

Introduction to HTML Training

Crash Course in CSS

Lesson 1, Activity 2: **Benefits of Cascading Style Sheets**

In HTML 4.0, most HTML formatting elements have been deprecated, meaning that, although they are still supported by browsers, the World Wide Web Consortium (W3C) recommends that they no longer be used. Web designers are to use CSS instead.

The major benefits of CSS are:

1. Cleaner code
 - Easier to maintain
 - Speedier download
 - Better for search engine optimization
2. Modular code
 - Style rules can be applied to multiple pages
 - Consistency of design
 - Easier to maintain
3. Design Power
 - Precise control of position, size, margins, etc.
4. Division of labor
 - Developers develop / Designers design
5. Better Accessibility ([W3C Info on CSS's Accessibility Features](#))
 - No need to misuse tags (e.g, `<blockquote>` for formatting)
 - No need for invisible images for positioning
 - Users' style sheets override authors' styles

Lesson 1, Activity 3: CSS Rules

CSS *rules* are statements that define the style of an element or group of elements. The syntax is as follows:

Syntax

```
selector {property:value; property:value; property:value;}
```

Each *property:value* pair is a *declaration*. Multiple declarations are separated by semi-colons. The *selector* defines which elements are affected by the rule. Take a look at the following rule.

```
p {  
  color:darkgreen;  
  font-family:Verdana;  
  font-size:10pt;  
}
```

This rule specifies that all paragraph text should be darkgreen and use a 10-point Verdana font.

CSS Comments

Comments in CSS begin with "/*" and end with "*/". See the example below.

```
p {  
  color:red; /* All paragraphs should be red */  
}
```

If you are looking for a CSS editor, make sure to check out our partners at Blumentals: [Rapid CSS](#)

Lesson 1, Activity 4: Selectors

Selectors identify the element(s) affected by the CSS rule. There are several different types of selectors:

- Type
- Descendant
- Child
- Class
- ID
- Attribute
- Universal

Type Selectors

Type selectors specify elements by tag name and affect every instance of that element type. Looking again at a previous example:

```
p {  
  color:darkgreen;  
  font-family:Verdana;  
  font-size:10pt;  
}
```

This rule specifies that the text of every `<p>` element should be darkgreen and use a 10-point Verdana font.

Descendant Selectors

Descendant selectors specify elements by ancestry. Each "generation" is separated by a space. For example, the following rule states that `` tags within `<p>` tags should have red text.

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

With descendant selectors generations can be skipped. In other words, the code above does not require that the `` tag is a direct child of the `<p>` tag.

Child Selectors

Child selectors specify a direct parent-child relationship

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

The `>` sign indicates that the rule only gets applied when the `` tag is a direct child of the `<p>` tag. So, in the first case below, it *does* get applied, but it *does not* in the second case:

```
<p>Life is <strong>good</strong>!!</p>
```

```
<p>Life is <em><strong>really good</strong></em>!!</p>
```

Class Selectors

Almost all elements can take the `class` attribute, which assigns a class name to an element. Class names are created in style sheet with rules defined for class selectors. Class selectors begin with a dot and have arbitrary names. For example, the following rule creates a class called "warning," which makes the text of all elements of that class bold and red.

```
.warning {
  font-weight:bold;
  color:#ff0000;
}
```

Following are a couple of examples of elements of the warning class.

```
<h1 class="warning">WARNING</h1>
<p class="warning">Don't go there!</p>
```

If the class selector is preceded by an element name, then that selector only applies to the specified type of element. To illustrate, the following two rules indicate that `h1` elements of the class "warning" will be underlined, while `p` elements of the class "warning" should not be.

```
h1.warning {
  color:#ff0000;
  text-decoration:underline
}
p.warning {
  color:#ff0000;
  font-weight:bold;
}
```

Because both rules indicate that the color should be red (`#ff0000`), this could be rewritten as follows.

```
.warning {
  color:#ff0000;
}
h1.warning {
  text-decoration:underline;
}
p.warning {
  font-weight:bold;
}
```

Note that you can assign an element any number of classes simply by separating the class names with spaces like this:

Syntax

```
<div class="class1 class2 class3">...
```

ID Selectors

As with the `class` attribute, almost all elements can take the `id` attribute, which is used to uniquely identify an element on the page. ID selectors begin with a pound sign (`#`) and have arbitrary names. The following rule will indent the element with the `"maintext"` `id` 20 pixels from the left and right.

```
#mainText {
  margin-left:20px;
  margin-right:20px;
}

<div id="mainText">
  This is the main text of the page...
</div>
```

Attribute Selectors

Attribute selectors specify elements that contain a specific attribute. They can also specify the value of that attribute.

The following selector affects all links with a `target` attribute.

```
a[target] {
  color:red;
}
```

The following selector would only affect links whose `target` attribute is `"_blank"`.

```
a[target="_blank"] {  
  color:red;  
}
```

The Universal Selector

The universal selector is an asterisk (*). It matches every element.

```
* {  
  color:red;  
}
```

Grouping

Selectors can share the same declarations by separating them with commas. The following rule will underline all `i` elements, all elements of the class `"warning"` and the element with the `id` of `"important."`

```
i, .warning, #important {  
  text-decoration: underline;  
}
```

For further information on CSS selectors see the [W3C site](#).

