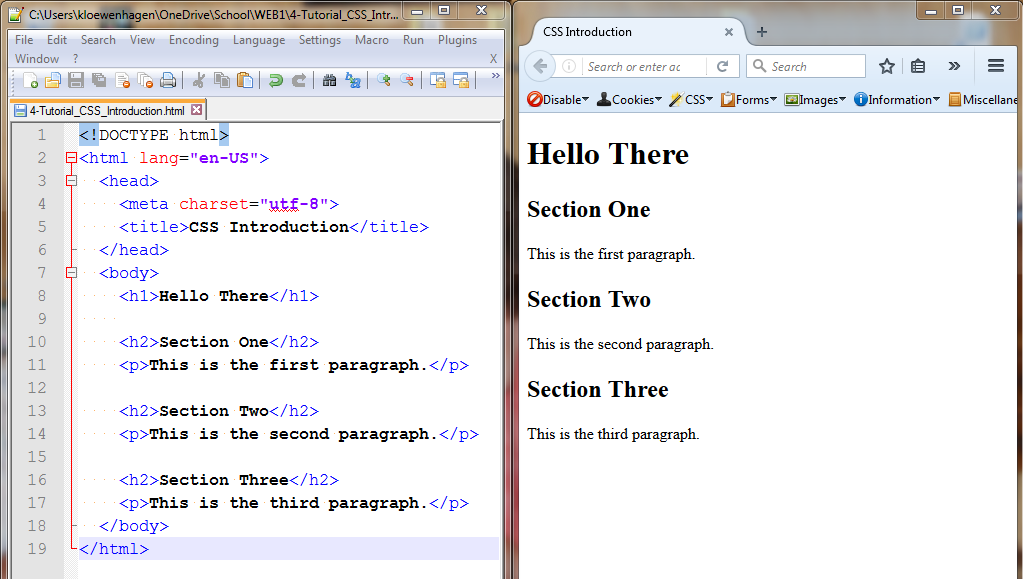
**How CSS Works – CSS Introduction**

In this tutorial, we're going to explore how we use Cascading Style Sheets to change the way a web page looks. This is the first in a series of tutorials on CSS, and this one focuses on the basics of CSS. We'll cover how to style individual elements and groups of elements, and we'll look at the various ways to use CSS in a web page.

The important concepts this week are how to use CSS to change the way elements look. Soon you'll use CSS to create a layout for your site, but right now it's more important to learn how CSS works.

What is CSS?

When the browser renders our elements, it then "paints" them with some styles. Once the tree of elements exists in the browser, the browser then applies its default styling.



I've built a very simple test page, with a few paragraphs and headers:

**<h1>Hello There</h1>**

**<h2>Section One</h2>**

**<p>This is the first paragraph. </p>**

**<h2>Section Two</h2>**

**<p>This is the second paragraph. </p>**

**<h2>Section Three</h2>**

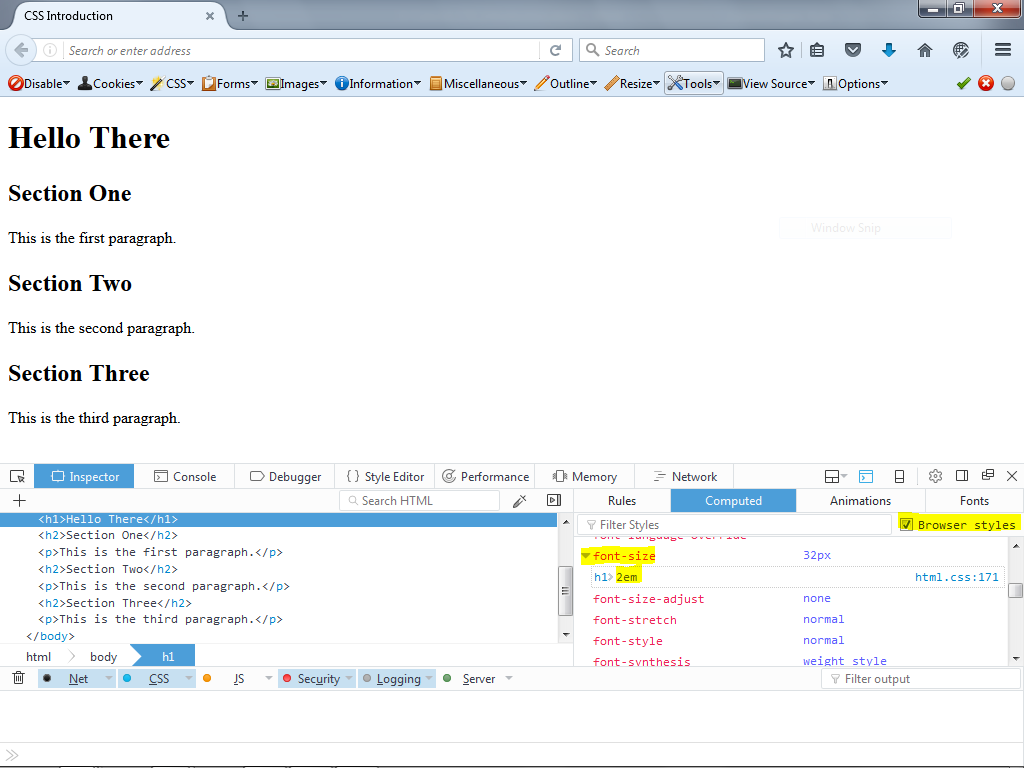
**<p>This is the third paragraph. </p>**

