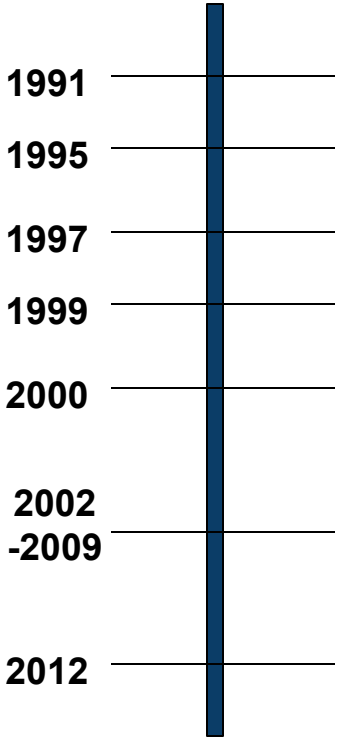
HTML and CSS

By: namth4

Introduction to HTML5

History of HTML



HTML first published

HTML 2.0

HTML 3.2

HTML 4.01

XHTML 1.0

XHTML 2.0

HTML5

After HTML 4.01 was released, focus shifted to XHTML and its stricter standards.

XHTML 2.0 had even stricter standards than 1.0, rejecting web pages that did not comply. It fell out of favor gradually and was abandoned completely in 2009.

HTML5 is much more tolerant and can handle markup from all the prior versions.

Though HTML5 was published officially in 2012, it has been in development since 2004.

What is HTML5?

- HTML5 is the newest version of HTML, only recently gaining partial support by the makers of web browsers.
- It incorporates all features from earlier versions of HTML, including the stricter XHTML.
- It adds a diverse set of new tools for the web developer to use.
- It is still a work in progress. No browsers have full HTML5 support. It will be many years – perhaps not until

Goals of HTML5

- Support all existing web pages. With HTML5, there is no requirement to go back and revise older websites.
- Reduce the need for external plugins and scripts to show website content.
- Improve the semantic definition (i.e. meaning and purpose) of page elements.
- Make the rendering of web content universal and independent of the device being used.

New Elements in HTML5

<article>	<figcaption>	<progress>
<aside>	<footer>	<section>
<audio>	<header>	<source>
<canvas>	<hgroup>	<svg>
<datalist>	<mark>	<time>
<figure>	<nav>	<video>

These are just some of the new elements introduced in HTML5. We will be exploring each of these during this course.

Other New Features in HTML5

- Built-in audio and video support (without plugins)
- Enhanced form controls and attributes
- The Canvas (a way to draw directly on a web page)
- Drag and Drop functionality
- Support for CSS3 (the newer and more powerful version of CSS)
- More advanced features for web developers, such as data storage and offline applications.

First Look at HTML5

Remember the DOCTYPE declaration from XHTML?

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

In HTML5, there is just one possible DOCTYPE declaration and it is simpler:

```
<!DOCTYPE html>
```

Just 15 characters!

The DOCTYPE tells the browser which type and version of document to expect. This should be the last time the DOCTYPE is ever changed. From now on, all future versions of HTML will use this same simplified declaration.

The <html> Element

This is what the <html> element looked like in XHTML:

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
   lang="en">
```

Again, HTML5 simplifies this line:

```
<html lang="en">
```

The **lang** attribute in the <html> element declares which language the page content is in. Though not strictly required, it should always be specified, as it can assist search engines and screen readers.

Each of the world's major languages has a two-character code, e.g. Spanish = "es", French = "fr", German = "de", Chinese = "zh", Arabic = "ar".

The <head> Section

Here is a typical XHTML <head> section:

```
<head>
  <meta http-equiv="Content-type" content="text/html; charset=UTF-8" />
  <title>My First XHTML Page</title>
  <link rel="stylesheet" type="text/css" href="style.css" />
</head>
```

And the HTML5 version:

