2. HTML Document Basics

Chapter 2 describes the characteristics of an HTML document, including some of the basic HTML elements and their attributes. The list of attributes is not necessarily complete, but rather includes a subset that is used in this book. The chapter includes a description of how to set colors in documents and a brief introduction to cascading style sheets.

2.1 Documents, Elements, Attributes, and Values

2.1.1 Essential Elements

As noted in Chapter 1, JavaScript needs an HTML document to serve as a user interface. Or, stated the other way around, HTML documents need a scripting language such as JavaScript to manage interactions with users. A basic HTML document consists of four sections defined by four sets of elements, arranged as follows:

```
<html>
      <head>
            <title> … </title>
            …
      </head>
      <body>
            …
      </body>
</html>
```

Each of these elements has a start tag and an end tag. Tags are always enclosed in angle brackets <…> and the end tag always includes a forward slash before the element name. The `body` element supports attributes that can be used to control the overall appearance of an HTML document. Documents, elements, attributes, and values are organized in a specific hierarchy:

HTML document → elements → attributes → values

Elements exist within a document. Elements can have attributes and attributes (usually) have values. Note that some elements are nested

inside others. For example, all the other elements are nested inside the `html` element, and the `title` element is nested inside the `head` element.

Following is a brief description of the four elements that will be part of every HTML document. Attributes, if any, are listed for each element. Note, however, that not all the possible attributes are listed. Thus, a listing of "none" may mean that there are attributes for this element, but that they are not used in this book. Consult an HTML reference manual for a complete list of attributes. As several elements can share common attributes, attributes and their values are listed separately, following the list of elements.

`<html> … </html>`

The `html` element surrounds the entire document. All other HTML elements are nested within this element.
Attributes: none

`<head> … </head>`

The `head` element contains information about the document. The `head` element must contain a `title` element and under XHTML rules, the `title` must be the first element after `head`. From our perspective, the other important element to be included in `head` is `script`, which will contain JavaScript code.
Attributes: none

`<title> … </title>`

The `title` element contains the text that will be displayed in the browser's title bar. Every HTML document should have a title, included as the first element inside the `head` element.
Attributes: none

`<body> … </body>`

The `body` element contains the HTML document content, along with whatever elements are required to format, access, and manipulate the content.
Attributes: `background`, `bgcolor`, `text`

2.1.2 Some Other Important Elements

The four basic elements discussed above constitute no more than a blank template for an HTML document. Other elements are needed to display and control the appearance of content within the document. Following are some important elements that you will use over and over again in your HTML documents, listed in alphabetical order. The list of attributes

is not necessarily complete, but includes only those that are used in this book.

`<a> … </a>`

The `a` (for "anchor") element provides links to an external resource or to an internal link within a document.
Attributes: `href`, `name`

`<b> … </b>`

The `b` element forces the included text to be displayed in a bold font. This is a "physical element" in the sense that it is associated specifically with displaying text in a bold font, even though the actual appearance may depend on the browser and computer used. In contrast, see the `strong` element below.
Attributes: none

`<br /> or <br>`

The `br` element inserts a break (line feed) in the text. Multiple breaks can be used to insert multiple blank lines between sections of text. The break element has no end tag because it encloses no content. Under XHTML rules, a closing slash (after a space) must be included: `<br />`. The slash is rarely seen in older HTML documents, so its use will be encouraged but not required.
Attributes: none

`<center> … </center>`

The `center` element causes displayed text to be centered on the computer screen.
Attributes: none

`<em> … </em>`

This is a "logical element" that will typically cause text to be displayed in italics, but it can be redefined to produce different results in different environments. For most purposes, `em` and `i` are interchangeable. See the `i` element below.
Attributes: none

`<font> … </font>`

The `font` element controls the appearance of text. The two most commonly used attributes control the size and color of the text.
Attributes: `size`, `color`, `face`

`<hr />` or `<hr>`

The horizontal rule element draws a shaded horizontal line across the screen. It does not have an end tag. A closing slash (after a space) is required in XHTML. A `noshade` attribute displays the rule as a solid color, rather than shaded.

Attributes: `align`, `color`, `noshade`, `size`, `width`

`<h`*n*`> … </h`*n*`>`

Up to six levels of headings (for *n* ranging from 1 to 6) can be defined, with decreasing font sizes as *n* increases from 1 to 6.