Lesson 1, Activity 5: Precedence of Selectors

In the event that rules conflict:

- The rule with the more specific selector takes precedence.
- In the event that two selectors have the same specificity, the rule specified later in the document takes precedence.

Determining a Selector's Specificity

Note: this is important only for debugging; **you'll almost never have to use it**, except when you have conflicting styling declarations and can't figure out why one takes precedence over the other.

Imagine your selectors are stacked as follows with the ones on top having the highest specificity:

Declarations in the style attribute	
<code>style=""</code>	(1)
Selectors with id attributes	
<code>h1#foo {}</code>	(2)
Selectors with other attributes	
<code>h1.foo {}</code> <code>a[target] {}</code>	(3)
Selectors with element names but no attributes	
<code>h1 {}</code>	(4)
The universal selector	
<code>* {}</code>	(5)

1. Declarations in the `style` attribute have no selector and have the highest precedence.
2. Selectors with `id` attributes (e.g, `h1#foo {}`) have the next highest precedence.
3. Selectors with other attributes (e.g., `h1.foo` and `a[target]`) or pseudo-classes (e.g, `a:hover`) have the next highest precedence.
4. Selectors with element names (e.g, `h1`) but no other attributes have the next highest precedence.

5. The universal selector (*) has the lowest precedence.

To figure out the exact specificity, follow this process:

1. Start with *0,0,0,0*.
2. If the declaration is found in the `style` attribute, change the first digit to 1, giving you *1,0,0,0*. In this case, you have the highest possible specificity and can stop calculating.
3. For each time the condition in level 2 is met, add 1 to the second digit.
 - For example, for `ol#foo li#bar` add 2 (1 for each id), giving you *0,2,0,0*.
 - Note: we're not counting when the conditions in the other levels are met here, just the condition for level 2.
4. For each time the condition in level 3 is met, add 1 to the third digit.
 - For example, for `ol#foo li#bar a[target]` add 1, giving you *0,2,1,0*.
 - Note: here we're only counting when the conditions in level 2 and 3 are met.
5. For each time the condition in level 4 is met, add 1 to the fourth digit.
 - For example, for `ol#foo li#bar a[target]` add 3 (1 for each element name), giving you *0,2,1,3*.
 - Note: here we're counting when the conditions in level 2, 3, and 4 are met.

When comparing two selectors' specificity, start with the left-most numbers. If one has a higher number than the other, than it is more specific. If they are the same, look to the next number and so on.

Lesson 1, Activity 8: The Cascade

Web designers can define style rules in three different places:

1. In an embedded style sheet.
2. In an external (or linked or imported) style sheet.
3. Inline in an element.

Embedded Style Sheets

Embedded style sheets appear in the `style` element in the head of an HTML page. The code below shows a page with an embedded style sheet.

Code Sample:

[CrashCourse/Demos/EmbeddedStyleSheet.html](#)

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Embedded Style Sheet</title>
<style type="text/css">
  .warning {color:#ff0000}
  h1.warning {text-decoration:underline}
  p.warning {font-weight:bold}
</style>
</head>
<body>
<h1 class="warning">WARNING</h1>
<p class="warning">Don't go there!</p>
</body>
</html>
```

As you can see, the `<style>` tag takes the `type` attribute, which is used to indicate that this is a CSS style sheet. This page will render as follows:



External Style Sheets

External style sheets are created in separate documents with a ".css" extension. An external style sheet is simply a listing of rules. It cannot contain HTML tags. [CrashCourse/Demos/Stylesheet.css](#) is an example of an external style sheet.

Code Sample:

CrashCourse/Demos/StyleSheet.css

```
.warning {color:#ff0000}
h1.warning {text-decoration:underline}
p.warning {font-weight:bold}
```

The above CSS file can be included in any number of HTML pages. The <link> tag, which goes in the head of an HTML page, is used to link to an external style sheet.

Code Sample:

CrashCourse/Demos/ExternalStyleSheet.html

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>External Style Sheet</title>
<link href="StyleSheet.css" rel="stylesheet" type="text/css">
</head>
<body>
<h1 class="warning">WARNING</h1>
<p class="warning">Don't go there!</p>
</body>
</html>
```

<link> Attributes

Attributed	Description
href	points to the location of the external style sheet
rel	must be set to "stylesheet" for linking style sheets
type	must be set to "text/css" for linking to cascading style sheets

There is no limit to the number of external style sheets a single HTML page can use. Also, external style sheets can be combined with embedded style sheets.

