**Tutorial: Creating Your First Web Page**

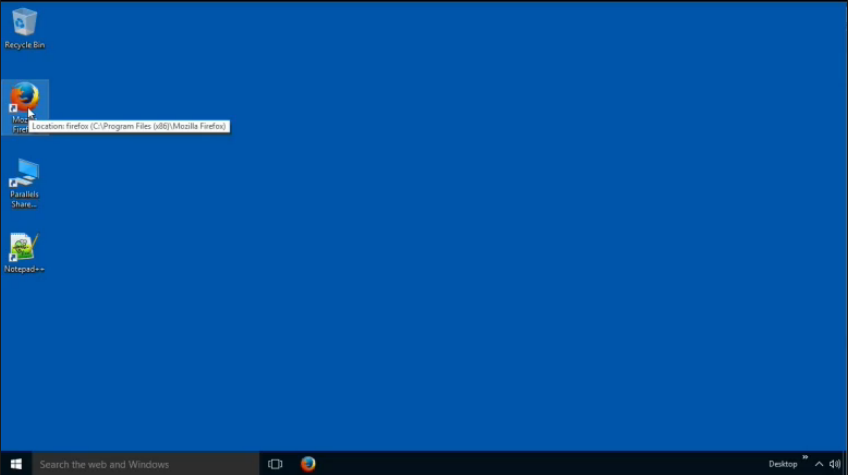
In this class, we'll build web pages with a web browser and a simple text editor. In this tutorial, I'll show you how to set up these tools and build a basic web page.

I encourage you to just read this tutorial first, and then reread it and follow along. This is a tried and true way of ensuring that the content sticks. See it, then do it. As you do it, you'll have the previous reading experience. I strongly recommend this approach; students that read the tutorials several times perform much better on assessments.

For this demonstration, I'll use Notepad++ as my text editor, as its free and very common on Windows computers. You may use any text editor you want, if you configure it the same way.

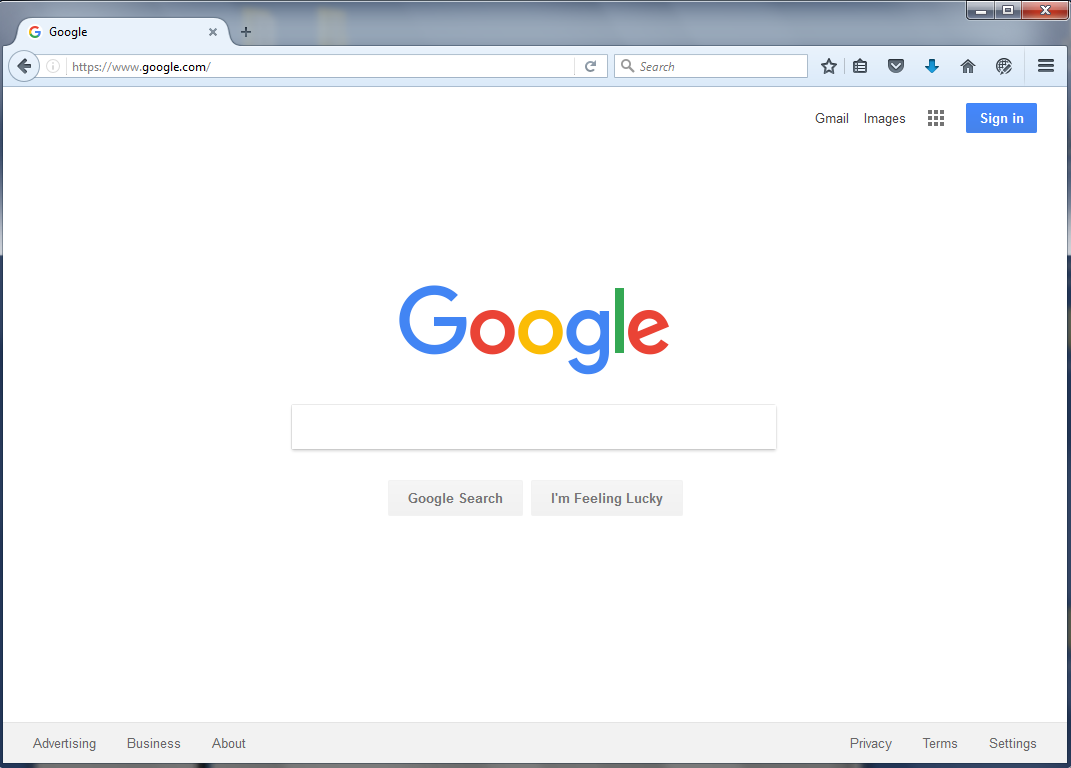
In this class, we'll use the Firefox web browser for development. Firefox is the strictest browser, so if we can make our pages work there, they'll work everywhere. Firefox also has some unique developer tools. You should test your web pages on every browser before turning them in, though, as I may grade assignments in a variety of browsers.

I am going to assume that you have installed Firefox and Notepad++ on your computer already, using the default settings. First, we'll explore Firefox.