Attributes: `align`

`<i> … </i>`

`i` is a "physical element" that forces the included text to be displayed in italics. The actual appearance may depend on the browser and computer used. Compare with the `em` element above.

Attributes: none

`<img />`

The `img` element provides a link to an image to be displayed within a document. The image is stored in a separate file, perhaps even at another Web address, the location of which is provided by the `src` attribute.

Attributes: `align`, `border`, `height`, `src`, `vspace`, `width`

`<p> … </p>`

The `p` element marks the beginning and end of a paragraph of text content. Note that HTML does not automatically indent paragraphs. Rather, it separates paragraphs with an empty line, with all the text aligned left. It is common to see only the start tag used in HTML documents, without the corresponding end tag. However, the use of the end tag is enforced by XHTML, and this is the style that should be followed.

Attributes: none

`<pre> … </pre>`

The default behavior of HTML is to collapse multiple spaces, line feeds, and tabs to a single space. This destroys some of the text formatting that you may wish to preserve in a document, such as tabs at the beginning of paragraphs.

The `pre` element forces HTML to recognize multiple spaces, line feeds, and tabs embedded in text. The default action for `pre` is to use a monospaced font such as `Courier`. This may not always be appropriate, but as line feeds and other text placement conventions are

recognized, `pre` is very useful for embedding programming code examples within an HTML document.
Attributes: none

`<strong> … </strong>`
`strong` is a "logical element" that typically causes text to be displayed in a bold font, but it can be redefined to produce different results in different environments. For most purposes, `b` and `strong` are interchangeable. Compare this with the `b` tag above.
Attributes: none

Note that most of the elements described here require both start and end tags. The general rule is that any element that encloses content requires both a start and an end tag. The `br` and `hr` elements do not enclose content, so no end tag is needed. However, `br` and `hr` should include a closing slash in their tags in order to be XHTML-compatible—for example, `<br />` rather than `<br>`, with a space before the slash.

Description of attributes:
These descriptions may not include all possible values. For a complete listing, consult an HTML reference manual.

`align = "…"`
Values: `"left"`, `"right"`, or `"center"`
Aligns text horizontally.

`background = "…"`
Value: the URL of a gif- or jpeg-format graphics file
Setting the background attribute displays the specified image as the background behind a displayed HTML document page. Depending on the image size (in pixels), background images may automatically be "tiled," resulting in a repeating image that can be visually distracting. It is not necessary to use background images, and they should be used with care.

`bgcolor = "…"`
Values: Background colors can be set either by name or by specifying the intensity of the red, green, and blue colors. This topic is addressed in Section 2.5.

`border="…"`
Value: The width, in pixels, of a border surrounding an image

`color = "…"`
Values: Text colors can be set either by name or by directly specifying the intensity of the red, green, and blue colors. See Section 2.5.

`face = "…"`
Values: Font typefaces can be set either generically, with `cursive`, `monospace`, `sans-serif`, or `serif`, or with specific font names supported by the user's computer.

The generic names should always produce something that looks reasonable on any computer, but specific font names that are not available on the user's computer may produce unexpected results.

`height = "…"`
Value: The height, in pixels, of an image.

`href = "…"`
Value: The URL of an external or internal Web resource or the name of an internal document reference.

`hspace = "…"`
Value: The horizontal space, in pixels, between an image and the surrounding text.

`name = "…"`
Value: The name assigned to an internal document reference through an "`a`" element.

`size = "…"`
Values: An unsigned integer from 1 to 7 or a signed number from +1 to +6 or –1 to –6.

An unsigned integer is an absolute font size, which may be system-dependent. The default value is 3. A signed integer is a font size relative to the current font size, larger for positive values and smaller for negative values.

For the `hr` element, `size` is the vertical height of the horizontal rule, in pixels.

`src = "…"`
Value: The URL of a graphics file. For local use, images and their HTML document are usually stored in the same folder.

`text = "…"`
Values: The `text` attribute, used with the `body` element, selects the color of text in a document, which prevails unless overridden by a `font` attribute.

`vspace = "…"`
Value: The vertical space, in pixels, between an image and the surrounding text.

`width = "…"`
Values: The width of an image or horizontal rule, in pixels or as a percent of total screen width. For example, `width="80"` is interpreted as a width of 80 pixels, but `width="80%"` is a width equal to 80 percent of the total screen width.