Inline Styles

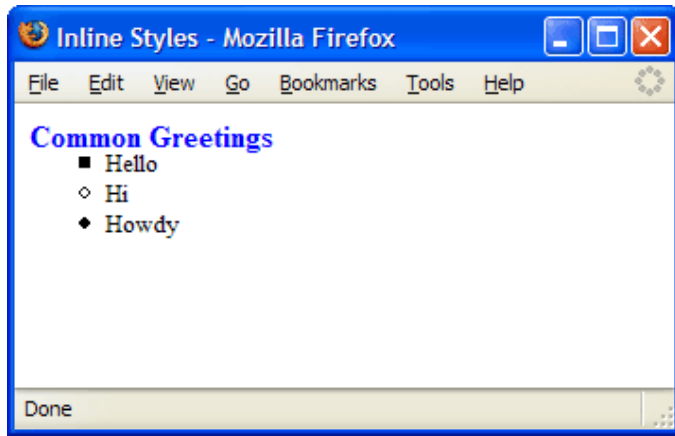
Inline styles are created by adding the `style` attribute to a tag. As with the `class` and `id` attributes, almost all elements can take the `style` attribute. The value of the style attribute is a list of one or more property-value pairs separated by semi-colons. The code sample below illustrates how inline styles are used.

Code Sample:

[CrashCourse/Demos/InlineStyles.html](#)

