

CSS

Cascading Style Sheets

Module 1

(CSS An Introduction)

CSS Basics
Style Sheet Syntax
Designing with Style Sheets
The DIV and SPAN Elements
Linked (external) Style Sheets
The Cascade: Dealing with Conflicting Rules
Typographical Properties
Font Properties

1.) CSS Basics	4
<i>Introduction</i>	4
<i>What Does a Style Sheet Look Like?</i>	5
2.) Style Sheet Syntax	5
<i>Grouping Properties</i>	6
<i>Adding White Space</i>	6
<i>Grouping Selectors</i>	6
3.) Designing with Style Sheets	7
<i>The Separation of Structure and Presentation</i>	7
<i>Cleaner, Better Code</i>	7
<i>Inconsistent Support for CSS</i>	7
<i>CSS Workarounds</i>	8
Types of Style Sheets	8
<i>Inline Styles</i>	8
<i>Embedded Style Sheets</i>	8
<i>Linked or External Style Sheets</i>	9
<i>Imported Style Sheets</i>	9
Classes and Selectors	9
<i>Class Selectors</i>	9
<i>Independent Classes</i>	10
<i>Pseudo-classes</i>	10
<i>ID Selectors</i>	10
4.) The DIV and SPAN Elements	11
<i>Applying the DIV Tag</i>	11
<i>Applying the SPAN tag</i>	12
5.) Linked (external) Style Sheets	12
<i>Adding Style Sheet Comments</i>	12
<i>Linking Pages to an External Style Sheet</i>	13
<i>The LINK Tag</i>	13
6.) The Cascade: Dealing with Conflicting Rules	13
<i>Inheritance</i>	14
<i>Specificity</i>	15
<i>Precedence in Selector Types</i>	15
<i>Order of Appearance</i>	15
<i>Styles Versus HTML Tags</i>	16
<i>Cascading Rules for Multiple Style Sheets</i>	16
<i>Using the Cascade to Manage a Site's Styles</i>	16
<i>Creating Inline Styles</i>	16
<i>Overriding Cascading Order</i>	17
7.) Typographical Properties	17
<i>The Text-Align Property</i>	17
<i>The Text-Indent Property</i>	18
Units of Measurement	18
<i>Pixels as Length Units</i>	18
<i>Scalable Documents</i>	19
<i>Value-Unit Syntax</i>	19
The Line-Height Property	19
<i>Line Height Measurements</i>	19
<i>Line Height Inheritance</i>	20
The Text-Transform Property	20
The Text-Decoration Property	20
The Letter-Spacing Property	20
8.) Font Properties	21

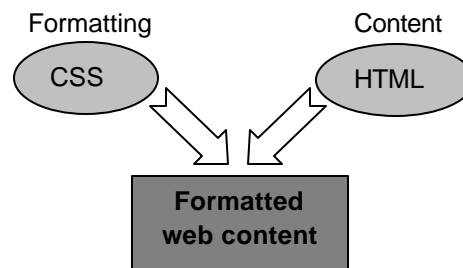
<i>The Font-Family Property</i>	21
<i>Font Names</i>	21
<i>The Font-Size Property</i>	22
<i>Font Sizing Across Browsers and Platforms</i>	22
<i>Font Sizing with Keywords</i>	22
<i>The Font-Weight Property</i>	22
<i>The Font-Style Property</i>	23
<i>The Font-Variant Property</i>	23
<i>The Font Property</i>	23

Introduction

Cascading Style Sheets, or CSS for short, is a web standard technology that has become integrated into web design since 1998. CSS is a far superior system for applying visual formatting to markup documents such as HTML files. Not only does it give web designers more options and increased control over the formatting and layout of documents that was not available under traditional tag-based HTML markup, but in the case of *external Style Sheets*, it also allows easy formatting and update of an entire web site by editing a single text file.

CSS is the World Wide Web Consortium (W3C)'s standard for controlling the visual presentation of web pages. It is a core technology of dynamic HTML (DHTML); along with HTML, JavaScript, and the document object model (DOM). CSS is a tool for modifying how web content is displayed by a browser. It does not replace the need to use HTML; rather it augments the way HTML content is *formatted* in a web page. CSS is designed to allow web designers greater control over how a web page is displayed. It does not control the *behavior* of HTML elements or affect fundamental functions such as hyperlinks or the behavior of embedded elements – only the formatting and display of elements.

HTML was not originally designed with sophisticated page layout properties in mind; rather it was set up to display the logical structure of a document by defining how various parts of the content should be displayed. This focus on *content* has proven to be one of the big weaknesses of HTML as the internet has grown and web designers have begun to demand more control over their web pages. The W3C attempted to standardize the HTML tags to make web pages more universally compatible with the HTML 4.0 specification. Eventually the W3C came up with the idea of Cascading Style Sheets to prevent the need to create more and more HTML tags to perform the complex layout demands needed by designers. By separating *content* and *formatting*, HTML and CSS are able to work hand-in-hand to make pages with much more rich styling than before. By allowing HTML to handle the content and its logical structure – the job it was born to do – and handing over the formatting to CSS – which was built to meet the needs of complex, consistent formatting – we are now able to create elegant, streamlined pages without the clutter and inconsistencies inherent in formatting with HTML alone.



Some examples of the increased formatting power of CSS include:

- Specify the exact size of text and objects using points, pixels, inches, cm, mm, pica, etc.
- Add indentations to text
- Set margins in a web page
- Additional formatting elements, such as borders to images
- Create distinctive, consistent styles for individual pages or groups of pages
- Change the fundamental a given tag is displayed by the browser
- Specify precisely where a background image is displayed and whether is should repeat horizontally, vertically, or both
- Alter the spacing between letters, between words or lines of text and more
- Add background colors and images to blocks of text
- Allow global updating of common elements within a page or a group of pages

With all of the benefits of Cascading Style Sheets, it seems that CSS would be quickly and widely adopted and embraced by web designers. Unfortunately, the implementation of CSS has been slow since its 1996 debut by the W3C. Mainly, the browser manufacturers (primarily Netscape and Microsoft) have been slow to implement full CSS support into their browsers. Over time they are getting better, but there is still a significant portion of the language that is not yet supported. Also, for CSS to reach its full potential, a page (or site) should be designed from the ground up with the idea of implementing CSS in mind. CSS thrives in an environment of consistent, logical structure and few (or none) extraneous HTML

tags controlling the format. The reasons for this will become evident as we learn how CSS works. Many designers would have to go back and completely redesign existing sites in order to use CSS effectively and a complete rebuild of a large site may not be an attractive option. One other reason CSS has been slow to gain widespread adoption is that the common WYSIWYG design tools such as Microsoft's Front Page, Macromedia's Dreamweaver and Adobe's Go Live! have not had user-friendly CSS implementation features. As these tools have become the preferred means of developing web pages (over hand-coding), it has been difficult to use CSS without coding it by hand. There is progress in this area, especially with Macromedia Dreamweaver MX 2004 which is the first major design tool to use CSS as the preferred means of formatting. As more development tools become 'CSS-centric' we will continue to see a rise in CSS use.