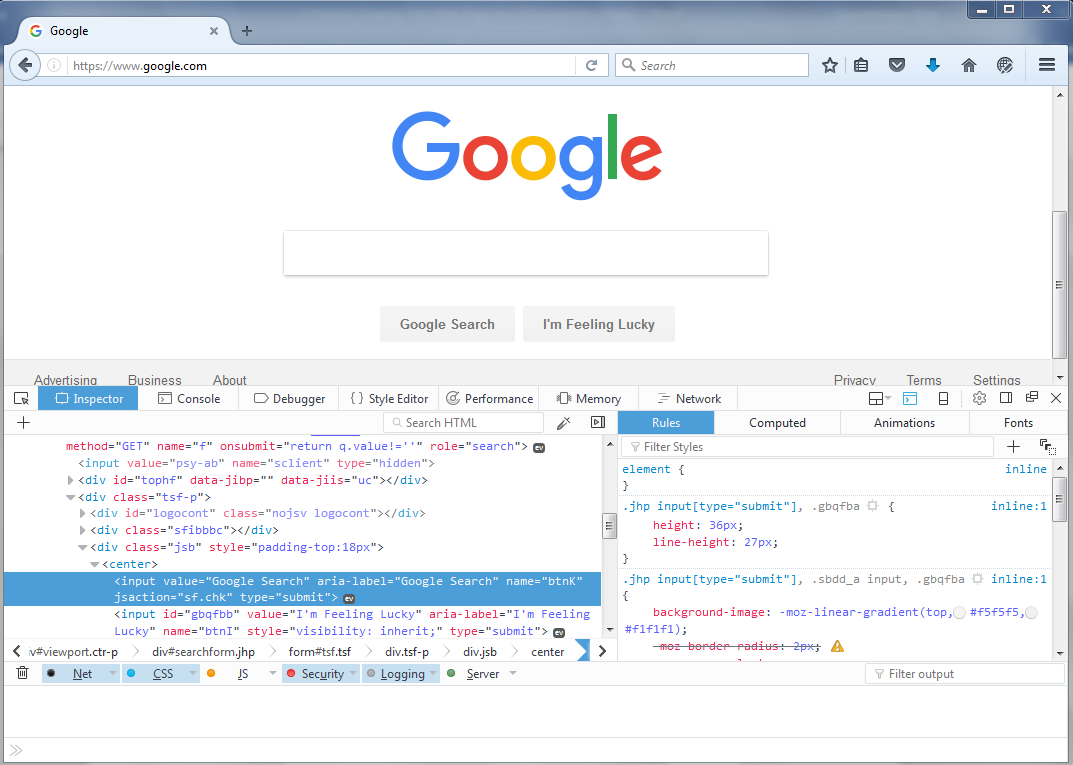


Let's explore a couple of web pages with some of the developer tools Firefox gives us.

First, let's open an existing web page; <http://google.com>.



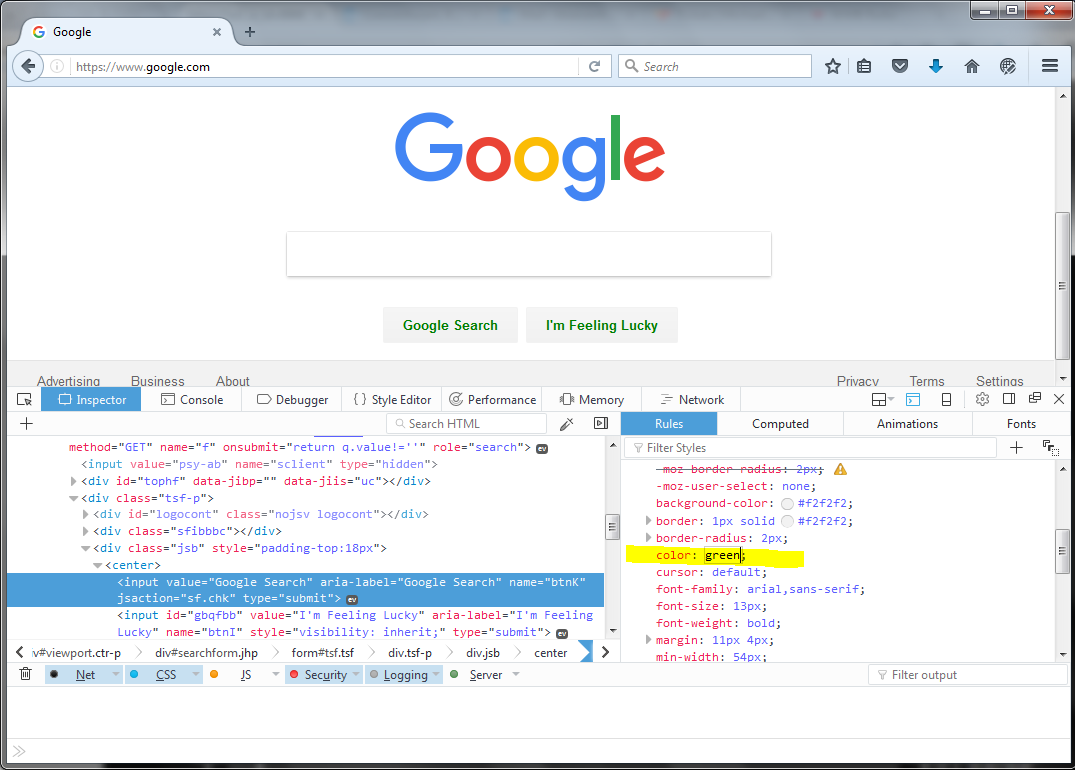
Then, we'll **right click** on the **Google Search** button element and choose **Inspect Element**. This opens the Firefox Web Console, and shows us the code associated with the element on the screen.



HTML, or HyperText Markup Language, is the code we use to make web pages. The web browser then reads that code and constructs Elements from that code. Elements compose a tree called the Document Object Model, or the DOM, and elements have child elements which may have other child elements. The Inspector shows you this tree, and we'll use this throughout the class.

If we choose an element, we'll also be able to see rules that determine how the browser displays the element. The browser has its own built-in rules that determine how elements look, but programmers can use Cascading Style Sheets, or CSS, to change those rules.

We can use the web inspector to change those styling rules. Let's change the color of the text of the buttons, just for fun. We select the element, and then we find the rule that controls its color. We can click on it to change it.



There. Now we have some green text buttons. Later you'll learn to do this in code. For now, just understand that we use HTML tags to define the elements on a page, and we use CSS to change how those elements are displayed in the browser.

**Building a webpage with Notepad++**

Now let's open Notepad++ and build a basic web page.

The code that makes up a web page is just plain text, using a language called HTML, or Hyper Text Markup Language. We use \*tags\* that describe our page's structure and content. A web browser then reads that text file in and transforms those tags into \*elements\*.

A basic web page needs to start with a document type declaration. For our class, the first line of every HTML file will look just like this:

<!DOCTYPE html>

This line tells browsers that this file is an HTML file. This determines the rules that a web browser will use to determine if our file has the right code in it. We call this the DOCTYPE.

Next, the file needs an HTML tag.

<html>

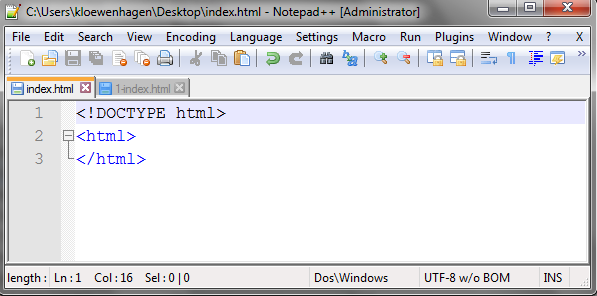
and it needs a closing HTML tag.

</html>

**Saving our File**

Before we get too far, let's save our file. HTML files should be saved with the .html extension. Go to `File` and choose `Save As`, and then save this new page to your desktop. Call it `**index.html**`. This `.html` extension is incredibly important. Don't use `txt` or something else.