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Inline Styles</title>
</head>
<body>
<p style="color:blue; font-weight:bold; font-size:12pt">Common Greetings</p>
<ul style="margin-top:-20px; font-size:10pt">
  <li style="list-style-type:square">Hello</li>
  <li style="list-style-type:circle">Hi</li>
  <li style="list-style-type:disc">Howdy</li>
</ul>
</body>
</html>
```

This page will render as follows.

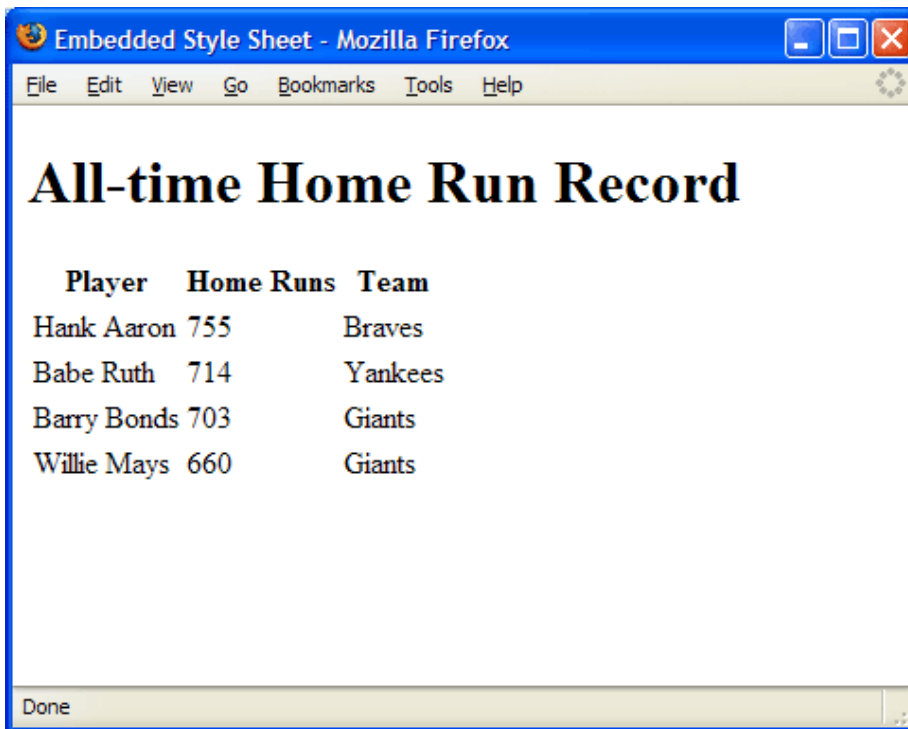


Lesson 1, Activity 10: Creating an Embedded Style Sheet

Duration: 20 to 30 minutes.

In this exercise, you will add an embedded style sheet to [CrashCourse/Exercises/EmbeddedStyleSheet.html](#). You will be adding several rules to a simple HTML file. Do not worry about learning the CSS properties and values at this point. This exercise is just to give you some practice creating a style sheet.

1. Open [CrashCourse/Exercises/EmbeddedStyleSheet.html](#) in a browser. It should look like this.



2. Open [CrashCourse/Exercises/EmbeddedStyleSheet.html](#) for editing.
3. Add a style block in the head of the page. Don't forget to include the type attribute.
4. Add a rule for the body element that contains the following property-value pairs:
 - background-image:url('Images/Baseball.gif');
 - background-repeat:repeat-x;
 - background-position:bottom;
 - background-attachment: fixed;
 - margin-left:50%;
 - margin-top:20px;
5. Add a rule for div elements that contains the following property-value pairs:
 - padding:10px;
 - border:10px groove red;
 - width:300px;
 - background-image:url('Images/YankeeStadium.gif');
 - margin-left:-170px;
6. Add a rule for h1 elements that contains the following property-value pairs:

- text-align:center;
- font-size: 12pt;
- color:#000099;
- margin-bottom:5px;
- text-decoration:underline;

7. Add a rule for table elements that contains the following property-value pairs:

- margin:5px;
- width:290px;

8. Add a rule for th elements that contains the following property-value pairs:

- padding:3px;

9. Add a rule for td elements that contains the following property-value pairs:

- padding-left:8px;
- padding-right:8px;
- border:1px solid #990000;
- background-color:#ffffcc;

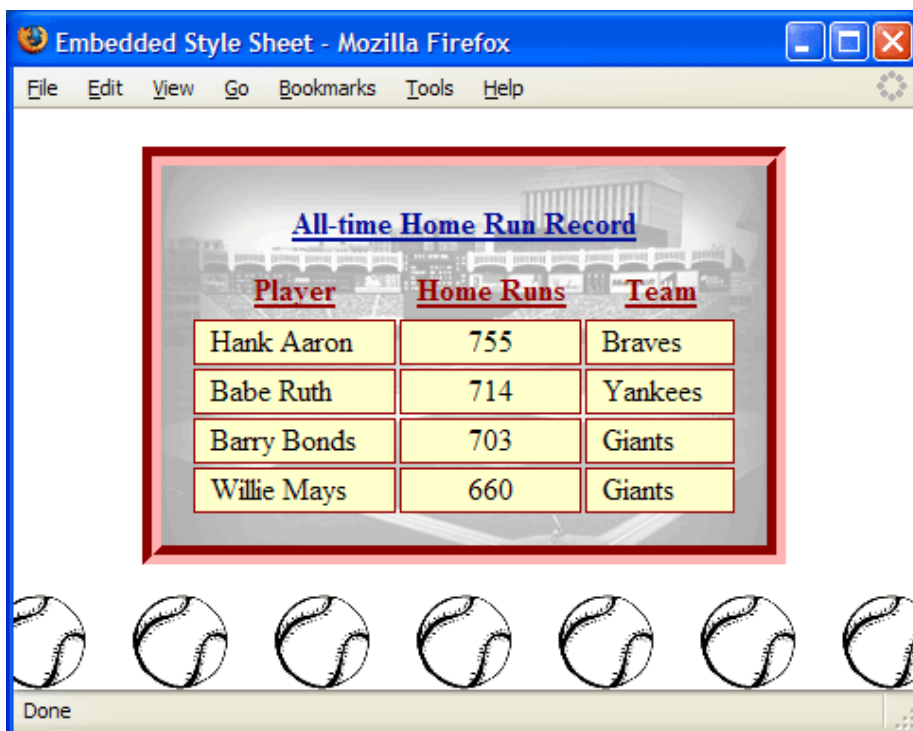
10. Assign an id of "trHeader" to the first table row and add a rule for this id that contains the following property-value pairs:

- text-decoration:underline;
- color:#990000;

11. Assign a class called "centerCell" to all of the center table data cells and add a rule for this class that contains the following property-value pairs:

- text-align:center;

12. Re-open [EmbeddedStyleSheet.html](#) in a browser. It should look like this:



Solution:

CrashCourse/Solutions/EmbeddedStyleSheet.html

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Embedded Style Sheet</title>
<style type="text/css">
  body
  {
    background-image: url('Images/Baseball.gif');
    background-repeat: repeat-x;
    background-position: bottom;
        background-attachment: fixed;
    margin-left: 50%;
    margin-top: 20px;
  }

  div
  {
    padding: 10px;
    border: 10px groove red;
    width: 300px;
    background-image: url('Images/YankeeStadium.gif');
    margin-left: -170px;
  }

  h1
  {
    text-align: center;
    font-size: 12pt;
    color: #000099;
    margin-bottom: 5px;
    text-decoration: underline;
  }

  table
  {
    margin: 5px;
    width: 290px;
  }

  th
  {
    padding: 3px;
  }

  td
  {
    padding-left: 8px;
    padding-right: 8px;
    border: 1px solid #990000;
```



```

    background-color: #ffffcc;
}

#trHeader
{
    text-decoration: underline;
    color: #990000;
}

.centerCell
{
    text-align: center;
}

</style>
</head>
<body>

<div>
<h1>All-time Home Run Record</h1>
<table>
<tr id="trHeader">
    <th>Player</th>
    <th>Home Runs</th>
    <th>Team</th>
</tr>
<tr>
    <td>Hank Aaron</td>
    <td class="centerCell">755</td>
    <td>Braves</td>
</tr>
<tr>
    <td>Babe Ruth</td>
    <td class="centerCell">714</td>
    <td>Yankees</td>
</tr>
<tr>
    <td>Barry Bonds</td>
    <td class="centerCell">703</td>
    <td>Giants</td>
</tr>
<tr>
    <td>Willie Mays</td>
    <td class="centerCell">660</td>
    <td>Giants</td>
</tr>
</table>
</div>
</body>
</html>

```

Lesson 1, Activity 11: Creating an External Style Sheet

Duration: 5 to 10 minutes.

In this exercise, you will extract the embedded stylesheet that you created in the last exercise to a new external style sheet.

1. Open [CrashCourse/Exercises/EmbeddedStyleSheet.html](#) in your editor and save it as [LinkedStyleSheet.html](#).
2. Create a new file and save it as [StyleSheet.css](#) in the same directory.
3. Move all the CSS rules from [LinkedStyleSheet.html](#) to [StyleSheet.css](#).
4. In [LinkedStyleSheet.html](#), remove the style block and add a link tag that points to [StyleSheet.css](#).
5. Open [LinkedStyleSheet.html](#) in a browser. It should look the same as [EmbeddedStyleSheet.html](#).