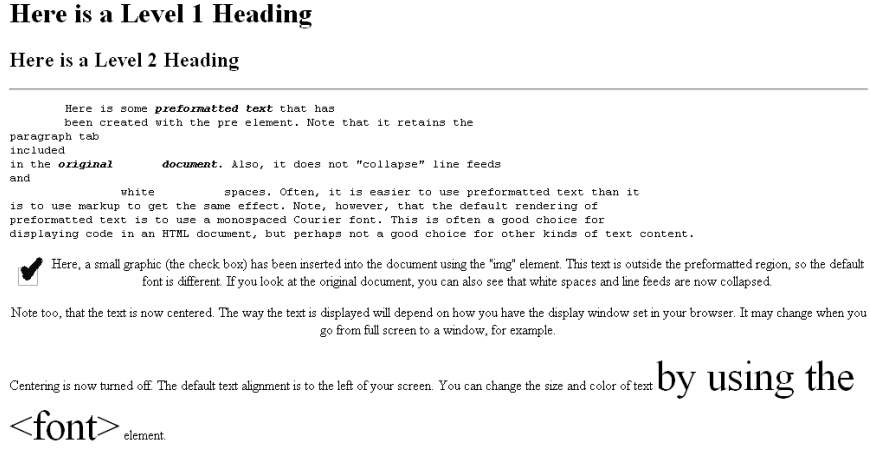
Document 2.1 illustrates how some of these elements are used.

Document 2.1 (`tagExamples.htm`)

```
<html>
<head>
<title>Tag Examples</title>
</head>
<body bgcolor="white">
<h1>Here is a Level 1 Heading</h1>
<h2>Here is a Level 2 Heading</h2>
<hr />
<pre>
      Here is some <strong><em>preformatted
text</em></strong> that has
      been created with the pre element. Note that it
retains the
paragraph tab
included
in the <b><i>original       document</b></i>. Also, it does
not "collapse" line feeds
and
             white          spaces. Often, it is easier to
use preformatted text than it
is to use markup to get the same effect. Note, however, that
the default
rendering of
preformatted text is to use a monospaced Courier font. This
is often a good choice for
displaying code in an HTML document, but perhaps not a good
choice for other kinds of text content.
</pre><p><center>
<img src="checkmark.gif" align="left" />Here, a small
graphic (the check box) has been inserted into
the document using the "img" element. This text is outside
the preformatted
region, so the default font is different. If you look at the
original document, you can also see that
white         spaces and line   feeds are now collapsed.
</p><p>
Note too, that the text is now centered. The way the text is
displayed will
```

```
depend on how you
have the display window set in your browser. It may change
when you go from full screen to a window, for example.
</center></p><p>
Centering is now turned off. The default text alignment is
to the left of your screen.
You can change the size and color of text <font size="7"
color="blue"> by using the &lt;font&gt;</font>
<font color="purple">element.</font>
</body>
</html>
```

Below is one rendering of Document 2.1. The small checkbox graphic has been created with the Windows Paint program. The actual text displayed in your browser is larger than this, but the output image has been reduced in size (perhaps to the extent of not being readable) to fit on the page. Moreover, because of the line feeds imposed on the text of this code example by the page width, the output looks a little different from what you might expect. So, you have to try this document on your own browser.



Here is a Level 1 Heading

Here is a Level 2 Heading

Here is some ***preformatted text*** that has
been created with the pre element. Note that it retains the
paragraph tab
included
in the ***original*** ***document***. Also, it does not "collapse" line feeds
and
white spaces. Often, it is easier to use preformatted text than it
is to use markup to get the same effect. Note, however, that the default rendering of
preformatted text is to use a monospaced Courier font. This is often a good choice for
displaying code in an HTML document, but perhaps not a good choice for other kinds of text content.

Here, a small graphic (the check box) has been inserted into the document using the "img" element. This text is outside the preformatted region, so the default font is different. If you look at the original document, you can also see that white spaces and line feeds are now collapsed.

Note too, that the text is now centered. The way the text is displayed will depend on how you have the display window set in your browser. It may change when you go from full screen to a window, for example.

Centering is now turned off. The default text alignment is to the left of your screen. You can change the size and color of text by using the <font> element.