```
<head>
  <meta charset="utf-8">
  <title>My First HTML5 Page</title>
  <link rel="stylesheet" href="style.css">
</head>
```

Notice the simplified character set declaration, the shorter CSS stylesheet link text, and the removal of the trailing slashes for these two lines.

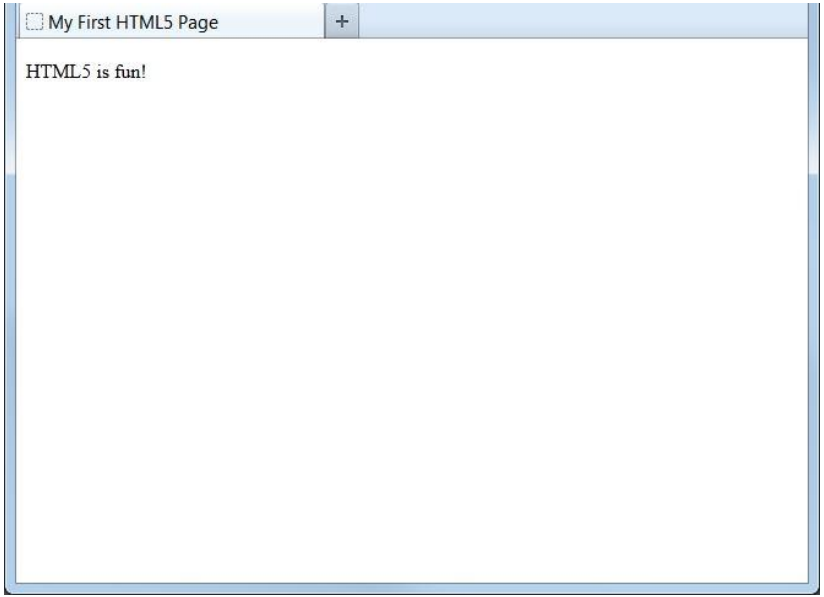
Basic HTML5 Web Page

Putting the prior sections together, and now adding the <body> section and closing tags, we have our first complete web page in HTML5:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>My First HTML5 Page</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <p>HTML5 is fun!</p>
```

Let's open this page in a web browser to see how it looks…

Viewing the HTML5 Web Page



Even though we used HTML5, the page looks exactly the same in a web browser as it would in XHTML. Without looking at the source code, web visitors will not know which version of HTML the page was created with.

Introduction to CSS

What is CSS?

- CSS ("Cascading Style Sheets") determines how the elements in our XHTML documents are displayed and formatted.
- By using CSS, we separate the content of a web page from the presentation (format and styling) of that content.
- CSS enables us to make all pages of our website look similar and consistent.
- The power of CSS is that it allows us to make site-wide formatting changes by making edits to a single file.

Three Ways to Use CSS

We can add CSS code in any combination of three different ways:

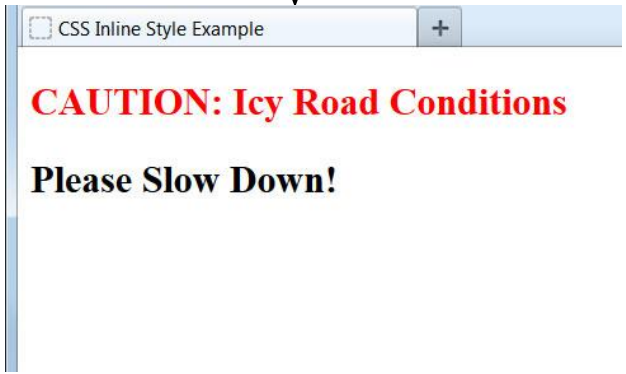
1. Inline Style - CSS code is placed directly into an XHTML element within the <body> section of a web page.
2. Internal Style Sheet - CSS code is placed into a separate, dedicated area within the <head> section of a web page.
3. External Style Sheet - CSS code is placed into a separate computer file and then linked to a web page.

Let's take a look now at examples of each of these methods.