Solution:

[CrashCourse/Solutions/LinkedStyleSheet.html](#)

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Embedded Style Sheet</title>
<link href="StyleSheet.css" rel="stylesheet" type="text/css">
</head>
<body>

<div>
<h1>All-time Home Run Record</h1>
<table>
<tr id="trHeader">
  <th>Player</th>
  <th>Home Runs</th>
  <th>Team</th>
</tr>
<tr>
  <td>Hank Aaron</td>
  <td class="centerCell">755</td>
  <td>Braves</td>
</tr>
<tr>
  <td>Babe Ruth</td>
  <td class="centerCell">714</td>
  <td>Yankees</td>
</tr>
<tr>
  <td>Barry Bonds</td>
  <td class="centerCell">703</td>
  <td>Giants</td>
</tr>
<tr>
  <td>Willie Mays</td>
```

```

<td class="centerCell">660</td>
<td>Giants</td>
</tr>
</table>
</div>
</body>
</html>

```

Solution:

CrashCourse/Solutions/StyleSheet.css

```

body
{
    background-image: url('Images/Baseball.gif');
    background-repeat: repeat-x;
    background-position: bottom;
    background-attachment: fixed;
    margin-left: 50%;
    margin-top: 20px;
}

div
{
    padding: 10px;
    border: 10px groove red;
    width: 300px;
    background-image: url('Images/YankeeStadium.gif');
    margin-left: -170px;
}

h1
{
    text-align: center;
    font-size: 12pt;
    color: #000099;
    margin-bottom: 5px;
    text-decoration: underline;
}

table
{
    margin: 5px;
    width: 290px;
}

th
{
    padding: 3px;
}

td
{

```

```
padding-left: 8px;
padding-right: 8px;
border: 1px solid #990000;
background-color: #ffffcc;
}

#trHeader
{
    text-decoration: underline;
    color: #990000;
}

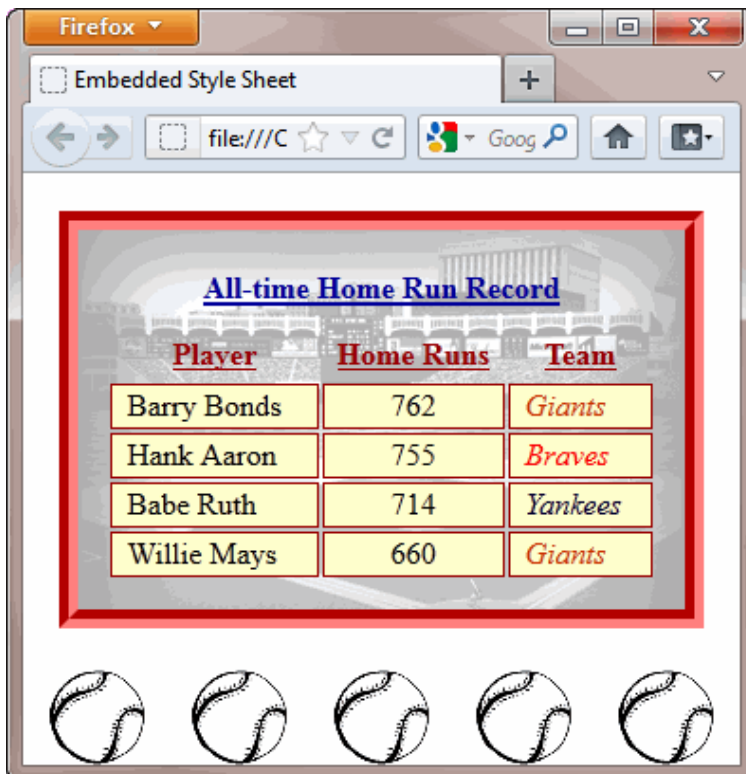
.centerCell
{
    text-align: center;
}
```

Lesson 1, Activity 12: Adding Inline Styles

Duration: 10 to 20 minutes.

In this exercise, you will add some inline styles to the page you have been working on.

1. Open [CrashCourse/Exercises/LinkedStyleSheet.html](#) and save it as [InlineStyles.html](#).
2. Add an inline style to the table data cell containing the word "Braves". The style should contain the following property-value pairs:
 - font-style: italic;
 - color: #ff0000;
3. Add an inline style to the table data cell containing the word "Yankees". The style should contain the following property-value pairs:
 - font-style: italic;
 - color: #000033;
4. Add inline styles to the table data cells containing the word "Giants". The style should contain the following property-value pairs:
 - font-style: italic;
 - color: #cc3300;
5. Open [InlineStyles.html](#) in a browser. It should look like this:



Note: In the image above, the browser has been resized to minimize white space. When viewed in a maximized window, there will be considerable white space between the table and the baseballs along the bottom edge.

Solution:

CrashCourse/Solutions/InlineStyles.html

```

<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Embedded Style Sheet</title>
<link href="StyleSheet.css" rel="stylesheet" type="text/css">
</head>
<body>