When you successfully save the file, Notepad++ will suddenly change the color of your code! Because you named the file with the `.html` extension, your editor now knows what language you're using.



Most HTML tags come in pairs, with an opening tag and a closing tag. The opening tag says, "This is the start of the element" and the closing tag says, "and this is the end of the element." Anything we put between the opening tag and the closing tag is "inside" of that element.

I recommend that you create the opening tag and closing tag at the same time, and then move your cursor between the tags. This ensures that you won't forget a closing tag.

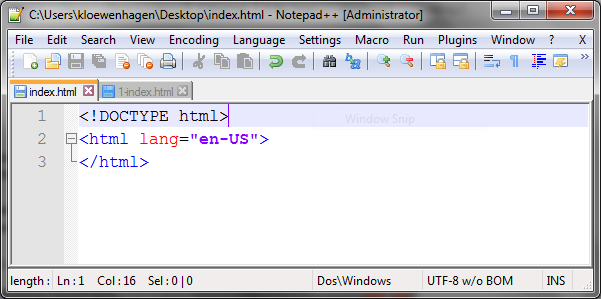
HTML tags also have attributes that further describe the element they'll create. We're going to add the `lang` attribute to the `html` tag so we can tell the browser what language the text of this document will be.

<html lang="en-US">

An attribute has a name, an equals sign, and a value. You must look at the HTML specification to find out what attributes are available on each tag, but there are many universal ones that we'll use in this class.

For this class, we'll use lowercase for our tags and attributes, and we'll use double quotes for the value. In this case, the value is 'en-US' which is a universally recognized shorthand for "US English."

This defines the HTML element. All our web page content must be "inside" of this element.



The next thing our page needs is a `head` element that holds the data about the web page, such as its title. We define that element with the `head` tag, which has an opening tag and a closing tag, just like `html` does.

Remember, every other element needs to be "inside" of the `html` element. So, this new tag must go between the opening and closing `html` tags,

To make it readable, we will indent tags we put in between an opening tag and a closing tag.

This indentation makes it very clear that the `head` element is "inside" of the `html` element.

**Tabs and spacing**

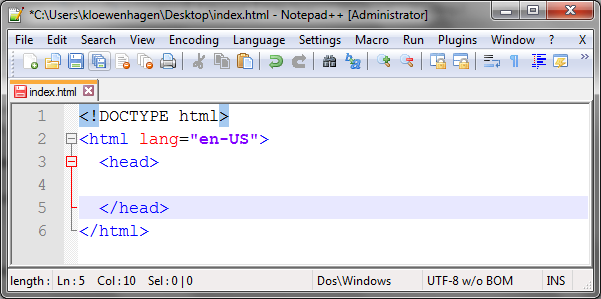
So, to indent, I'll hit the `tab` key and I’ll type in the opening `head` tag, and the closing `head` tag:

<html lang="en-US">

<head>

</head>

</html>

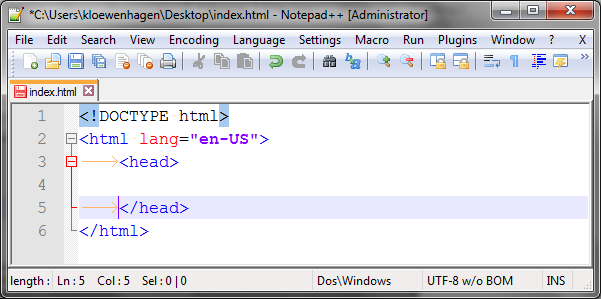


But we have a bit of a problem.

We'll be indenting code a lot in this class, and you'll be doing that a lot as a software developer. But web developers favor using spaces for indentation instead of tab characters.

When we press the `tab` key, we're getting a `tab` character inserted into the document instead of spaces, and that's going to look different on different people's machines. Let me show you:

Now, I'll go to View -> Show Symbol -> Show White Space And Tabs



Look at my output. You'll see an arrow where I've pressed the Tab key; the editor inserted the tab character.

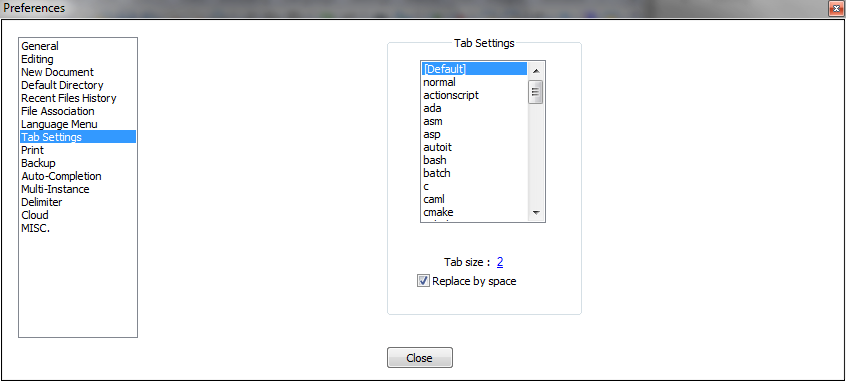
Now, the choice of tabs vs spaces is a large debate that differs among programmers and languages, but we're adopting a coding standard for web developers. By using a standard, everyone's code looks the same on any machine. If you use tabs, my tab size might be different than yours, so your code will look funny to me. If we both use spaces, then your code will have the same indentation everywhere.

So, let's configure Notepad so that when we press the Tab key, we get \*\*two spaces\*\* instead.

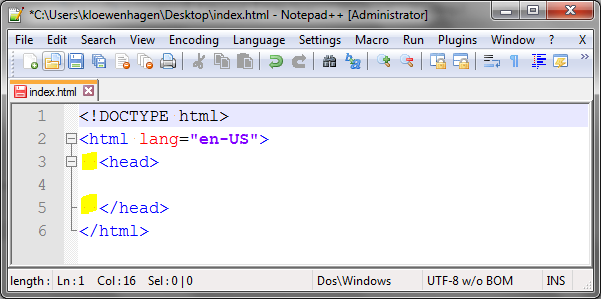
First, we'll go to **Settings**, and choose **Preferences**.

Then we'll choose **Languages.**

From there, we'll change the **Tab size** to **2** and **check** the box that says, **"Replace by spaces."**



Now when I press the tab key to indent something, I see space characters instead of tab characters. Changing this setting does not change the code we've already typed though, so I must fix that. Delete the tab arrows before <head> and </head> tags and reapply the tab key.



