the rules at different settings where you can decide whether you want just the vertical lines on the border or the horizontal lines, the rules--have more strict control over what's available inside the table in terms of the vertical and horizontal rules that go on in there. These again are things that are available on table, they're still there, but you can also leverage style sheets which is probably more common to go in and apply to the table cells or the table rows, various styles where border would be one of those things. So you may go into particular table cells based on their class or all table data elements and apply a particular style to set the border for that item. Now I set this back to a border of two because the last thing I want to show you is the cell padding and the cell spacing and that only works really well with the, the border set up this way. If you use rules, you don't see all of this as visually. So for the cell padding, I'm going to set that to let's make it easy 10 pixels and we'll save that and what you'll notice is you can see how unordered list right now is jammed up against the left side as are all the tags and so if we change that now and say that we want the padding to be 10 pixels, I've now added this buffer of 10 pixels between my content and the edge of the cell. So the table padding really gives you the pad around that content but within the borders of the cell. If we add the cell spacing in similar, similarly added is 10 pixels then when we refresh, we'll see now it's not within the cells, it's the spacing between the cells now that set up as 10 pixels. So now we get some spacing between the cells themselves. So you can use both of those things to functionally lay out that table. If you need a little more room around the content, use the cell padding. The cell spacing gives you the ability to provide room in between the rows and columns as you're going. And note that even though this, this particular table data element spans to rows, so we don't see it in between there, it still applies the right spacing, it still lines up correctly along these items based on the cell spacing and cell padding because this is treated as one cell as it was declared in the document.

Summary

In this module we've seen the basics of creating tables for displaying information, we've seen how to create headers, footers and the body and how to use those table cells whether it's a single cell or it spans rows or columns to show the particular data and to provide that structure. You want to use that caption for the visible title of your table and the summary, for a detailed summarization of what's in the table to help those folks who are viewing your site with assisted devices or other kinds of clients.

HTML Images and Objects

Introduction

Hello and welcome to this module in the HTML Fundamentals Course on Images and Objects. We're going to talk about adding some visual appeal to our page by pulling in images or using objects to do things that perhaps the client browser can't do. We're going to look at pulling in images. These could be from the same director where all of our web content is, our HTML files for our markup, our style sheets and scripts, or they may be images that are found somewhere else on the internet. And we'll also talk about including objects, like plug-ins for Flash or Silverlight, or other technologies that allow us to extend what the browser can do in terms of interactivity, video and interaction. Now, with that we'll talk about fallback and accessibility, or what happens if you want to put that Flash player in your cage, but the client doesn't support Flash. Maybe they're on an Apple device, an iPad or an iPhone and they don't support Flash. You still want to give them some sort of experience or at least want to let them know that there's supposed to be some Flash animation going on here, but it's not going to happen because your browser doesn't support it.

Image and Object Concepts

When we think of our web pages we've got a lot of markup, and so far we've been looking at how we can use that markup to structure our elements into things like lists, and we've got the ability to create tables and links. We also want to be able to bring in visual elements, such as images, videos, or perhaps a more interactive applets or small applications into the browser. One way we can do that for images and it's use an image tag with the IMG element. And we simply reference the image file. So we're going to point out to an image file, again using a path that points to something next to or near the document itself, or the relative address, or using an absolute address to something out on the internet. We are, however, going to have some issues with images that we want to be aware of in terms of image sizes, both in terms of the visual size or the layout on the page, as well as the size of the file. And we also want to provide alternate text for images such that someone with a visual impairment or using various clients will still understand that the image was there and they have a representation in text that they can use rather than the visual. Objects allow us to add other items into the page that don't fit the image mold. And often this is used for things like plug-ins, so you'll see Flash being used a lot for games within web pages, advertising with animations. You can use Silverlight for the same sorts of things. We also have applets written in Java, you have videos that you might want to embed inside the page. And so the object element allows you to embed these items in the page, point to the source, generally, so where is the Silverlight application or the Flash application? Where is the video located? And the browser or the client then could determine whether it knows how to handle that object content. So if a browser sees a object tag that says that you want to play Flash and it doesn't have a Flash plug-in or doesn't have any registered way of handling that, it's going to look for a fallback to tell you that. If it's looking for or sees an object tag for a video with a particular content type and it doesn't know how to play that, then it will, again, for a fallback of something else to show you, or it will ignore it. So the objects we'll look at pulling in some of these various types of information and we'll see how to configure those and to provide the fallback experience for your clients. We're not going to go into detail on Flash or Silverlight. You can look into some of our other courses if you really want to go deeper on how to use these various plug-ins. We're going to look at how to use the object tag to reference those things.