<div>
<h1>All-time Home Run Record</h1>
<table>
<tr id="trHeader">
  <th>Player</th>
  <th>Home Runs</th>
  <th>Team</th>
</tr>
<tr>
  <td>Barry Bonds</td>
  <td class="centerCell">762</td>
  <td style="font-style: italic; color: #cc3300">Giants</td>
</tr>
<tr>
  <td>Hank Aaron</td>
  <td class="centerCell">755</td>
  <td style="font-style: italic; color: #ff0000">Braves</td>
</tr>
<tr>
  <td>Babe Ruth</td>
  <td class="centerCell">714</td>
  <td style="font-style: italic; color: #000033">Yankees</td>
</tr>
<tr>
  <td>Willie Mays</td>
  <td class="centerCell">660</td>
  <td style="font-style: italic; color: #cc3300">Giants</td>
</tr>
</table>
</div>
</body>
</html>

```

Lesson 1, Activity 13: Media Types

Styles can be defined for different media. For example, you may want to style a page one way for viewing with a browser and a different way for viewing in print. The media type is defined in the `<link>` tag for external style sheets and in the `<style>` tag for embedded style sheets.

Syntax

```
<link href="stylesheet.css" rel="stylesheet" type="text/css" media="screen">

<style type="text/css" media="all">
  /* rules */
</style>
```

If the media is undefined then the style rules will apply to all media. Possible values for media are:

- all
- aural
- braille
- embossed
- handheld
- print
- projection
- screen
- tty
- tv

The following code sample illustrates how you can use CSS to design for different media.

Code Sample:

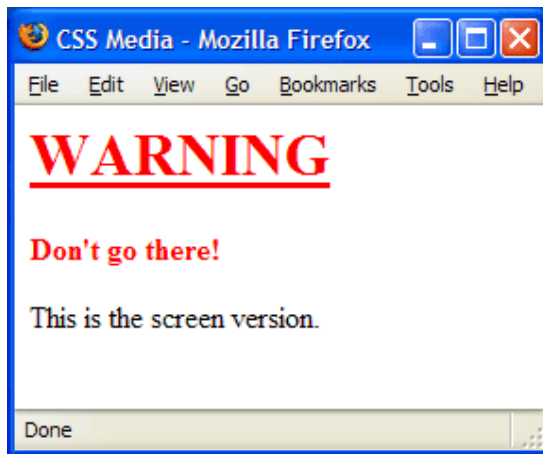
[CrashCourse/Demos/Media.html](#)

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>CSS Media</title>
<style type="text/css" media="screen">
  .warning {color:#ff0000}
  h1.warning {text-decoration:underline}
  p.warning {font-weight:bold}
  .printDisplay {display:none}
</style>
<style type="text/css" media="print">
  .warning {color:#660000;}
  h1.warning {text-decoration:underline; font-size:1in;}
</style>
```

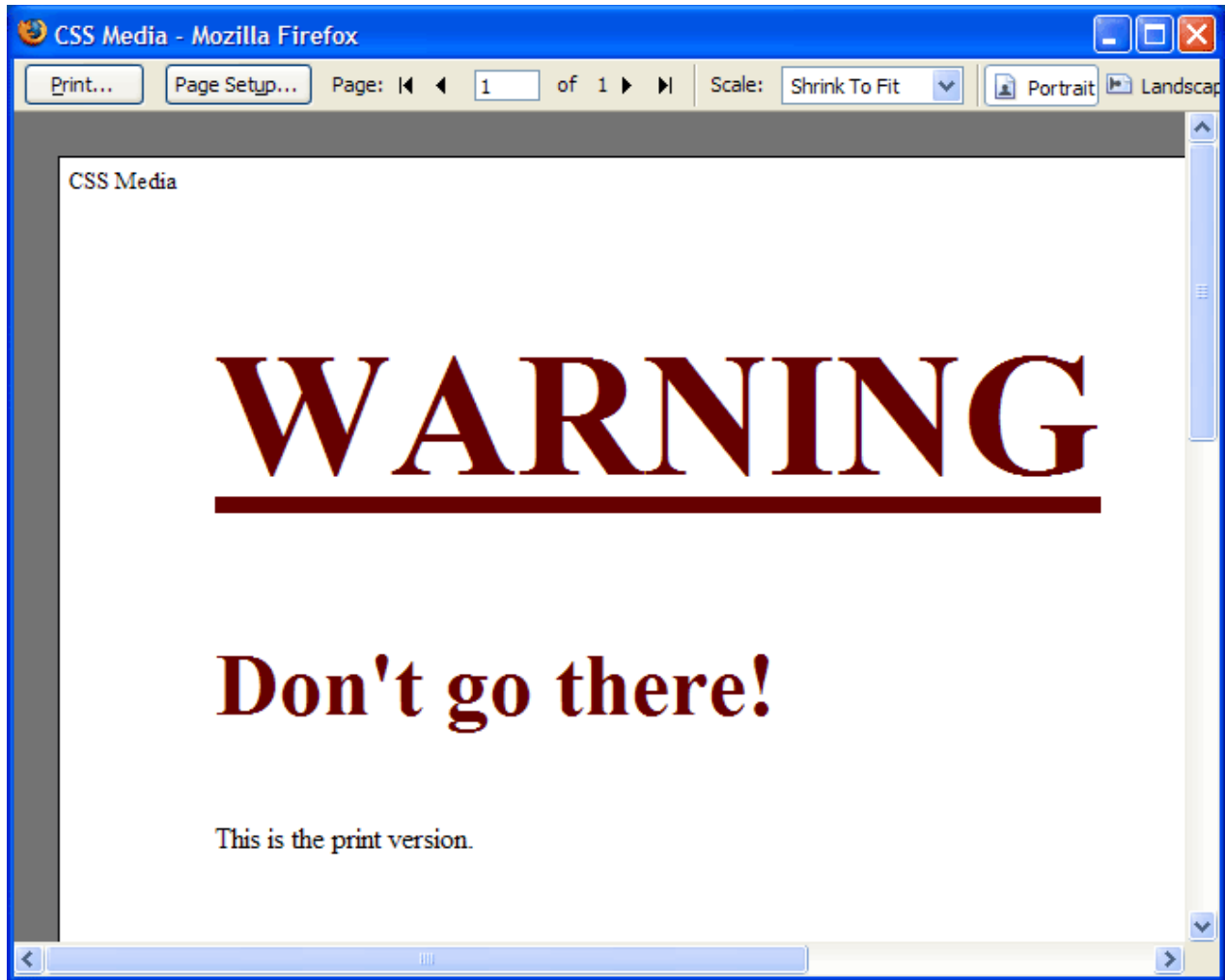
```
p.warning {font-weight:bold; font-size:.5in;}
.screenDisplay {display:none}
</style>
</head>
<body>
<h1 class="warning">WARNING</h1>
<p class="warning">Don't go there!</p>
<p class="printDisplay">This is the print version.</p>
<p class="screenDisplay">This is the screen version.</p>
</body>
</html>
```