Please create this web page now if you have not done so already. Save the file as 4-CSS\_Introduction.html.

If we look at the `h1` element here, we see that it's displayed in a larger font size than the rest of the text on the page. The browser sees the h1 element, and then it changes the font size based on a built-in rule that's mapped to the `h1` element.

In the developer tools, we can see exactly how that works. By right-clicking on the heading and choosing Inspect Element, and then choosing the "Rules" tab, we'll see the CSS rules we've written. It's empty since we haven't written any CSS yet. But if we look at the "Computed" tab, and then click on the box for "browser styles", we'll see all the defaults that Firefox applies.

We can narrow down the results by typing the word; **font** into the filter search box on the bottom of this pane. Here we see that the font size has been set by the browser, 2em (Twice the normal size font). Cascading stylesheet pools are composed of a property and a value. In this case, we have the font size property and its associated value.



To use CSS, we first find an element and then apply some rules that define how that element should be displayed.

I'm going to show you the easiest, and probably the worst way, to apply style to an element.

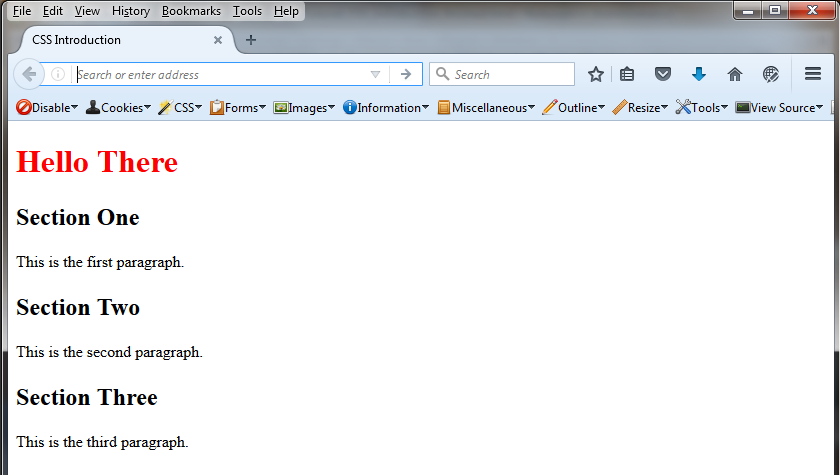
**Inline styles**

We can apply CSS rules to an existing element by using the style attribute of any HTML tag. This approach is bad because it mixes style with the content, and we always want to separate the way things look from the way things are on a webpage. However, this method is often necessary if you don't have the ability to change the stylesheets of the web page you're working on, or if you just need to make a very specific style adjustment to a single element. You should consider what I'm about to show you as a last resort. While your book shows you how to do this, we will not use this technique in this class.

Let's make the text of our main heading show up as red. To do this, we add the 'style' attribute to our `h1` element. Then, between the quotes, we'll add the property "color", a colon, and the word "red".