Just as working with any web technology, before using a tool such as Dreamweaver or GoLive!, it is best to have a thorough understanding of the structure and capabilities of the technology you are implementing. That is what we are setting out to do: gain a thorough understanding of the CSS language.

What Does a Style Sheet Look Like?

A *style sheet* is any rule or sequence of rules that affect the appearance of a document. With CSS, you can modify a document's appearance by changing the formatting of page elements, much like a word processor template. Below is an example showing a few of the style sheet rules that influence the appearance of a web page. Elements like the <body> of the document are given rules that declare how they are rendered in a browser.

```
body {
    background-color: #ffffff;
    font-size: 14pt;
    color: #336666;
    font-family: arial, Helvetica, sans-serif;
}

.slogan {
    position: absolute;
    left: 2.5in;
    top: 70px;
    color: #0000cc;
    font-family: comic sans ms, verdana, sans-serif;
    letter-spacing: 2px;
    font-weight: bold;
    font-size: 16pt;
    text-align: center;
}
```

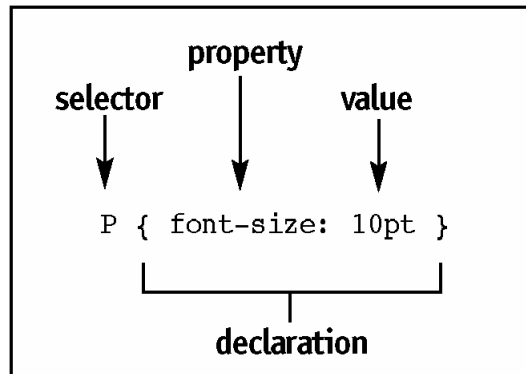
2.) Style Sheet Syntax

CSS syntax is reasonably simple. If you found the structure of HTML easy to grasp, CSS will be a snap!

In essence, style sheets establish *rules* that declare how certain elements are rendered on a page.

Every style sheet rule begins with a *selector*, which determines what element is connected to a particular style. Any valid HTML element (such as <body>, <p>, or <td>) can be a style sheet selector.

A *declaration* applies the style to the selector; it consists of braces { } that contain a property or series of properties. Each property has an associated *value*, an exact specification of the property. The selector precedes the declaration, which consists of everything inside the braces. See the example below.



Grouping Properties

A rule can have multiple properties. For example, if you want all paragraphs to have blue text, a gray background, and a 10-point font size, you can group all the properties together within one rule by separating each property with a semicolon:

```
p { color: blue;
    background-color: gray;
    font-size: 10pt; }
```

Adding White Space

The amount of space you insert between the opening brace, rules, and the closing brace is a matter of preference. It's good practice to make sure your styles can be easily read. For example, you can keep braces and rules on their own lines:

```
p {
  color: blue;
  background-color: gray;
  font-size: 10pt;
}
```

Grouping Selectors

If you want the same styles to apply to more than one element, you can group selectors. By grouping multiple selectors into one rule, you avoid repeating statements. For example, if three elements require the same style characteristics, rather than declare each style separately, like this;

```
TD { color: orange; background-color: silver; }
H1 { color: orange; background-color: silver; }
H2 { color: orange; background-color: silver; }
```

you can group all three selectors into one rule simply by separating each selector with a comma:

```
TD, H1, H2 { color: orange; background-color: silver }
```

This is a faster, more efficient way of declaring the same style characteristics for multiple elements. Selectors are not case sensitive, so it doesn't matter if the selector is lowercase and the HTML element is uppercase, or vice-versa.

3.) Designing with Style Sheets

HTML was originally intended to denote a document's *structure*. Elements like `<h1>` tags (headings) and `<p>` tags (paragraphs) make up the structural components of a document. Browsers determine how these elements are displayed, which doesn't afford the author much control over the document's design. As the Web has grown, browsers have included several auxiliary attributes to increase the design capacity of HTML. With the introduction of these attributes (and a bevy of proprietary tags) have come many unhappy HTML purists.

Although it's true that some of what can be accomplished with CSS can also be done with standard HTML, in most cases, this requires tedious workarounds and convoluted tables, resulting in a mess of HTML code.

If you take a look at some of the more common HTML workarounds – particularly the `` tag, along with the *face*, *color*, and *size* attributes – you'll find that they're often strewn into every instance of a given element. You'll also see that tables, originally meant to hold tabular data, are the most frequently used tool to force elements into a desired position on a page. When it comes time to change the appearance of pages designed with these HTML workarounds, you're left with cluttered code that's tedious and time consuming to modify.

The Separation of Structure and Presentation

With style sheets, you can separate the structure of a Web page from its design—as a result, you can control a document's presentation in ways HTML can't.

Style sheets offer unique control over how page elements are rendered (including font colors and sizes, indents, and borders) without changing the original structure of the HTML document. A preserved structure helps a Web page to maintain platform independence; it also enables internet search engines to index pages more efficiently.

By separating a document's structure from its design, you can change the styles for multiple pages with relative ease. Imagine having to update the styles of a site consisting of over 100 pages, chock full of tables and `` tags with various attributes necessary to style each occurrence of a given element. Chances are you wouldn't want to do it twice. With style sheets, updating the styles of countless pages is a matter of changing a few lines in one document.

Cleaner, Better Code

Another advantage to authoring with style sheets is that the simplicity and precision CSS offers for controlling a document's design characteristics translates into cleaner, more elegant HTML code. Pages will load faster and are far easier to update because the amount of HTML needed to design a page is reduced. The need for "empty" images to position page elements and create white space is also eliminated. And of course, CSS enables you to change the presentation of multiple documents without having to re-code any HTML.

There are many advantages to designing with style sheets, as well as some pitfalls that you should consider.

Inconsistent Support for CSS

Although there are several essential CSS properties that are supported by the most popular browsers (including Netscape Navigator 4.x, Microsoft Internet Explorer 4.x and 5, and Opera 3.x), these browsers interpret some style sheet properties differently – just as they interpret some HTML tags differently. Even subtle rendering differences can cause problems for Web developers, who need to consider all audiences. Hopefully, future browsers will feature full and accurate support of CSS, but until then, it's vital that you routinely test your style sheets in all major browsers, in the various versions of those browsers, and across all platforms. (Of course, if you're developing for a company intranet, and the company uses one browser exclusively, you need not worry so much about how your pages will be rendered in other browsers.)

CSS Workarounds

Unfortunate but true! Because of inconsistent and incomplete support for the CSS-1 specification in Navigator 4.x, Internet Explorer 4.x and 5, and their earlier versions, sometimes you'll need to tweak your style rules to make them work in multiple browsers. CSS workarounds aren't as time consuming or tedious as those using HTML, but you'll see that they do at times seem strange and arbitrary.

Types of Style Sheets

There are four style sheet types in the CSS-1 specification: *inline styles*, *embedded style sheets*, *linked style sheets*, and *imported style sheets*. Embedded and linked style sheets are most commonly used. Each type is described below.