Inline Style

To define an inline CSS style, we simply add the **style** attribute to an XHTML element with the CSS declaration as the attribute value:

```
<h2 style="color:red;">CAUTION: Icy Road Conditions</h2>
<h2>Please Slow Down!</h2>
```



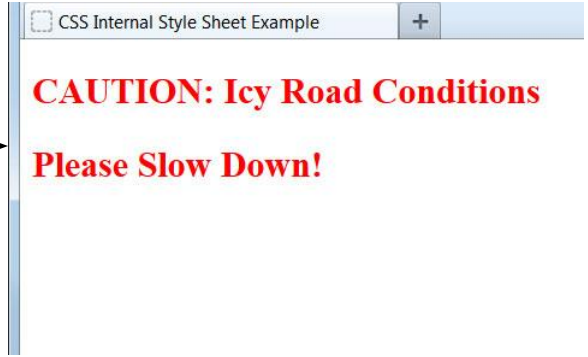
An inline style declaration is highly specific and formats just one element on the page. No other elements, including other <h2> elements on the page, will be affected by this CSS style.

Since inline styles have limited scope and do not separate content from presentation, their use is generally discouraged. We won't be using inline styles much in this class.

Internal Style Sheet

To use an internal CSS style sheet, we add a <style> section within the <head> of the page. All our CSS declarations go within this section:

```
<head>
  ...
  <style type="text/css">
    h2 {color:red;}
  </style>
</head>
<body>
  <h2>CAUTION: Icy Road Conditions</h2>
  <h2>Please Slow Down!</h2>
</body>
```



Styles declared in the internal style sheet affect all matching elements on the page. In this example, all <h2> page elements are displayed in the color red.

Since formatting declarations are entirely in the <head> section, away from the actual page content, internal CSS style sheets do a much better job than inline styles at separating content from presentation.

External Style Sheet

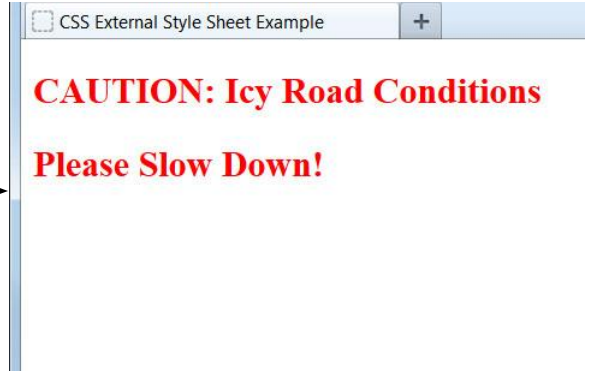
To use an external CSS style sheet, we create a new file (with a .css extension) and write our style declarations into this file. We then add a <link> element into our HTML file, right after the opening <head> tag:

style.css (separate file):

```
h2 {color:red;}
```

example.html file:

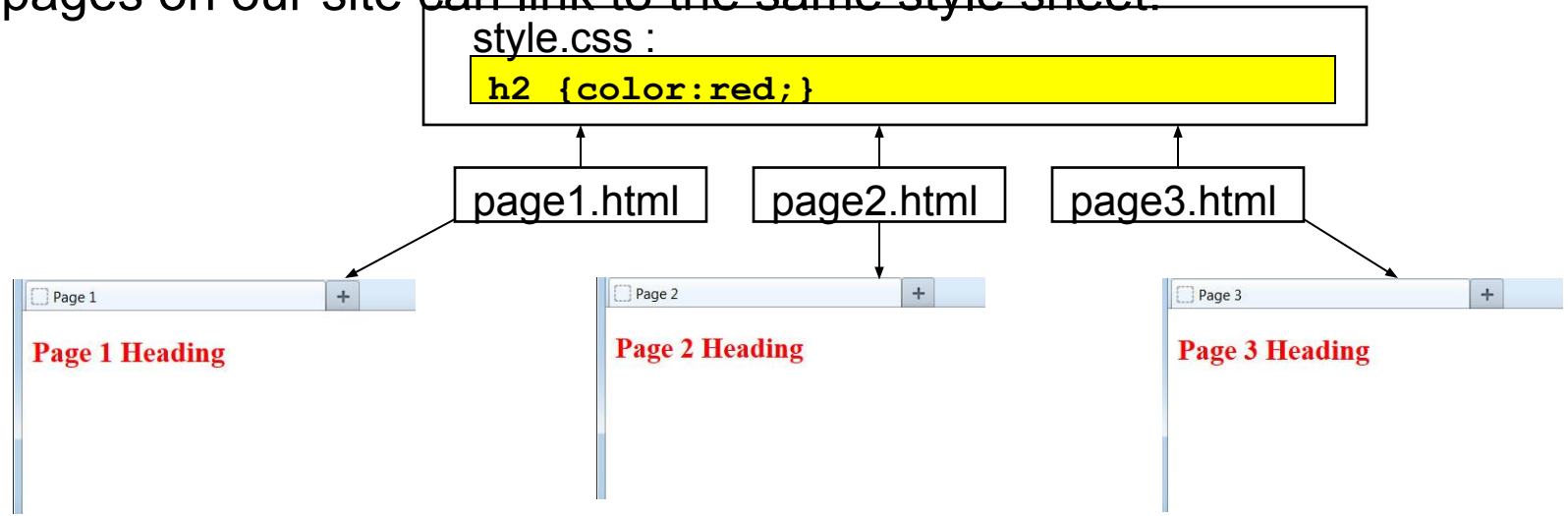
```
<head>
  <link rel="stylesheet" type="text/css"
    href="style.css" />
  ...
</head>
<body>
  <h2>CAUTION: Icy Road Conditions</h2>
  <h2>Please Slow Down!</h2>
</body>
```



The <link> element instructs the browser to load the external file specified by the href attribute and to apply the CSS style declarations contained there.

Benefit of External Style Sheet

The real power of using an external style sheet is that multiple web pages on our site can link to the same style sheet:



Styles declared in an external style sheet will affect all matching elements on all web pages that link to the style sheet. By editing the external style sheet, we can make site-wide changes (even to hundreds of pages) instantly.

Internal vs. External Style Sheets

Internal Style Sheets:

- are appropriate for very small sites, especially those that have just a single page.
- might also make sense when each page of a site needs to have a completely different look.

External Style Sheets:

- are better for multi-page websites that need to have a uniform look and feel to all pages.
- make for faster-loading sites (less redundant code).
- allow designers to make site-wide changes quickly and easily.

External style sheets create the furthest separation between content and presentation. For this reason - and the others listed above - we'll consider external style sheets to be the best option when creating a new site.

CSS Terminology and Syntax:

Now let's take a closer look at how we write CSS code. The correct syntax of a CSS declaration is: **selector {property:value;}**

```
p {color:red;}
```

selector

property

value

Internal and external style sheets use this identical CSS syntax. Internal style sheets must use the opening and closing <style> tags to surround the CSS code, while external style sheets do not use the <style> element.

A semicolon must be placed after each CSS declaration. Omitting this semicolon is the single most common mistake made by those learning CSS.

Setting Multiple Properties

We can define as many properties as we wish for a selector:

```
p {color:red;font-style:italic;text-align:center;}
```

In this example, all text within paragraph elements will show in red italics that is centered on the page.

```
p {
   color: red;
   font-style: italic;
   text-align: center;
}
```

Just as with HTML, browsers ignore space characters in CSS code. Many designers take advantage of this fact by placing the opening and closing curly brackets on their own dedicated lines. Each of the property and value pairings are placed on their own indented line, with a space after the colon. This makes the code far easier to read.

CSS Text Properties

The following properties can be specified for any element that contains text, such as <h1> through <h6>, <p>, <ol>, <ul>, and <a>:

Property	Some Possible Values
text-align:	center, left, right, justify
text-decoration:	underline, line-through, blink
color:	blue, green, yellow, red, white, etc.
font-family:	Arial, Verdana, "Times New Roman"
font-size:	large, 120%, 20px (pixels)
font-weight:	bold, normal

The actual list of available properties and values is quite long, but the ones listed above are the most common for formatting text via CSS.