As shown below, the screen output of the file above is different from the print output.

The screen output will look like this:



The print output will look like this:



Lesson 1, Activity 15: <div> and

The <div> and tags are used in conjunction with Cascading Style Sheets. By themselves, they do very little. In fact, the tag has no visual effect on its contents. The only effect of the <div> tag is to block off its contents, similar to putting a
 tag before and after a section on the page.

Like most tags, the <div> and tag can take the `class`, `id`, and `style` attributes. It is through these attributes that styles are applied to the elements. The tags are used like any other HTML tags and can be nested within each other any number levels.

Code Sample:

[CrashCourse/Demos/DivAndSpan.html](#)

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Div and Span</title>
</head>
<body>
<div style="position:absolute; left:0px; top:0px;
font-family:Verdana; font-size:10pt;
border-style:groove; border-width:30px; border-color:blue; padding:4px">
This text appears in the
<span style="font-style:italic; color:red">upper-left hand corner</span>
of the page.<br>
It also has a big blue groovy border around it.
</div>
</body>
</html>
```

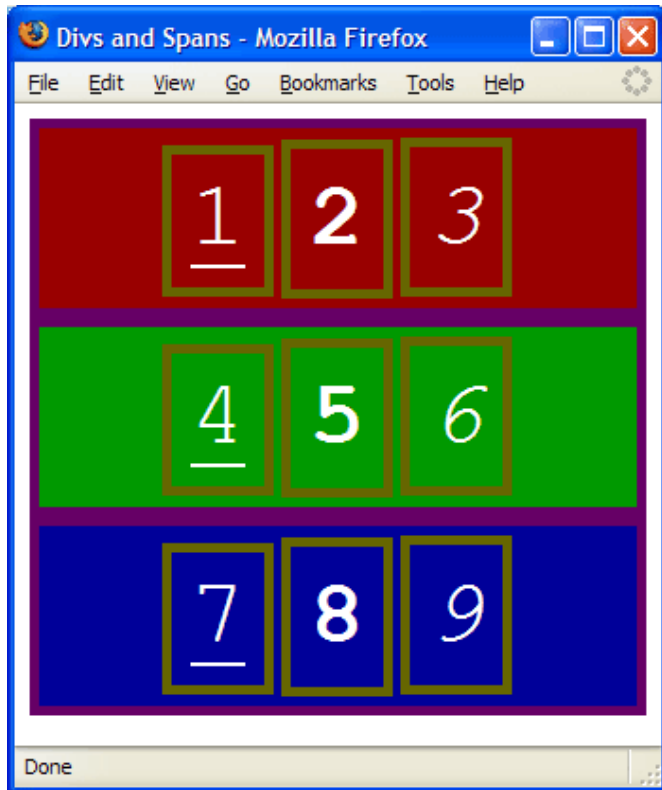
This page will render as follows.



Lesson 1, Activity 17: Divs and Spans

Duration: 10 to 20 minutes.

In this exercise, you will add `class` and `id` attributes to `div` and `span` tags to an already existing HTML page. The HTML page already contains an embedded style sheet, which you will not need to modify. Your goal is to make the page render as follows.



There are no step by step instructions. Review the rules in the embedded style sheet and apply classes and ids as appropriate.

Solution:

CrashCourse/Solutions/DivsAndSpans.html

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Divs and Spans</title>
<style type="text/css">
div
{
border: 5px solid #660066;
padding: 20px;
text-align: center;
}
```

```

span
{
  border: 5px solid #666600;
  padding:10px;
  text-align: center;
  font-size: .5in;
  color: #ffffff;
  font-family: monospace;
}

#topDiv
{
  background-color:#990000;
}

#midDiv
{
  background-color: #009900;
}

#bottomDiv
{
  background-color:#000099;
}

.leftSpan
{
  text-decoration: underline;
}

.midSpan
{
  font-weight: bold;
}

.rightSpan
{
  font-style: italic;
}
</style>
</head>
<body>
<div id="topDiv">
  <span class="leftSpan">1</span>
  <span class="midSpan">2</span>
  <span class="rightSpan">3</span>
</div>
<div id="midDiv">
  <span class="leftSpan">4</span>
  <span class="midSpan">5</span>
  <span class="rightSpan">6</span>
</div>
<div id="bottomDiv">
  <span class="leftSpan">7</span>
  <span class="midSpan">8</span>