Type	Description
Inline Style	Inserted directly into any HTML tag by using the <i>style</i> attribute. Inline styles only affect the instance of the element they're inserted into.
Embedded Style Sheet	Placed inside the <head> section of an HTML document; defines document-wide styles (on its own page only).
Linked (external) Style Sheet	Used as a global template; an external document consisting only of style sheet rules (no HTML) that you can point multiple HTML documents to.
Imported Style Sheet	Used when a document requires styles in addition to those applied document-wide. Imported style sheets are external CSS files that can be used to cascade with other imported style sheets (not supported by Navigator 4.x).

Inline Styles

You can add *inline styles* by using the *style* attribute to insert a style rule directly into an HTML tag.

```
<h1 style="color: red">
```

Using inline styles repeatedly in a document, however, defeats the fundamental purpose of using style sheets: *the separation of structure from design*. If you choose to make changes to your document styles, you'll need to hunt through the HTML code and change each inline style one by one, just like you would if you had used HTML to handle the formatting. Inline styles are useful when you need a single instance of a particular style.

Embedded Style Sheets

Embedded style sheets are placed within the <head> section of a document and are applied to the entire current document and only the current document. With embedded style sheets, it's important to place style sheet rules within HTML comment tags. Otherwise, some older browsers might display the style sheet code as text. As older browsers continue to be phased out, so will the need for comment tags to "hide" the code.

```
<head>
<style type="text/css">
<!--
h1 { font-size: x-large}
td { color: blue }
-->
</style>
</head>
```

Linked or External Style Sheets

Linked style sheets present the real power of the CSS technology. By pointing multiple documents to a linked style sheet, you can redesign a large web site simply by modifying one document. Linking multiple pages to an external style sheet is an effective tool for controlling the appearance of related web documents, particularly if the site requires corporate or organizational styles on every page.

External style sheets are connected to an HTML document using the `<link>` tag in the `<head>` section of the document. The `<link>` tag contains a `rel` attribute which indicates that we are referencing a style sheet sub-document (`rev` is used instead if referencing a parent document). The `type` attribute is used to establish what type of style sheet is being referenced and the `href` attribute establishes the location of the *.css file.

```
<link rel="stylesheet" type="text/css" href="URL" />
```

Imported Style Sheets

Imported style sheets use the `@import` notation to link documents to style sheets. An `@import` rule links a document to a style sheet by referencing the URL where the style sheet is located. For example:

```
@import url(http://www.zd.com/mystyles.css)
```

An `@import` rule can be inserted into a style sheet, and it must be the first rule in the style sheet. You can use multiple `@import` rules to apply styles from various style sheets, which cascade in the order in which `@import` rules appear in the document. With imported style sheets, you can apply broad, company-wide styles with one style sheet and specific styles for a particular department or function with another.

Classes and Selectors

Class Selectors

With *class selectors*, you can create *different* styles for the *same* element or the *same* styles for *different* elements. They are powerful because you can personalize the styles you create by giving special names to certain features of your document. For example, if you want to create a page element called an "endnote," and for consistency you want all endnotes to have a gray background, you can create a class of the `<p>` element (paragraph) and declare styles for it. To do this, you use a "flag character" to separate the selector and the class name:

```
p.endnote { background-color: gray }
```

The flag character for a class selector is a period. Once you create the class selector and its corresponding style rule, you can specify which class you wish to use for a particular paragraph by adding a `class` attribute that matches the class name:

```
<p class="endnote">
```

There are a couple more things you should keep in mind when using classes. First, avoid using numbers in your class names. Some browsers, including Navigator, will not support a class name that begins with a number. Also, for the 4.x browsers, only one class is allowed per selector. In other words, you can't have a selector that looks like this:

```
p.endnote.footnote
```

Class attributes should not be used more than once within the same tag. If you're thinking of giving an element the styles of more than one class, this isn't the way to do it. Browsers treat an occurrence of multiple class attributes differently so your results will be inconsistent. For example: Internet Explorer uses the first class specified, while Navigator 4.x uses the last.

Independent Classes

You can use class selectors to declare styles for an element without having to include the HTML element itself in the selector. In effect, you can create your own custom page elements by using classes that don't correspond to any HTML element in particular.

For example, you might create the following rule:

```
.author { color: green;
          font-style: italic; }
```

Since there's no HTML element associated with this rule, you can use `class="author"` to classify any element in a document. Since you are not restricted to any one HTML element, you'll find that this approach is more flexible than using class selectors.

```
<span class="author">Ernest Hemingway</span>
```

It's good practice to name classes according to how they are applied, rather than only by their appearance. The "author" class in the preceding example could be named *green*, but what happens if you later decide to change the style so that the text is rendered red instead of green? You're stuck with a class name that's unsuitable and potentially confusing, or you'd have to go into the HTML and change the class names. Either way, it's more work.

Pseudo-classes

Rather than corresponding to an element's `class` attribute, *pseudo-classes* denote a specific quality of an element. Navigator 4.x and Internet Explorer 4.x support three pseudo-classes, all of which are used with the `<a>` (anchor) element to distinguish the status of hyperlinks in a document. (Internet Explorer also supports the `hover` pseudo-class. Like the `link`, `alink`, and `vlink` attributes used with the `<body>` tag in HTML, CSS pseudo-classes are used to override default link colors. Anchor pseudo-classes are declared with `a`, followed by a colon and the link status.

Pseudo-class	Description
<code>a:link</code>	Sets the style for links that have not yet been visited.
<code>a:active</code>	Sets the style for links when clicked on.
<code>a:visited</code>	Sets the style for visited links.
<code>a:hover</code> (IE only)	Sets the style for links to change as the cursor passes over the link

Pseudo-classes can also be used with regular classes to create a variety of different link styles. For example, the following rule will style only visited links of the "home" class.

```
a.home:visited { color: gray }
```

ID Selectors

ID selectors work similarly to classes: you can create unique styles for any element. If you have a particular page element that isn't duplicated elsewhere, and you want to give it its own one-of-a-kind style, ID is the tool of choice. Whereas class selectors can be applied multiple times to any element throughout a document, ID selectors are intended for use on an individual basis to single out an element's style. You cannot use more than one ID attribute with the same value in a document.

ID selectors have a slightly different syntax than class selectors; the flag character is a pound sign (`#`) rather than a period. An ID style might look like this:

```
#name { color: olive;
        font-weight: bold; }
```

Once you create the unique ID style, you specify the ID in the desired element:

```
<blockquote id="name">
```

IDs are extremely valuable in the world of dynamic HTML (DHTML) because they allow you to uniquely identify any element within a document in your scripts. This is particularly useful in Internet Explorer, where you can reference all HTML elements as properties of the document object. While the object references can be more complex in Navigator, the same principle applies. But all that stuff is beyond the scope of this course.

Although you can't give an element more than one *class* or *id* attribute, you can use the two attributes together. That way, you can classify an element with styles shared by other elements, and give it a unique *id* to distinguish it further.

