









**Slide Notes:** When you’re first starting out with HTML, you start thinking that you use HTML tags to make things look a certain way. Like, you might pick a <strong> tag to make words bold. However, that’s not quite the entire story.



**Slide Notes:** To use HTML properly, you use “semantic markup”. That means you use HTML tags to describe the purpose of the text you’re marking up. When we add a strong tag around some text, we’re saying that this text is a little more important. The web browsers all have their own default way of showing some tags. So, when Firefox finds a strong tag it makes it bold. But you don’t want to use HTML tags to make things look a certain way, you use tags to describe your content. Then you use CSS to make things look a certain way. If you have something that’s clearly a paragraph, you put it in paragraph tags, even if you don’t like how it looks, because you can always use CSS to change how it looks.



**Slide Notes:** See, here we have a heading tag used properly, but the text itself is just placed underneath the h1 tag. Then, break tags are placed between the paragraphs to give it the spacing.



**Slide Notes:** Here it’s very clear what everything does. Each paragraph is clearly marked up. We can then use CSS to change how it looks. Using HTML markup properly makes it easier for computer programs, like browsers, and computer programming languages like JavaScript, to work with it. For example, it’s very easy for me to use JavaScript now to find all the paragraphs in a document.



**Slide Notes:** So, we want to make sure we know what the basic HTML tags are so we can use them properly. We have the h1 through h6 tags which we use for headings based on the level of importance of the heading. We only want one level1 heading per page, but there are cases when it’s appropriate to have more. We have the p tag for paragraphs, so any text that really is a paragraph should be wrapped with that tag. Then we have the strong and em tags which we use to markup text to give it a little more meaning.



**Slide Notes:** We can also list things in an outline form. We have two specific tags.



**Slide Notes:** Here are two lists. The only difference between these lists is that with one we care about the ordering of the items, and with the other, we don’t. Now, our browser is going to make the first list a numbered list, and the second a bulleted list, but it’s important to understand that that’s what the browser does, not what the tags do.



**Slide Notes:** HTML ignores spaces and line breaks, but sometimes we need to preserve spaces and line breaks. We can do that by using the <pre> tag.



**Slide Notes:** Let’s talk about divs and spans. Both are invisible elements that we use to divide up our page. A div is a block-level element, like headings and paragraphs. A span, on the other hand, is an inline element, like strong. The difference is subtle but important; we don’t put block level elements inside inline elements.







**Slide Notes:** In HTML5 we have many new tags we can use instead of DIVs to divide up our pages and they help us give more meaning to our content than DIVs do.



**Slide Notes:** It’s really important to use the right tag for the job. It makes it easier for people, and computer programs, to figure out what the text on the page means.



**Slide Notes:** So now, let’s talk about JavaScript.



**Slide Notes:** JavaScript was created in about 13 days by Brendan Eich. It was originally called “LiveScript” and Netscape wanted a language to compete against the Java programming language, so that web pages in the Netscape browser could be more dynamic. Eich was really under the gun, but he managed to develop a language that, despite its quirks, is flexible, popular, and powerful.



**Slide Notes:** JavaScript is an interpreted scripting language that can run in the browser but can also run on servers. It’s interpreted, meaning that it’s translated to machine code on the fly, as it runs. One of the biggest issues people have with JavaScript is that each interpreter is a little different. For example, the interpreter in Microsoft Internet Explorer doesn’t support the same things that the interpreter in Google Chrome supports. That means we must be careful about how we write our code if we want to make sure our code runs everywhere. If you develop JavaScript for web applications, you absolutely must test it on all browsers just like you test your page designs.



**Slide Notes:** HTML is a display language, not a programming language, so JavaScript lets us make HTML pages come alive with interactivity. But if we combine HTML and JavaScript together, we can go beyond simple interactivity and create entire applications and games that run entirely on the end user’s computer rather than on a server. We can also use JavaScript to build mobile applications. Most importantly, because JavaScript is easy to learn, it’s a great way to learn the basics of programming.



**Slide Notes:** A JavaScript program is made up of statements, called “Code” that follow a specific set of syntax rules. These lines of code consist of a series of characters which form words, or “tokens”, with symbols called “operators” and parameters, or identifiers, to accomplish a task.



**Slide Notes:** Here’s a little bit of JavaScript code.



**Slide Notes:** The ‘alert’ keyword tells the web browser to pop up an alert dialog on the screen.



**Slide Notes:** We pass the text we want to display as a parameter.



**Slide Notes:** Notice there are a lot of little syntax things we have to worry about. We have to end each line with a semicolon, and we have to make sure we have quotes and parentheses in the correct places.



**Slide Notes:** For the computer to run our code, it has to be written correctly. Keywords have to be spelled right, they have to be properly capitalized, and we have to watch out for missing semicolons, parenthesis, and quotations. Messing up these little details can cost you hours of time, so be careful.



**Slide Notes:** So, we need to get JavaScript code into a place where it can be interpreted. The easiest way to do that is to use the <script> tag to embed JavaScript code in our pages. Within the script tag, we simply write our JavaScript code. You’ll be doing a lot of this in your labs. You’ll find a script tag and have to type in the code specified by your assignment. This is called “embedded” JavaScript.



**Slide Notes:** There are two ways we can run JavaScript with an HTML page. The first way is, as you saw, to place the JavaScript code right in the HTML file within a Script tag. The current industry practice, however, is to save JavaScript code to a separate file, and then pull it in using a script tag and the src attribute. This approach is more advanced, so we won’t be using it until later in the class. But if you’ve come across tutorials that use popular libraries like jQuery, you’ve seen this approach before. This is often called Unobtrusive JavaScript because it doesn’t get mixed in with any HTML.



**Slide Notes:** JavaScript running in the browser is limited by what the browser will allow. It’s also limited by what the interpreter supports and by the speed of the machine the code is running on.



**Slide Notes:** You can’t assume that code you write on your machine works everywhere else. You must test your code on other machines to ensure the results are accurate. IN addition, you should know that while JavaScript runs everywhere, some people turn it off intentionally because they view it as a security concern. Other people with disabilities such as vision impairments may be unable to use web sites that rely heavily on JavaScript because the software they use to read web pages aloud can’t understand JavaScript as well. To get around that, we use unobtrusive JavaScript, and develop web applications that work well without JavaScript, but then work even better when it’s available. This way our users can still get work done even if they can’t use JavaScript.



**Slide Notes:** One of the biggest limitations is that in order to run JavaScript code in the browser, the source code itself must be downloaded by the user. That means that you can’t really write code that’s completely secure and you can’t use JavaScript on the client to write secure applications. The way we get around this is that we use JavaScript for the user interface, but we keep the important business logic on servers. Our web pages then talk to the servers and exchange data.