**More Elements**

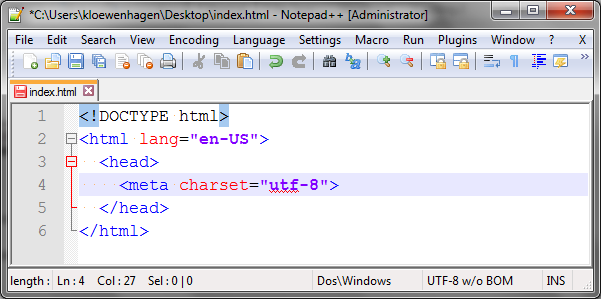
The `head` element contains a couple of other elements that we'll need in every one of our pages.

First, we'll add this special tag called a `meta` tag inside the head element:

**<meta charset="utf-8">**

This tag specifies what characters we want to allow on our page. This determines whether our document will contain special symbols and international characters.

Notice the meta tag is indented two more spaces, to show that it is "within" the `head` elements. We're continuing to use indentation to show to other developers that our page has a hierarchy. `meta` is inside `head` which is inside `html`.



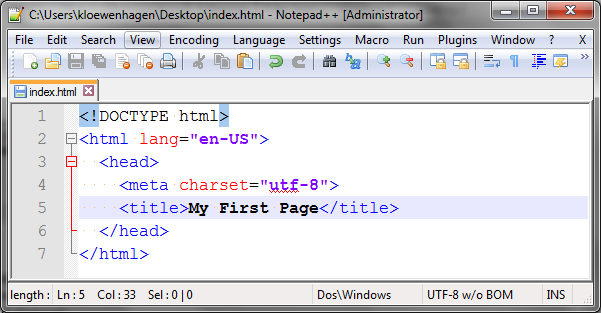
When the browser renders a page, these tags get transformed into \*elements\* in a tree, and the elements on that tree have parents and children. Some of those elements will have their own child elements, and sometimes they won't. And sometimes they can't have child elements.

The `meta` tag doesn't have a closing tag. The meta element cannot contain any other elements, so we just leave the closing tag off. There are only a handful of elements that don't have closing tags, and as you learn more about HTML, you'll be able to easily remember them.

Next, we'll use the `title` tag to define the title of the page.

**<title>My First Page</title>**

The `title` tag has an opening and closing tag, but this is one place where we can do it all on one line instead of multiple lines with indentation.

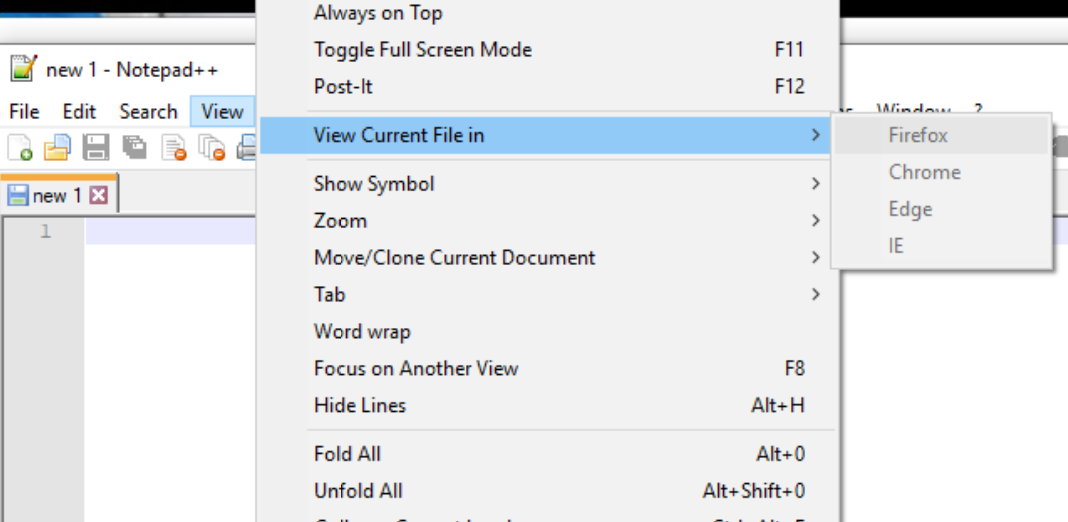


Notice that once again, I created the opening and closing tags.

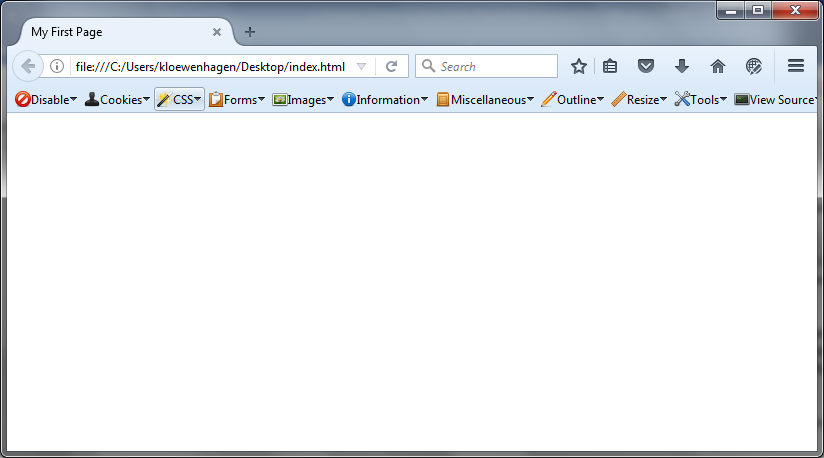
**Viewing in the Browser**

Let's look at our page in the browser. You might be used to viewing a web page in your web browser by searching for it on Google or by typing in the page's address. In those cases, the web page is stored on some other computer on the Internet and made available to you by a web server. But web browsers can read files from your local hard drive, which is exactly what we'll do.

Before we continue, ensure your file is saved. In Notepad++ the disk icon will be red if it's not saved. You can use the File menu and choose Save, but I recommend you start using keyboard shortcuts. In Windows, you can almost always save the file you're working on by pressing **Control S**.



Next, you can run this file in your browser a few ways. First, you can use Notepad++'s ‘**View` - ‘View Current File in**’ menu and choose `**Firefox`**. This will open the Firefox web browser and display your file. Notice that the URL references the file on your desktop. You should compare this URL with the file path displayed in Notepad++'s title bar. It's easy to accidentally edit the wrong file, especially when you're working on assignments later in the course. You can also close Notepadd++, find the index.html file and right click on the file – open with – Firefox to open your page in Firefox.



The other way we can open this file is by right clicking on the tab, copying the file path to the clipboard, and pasting it into a running web browser.