4.) The DIV and SPAN Elements

<div> and are HTML tags that are most often used to reference style sheet rules. The <div> tag (logical division) is a block-level HTML element that can contain document sections, such as headings, lists, and tables. If you want to control the appearance of a section of a document that contains various elements, you can use <div> to establish global styles for that section. As a block-level element, the <div> tag creates a line break.

Applying the DIV Tag

In the following example, a <div> defines the styles for a section of a document with various elements: a heading, a paragraph, and two text-level elements, and .

```
<div class="definition">
<h1>The Green Sea Turtle <em> (Chelonia Mydas)</em></h1>
<p>Sea turtles evolved <strong>180 million years</strong>, ago, long before
the Hawaiian Islands were formed. Hawaii's isolation in the Pacific has
helped maintain a healthy population of Green Sea Turtles.</p>
</div>
```

Here, the <div> tag controls a specific section of a page. All elements within this division will be rendered with the *definition* style, provided that a class named *definition* exists in the document's style sheet. <div> can be used with *class* and *id* attributes to style document sections according to document-wide or external style sheet rules, and with the *style* attribute to create inline styles.

```
<div class="section">

<p>To become a member, <em>follow these easy steps:</em></p>
<!-- ordered list -->
<ol>
<li>Fill out the form above, or;</li>
<li>Send a self-addressed, stamped envelope to:
    <strong>LearniT!, PO Box 123 San Francisco, CA, 94104</strong></li>
<li>When you receive the application by mail, check the
<strong>"yes"</strong> box, and return it; or, if you're not sure what "yes"
entails:
    <ol> <li>Simply trust us; or</li>
        <li>Use the Force</li>
    </ol>
</li>
</ol>
<p>Your membership will give you the chance to met and correspond with other
students in the classes you are taking. If you refer a friend to us, you'll
appear on our list of really cool people!</p>
```

```
</div>
```

Applying the SPAN Tag

Like the `<div>` tag, `` is one way to link style sheet rules to sections of a document. The `` tag is an inline or text-level HTML element, which means it's placed around text (rather than blocks or containers) and doesn't create a line break. You can use `` with the `class` or `id` attribute to apply a style as defined in a style sheet, or you can use the `style` attribute within the `` tag to create an inline style. You can use the `` element to style a letter, word, or section of text:

```
The <span class="highlight">Green Sea Turtle</span> lays its eggs on secluded beaches on isolated islands in the Hawaiian chain.
```

This gives the text string "Green Sea Turtle" a green background, assuming that the following rule is defined in a style sheet associated with that page:

```
.highlight { background-color: green }
```

The same effect can also be achieved by using the `style` attribute to create an inline style, as shown in the following example:

```
The <span style="background-color: green">Green Sea Turtle</span> lays its eggs on secluded beaches on isolated islands in the Hawaiian chain.
```

5.) Linked (external) Style Sheets

At this point, you should be familiar with the utility of linked, or external style sheets. They're the most powerful and flexible tool for styling multiple documents simultaneously. To link pages to an external style sheet, the `<link>` tag is placed in the `<head>` section to reference the location (URL) of the external style sheet. The external style sheet is simply a text file containing only style sheet rules. No HTML tags are necessary in this file and the file must be saved with a `.css` extension. Once you create the external style sheet, you can link documents to it with the `<link>` tag.

Adding Style Sheet Comments

It's good practice to add comments to your style sheets just as you would throughout your HTML code. Adding comments next to a style rule or group of rules enables you or another viewer of the style sheet to easily parse the code and determine the effect that each style has on the page. Like HTML comments, style sheet comments don't show up in the browser window.

CSS comments are denoted with the forward-slash (`/`) and asterisk (`*`) characters:

```
/* This is a CSS comment */
```

A typical application of a style sheet comment might look like this:

```
.note { text-indent: .25in }
  /* "Note" class elements are indented 1/4 inch */
```

Or, above the rule:

```
/* Table cells have a silver background */  
TD { background-color: silver }
```

You can even put comments inside rules, like this:

```
p { /* All normal paragraphs dark green */  
  color: darkgreen }
```

Linking Pages to an External Style Sheet

Once you create the *.css file, you can link multiple documents to it with the <link> tag. Because of this capability, linked style sheets are a powerful and effective tool, especially if you develop large web sites with multiple documents. With an external style sheet, you can create a template that provides a uniform presentation for corporate or organizational styles. You can then go back and change the styles of multiple pages simply by modifying one document, the external style sheet (*.css file).

The LINK Tag

The <link> tag creates a relationship between the current document and the document you are linking to. Several attributes are necessary in the <link> tag when you link to an external style sheet: *rel*, *type*, and *href*.

The *rel* attribute specifies the relationship of the referenced file to the current document. For example, *rel="stylesheet"* indicates that the current document's link relationship is a style sheet. The *type* attribute is used to specify the type of style sheet used in the document, and the *href* attribute declares the URL or filename of the external style sheet you are linking to. *Title* can also be used to indicate the style sheet's name.

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

6.) The Cascade: Dealing with Conflicting Rules

Cascading refers to the hierarchical order in which different style sheet types interact when conflicts arise. The prevailing principle of cascading order is that rules cascade from *general* to *specific* and from *external* to *local*.

Premise: A linked style sheet declares that all level-one headings <h1> are displayed in blue. HTML documents linked to the style sheet contain inline styles declaring that each <h1> is displayed in red.

Outcome: The inline styles take precedence over the linked style sheet because they're more *local* – text contained by <h1> would be displayed in red. If the two rules have different properties assigned to the <h1> element, the styles from both sources are applied. Only when the same properties with differing values are assigned from different style sheets do conflicts arise.

The cascading order of rules is part of the CSS-1 specification that assigns levels of importance to the different style sheet types. Style sheets cascade according to the following order of priority:

1. Inline styles
2. Embedded style sheets
3. Linked style sheets

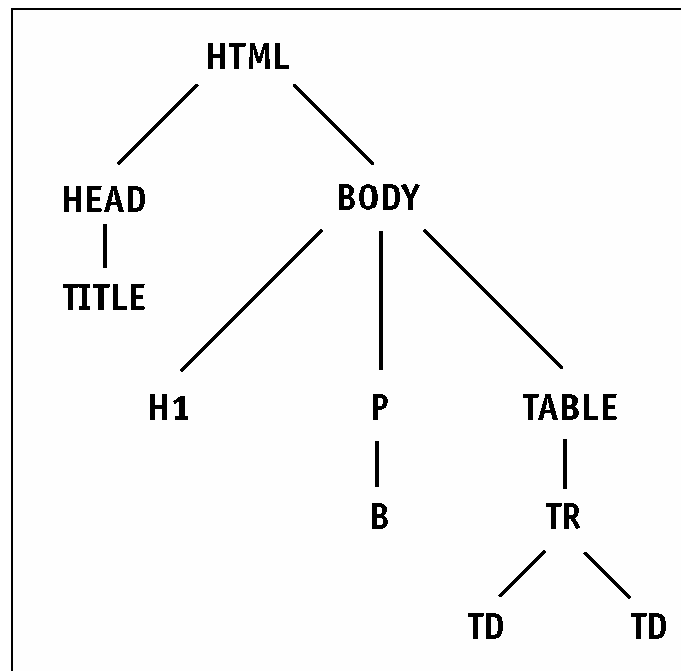
4. Imported style sheets

Inheritance

Inheritance refers to the way CSS properties flow through a document. HTML elements are organized by a *tree structure*. The tree structure is the hierarchy of elements in any given document wherein elements are connected by *parent-child relationships*. The rules of inheritance determine which properties carry through these relationships, and which do not. (Some CSS properties can be inherited, while others can't.)