**<h1 style="color: red">Hello There</h1>**

When we save, and refresh the page, the color of the text is now red. This was incredibly simple to do, but like I said we want to keep the style information separate from the HTML markup. This way we'll be able to use the same styles over and over again.



Remove the inline style attribute in the h1 element back to:

**<h1>Hello There</h1>**

Save the .html File.

**The Style Tag**

Instead of adding CSS rules to an HTML element using the style attribute, we define a stylesheet that contains code that looks up elements and applies the rules we want to those elements.

There are two types of stylesheets: external stylesheets, where all the CSS code is in a separate file with a CSS extension, and an embedded stylesheet inside of the style tag which is usually placed in the head of the document. For this demonstration, we’ll use an in-page stylesheet. Soon you'll learn how to move CSS into an external stylesheet.

At the top of the page inside of the head section, add a `<style>` tag, followed by a closing `</style>` tag.

Then, in between those new tags, we'll create our firsts CSS rule. Once again, we're going to make the text of our `h1` element red.

To do that we use something called a CSS selector. A CSS selector lets us locate an element on the page by using the tag name of the element, the class name of element, or the ID of an element.

We'll start out with an "element" selector.

A selector looks like this:

**h1 {**

**}**

We place the selector followed by an opening curly brace, and then a closing curly brace. In between the braces, we place the CSS properties and their values.

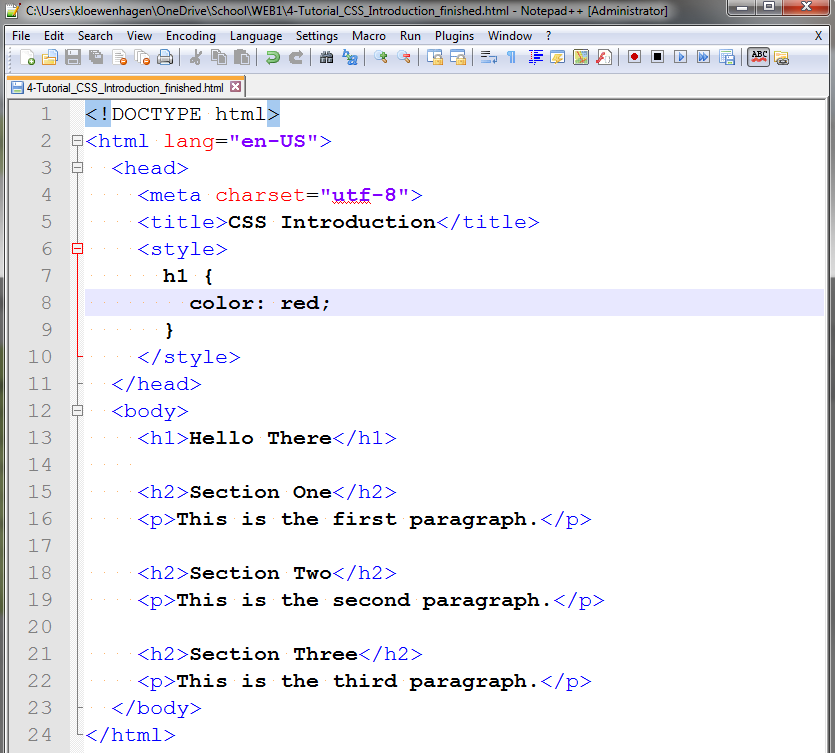
So, to make our heading red, we write this:

**h1 {**

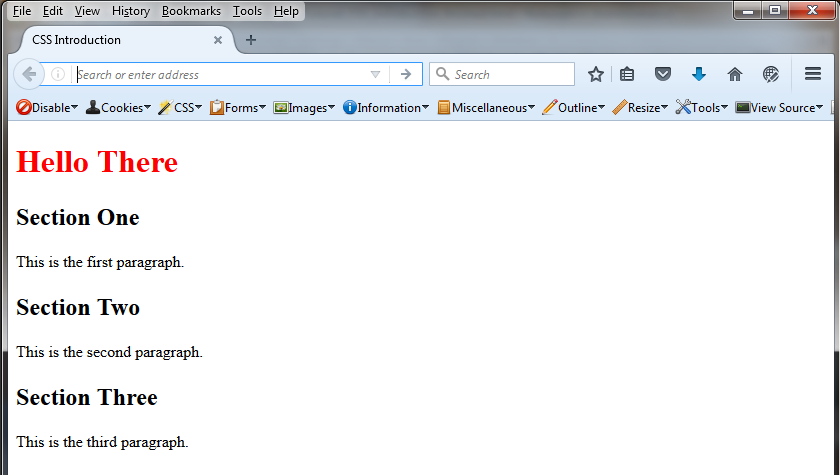
**color: red;**

**}**

Just like an HTML we're indenting the contents of this CSS rule so that it's clear that there's a hierarchical relationship here. You'll be required to follow this coding standard in your assignments.