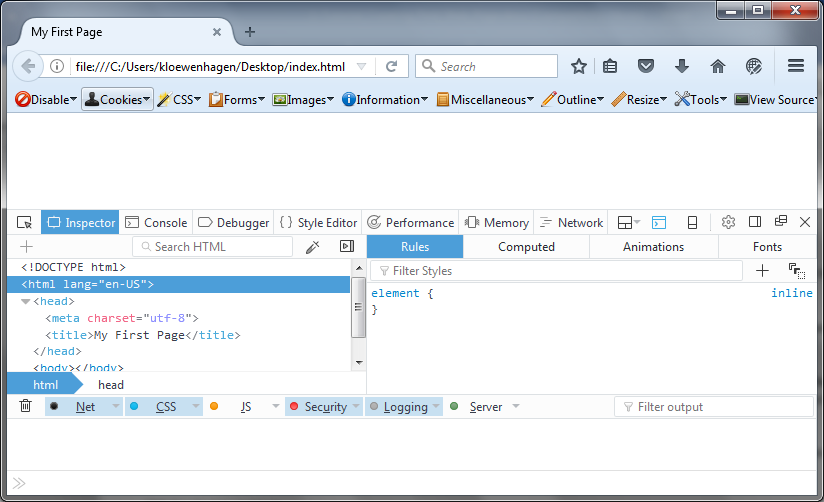
Finally, we can drag the file from our Desktop onto a running web browser window.

**Exploring the Page**

When we bring up the page, we see nothing. We've done all that typing and don't have much to show for it. The title of the page does show up though, in the tab of the browser.

All the stuff we typed already is just information about the page. The stuff that the user interacts with in the browser window hasn't been defined on this page, yet.

But let's inspect what we do have; **right click** on the empty web page and select **Inspect Element**. As you saw before, the web inspector shows the document object model, or the DOM, as a tree. Each of the tags in our document are now represented as elements on this tree. You'll also notice that the browser has indented these items just like we did in our code.



Use this to your advantage. The browser doesn't care if your code is indented, but other developers will. Compare your code with how the browser's inspector indents things. Let it be your guide.

**Displaying Content**

The next element we need to add to the page is the `body`.

The viewport, or the part of the browser that displays the page, contains any element we specify inside of this element.

So, in our page, below the closing `head` tag, we place a `body` tag. And I'll put the closing tag in too, and leave some space between them.

<html lang="en-US">

<head>

<meta charset="utf-8">

<title>My First Page</title>

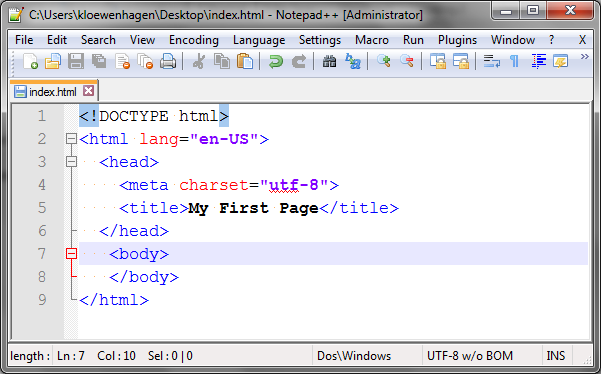
</head>

**<body>**

**</body>**

</html>

The `head` and `body` elements are siblings. They are both children of `html`, therefore we make sure that they are indented at the same number of spaces.



With the `body` tag defined, let's add some content.

Tags that make up child elements of `body` get indented two more spaces.

Let's add the `h1` tag, which will create a heading1 element on our page.

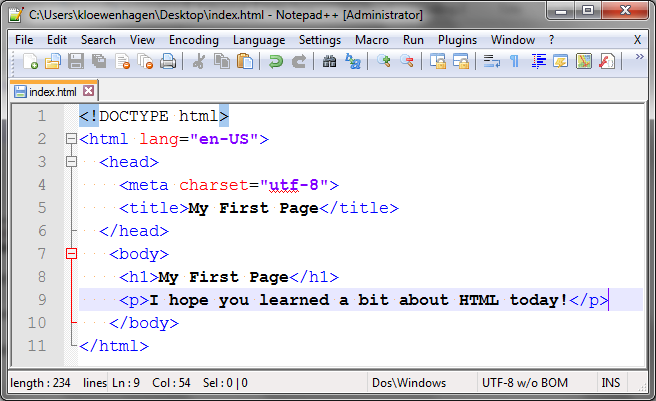
**<h1>My First Page</h1>**

The text `My First Page` is "wrapped" in `h1` tags. The text is the "content" of this element when rendered on the page.

Let's add a paragraph to this page now.

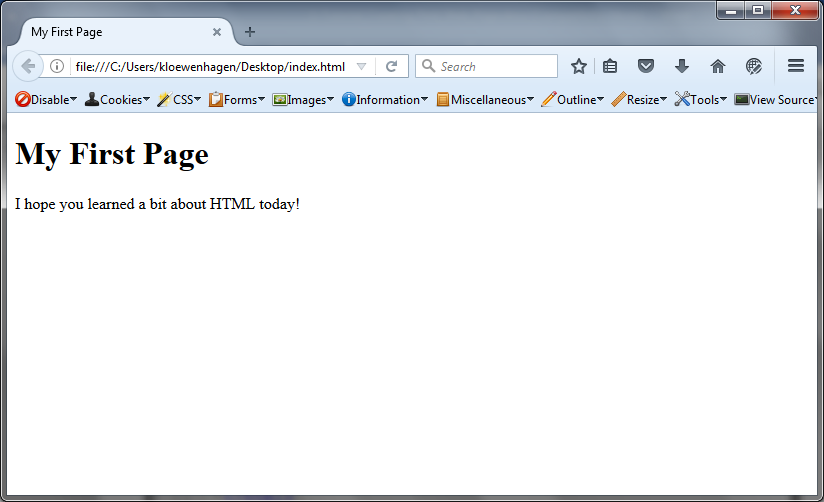
**<p>I hope you learned a bit about HTML today!</p>**

The `p` tag creates a `paragraph` element. All the text between the opening and closing tags is the content of the paragraph element.



Let's view it in the browser. Save the page and reload.

Once I've loaded the page in the browser, I can just switch to the browser and press the reload button, or use the Control R keyboard shortcut to reload the page. I don't have to do it from Notepad++.