In the example below, the <html> element is the parent of the <head> and <body> elements, the <head> element is the parent of the <title> element, and the <body> element is the parent of several child elements including <h1>, <p> and <table>. Following is a diagram showing the tree structure of this document.

```
<html>
<head>
<title>Document Title</title>
</head>
<body>
<h1>A level-one heading.</h1>
<p>A paragraph with some <b>bold text</b>.</p>
<table>
<tr><td>Table cell contents</td>
<td>Table cell contents</td></tr>
</table>
</body>
</html>
```



More often than not, child elements inherit the styles of their parent elements. For example, if you declare properties such as font and color to the <body> element, then every element within the page will inherit the body styles because <body> is the top-level parent of all visible elements. There's no need to set the font and color again for each element, such as list items or paragraphs.

In some cases, Netscape Navigator and Microsoft Internet Explorer treat rules of inheritance differently. In Navigator 4.x, declaring all text in a document the same size – say, 10 points – will do just that. In Internet Explorer, however, the heading structures are preserved – any headings in the document will retain their default heading size until the style is specifically declared for the element. Moreover, tables viewed with

Navigator don't inherit styles from their parent elements. A simple workaround is to directly assign to the <td> element the styles you want a table to inherit.

Specificity

You can always override inherited styles. If you don't want an element to inherit the styles of its parent, all you need to do is declare styles for that element specifically (using the same properties). By specifically declaring a style for an element, you override any inherited style that conflicts with it. For example, if one embedded rule declares <blockquote> text red, and another embedded rule declares it blue, browsers will render the text according to the most specifically declared style. Here's how that might look:

```
body { color: green; }
blockquote { color: blue; }
```

The *blockquote* text is specifically declared blue, but it also inherits the green value from the *body* style. The specifically declared style outweighs the inherited style, so *blockquote* text will be rendered blue.

If, however, the *body* rule contains a property that doesn't conflict with the specifically declared *blockquote* rule,

```
body { color: green; font-size: 10pt; }
blockquote { color: blue; }
```

then the *blockquote* will continue to inherit the font-size property from the <body> element while overriding the inherited color.

Precedence in Selector Types

Rules of precedence also apply to each selector type: the *id*, *class*, and HTML element selectors. The *id* has the highest level of importance; the *class*, the second-highest; and the HTML element selector, the lowest. In other words, if a particular element is attached to both an *id* and a *class*, and the two rules contain conflicting styles, the *id* styles take precedence over the *class* styles.

Specificity is calculated as follows: Each element selector counts as 1 point. A class selector or pseudo-class counts as 10 points. An *id* selector is worth 100 points. The points are then combined for a total. For instance, a rule with **td.header** as its selector has a total of 11 specificity points, since the **td** gets one and **.header** is worth 10. In other words, the following rule takes precedence over the rule that follows it, since it has 11 specificity points and the latter rule has only one:

```
i.home { color: blue }
i      { color: red }
```

So, any <i> element of the class "home" would be blue. Here are a couple more examples:

```
a.home:visited { color: gray }
a#navbar:link  { color: red }
```

The first rule has a normal element selector, *a*, which is worth 1 point, a class, worth 10 points, and a pseudo-class, also worth 10. So, this rule has a specificity of 21. The second rule has an element selector, an *id* selector, and a pseudo-class, worth a total of 111 points.

For a rule with grouped selectors, the specificity of each rule is calculated separately, as follows:

```
h1, div.home { color: navy }
```

In the preceding rule, the **h1** rule has a specificity of 1, while the **div.home** rule has a specificity of 11. Calculating specificity can get more complicated than this. Visit the World Wide Web Consortium's site for a more in-depth analysis.

Order of Appearance

In cases where a property is declared for an element more than once in the same style sheet, preference is given to the style declared last (provided the two rules have the same specificity). You might never choose to set up your styles in this way, but knowing all the rules of the cascade is important. Say for instance that you need to toggle between two styles often. Instead of deleting the style you want to “switch off,” you can simply reverse the order of the two rules.

```
p { margin-top: 10%; }  
p { margin-top: 25%; }
```

In the preceding code, all paragraphs would have a 25% top margin because that’s what the last rule declared.

An exception to this rule is as follows: Suppose a paragraph contains a <sup> (superscript) element, which in turn contains a element that directly contains text. You have set up the following rules:

```
strong { color: green; }  
sup { color: red; }
```

The text contained directly by the element will be rendered green because is the element closest to the actual text, even though the <sup> element is the last rule to be declared in the style sheet.

Styles Versus HTML Tags

Although the CSS-1 specification states that styles are given precedence over HTML tags, it’s not the case with today’s browsers; they give HTML tags with style attributes a higher level of importance. For example, if an embedded style sheet declares that all text is blue, but a tag contains all of the text in the document, all text will be rendered green in accordance with the tag. Future generations of the browsers might correct this inconsistency.

Cascading Rules for Multiple Style Sheets

The style given to a Web document can come from a variety of sources at once. You can have a page that contains an embedded style sheet and inline styles but that is also linked to an external style sheet. Whenever the same properties for like elements collide from various sources, the more local styles take precedence. In other words, for the same property, an embedded style wins out over a linked style; an inline style, being the more local rule, takes precedence over the embedded style.

Rules from various sources can also work together:

Premise: An embedded style sheet contains a rule for the <body> element declaring all text blue; the document is also linked to an external style sheet that contains a rule for the <body> element declaring all text at 10 points.

Outcome: The page will be rendered with 10-point blue text. The color property from the embedded style sheet and the font-size property from the external style sheet cooperate to provide the <body> styles.

Using the Cascade to Manage a Site’s Styles

Because style sheet rules cascade according to origin, you can set up your styles in ways that make them easy to understand and modify. If you’re managing a large site that requires global styles, you should always use an external style sheet and link your documents to it. If a certain page on your site requires its own special styles in addition to or in place of those global styles, you can embed those styles in that document, and the rules will cascade. Any property that doesn’t conflict will pass through from the external style sheet, while those that do conflict will display in accordance with the rules of the embedded style sheet, and/or the rules of specificity.