Document 2.1 answers an interesting question: How can HTML display characters that already have a special meaning in the HTML language or that do not appear on the keyboard? The angle brackets (< and >) are two such characters because they are part of HTML tags. They can be displayed with the `<` and `>` escape sequences (for the "less than" and "greater than" symbols from mathematics). There are many standardized escape sequences for special symbols. A list of some of them is given in Appendix 2.

2.2 HTML Syntax and Style

A general characteristic of programming languages is that they have very strict syntax rules. HTML is different in that regard, as it is not highly standardized. The positive spin on this situation is to call HTML an "open standard," which means that self-described bearers of the standard can treat the language as they see fit, subject only to usefulness and market acceptance. HTML has an established syntax, but it is very forgiving about how that syntax is used. For example, when a browser encounters HTML code that it does not understand, typically it just ignores it rather than crashing, as a "real" program would do.

Fortunately, market forces—the desire to have as many people as possible accept your browser's interpretation of HTML documents—have forced uniformity on a large subset of HTML. This book adopts some HTML style conventions and syntax that are as platform-independent as possible. Although these "rules" might seem troublesome if you are not used to writing stylistically consistent HTML documents, they should actually help beginners by providing a more stable and predictable working environment. The only things worse than having syntax and style rules are having no rules or rules that nobody follows.

Some of the style rules used in this book are listed below. Under the circumstances of HTML, they are more accurately referred to as "guidelines." Some of them will make more sense later on, as you create more complicated documents.

1. Spell the names of HTML elements in lowercase letters.

Unlike JavaScript and some other languages, the HTML language is not sensitive to case. Thus, `<html>`, `<HTML>`, and `<hTmL>` are equivalent. However, the XHTML standard requires element names to be spelled with lowercase letters. In the earlier days of HTML, many programmers adopted the style of using uppercase letters for element names because they stood out in a document. You will often still see this style in Web documents. Nonetheless, we will consistently use lowercase letters for element names.

2. Use the `pre` element to enforce text layout whenever it is reasonable to use a monospaced font (such as `Courier`).

HTML always collapses multiple "white space" characters—spaces, tabs, and line breaks—into a single space when text is displayed. The easiest way to retain white space characters is to use the `pre` element. Other approaches may be needed if proportional fonts are required. Furthermore, tabbed text may still not line up, as different browsers have different default settings for tabs.

3. Nest elements properly.

Improperly nested elements can cause interpretation problems for your browser. Even when browsers do not complain about improperly nested elements, HTML is easier to learn, read, and edit when these restrictions are enforced.

Recall the following markup in Document 2.1:

```
Here is some <strong><em>preformatted
text</em></strong>
```

If you write this as

```
Here is some
<strong>
      <em>
            …{text}
      </em>
</strong>
```

it is easy to see that the `em` element is properly nested inside the `strong` element. If this is changed to

```
<strong><em> …{text} </strong></em>
```

your browser probably will not complain, but it is not good programming style.

4. Enclose the values of attributes in single or double quotes.

In Document 2.1, `bgcolor="white"` is an attribute of `<body>`. Browsers generally will accept `bgcolor=white`, but the XHTML standard enforces the use of quoted attribute values. This book is consistent about using double quotes unless attribute values appear inside a string that is surrounded with double quotes (for example, an attribute value embedded in a parameter in the `document.write()` method). Then attribute values will be single-quoted.

2.3 Using the `script` Element

The `script` element usually (but not always) appears inside the `head` element, after the `title` element. Following is a description of `script` along with its essential attributes:

```
<script language="javascript" type="text/javascript">
  …
```

```
</script>
```
Attributes: `language`, `type`, `src`

The values usually assigned to the `language` and `type` attributes are `language="javascript"` and `type="text/javascript"`. The values shown in the description are default values, so for documents using JavaScript, inclusion of these attributes is usually not actually required.

The `src` attribute has a value corresponding to the name of a file containing JavaScript script, usually (but not necessarily) with a `.js` extension. This attribute is used in a later chapter.

2.4 Creating and Organizing a Web Site

Obviously this is a major topic, a thorough investigation of which would go far beyond the reach of this text. There is an entire industry devoted to hosting and creating Web sites, including helping a user obtain a domain name, providing storage space, developing content, and tracking access. For the purposes of a course based on this text, the goal is extremely simple: create a Web site sufficient to display the results of work done during the course.