How Browsers Process CSS

- A web browser will process all CSS code it encounters, even if it is from all three methods.
- For example, an external style sheet could define the font of a heading, an internal style sheet could specify the font size of the heading, and an inline style could italicize the heading. All three would be applied.

Browsers need a consistent way of settling these formatting conflicts in a consistent fashion. That is where the "cascade" of cascading style sheets comes into effect.

What Does "Cascading" Mean?

We use the term "cascading" because there is an established order of priority to resolve formatting conflicts:

1. Inline style (highest priority)
2. Internal style sheet (second priority)
3. External style sheet (third priority)
4. Web browser default (only if not defined elsewhere)

For each XHTML element, the browser will see which styles are defined inline and from internal and external style sheets. For any conflicts detected, it will use this priority system to determine which format to display on the page.

In the prior example, the heading text would display in the color specified by the inline style, which outranks all the others.

If multiple, conflicting styles are defined in the same style sheet, only the final one will be applied. Be careful, as this is another common mistake committed by beginners.

Introduction to Sass, SCSS

Sass/SCSS

Sass is an acronym and typically denotes; Syntactically Awesome Stylesheets and SCSS is simply ; Sassy CSS. They are both preprocessing languages that are compiled to CSS, but they use different technical syntaxes.

```
//Sass Example
$text-color: #333333
body
color: $text-color
```

```
//SCSS Example
$text-color: #333333;
body {
color: $text-color;
}
```

Less

Just like Sass, Less is a CSS preprocessor and it allows web developers to build modular, scalable, and more manageable CSS styles. Remember that preprocessors extend the CSS language and add features that allow variables, mixins, and other functions that enable you to easily maintain CSS. The compiler is what turns Less code into standard CSS that a web browser can read and process. Less is written in JavaScript so it runs inside NodeJS, in the browser, and inside Rhino. There are also many third-party tools that allow you to compile your files in Less and watch for changes.

Less

```
//Less Variable
@link-color: blue;


//SCSS Variable
$link-color: blue;
```

Mixins also have a small syntax differences between Less and Sass.

```
//Less mixin for rounding borders
.round-borders (@radius) {
border-radius: @radius;
}
//Add the round-borders mixin with Less
.box { round-borders(0.5rem); }
```

The same round-border mixin in Sass looks as follows:

```
//Sass mixin for rounding corners
@mixin round-borders (@radius) {
border-radius: $radius;
}
//Add the round-borders mixin with Sass
.box { @include round-borders(0.5rem); }
```

Initially, Sass was part of another preprocessor called Haml, designed and written by Ruby developers. Because of that, Sass stylesheets were using a Ruby-like syntax with no braces, no semi-colons and a strict indentation, like this:

```
// Variable
!primary-color= hotpink

// Mixin
=border-radius(!radius)
    -webkit-border-radius= !radius
    -moz-border-radius= !radius
    border-radius= !radius

.my-element
    color= !primary-color
    width= 100%
    overflow= hidden

.my-other-element
    +border-radius(5px)
```

As you can see, this is quite a change compared to regular CSS! Even if you're a Sass (the preprocessor) user, you can see this is pretty different from what we are used to. The variable sign is `!` and not `$`, the assignment sign is `=` and not `:`. Pretty weird.
But that's how Sass looked like until version 3.0 arrived in May 2010, introducing a whole new syntax called SCSS for *Sassy CSS*. This syntax aimed at closing the gap between Sass and CSS by bringing a CSS-friendly syntax.

```
// Variable
$primary-color: hotpink;

// Mixin
@mixin border-radius($radius) {
    -webkit-border-radius: $radius;
    -moz-border-radius: $radius;
    border-radius: $radius;
}

.my-element {
    color: $primary-color;
    width: 100%;
    overflow: hidden;
}

.my-other-element {
    @include border-radius(5px);
}
```

Homework