Creating Inline Styles

Inline styles are the most local style rules: they take precedence over all conflicting rules from a linked or embedded style sheet. Inline rules can be useful for creating single-instance styles for an element. For example, if you maintain a site of 100 pages with a common style defined by an external style sheet, but one page requires a special rendering of the element for extra emphasis, it makes sense to use an inline style, rather than create a class for the element, since it's not likely to be duplicated. Even creating an *id* for the element, which would suit the single-instance nature of the style, would entail more work than defining the style inline. The obvious drawback of using inline styles, however, is the lack of separation between structure and presentation.

Overriding Cascading Order

Now that the rules of cascading order are clear, we can begin to break them. By adding **important** to a particular style in a declaration, you guarantee that it's sorted with the highest priority, regardless of where it exists in the normal cascade. Consider the following example:

```
p, div { color: darkgreen;
        font-size: 10pt !important }
```

Here, the font-size property is declared important. This means that all paragraphs and divisions will be rendered with a 10-point font size, even if a more local or specific rule is declared that conflicts with it. In examples like this where there's more than one property in a declaration, you'll need to use the **important** notation for each property you wish to give precedence. The color property in the preceding example is not declared important; it has normal priority in the cascade.

When "important" styles conflict with other "important" styles, the normal rules of the cascade return. In other words, an important property in an embedded style sheet takes precedence over the same important property in a linked style sheet, because it's more local. Also, note that there is no space between the exclamation point and the word "important" in the example above. This isn't important. The number of white spaces between the ! and the word **important** does not make a difference. It's all a matter of preference.

Note: Netscape Navigator 4.x doesn't support the *!important* notation, and neither does Internet Explorer 4.x for the Macintosh.

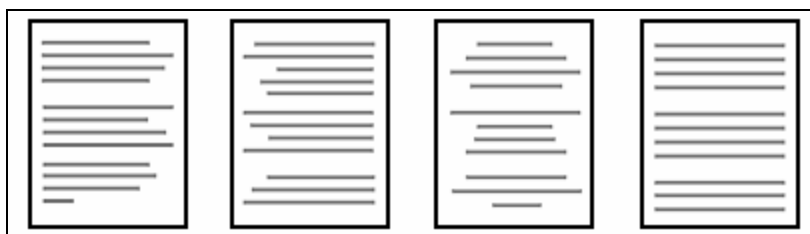
7.) Typographical Properties

The fonts you choose for your site and the readability of your text is at the core of smart Web design. You should avoid fonts that are difficult to read, or overly garish, and try not to use too many different fonts. Choose alignments and spacing that complement the overall layout of the page, and add to its readability.

The Text-Align Property

```
left | right | center | justify
```

Text alignment can be handled with HTML attributes, but controlling alignment with CSS is a better idea, due mainly to its fundamental benefit: the separation of content from design in formatting web documents.



text-align:
left**text-align:**
right**text-align:**
center**text-align:**
justify***The Text-Indent Property***

You're probably familiar with text indents – they're common in books, correspondence, and other forms of print communication. With the *text-indent* property, you can precisely control (in absolute or relative units) how much indentation the first line of a text element receives. Absolute and relative units are discussed in the next section.

Units of Measurement

Any CSS property associated with specific lengths, such as *text-indent* or *font-size*, can be assigned with a variety of different *absolute* and *relative* units. Absolute units specify a precise measurement (such as inches or pixels), and are best used when the settings of the output device (such as a printer or monitor) are known. In the world of web development, there's no real way to ascertain the output device settings of your varied audience. For that reason, relative units of measurement are often the better choice because they scale automatically from user to user.

Absolute units of measurement include the following:

Absolute unit	Description
inches (in)	One inch is equal to 2.54 centimeters.
centimeters (cm)	One centimeter is equal to almost 2/5 of an inch.
millimeters (mm)	10 millimeters are equal to one centimeter.
points (pt)	One point is equal to 1/72 of an inch.
picas (pc)	One pica is equal to 12 points.

You should be careful with absolute units of measurement. You'll find that they're not as absolute as they might seem. Your audience will view your pages on a wide variety of browsers, platforms, screen resolutions, and other output devices, like printers – all of which can conspire to make your measurements more arbitrary than absolute. Besides, you probably want your documents to scale from user to user, and from one output device to the next, which is where relative units come in.

Relative units of measurement include the following:

Relative unit	Description
ems (em)	Relative to the height of the current element's font, or the inherited font size when applied to the font-size property.
x-height (ex)	Defined by the size of the element's x character, or the size of the x character of the inherited font size when applied to the font-size property.
pixels (px)	One pixel equals one physical pixel on a computer display, which varies according to screen resolution.
percent (%)	Based on the computed value of an inherited length.

Pixels as Length Units

All relative units (other than pixels) specify a length relative to an inherited length. Pixels are often considered an absolute unit, since a pixel refers to one physical pixel on a computer screen and you can guarantee that a value of 10px will occupy ten screen pixels. But since screen resolutions vary from user to user, you should use pixel values for properties like font-size with caution, since the resulting size of text will vary.

Scalable Documents

Because you can't control what output devices and settings your varied audience will view your pages with, it's important to make your documents scalable. You can do this by using relative units of measurement. Ems (em) and percentages (%) are good choices; they're based on the size of the font in use. However, as with absolute units, there are some problems with relative scales between Navigator 4.x and Internet Explorer, and on the Macintosh versions of both.

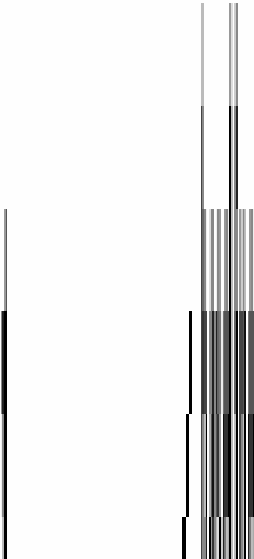
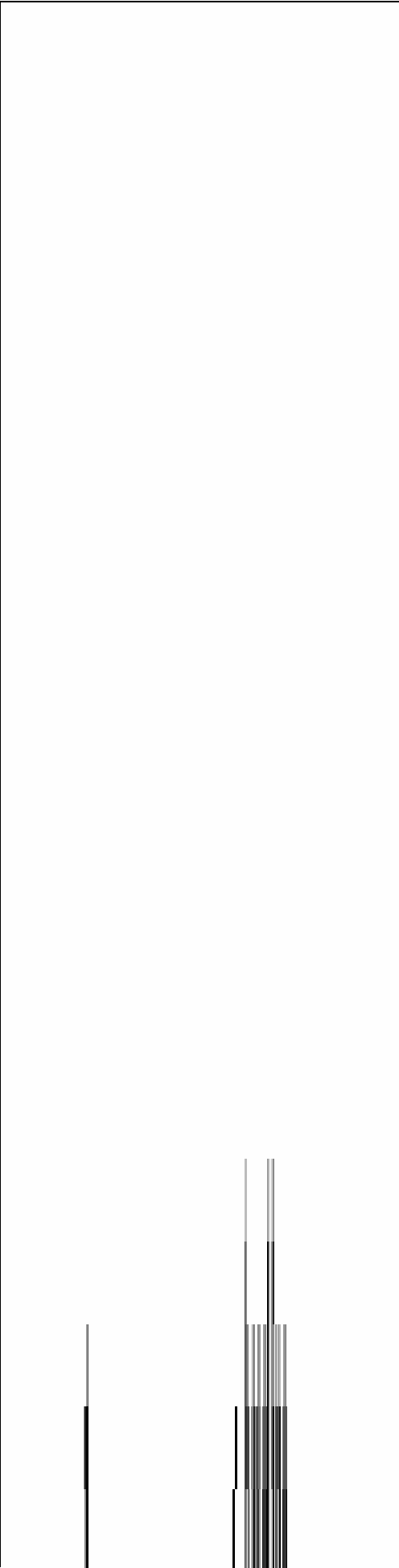
Whether you decide to use absolute or relative units of measurement comes down to your own design philosophy. Be aware that some earlier browsers, including early versions of the 4.x browsers, don't support many units of measurement, including ems, x-height, and picas. In some older browsers, only pixel measurements work. It's a nightmare out there, and it's nearly impossible to make all pages work for all users in the way that you intend.