When we save the page, and bring up this page in our web browser we see that this page has heading text that's red.



However, this CSS rule that we just created doesn't just apply to this one h1 element. It applies to all h1 elements on the page. Let me show you:

I'll create a new style inside the style tag that makes all the h2 tags blue:

**h2 {**

**color: blue;**

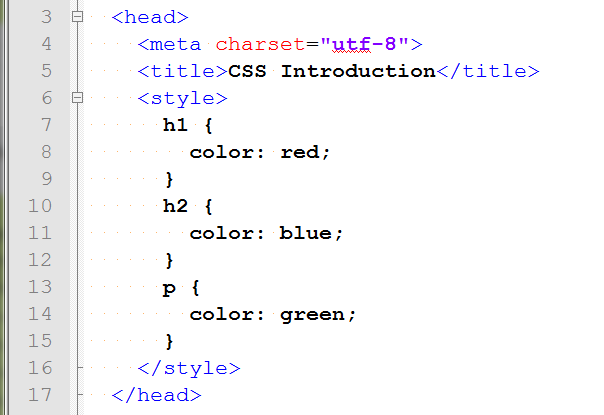
**}**

and all the p tags green:

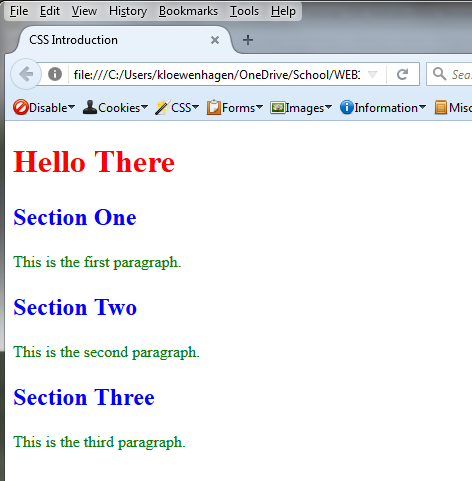
**p {**

**color: green;**

**}**



When we look at the page, we'll see that these elements, not just the first one the browser finds, get the treatment.



**ID Selectors**

We may want to apply style to a specific element on the page. If we have more than one h1 tag, we may need to say, "only do this one." One way to do that is to alter the HTML and add an ID attribute. I'm going to take the Section One h2 tag and add an ID to it called "section1":

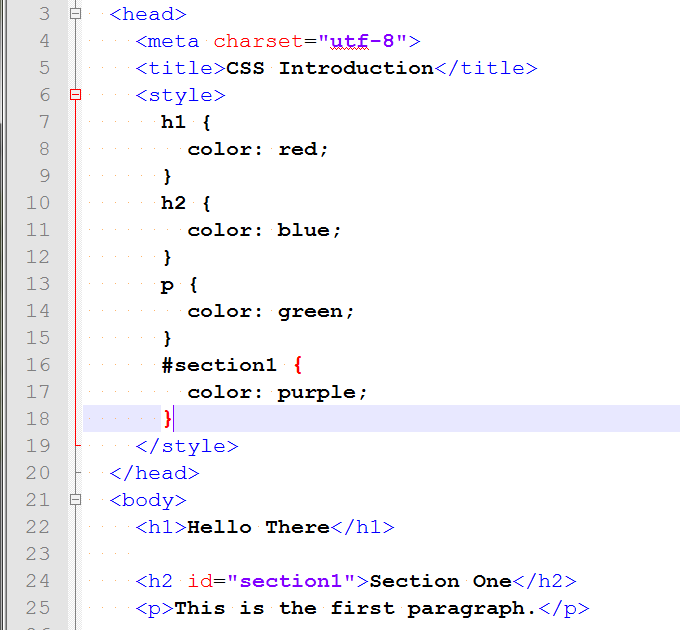
**<h2 id="section1">Section One</h2>**

Then in my stylesheet I can make a new rule for this. Instead of the element I want to find, I use an ID as my selector, and to do that I use the pound sign, followed by the name of the ID:

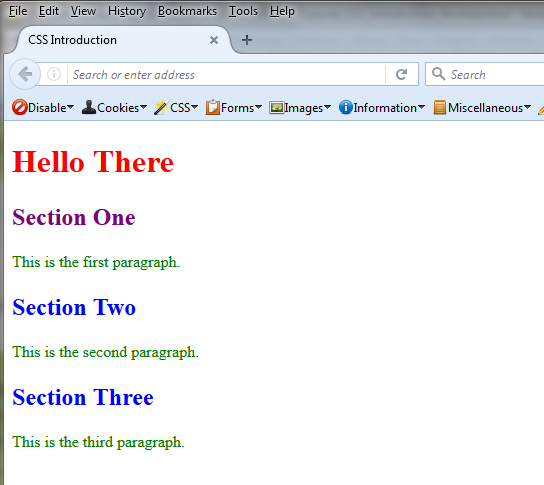
**#section1 {**

**color: purple;**

**}**

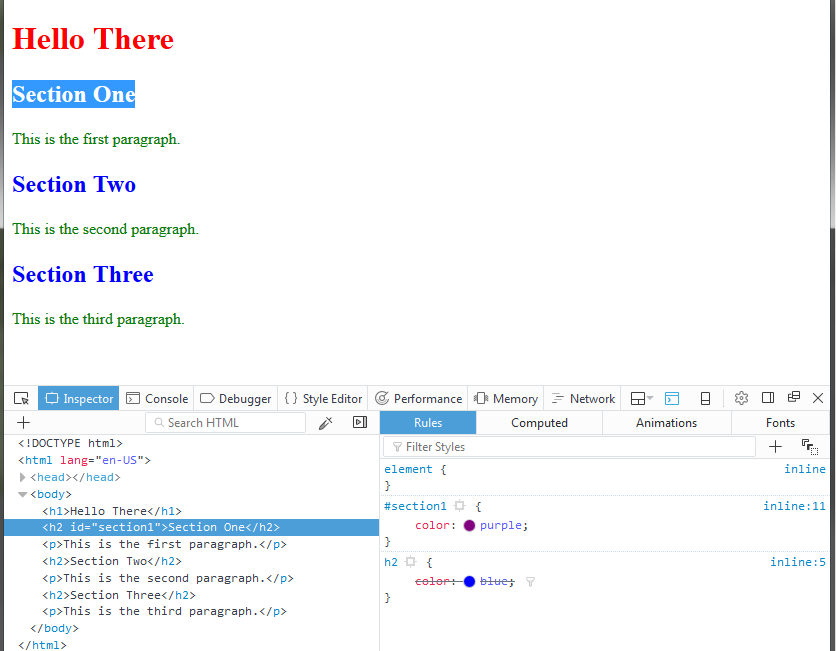


This makes the "Section One” header purple now.

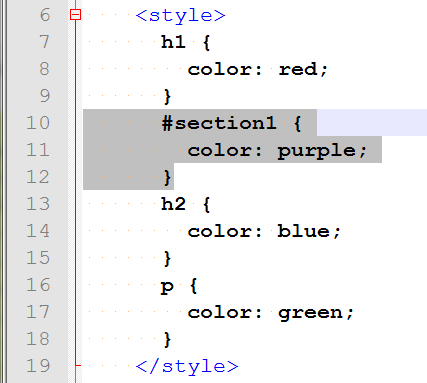


But something interesting is going on if we look at the element inspector: The original rule that colored this text blue is crossed out. And a rule that's crossed out is a rule that is overridden. In other words, the browser made the header blue, and THEN made it purple.