The first step toward creating a Web site is establishing its location. In an academic environment, a college, university, or department computer may provide space for web pages. A URL might look something like this:

```
http://www.myuniversity.edu/~username
```

where the "~" symbol indicates a directory where Web pages are stored. Together with a user name, this URL directs a browser to the home Web directory for that user. As noted in Chapter 1, as HTML documents are not automatically Internet-accessible, your Web pages for this book may be accessible only locally on your own computer.

In this home directory there should be at least one file called `index.htm` (or `index.html`). UNIX systems favor the `.html` extension, but Windows users should use the three-character `.htm` extension to remain compatible with Windows file extension conventions. This is the file that will open automatically in response to entering the above URL. That is, the `index.htm` file is the "home page" for the Web site. This home page file could be named something else, but then its name would have to be added to the URL:

```
http://www.myuniversity.edu/~username/HomePage.htm
```

An `index.htm` file can contain both its own content as well as links to other content (hyperlinks), including other pages on the user's Web site and to external URLs. Following are four important kinds of links:

1. Links to other sites on the World Wide Web.

The following is the basic format for globally linking Web pages:

syntax: `<a href="`*{URL of Web page}*`">`
{description of linked Web page}`</a>`

The URL may refer to a completely different Web site, or it may be a link to documents in the current folder or a subfolder within that folder.

2. Links to images.

The `img` element is used to load images for display or to use as a page background:

syntax: `<img src="`*{URL plus image name}*`" align="…" height="…" width="…" />`

The image may exist locally or it may be at a different Web site. The `align`, `height`, and `width` attributes, which can be used to position and size an image, are optional. However, for high-resolution images, it is almost always necessary to specify the height and width as a percentage of the full page or as a number of pixels in order to reduce the image to a manageable size in the context of the rest of the page. Resizing the image, if possible, will solve this problem.

You can also make a "clickable image" to direct the user to another link:

Syntax: `<a href="`*{URL of Web page}*`">`
`<img src="`*{URL plus image name}*`" align="…" height="…" width="…" /></a>`

3. Links to e-mail addresses.

An e-mail link is an essential feature that allows users to communicate with the author of a Web page.

syntax: `<a href="mailto:`*{e-mail address}*`">`
{description of recipient}`</a>`

Often, but not necessarily, the *{description of recipient}* is also the e-mail address. The actual sending of an e-mail is handled by the default mailer on the sender's computer.

4. Internal links within a document.

Within a large document, it is often convenient to be able to move from place to place within the document using internal links.

Syntax: `<a href="{#`*internal link name*`}">`
{description of target position}`</a>`
…
`<a name="`*{internal link name}*`">`*{target text}*`</a>`

The "`#`" symbol is required when specifying the value of the `href` attribute, in order to differentiate this internal link from a link to another (external) document.

The careless use and specification of hyperlinks can make Web sites very difficult to maintain and modify. As noted above, every Web site should have a "home" directory containing an `index.htm` file. In order to make a site easy to transport from one computer to another, all other content should be contained either in the home directory or in folders created within that directory. References to folders that are not related in this way should be avoided, as they will typically have to be renamed if the site is moved to a different computer. Although it is allowed as a matter of syntax to give a complete (absolute) URL for a local Web page, this should be avoided in favor of a reference relative to the current folder.

This matter is important enough to warrant a complete example. Document 2.2a–c shows a simple Web site with a home folder on a Windows desktop called `home` and two subfolders within the `home` folder named `homework` and `personal`. Each subfolder contains a single HTML document, `homework.htm` in `homework` and `resume.htm` in `personal`.

Document 2.2a (`index.htm`)

```
<html>
<head>
<title>My Page</title>
</head>
<body>
<!-- These absolute links are a bad idea! -->
Here are links to
<a href="C:/Documents and Settings/David/desktop/
JavaScript/Book/homework.htm">homework</a> and
<a href="C:/Documents and Settings/
```

```
  David/desktop/JavaScript/Book/resume.htm">
personal documents.</a>
</body>
</html>
```

Document 2.2b (`resume.htm`)

```
<html>
<head>
<title>Resumé</title>
</head>
<body>
Here is my resumé.
</body>
</html>
```

Document 2.2c (`homework.htm`)

```
<html><head>
<title>Homework</title>
</head>
<body>
Here are my homework problems.
</body>
</html>
```

Note that Document 2.2a uses forward slashes to separate the directories and file names. This is consistent with UNIX syntax, but Windows/DOS systems use backward slashes. Forward slashes are the HTML standard, and they should always be used even though backward slashes may also work. Another point of interest is that UNIX directory paths and filenames are case-sensitive, but Windows/DOS paths and filenames are not. This could cause problems if you develop a Web page on a Windows/DOS computer and then move it to a UNIX-based system. As a matter of style, you should be consistent about case in directory and file names even when it appears not to matter.