Value-Unit Syntax

Property values that require units of measurement consist of an integer or decimal, followed *directly* by the unit, such as 1in, 2.2em, -6cm, and so on. Your style won't work if you add space between the number and its unit.

The Line-Height Property

The *line-height* property enables you to control the amount of space between lines of text. (The space between lines of text is commonly referred to as *leading*.) Browsers incorporate their own default line height, or leading amount, into an element's font size. With the line-height property, you can override the browser's default line height, which isn't possible with straight HTML. Line height is measured by the distance between the baselines of two adjacent lines of text.



The total amount of line height is halved between the top and bottom of each line of text. For example, if a line-height of 24 points is specified for 12-point text, then 12 points of extra line height are added; 6 points above the text and 6 points below.

Note: There's a subtle rendering difference between Navigator 4.x and Internet Explorer. Internet Explorer seems to equally divide the amount of line height above and below a given line of text, while Navigator 4.x seems to apply the total line height amount to the top of each line. In Navigator, line height with text in table cells seems to do the exact opposite; the total line height is applied to the bottom of each line.

Line Height Measurements

You can specify line height with a percentage value, a specific length, or a number. Specifying line height by percentage means that the leading will be calculated relative to the font size of the text. For example, if you have 12-point text, and you specify a line height of 150%, the actual line height will be 150% of 12 points, or 18 points.

Using a number to specify line height works similarly to percentage; the browser determines the amount of leading by multiplying the font size of the text by the number specified. For example, with 10-point text, a line-height value of 2.6 will result in 26 points of leading. You can also specify the line-height amount directly by using an absolute length value, such as 14pt or 1cm. To use the browser's default leading, you can use *normal* as the line-height value.

Note: Navigator 4.x has trouble calculating the height of lines. Generally, Navigator 4.x adds slightly more leading than Internet Explorer when applied with percentage or integer values. Applying the line-height property with these relative units is, in theory, the best method. But once again, rendering differences between browsers interferes with otherwise smart code. For more accurate line height results in Navigator 4.x, try using absolute units.

Line Height Inheritance

In Navigator, when line height is applied with an integer, child elements inherit the calculated line height value, rather than the integer itself. In other words, if you have a <div> at 12 points with a line-height of 2, and in that <div> you have a paragraph <p> with a 14 point font size, the paragraph will inherit the div's calculated line-height, 24 points, rather than the actual line-height value, which would give the paragraph a line-height of 28 points.

The Text-Transform Property

capitalize		uppercase		lowercase		none
------------	--	-----------	--	-----------	--	------

With the *text-transform* property, you can create basic text qualities like initial-letter capitalization, and upper-and-lower casing. The effects of each text-transform value are fairly logical. For example, *capitalize* capitalizes the first letter of each word in a paragraph. *Uppercase* styles all text in uppercase characters, and so on.

Okay. You might be thinking, "Why on Earth would I bother using CSS for things like capitalization, when I can simply enter my text with all necessary upper- and lower-casing?" And the answer to that would be, well, fine – that's how most people would handle that mole hill of a problem. But what if you had a large site with, say, headings of one sort or another on every page that feature normal capitalization. Then your boss tells you that the heading styles need to change: "We're moving to all caps", he or she might say. Then you'd have a mountain of a problem. That's where the text-transform property can come in really, really handy.

The Text-Decoration Property

underline		overline		line-through		blink		none
-----------	--	----------	--	--------------	--	-------	--	------

The *text-decoration* property has five possible values: *underline*, *overline* (Internet Explorer only), *line-through*, *blink* (Navigator only), and *none*. The *none* value is useful if you don't want your hyperlinks to be underlined. (Browsers underline all hyperlinks by default.) By specifying *none*, you can override default browser styles to ensure that your hyperlinks won't be underlined. A note on *blink*: nobody likes blinking text. Forget it exists – please!

The Letter-Spacing Property

With the *letter-spacing* property, you can control the amount of spacing between characters. This is commonly referred to as *Kerning*. The value of the letter-spacing property is applied in addition to the browser's default letter spacing. You can increase the kerning by applying any positive value, or you can decrease the kerning with a negative value.

You can create some interesting effects by adjusting the kerning of letters; however, the letter-spacing property isn't supported by Navigator 4.x. Internet Explorer 4.x and later does support it, and future versions of Navigator and other browsers will too. It's an especially powerful tool because there's no method for kerning with straight HTML, and it can replace the need for images to create eye-catching text and increase readability.

8.) Font Properties

Browsers have default fonts and font sizes that users can change to suit their tastes. CSS overrides these styles, giving the author control over how a page is displayed. Choosing fonts can be tricky. As a web author, you should make every effort to accommodate your varied audience, which means you need to specify fonts that all or most users will have available to them. Macintosh users, for example, have several fonts that a PC user might not have, and vice versa. Specifying a platform-specific font will result in a default font for any users who don't have that specific font installed on their computers, and that default font might not match your intentions.

The Font-Family Property

With CSS, you specify fonts with the *font-family* property. With it, you can declare a specific font-family, or a generic family, or both.

A *font family* is the name of a specific font, such as Times New Roman or Arial, and all of its variations (bold, italic, etc.). A *generic family* describes a type of font, rather than a specific font face. Generic font families include:

- *Monospace*: Fonts with uniformly spaced characters, such as Courier or Courier New. Each character takes up the same amount of space, regardless of the width of the individual character.
- *Serif*: Fonts with letters that have “decorations” (often called flourishes) that help to distinguish a letter. Serifs are usually applied to the ends of a letter (see figure below). Times New Roman and Garamond are examples of serif fonts.
- *Sans-serif*: Fonts without serifs (see figure below). Sans-serif fonts include Helvetica, Arial, and Verdana.
- *Cursive*: Fonts that look similar to handwriting. Navigator 4.5 doesn't support this generic family name, so it's a good idea to avoid using it until it's widely supported.