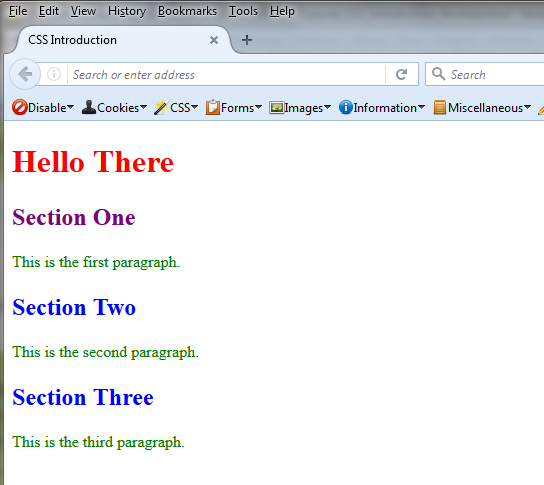
To view this: Right click on the purple “Section One” label on the Web page – Select “Inspect Element”. Then in the Inspector at the bottom of the screen click on the “Rules” tab.



Is it because we put the new "purple" declaration after the rules that made the blue headings? Let's find out: I'll move the purple rule higher in the stylesheet.

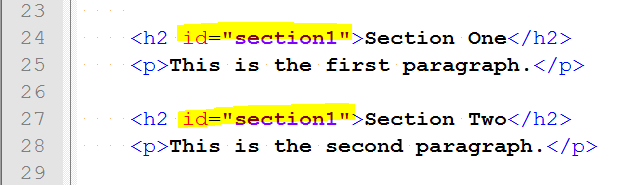


It doesn't look like it's made any difference at all. It turns out that *rules that select IDs are more important than rules that select tags.*

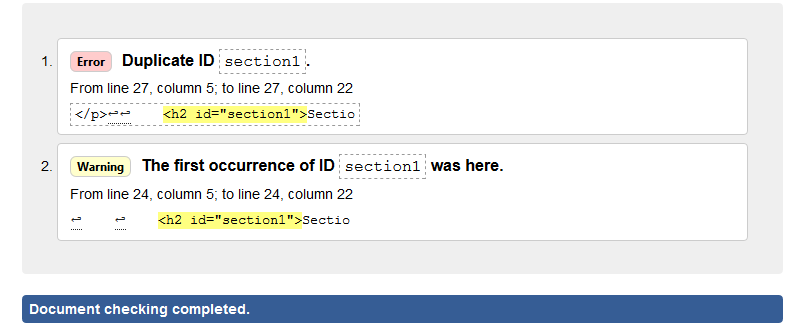


Now, IDs are great, but we can only use an ID once per page. I can't apply this same ID to any other elements on this page. Of course, I could use the same ID on a different page, but if I try to apply that same ID to any other element on this page, the page will no longer be valid. Let me show you:

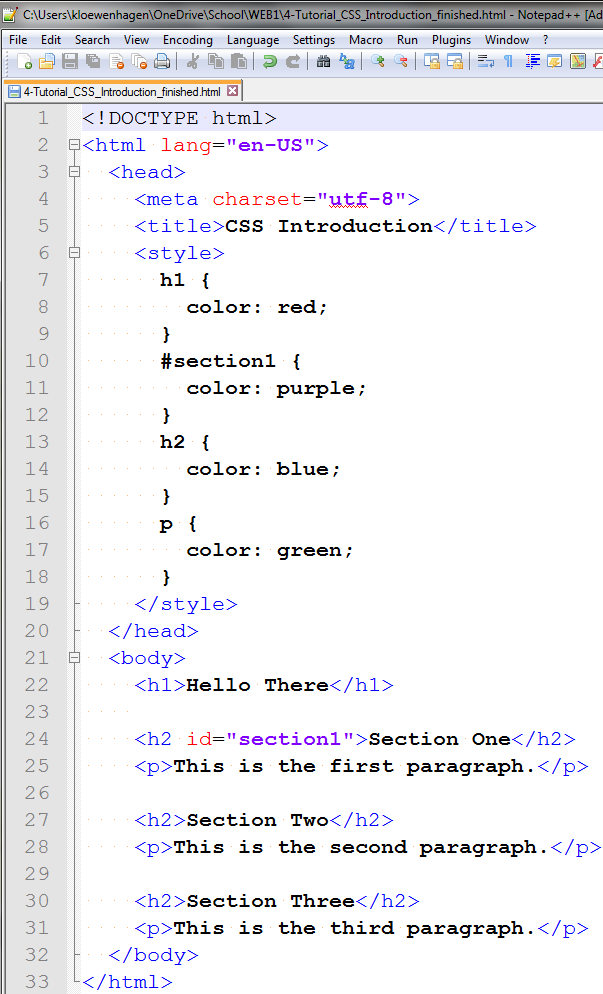
**Add id="section1" into “Section Two” h2 and save, then validate.**



See? We get an error that lets us know that IDs must be unique on the page.