Absolute references to a folder on a particular Windows computer desktop are a bad idea because such references will have to be changed if the `index.htm` file is moved to a different place on the same computer, or to a different computer—for example, to a UNIX university department computer with a different directory/folder structure. Document 2.2d shows the preferred solution. Now the paths to `homework.htm` and `resume.htm` are given relative to the home folder, wherever the `index2.htm` file resides. (Remember that this file, no longer named `index.htm`, will not be recognized as a default home page.) This document assumes that folders `homework` and `personal` exist in the home folder. The relative URL should work without modification when the Web site is moved to a different computer. If the Web

site is moved, only a single reference, the one to the `index2.htm` file, has to be changed.

Document 2.2d (`index2.htm`, a new version of `index.htm`)

```
<html>
<head>
<title>My Page</title>
</head>
<body>
<!-- Use these relative links instead! -->
Here are links to
<a href="homework/homework.htm">homework</a>
and <a href="personal/resume.htm">personal documents.</a>
</body>
</html>
```

When designing a Web site proper attention to the use of relative URLs from the very beginning will save a lot of time in the future!

2.5 Selecting and Using Colors

As previously noted, several attributes, such as `bgcolor`, are used to set colors of text or backgrounds. Colors may be identified by name or by a six-character hexadecimal numeric code that specifies the strength of the signal emitted from the red, green, and blue electron "guns" that excite the corresponding phosphors on a cathode ray tube color monitor screen. This convention is retained even when other display technologies are used. The **hex code** is in the format `#RRGGBB`, where each color value can range from `00` (turned off) to `FF` (maximum intensity).

There are many color names in use on the Web, but only 16 are standardized, representing the 16 colors recognized by the Windows VGA color palette.

Table 2.1. A list of 16 standard HTML color names and hex hodes

Color Name	Hexadecimal Code
aqua	#00FFFF
black	#000000
blue	#0000FF
fuchsia	#FF00FF
gray	#808080
green	#008000
lime	#00FF00
maroon	#800000
navy	#000080
olive	#808000
purple	#800080
red	#FF0000
silver	#C0C0C0
teal	#008080
white	#FFFFFF
yellow	#FFFF00

These colors are listed in Table 2.1. The problem with additional color names is that there is no enforced standard for how browsers should interpret them. Two examples: magenta probably should be, but does not have to be, the same as fuchsia; ivory is a nonstandard color that should be rendered as a yellowish off-white. The colors in Table 2.1 are standardized in the sense that all browsers should associate these 16 names with the same hexadecimal code. Of course, variations can still occur because monitors themselves respond somewhat differently to the same name or hex code; blue on my computer monitor may look somewhat different than blue on your monitor.

Note that the standardized colors use a limited range of hex codes. With the exception of silver (nothing more than a lighter gray), the RGB gun colors are off (`00`), on (`FF`), or halfway on (`80`).

What should you do about choosing colors? Favor standardized colors, and if you wish to make an exception, try it in as many browser environments as possible. Be careful to choose background and text colors so that the text will always be visible against its background. The safest approach for setting colors in the `body` element is to specify both background and text colors. This will ensure that default colors set in a user's browser will not result in unreadable text.

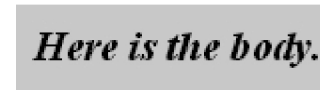
If you are not sure whether a color name is supported and what it looks like on your monitor, you have nothing to lose by trying it. If you set `bgcolor="lightblue"`, you will either like the result or not. If a color name is not recognized by your browser, the result will be unpredictable, but not catastrophic. There are (of course) numerous Web sites that can help you work with colors, including getting the desired result with hex codes.