- *Fantasy*: This generic font family isn't supported by Navigator 4.x. Internet Explorer will display unusual or exotic fonts, but it's generally not a good idea to use fantasy as a generic family name because these types of fonts vary in appearance greatly from browser to browser.



Serif and sans-serif fonts.

Font Names

To achieve a consistent appearance for all potential viewers, it's important to include a generic font family at the end of the declaration. If the browser is unable to locate the specifically-named font on the user's system, it will default to a similar font. For example, the font Verdana is a sans-serif Windows font widely used on the Web for its on-screen readability. Macintosh users, however, might not have Verdana installed on their systems. To ensure that your intended presentation is not lost for those who don't have access to the font specified in your style sheet, you can follow the specific font name with a generic font family:

```
body { font-family: verdana, sans-serif; }
```

If a user doesn't have the font Verdana on their system, the browser will skip over the font name "Verdana" and move on to the next choice, "sans-serif." Because this is a generic family name, the browser will display whatever sans-serif font is installed on the user's system. Note that multiple fonts in a string are separated by commas, a syntax unique to the font property.

When specifying specific font names, the name must be spelled exactly as it is displayed on the system. For example, "Comic Sans" is "Comic Sans MS" on Windows. Without the MS, the browser will ignore the font name. It is also important to place any font name that consists of more than one non-hyphenated word in quotes, like this:

```
p { font-family: "Comic Sans MS", "Times New Roman", serif; }
```

Remember, it's a good idea to choose fonts that are common on the Windows and Macintosh platforms. Since many Windows fonts aren't likely to be found on a Macintosh, and vice-versa, you should specify fonts from both platforms that are similar to each other. This ensures that your intended font, or at least something similar to it, will display on different platforms.

The Font-Size Property

The *font-size* property is fairly self-explanatory. You can specify a font's size with either absolute or relative units, or with keywords. Font sizes are most commonly specified in points (pt), ems (em), picas (pc), and percentages. It's generally not a good idea to use pixels (px) due to varying screen resolutions. Ems and percentage values are your best bet, since they're relative units and your text will scale to a user's settings.

For example, if you declare that a heading is displayed at 300% of a user's base font size (usually 12 points unless otherwise specified), the heading will be three times the size of regular text, regardless of screen resolution or other variables. When you need to ensure the relative sizes of text elements in your documents, ems and percentages are the answer.

Font Sizing Across Browsers and Platforms

The conflicting ways that browsers handle font sizes and relative scales is enough to drive a developer completely mad. What's worse is that the Windows and Macintosh platforms also have their own disparate display methods. For example, on a PC, text set at 12 points will look more like 9 points when viewed on a Macintosh browser. Text set at 9 points when viewed on a Macintosh browser is almost

unreadable. There are ways around this, including using browser identification scripts that link to separate style sheets, or choosing font sizes that look reasonable, if not optimal, on both platforms. Whatever you decide, remember to test your font sizes in browsers of both platforms.

Font Sizing with Keywords

Font size can be specified in a variety of ways: with keywords such as “small” or “large”; with relative keywords such as “smaller” or “larger”; with percentage units; or with specific length units. Depending on the context of your style rules and the structure of your documents, you might want to use a variety of measurement units throughout a given page or web site.

You can use absolute keywords to set the font size according to how a browser treats that keyword. (You’ll notice some differences in the way Navigator 4.x and Internet Explorer render absolute keyword font sizes.) Relative font sizes scale a font size up or down according to the font size of the element’s parent; otherwise, by the default font size. Below is a list of available keywords:

Absolute keywords: xx-small, x-small, small, medium, large, x-large, xx-large
Relative keywords: smaller, larger

The Font-Weight Property

100-900	bold	bolder	lighter	normal (default)
---------	------	--------	---------	------------------

Sure, with straight HTML, creating bold text is easy – just nest the text in a tag. But, bolding text with CSS is more flexible. With the font-weight property, you can assign different degrees of boldness. The weight values used with this property are *normal*, *bold*, the relative values *bolder* and *lighter*, and the numerical values *100*, *200*, *300*, *400*, *500*, *600*, *700*, *800*, and *900*. Each numerical value represents a specific weight, or degree of boldness – at least in theory.

With numerical values, the amount of bold given to a particular font depends on the variations available with that font. If there are only a few weight variations in a given font, only certain numerical values will have an effect on its degree of bold. And of course, Navigator and Internet Explorer handle this problem differently, so once again it’s vital that you test your font-weight styles in a variety of browsers and platforms.

Note: Unfortunately, support for the 100-900 number values is unreliable and dependent upon the font in use. The only number values that are likely to have any effect are 700 and 900. The font weight property will become more powerful once the new standard, OpenType, is supported, which will enable users to view fonts even if they’re not installed on the user’s system.

The Font-Style Property

normal (default)	italic
------------------	--------

You can use the *font-style* property to italicize text like the tag in HTML. Unlike the font-weight property, with its varied levels of bold, there are no degrees of italicization. There are two values that currently work in both Navigator and Internet Explorer: *italic* and *normal*. Setting the font-style to *normal* is useful if you want to prevent a particular element from inheriting an italic style.

The Font-Variant Property

normal (default)	small-caps
------------------	------------

Another no-brainer. You can either create small caps, or use the “normal” value to override inheritance. The problem with using font-variant is that Navigator 4.x doesn’t support it at all, and Internet Explorer 4.x and 5 incorrectly make all characters, even capital letters, the same height. So there’s no real difference, in effect, between small caps and uppercase letters applied with the text-transform property. Meanwhile, in Internet Explorer 4.x for the Macintosh, the first letter of every word gets capitalized, and all others are lowercase, and that’s what *text-transform: capitalize* is supposed to do. It’s a good idea to leave this property alone until it’s correctly supported.

The Font Property

With the *font* shorthand property, you can declare all the font properties in one. For example, you can declare the font-style, font-weight, font-size, line-height, font-variant, and font-family without having to specify each property, as follows:

```
p { font: italic bold .8em/2em Helvetica, sans-serif }
```

The preceding rule declares the following:

```
p { font-style: italic;
    font-weight: bold; font-size: .8em;
    line-height: 2em;
    font-family: Helvetica, sans-serif; }
```

In the shorthand property, each property is implicit in its value. For any property left out, its default, or initial value is applied. Note that all properties are separated by white space, and multiple values of the font-family property are separated by commas, as usual. The font-size and line-height properties are grouped together and separated by a forward slash. Font-size on the left of the slash, and line-height on the right. Also, to make sure the rule works in Navigator 4.x, you have to include font-family in the mix, even if the element already inherits an appropriate value. The rule must also begin with either the font-style or font-weight values, and font-family should be listed last, otherwise, the entire rule won't work.

The font shorthand is nice, but there are obvious risks; first, older browsers won't follow it, and it's still bound to all the browser/platform compatibility problems that each property might have.