Demo: Adding Images

I have my course outline page here. And I want to add some visual appeal. It's a little boring right now with just text and a white background. So over here on the right you can see I've added a little bit for the instructor. And what I want to be able to do is begrudgingly put a picture of myself on the page here underneath my name. So in the markup I've got my name here with a break, and underneath that I simply want to put an image. So I'll use the image tag and I'm going to apply the source. And here I'm going to apply a relative path to the image on the server; so the image source. And I'm also going to supply an alternate. And this is a required attribute; and again, this is so that if the browser can't render the image or the client is visually impaired and can't see it, there's some alternate representation of that image. And save that; and that image I'm referring to is just sitting right out here. So there's the matt.jpeg. And if we look, it's right up on my desktop so we can go up to the desktop. That's where the file is that I'm working with, so the relative app goes into images and finds that file. If we refresh then, you can see that right where I wanted it, the image appears, but I've got a problem. One, my image is way too big, and two, it's going way off the page, and it shows in way too much detail the gray in my beard there. We'll pretend that's frost. So this isn't working out for me. This isn't going to be a good layout for my page. So let's go back in here, and we can apply now some height and width information to this so I can get it down to a size that actually fits. So we'll do a width of 125, save, refresh, and now if we scroll back up, now that's a little better. So it fits into the page. Now it's in there. We've got a nice visualization and it's not thrown off and out of the screen. However, we do still have a problem. Let me pull up the tools here, and let's refresh this page. I'll show you two things. What you'll see if I pull up the developer tools in Chrome on the network tab this is going to show me the information it's pulled down and all the requests that were being made. So you can see we've made an HTTP get request for the HTML page itself. It means the browser requested the page and got that back. Then we could see that the content size was just under five kilobytes, and took about a millisecond to load. What happens when the browser sees the image tag is it then goes out and requests the image file as well. So we get the markup first. When it sees the image tag it goes to request that image file, and we could see now that it's pulled down the image slash jpeg. We've got that there. This is about 137, 138 kilobytes and took about two milliseconds. Now, that's not too big, but it's bigger than it needs to be. I've shrunk down the image in terms of its visual size, but I'm still taking the hit for having to pull down that entire image. Now, here I'm local, I'm going against the file system. There's no visual indication that that was slow, but if I'm on a mobile device or a phone, every image that you put on the page really counts. So let's go back now to our document, and we'll change this. And rather than use the large image, we'll use the small image here. And I could still apply the height and the width, in this case I'll even make it bigger and I'll say 185, and for the width we'll do 200. And now if we refresh, you'd see I've got the smaller file. Now it's less than 20KB, and it's coming back in there. It's still the perfect size for what I need in there, but it's a lot less resources in terms of the client experience. So you can use the sizing in terms of the height and width here, but don't use it to crop down a picture where the file size is way too large. Take those images, take the time to size them correctly for what you're going to use them for, get them down to a small size so they're optimized for the web. And then use the height and width simply to get the right optimization in terms of the visual layout, look and feel. And the last thing I'll mention is that of course here I used a relative URL to point to an image file that was along with my content. You can use an absolute URL, where we put HTTP and the server name and the full path. What you have to be careful with there is that that image is going to remain there. If I put a little error in my path here, we'll see if we refresh what will happen is you're going to get the broken image link, you'll see the alternate text here, but that image isn't going to appear. So if you don't own the image, well obviously there's copyright and rights issues, but also if somebody moves or renames the image, your page may be broken, and unless you go and look at it or scan it in some way you won't know that. So we'll fix that. Look once again. And we'll notice here that because I have the visual image displayed I don't need the alternate, but when it breaks or if there's another client rendering it then they can pull in the alternate text as well.