```

```
<span class="rightSpan">9</span>
</div>
</body>
</html>
```

Lesson 1, Activity 18: Units of Measurement

CSS allows you to specify font size, border size, margins, padding, etc. using many different units of measurement. The table below shows the units available.

Unit	Description	Example
px	Pixels	margin-top: 10px;
pt	Points	font-size: 12pt;
in	Inches	padding-top: .5in;
cm	Centimeters	top: 5cm;
mm	Millimeters	left: 45mm;
pc	Picas	bottom: 12pc;
em	Ems	font-size: 1.5em;
ex	Exs	font-size: 1.5ex;
%	Percentage	width: 80%;

Pixels (px)

The measurement unit most often used for designing web pages is pixels. A pixel is not an absolute measurement like, for example, an inch or a point. The actual size of a pixel depends on the resolution and size of a user's monitor. Consider an image that is 900 pixels wide. If the monitor resolution is set to 800 by 600 pixels, then the image will not fit on the screen. However, if the monitor resolution on the same computer is set to 1024 by 768 pixels, the image will fit with room to spare.

Points (pt)

Points should be reserved for print. There are 72 points in an inch.

Inches (in), Centimeters (cm), and Millimeters (mm)

Although these are the most common units of measurement in life, they should be avoided in Web design.

Picas (pc)

Picas should be reserved for print. There are 6 picas in an inch.

Ems (em)

An em (or Mutt) is a relative unit that refers to the size of the letter "M" in a font. Because em is a relative rather than absolute measurement, it is often used in Web design.

Exs (ex)

The "x-height" is the height of font's lowercase "x". Exs are used to set the size of content based on the size

of the surrounding font.

Lesson 1, Activity 19: The Inherit Value

Many properties take the value "inherit". This specifies that the property value should be inherited from the parent element. If a property is left undefined, the implicit value is "inherit".

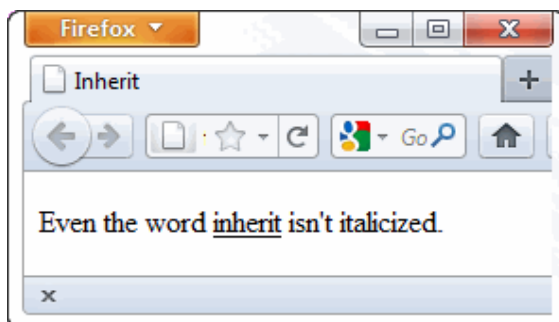
To illustrate, look at the following:

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Inherit</title>
<style>
p {
  font-style:none;
}

em {
  font-style: inherit;
  text-decoration:underline;
}
</style>
</head>
<body>
<p>Even the word <em>inherit</em> isn't italicized.</p>
</body>
</html>
```

The rule for `p` sets the `font-style` of paragraphs to *none*, which means plain, non-italicized text. We don't really have to do this as the default value for `font-style` for paragraphs is *none*.

The rule for `em` sets the `font-style` for emphasized text, which is usually *italic*, to *inherit*. That means that, in the HTML below the CSS, the `em` tag will inherit the *none* value from the containing `p` tag. We also set the `text-decoration` property to *underline*, so we can still see that the tag's content is emphasized:



@import

The `@import` rule is used to import one style sheet into another. There are two syntaxes for using `@import`:

Syntax

```
@import "styles.css" mediatypes;  
@import url("styles.css") mediatypes;
```

In the examples above, `mediatypes` would be replaced by a comma-delimited list of media types (as discussed in the earlier reading in the lesson) to which the imported style sheet applies.

In general, you should avoid using `@import` due to performance reasons, as described in an article called [don't use @import](#) written by Steve Souders.

Lesson 1, Activity 20: **@import**

The `@import` rule is used to import one style sheet into another. There are two syntaxes for using `@import`:

Syntax

```
@import "styles.css" mediatypes;  
@import url("styles.css") mediatypes;
```

In the examples above, `mediatypes` would be replaced by a comma-delimited list of [media types](#) to which the imported style sheet applies.