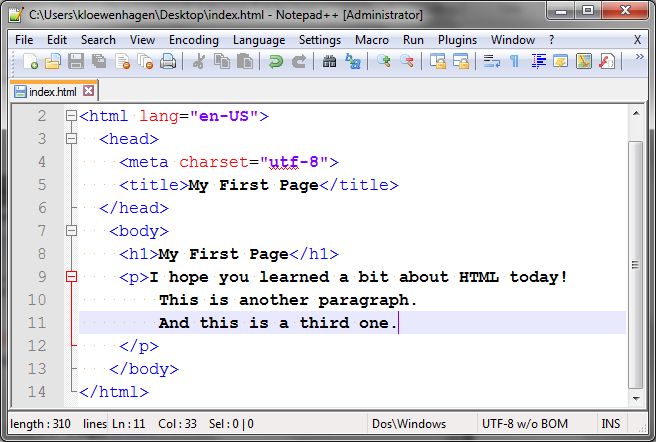
We now see our new paragraph on the page.

**Choosing the Right Tags**

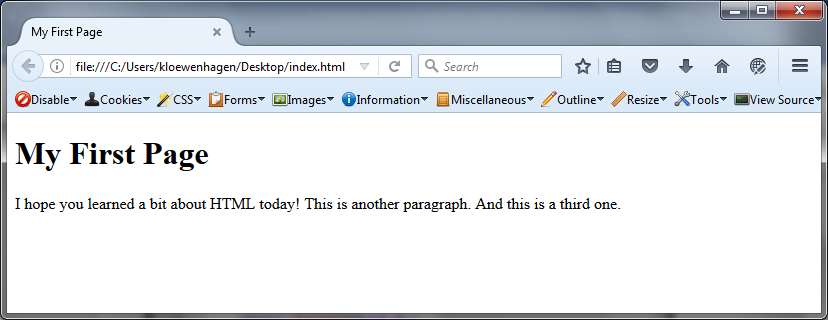
HTML doesn't require that we wrap our paragraphs with `p` tags. Let's add a few new paragraphs to the page, right below the previous one:

**This is another paragraph.**

**And this is a third one.**



When we view the page, we'll see something interesting: The two paragraphs run together! The browser ignores line breaks.

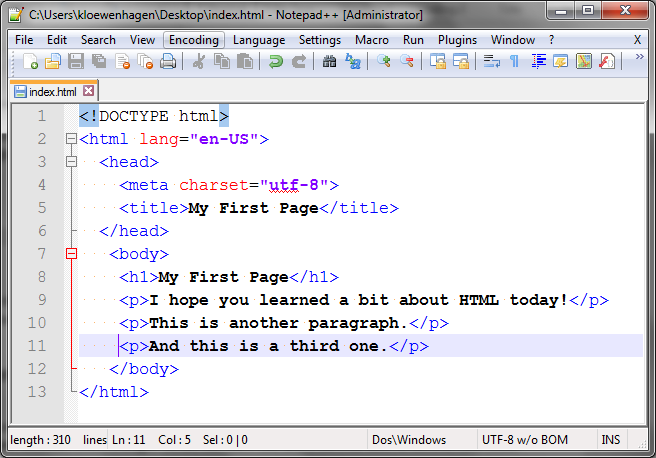


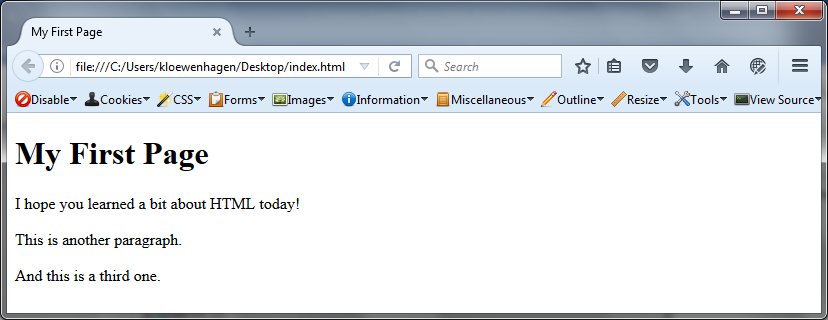
When building things on the web, it's important that you correctly describe the content you're placing. A paragraph should be wrapped with `p` tags. A heading should be wrapped with `h1` or another tag that's appropriate.

Notice the difference between the `h1` and the `p`. The `h1` tag makes the text huge and bold. But this isn't because of the tag we chose, it's how the browser decides to render the element by default. Using CSS, we'll be able to change how the browser displays our tags.

However, if we don't use the right tags, it'll be much harder for us to use CSS or other programming languages to read and display those elements.

So, let's wrap both paragraphs with `p` tags. Save the page and reload, and we see that they are now separated. They are all displayed with a line space between them though. When we learn CSS later, we'll be able to control how these are presented.

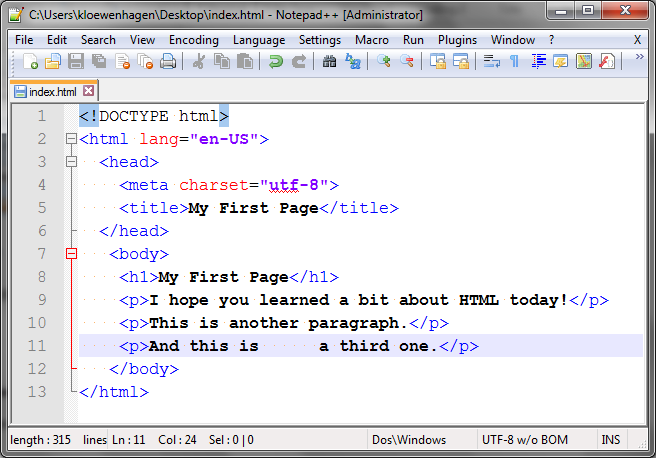


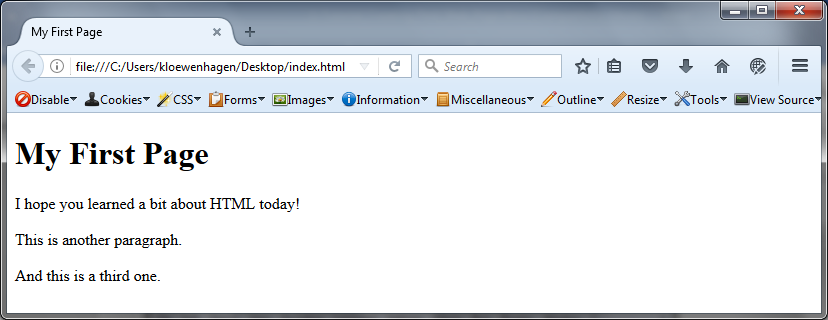


The browser also ignores spaces. So, if we changed the last paragraph to:

**And this is a third one**

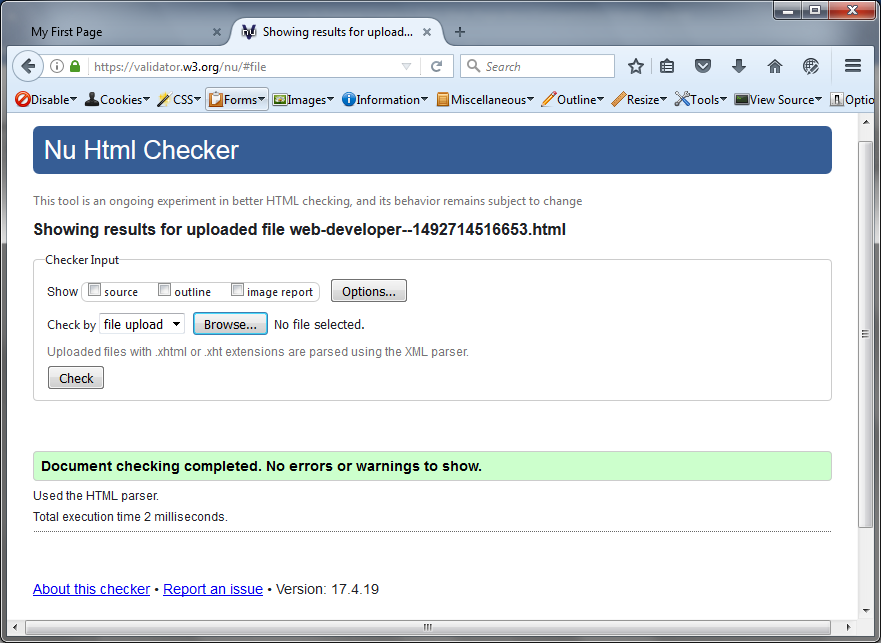
and we saved the page and viewed it again, we'd see that all but one space was ignored.





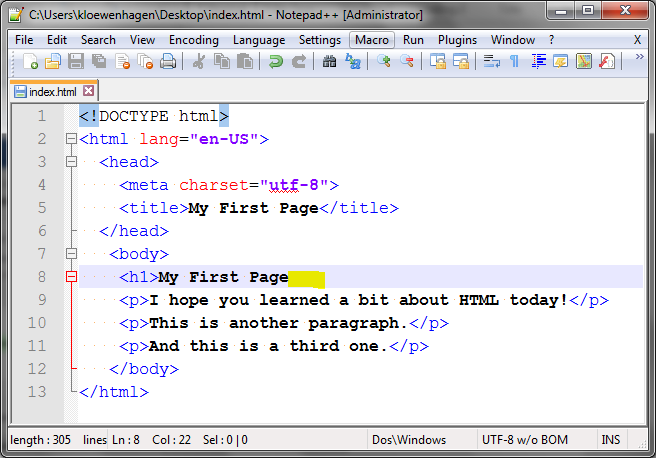
**Validation**

It's a good idea to check to make sure we haven't made any coding mistakes along the way. HTML is a language with rules. Some tags need to have closing tags, and others don't. We can periodically check if our code is valid by using the <https://jigsaw.w3.org/css-validator/> . This will take all the code on our page and send it to the W3C validation service, giving us a report.

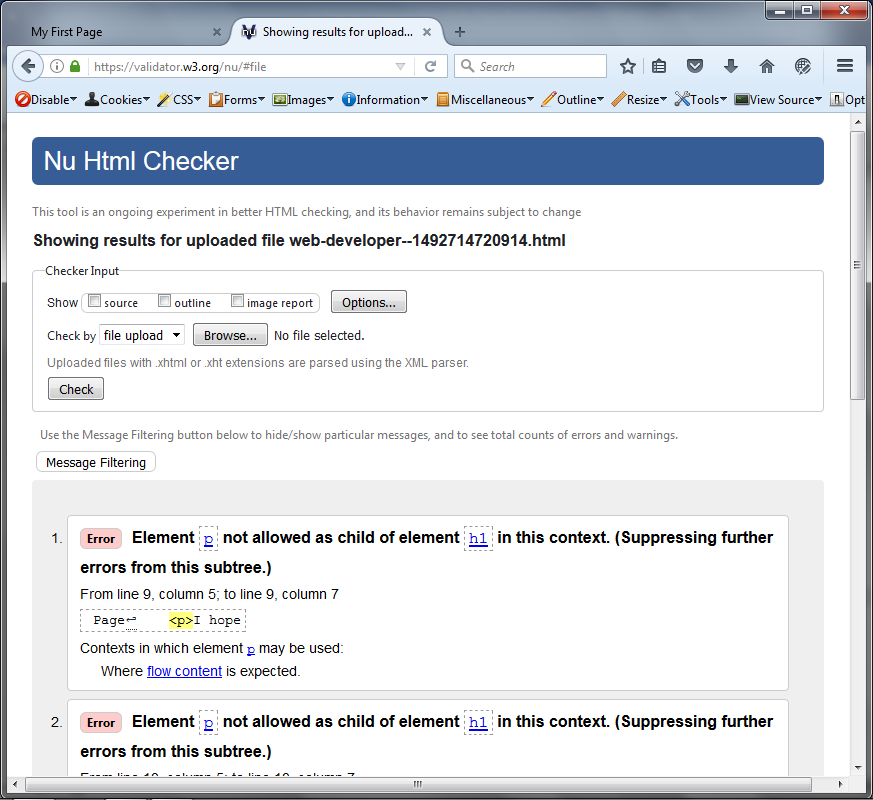


Things look great. We have a couple of informational messages but no errors. But let's make an intentional mistake so we can see the consequences.

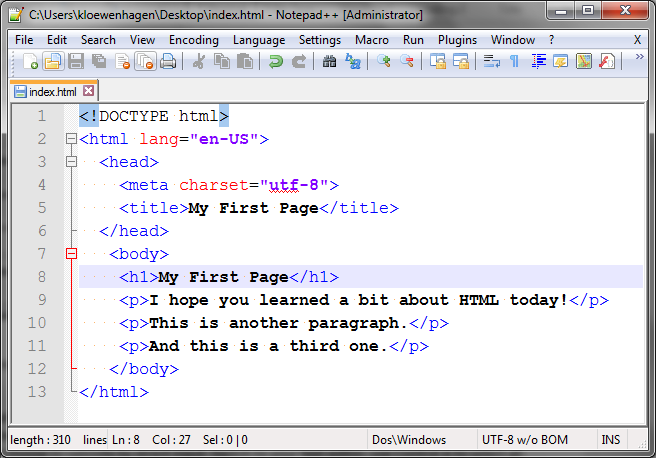
I'll remove the closing `h1` tag from our header. Then I'll save the page and reload in the browser. Then I'll run the validator again.