Demo: Adding Objects

With the object tag we can actually use something other than images. One of the common uses that I've mentioned is plug-ins. So I'm going to come in here under my page below my H1, and I want to add a plug-in and I want to use the Silverlight plug-in for Microsoft. And you could use Flash, you could be using a Java applet as well. I'm going to paste in this object text. You don't have to watch me type it. What you'll see is we have the object tag, and then we have a couple different pieces of information. So for the data I'm telling it that this is a particular content type, that this is an application and it happens to be a Silverlight application. And I'm also giving it that in the type, telling it that it's a Silverlight application. And then I'm just setting the width and the height of the application so I could put it on the page. Now, if we save this and we went and tried to run it, it's not going to work for us. You can see we have a blank space here, but it doesn't actually render anything because we haven't given it critical information that it needs. It doesn't know, for example, where the application is or where the code is for this. And for that we need to use the perim element. This is for perimeters that we want to pass into the particular object. So we have on these a name and value. So I'm going to give it a value that is the path to this particular code base, or the package that contains the code that we want to run. So for Silverlight that's a zap, and we're going to give it that perim. And now if we save and run you'll see now we've got a Silverlight object here running. And we get the marquis message saying, "ello HTML objects." Now, I would not recommend that you use plug-ins to put marquis on your page. I'm just using this as a demonstration here. Flash, Silverlight, Java applets are all very powerful for building an application functionality that you might need. But this does give you an example of how we can reference that object. And if I right click, you'll see that I get a specific context menu for Silverlight and I pull up the Silverlight information here. Now, the perim, this particular perimeter was used to pass information because Silverlight needed it to initialize that. But I can also pass perimeters into my applications. So I'm going to -- for Silverlight I can pass in a nit (phonetic) perims. So I can say that text message equals Do not use marquis in your HTML, in case it was unclear before. This particular application knows to look at this special input perimeter. This is a Silverlight perimeter, and it's going to be able to parse this out and see that the text message I want it to display is found here. So now we can see, Do not use marquis in your HTML, just to be perfectly clear about how you don't want to abuse these technologies. So the object tag allows us to specify a particular information about what kind of object we want to display, the type of that. We can size it. And then the perims allow us to initialize those items. You're often going to use the source in here, and you'll initialize Silverlight and Flash and Java applets, all slightly differently; not going to go into the details for every one of those. We do have courses on Silverlight of course where you can find out more about how to do that. You'll need more information than what we have here on HTML to go into using applets or using Flash, but now you have the basis to understand how to insert that with the object tag. Now, I mentioned that we wanted to have a fallback. What if my browser doesn't support this particular kind of application? What if it doesn't have the Silverlight plug-in? Well, inside the object here we can put any kind of markup. So I could come in here and put a span, and I could say, "We do not seem to have Silverlight." We can save that. Now, of course my Google Chrome does have Silverlight here. So I'm going to go and open this with Firefox, where I've got Silverlight disabled. And I'll pull that onto the screen. And you can see that the browser's smart enough to render the content within the object tag if it can't render what it's supposed to with the object tag. So it knows that, Aye, you don't seem to have Silverlight is the content I want to show if it can't do what's inside the object tag. But there's a couple of different things you can do with this. Certainly you can have richer content than just my span, so what's often used as an example for Silverlight is you can have a link here. You can see we've got the anchor tag with an HRF out to Microsoft site. We'll go get a particular version of Silverlight and that link is wrapping an image. So here you can see an absolute URL to Microsoft servers to get the image. And it has the alt. And so if we save that, bring back up Firefox and refresh, now you'll see we get the image that's a link. And so if I click on it, it will take me out to download the Silverlight IDXE and open that. So instead of showing me the object content, it can show me whatever markup I put inside of the object tag, so that I can give somebody a great fallback experience. Now, I happen to have put in here inside the object tag just a link for somebody to go and do something, but you can also nest the object tags. So maybe you've got Silverlight here and you want as your fallback Flash, or vice versa. Or maybe you want to have a Flash here but if they don't support that you want to put a separate object tag in here that simply points to an mpeg video, for example, or a different video format that somebody could use. You can nest those object tags as well to have fallbacks to different kinds of plug-ins or different kinds of native playback. So the object tag allows you to extend the browser or it allows you to go beyond text and images to include things such as video and other content.

Summary

In this module we've seen how to embed images or objects into our documents to add some visual appeal or to extend the capabilities of the browser itself. Those objects enable us to pull in richer content in some cases through objects or applications, but we need to be careful that we provide an experience for those clients so that they don't have that plug-in or they don't support that particular media type. They've got some option or some capability to render an alternative, or to let the user know in a clear way that they're not capable of rendering that particular content. And while you provide those fallbacks, it's important to consider those things because you want to give users the best experience, so choose wisely if you're going to use those plug-ins or things that go outside of the standard HTML that clients understand and know how to render.