**Remove the id=”section1” in the “Section Two” h2 tag.**



**Classes**

So, what if we wanted to target a group of elements and let them share styles, but we weren't necessarily interested in the element? Well, we can use a class. A class is another attribute we add to elements. Let's add the class "section" to all the h2 elements in the document:

**<h2 class="section">**

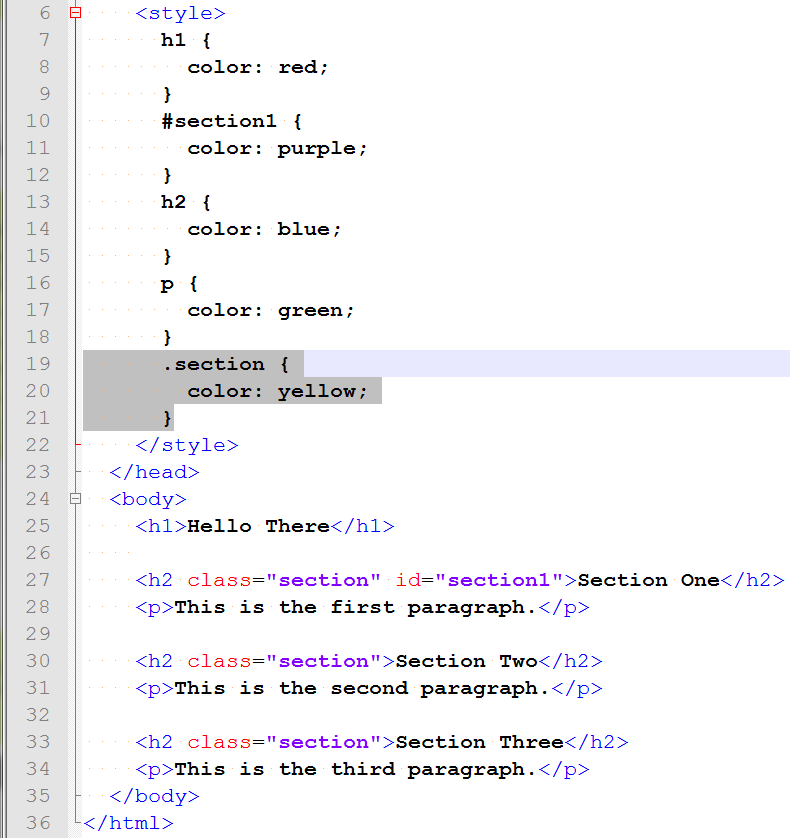
Now let's make a rule that turns these yellow instead of green:

**.section{**

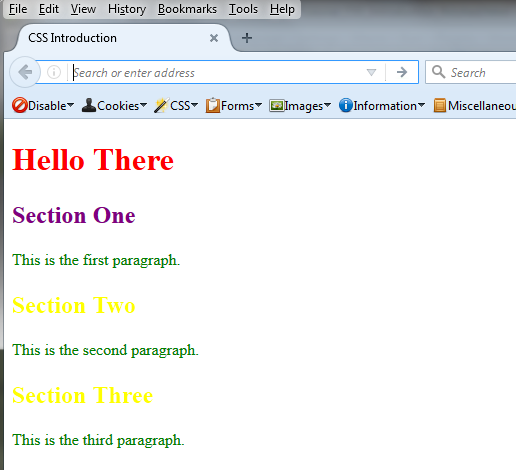
**color: yellow;**

**}**

This is a class selector.



A class selector starts with a period. Now when we look at the page these two headings are yellow, except for the first one, which is purple. Why's that? Well, the ID! IDs are more important than classes, too.