This time we see several error messages.



You will want to validate your markup frequently in this class to ensure that things work properly. Add your </h1> back into your HTML and resave your file.



Visit <https://validator.w3.org> and choose to validate by direct input. Switch to your text editor, use Control A to select all the HTML, then use Control C to copy it to your clipboard. Then switch back to the validator and paste the contents in using Control V. Press the validate button and you'll see the same report.

**"Block" and "Inline" Elements**

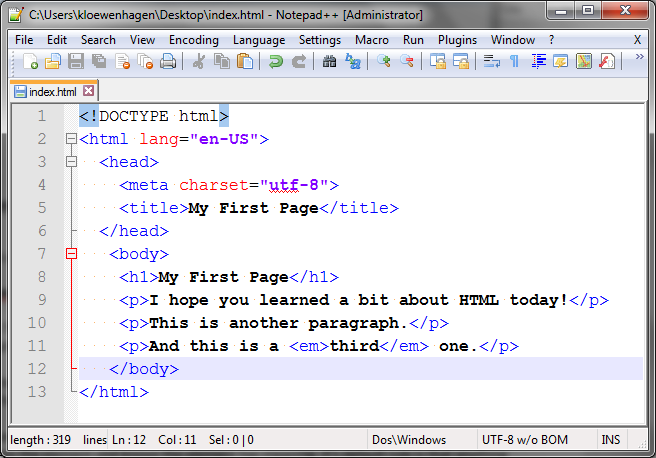
Tags like the `h1` and `p` tag create elements that, by default, start on a new line. These elements are called "flow content" elements because they are meant to contain content or other elements.

Another class of elements, sometimes called "inline" elements, are elements that don't start on a new line. They're meant to be placed inside of other elements. These elements are called "phrasing content" elements, and they describe the content they contain.

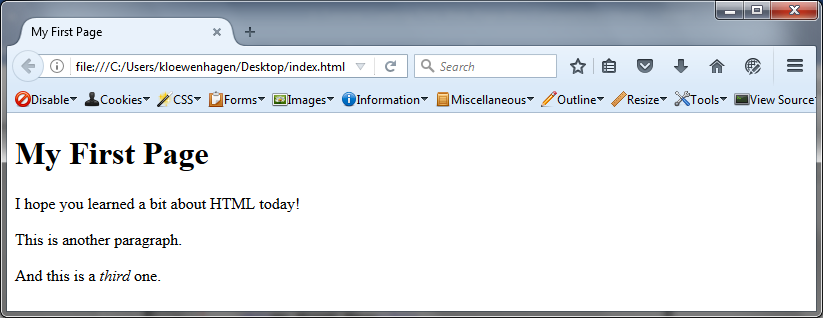
Let's change the last paragraph so it looks like this:

**<p>And this is a <em>third</em> one. </p>**

We're using the `em` tag here. The text between the opening and closing tags is now "emphasized". When the browser reads this, it creates a paragraph with some text, and then, inside that paragraph, it creates a new `em` element.

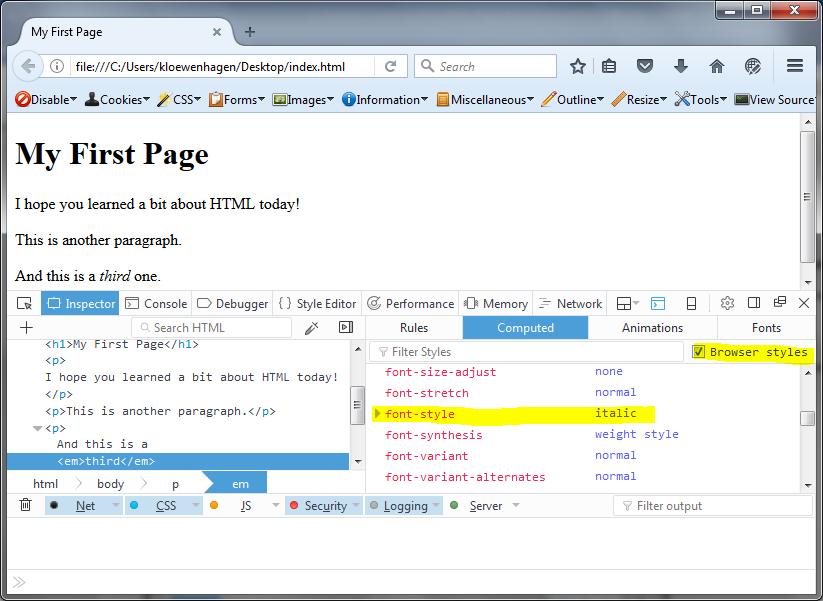


The browser then displays the text in the `em` element in italics. However, here's the tricky part. The `em` doesn't mean `italics`. It means that this text has emphasis, or should be read with emphasis. It has meaning.



Let's use the web inspector to explore this; right click on the page and then **Inspect Element**. If I inspect this element, the inspector shows us the EM tag. The browser is italicizing the text because it sees the element and knows the element has meaning. It's default rule is that anything that's wrapped in an `em` element should be in italics.

We don't see any rules, but if we click on the `computed` tab and then check the checkbox for `browser styles`, and in the search box we search for `font`, we'll find out that the browser has applied italics to that text.



We can use the `strong` tag here instead, to define that we want this content to be interpreted as content with strong emphasis. By default, the browser will make this bold text. But again, we don't use this tag because we want bold text, we use it because we want to say, "this text is really important text with emphasis."

It's a subtle distinction, but it's important. When creating web pages, use the right tag for the task at hand. HTML includes the `<i>` tag and the `<b>` tag for italics and bolding. Use those if you just want the appearance but don't care about the meaning.

But don't forget - if you're italicizing something, you probably have a reason to do that. What's the meaning? Usually it's because you want to emphasize something.

Using markup to describe our content is called "semantic markup" and it's incredibly important to producing good web content. We'll continue to explore this concept throughout the course.

**Wrapping up**

That about wraps it up. We've covered how to use the developer tools in Firefox and how to build a basic page with Notepad++. We've also gone over semantic markup and how to ensure things are valid.