2.6 Using Cascading Style Sheets

As you create more Web pages, you may wish to impose a consistent look for all of your pages or for groups of related pages. It is tedious to insert elements for all the characteristics you may wish to replicate—font size, font color, background color, and so forth. Style sheets make it much easier to replicate layout information in multiple documents..A complete discussion of style sheets is far beyond the scope of this book, as there are many different kinds of style sheets, many ways to make use of them, and many browser-specific nuances. This book uses **cascading style sheets** (CSSs), which are widely accepted as a default kind of style sheet, but presents only a *small* subset of all the possibilities! By way of introduction, Document 2.3 illustrates the use of a `style` element to establish the default appearance of the body of an HTML document.

Document 2.3 (`style1.htm`)

```
<html>
<head>
<title>Style Sheets</title>
<style title="David's default" type="text/css">
       body.bright {background: red; font: 16pt serif;
          color: blue; font-style: italic; font-weight: bold}
</style>
</head>
<body class="bright">
Here is the body.
</body>
</html>
```



The `style` element has an optional `title` attribute and a `type` attribute set equal to `"text/css"`, where the `css` stands for cascading style sheet. This `style` element gives the `body` style a name (`bright`) and sets the document background color to red and the default font to bold, 16-point serif, blue, and italicized. Note the use of the dot notation to assign a **class name** to the **style rule(s)** established for the element, and the use of the name later (`class="bright"`) with the `class` attribute in the `<body>` tag. Each style rule is terminated with a semicolon. So, for example, the line

```
{font: 16pt serif; color: blue;}
```

gives one rule for setting font properties and a second for setting text color. When multiple properties are set for the same element, they are enclosed in curly brackets.

For this simple example, with styles applying only to a single `body` element, the class name is optional. In general, several different style rules can apply to the same HTML element. For example, several different style rules could be established for paragraphs (`<p> … </p>`), each of which would have its own class name.

In summary, style specifications follow a hierarchy:

`style` element → other HTML elements*[.class name]* →
properties → value(s)

where the class name (without the brackets) is optional.

How did CSSs get that name? The answer is that the properties set for an element cascade down, or are "inherited," by other elements contained within it unless those elements are assigned their own style properties. So, for example, properties set for the `body` element are inherited by the `p` and `h1` elements because these are contained within the

`body` element. Properties set for the `head` element are inherited by content appearing in the `title` element.

CSSs can be used to modify the appearance of any HTML element that encloses content. Following are some properties that can be specified in style sheets.

Background properties

`background-color`

When used in a `body` element, `background-color` sets the background color for an entire document. It can also be used to highlight a paragraph, for example, when used with a `p` element.

`background-image`

This property is used with a URL to select an image file (gif or jpeg) that will appear as a background. Typically, this is used with a `body` element, but it can also be used with other elements, such as `p`. For other background properties that can be used to control the appearance of a background image, consult an HTML reference text.

`background`

This allows you to set all background properties in a single rule.

Color property

The `color` property sets the default color for text, using the descriptions discussed in Section 2.5.

Font properties

`font-family`

Font support is not completely standardized. However, browsers that support style sheets should support at least the generic font families listed in Table 2.2.

Table 2.2. Generic font families

Generic Name	Example
`cursive`	Zapf-Chancery
`monospace`	Courier
`sans-serif`	Arial
`serif`	Times

Example: `font-family: Arial, sans-serif;`

`font-size`

This property allows you to set the actual or relative size of text. You can use relative values, such as `large`, `small`, `larger`, `smaller` (relative to a default size); a percentage, such as `200%` of the default size;

or an actual point size such as `16pt`. Some sources advise against using absolute point sizes because a point size that is perfectly readable on one system might be uncomfortably small on another. For our purposes, specifying the point size is probably the easiest choice.

Example: `font-size: 24pt;`

`font-style`

This property allows you to specify `normal`, `italic`, or `oblique` fonts.

Example: `font-style: italic;`

`font-weight`

This property allows you to select the font weight. You can use values in the range from `100` (extra light) to `900` (extra bold), or words: `extra-light`, `light`, `demi-light`, `medium`, `demi-bold`, `bold`, and `extra-bold`. Some choices may not have a noticeable effect on some fonts in some browsers.

Example: `font-weight: 900;`

`font`

This property allows you to set all font properties within one style rule.