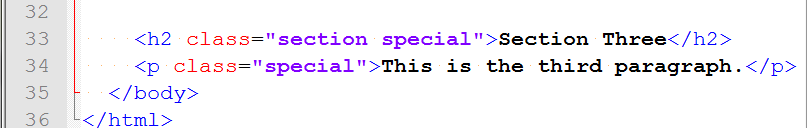


Let's add another class to the “Section Three” h2. We can have multiple classes on an element:

**<h2 class="section special">**

Let's also add this new "special" class to the paragraph below that header:

**<p class="special">**

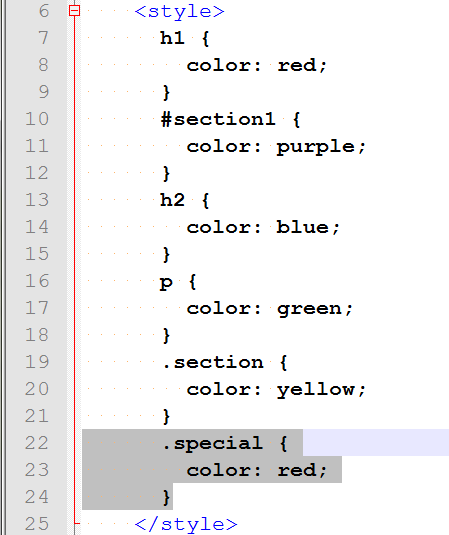


Then let's style it so that the text is red:

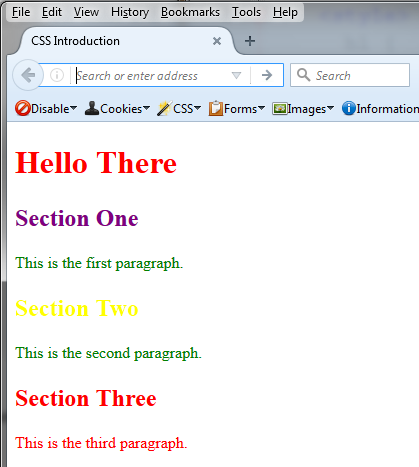
**.special {**

**color: red;**

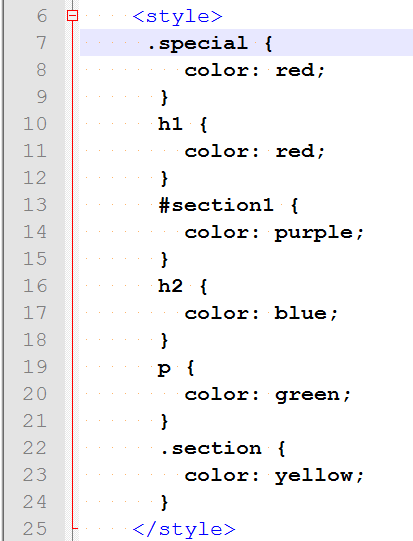
**}**



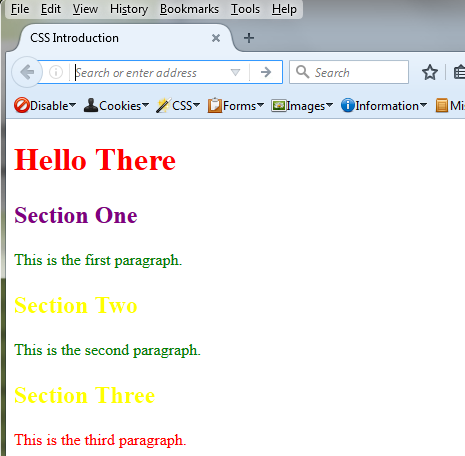
When we save the page, and review the page in the browser, the last header and last paragraph are both red.



What happens if we move the new CSS rule to the top of the stylesheet?



Well it looks like the yellow rule overrode it! IDs are more important than classes, but classes are not more important than other classes. So, the browser just uses the one that's lower in the CSS.



This order of importance is called "specificity." The most specific style rule will get applied. Understanding specificity is vital to being able to do anything with CSS.

Styles defined directly on the element, in-line styles, are the most specific. That's another reason we don't use that.

So, then IDs are the most important after that. Then classes, and then elements.

But there's one way we can force a rule to run:

**color: red !important;**

The !important declaration bumps the rule all the way to the top of the specificity list. However, using this in your own CSS is an indication that you don't understand CSS very well. This is a tool of last resort, when you cannot change the original stylesheet. Don't use this in your own work; fix the underlying problem instead.

**Wrapping Up**

This just touches the basics. Practice adjusting elements by using element selectors, ID selectors, and class selectors, and make sure you understand how specificity works!