Example: `font: italic 18pt Helvetica, sans-serif;`

How will your browser interpret a generic font name? For the generic name `serif`, it will pick the primary serif font that it supports—probably Times or Times Roman. Browsers will probably also recognize specific font names such as Times or Helvetica (a sans-serif font). If you specify a font name not supported by your browser, it will simply ignore your choice and use its default font for text. It is possible to list several fonts, in which case your browser will select the first one it supports. For example, consider this rule:

```
font-family: Arial, Helvetica, sans-serif;
```

Your browser will use an Arial font if it supports that, Helvetica if it does not support Arial but does support Helvetica, or, finally, whatever sans-serif font it does support. By giving your browser choices, with the generic name as the last choice, you can be reasonably sure that text will be displayed with a sans-serif font.

Text properties
Of the many text properties, just three that may be useful are shown below.

`text-align`
This is used in block elements such as `p`. It is similar in effect to the HTML `align` attribute. The choices are `left`, `right`, `center`, and `justify`. With large font sizes, `justify` may produce odd-looking results.

Example: `text-align: center;`

`text-indent`
Recall that paragraphs created with the `p` element do not indent the first word in the paragraph. (HTML inserts a blank line, but left-justifies the text.) This property allows you to set indentation using typesetting notation or actual measurements. I suggest the use of actual English or metric measurements—inches (`in`), millimeters (`mm`), or centimeters (`cm`).

Example: `text-indent: 0.5in;`

`white-space`
The value of this property is that you can prevent spaces from being ignored. (Remember that the default HTML behavior is to collapse multiple spaces and other nonprintable characters into a single blank space.) You can use the HTML `pre` element by itself, instead, but this causes the text to be displayed in a monospaced font such as Courier. (At the time this book was written, not all browsers supported this property.) The example given here retains white space regardless of the typeface being used.

Example: `white-space: pre;`

Styles are not restricted just to the body element. For example, paragraphs (`<p> … </p>`) and headings (`<h`*n*` > … </h`*n*`>`) can also have styles associated with them. You can also set styles in selected portions of text using the `span` element, and in blocks of text using the `div` element.

`<div> … </div>`
Attributes: `align`, `style`

`<span> … </span>`
Attributes: `align`, `style`
Values for align: `"left"` (default), `"right"`, `"center"`

You can create style sheets as separate files and then utilize them whenever you wish to use a particular style on a Web page. This makes it easy to impose a uniform appearance on multiple Web pages. Documents 2.4a and 2.4b show a simple example.

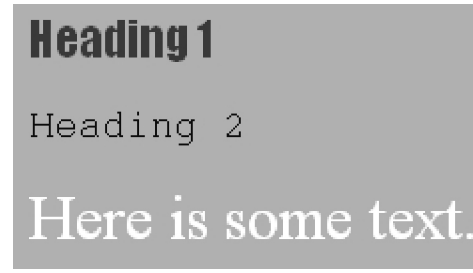
Document 2.4a (`body.css`)

```
body {background:silver; color:white; font:24pt Times}
h1 {color:red; font:18pt Impact;}
h2 {color:blue; font:16pt Courier;}
```

Document 2.4b (`style2.htm`)

```
<html>
<head>
<title>Style Sheet Example</title>
<link href="body.css" rel="stylesheet"
  type="text/css" />
</head>
<body>

  <h1>Heading 1</h1>
  <h2>Heading 2</h2>
  Here is some text.
</body>
</html>
```



(See Color Example 3 for full-color output.)

This example shows how to create a file, `body.css`, containing style elements that can be applied to any document by using the `link` element, as in Document 2.4b. The `.css` extension is standard, but not required. (You could use `.txt`, for example.) Although this example is very simple, the concept is powerful because it makes it easy to create a standard style for all your documents that can be invoked with the `link` element. The `Impact` font chosen for `h1` headings will not be supported by all browsers, in which case the default font will be used in its place.

The attributes of `link` include `href`, which contains the URL of the style sheet file, the `rel="stylesheet"` (relationship) attribute, which describes how to use the file (as a style sheet), and the `type`, which should be `"text/css"`, just as it would be defined if you created a `style` element directly in the `head` element. In this example, `body.css` is in the same folder as `style2.htm`. If you keep all your style sheets in a separate folder, you will need a more explicit URL.

It is worth re-emphasizing that this discussion of style sheets has barely scratched the surface of the subject. Style sheets can make your Web pages more visually appealing and can greatly simplify your work

on large Web projects. Some Web developers advocate replacing *all* individual formatting elements, such as `font` and its attributes, with style sheet specifications. In newer versions of HTML, and in XHTML, the use of individual formatting elements is "deprecated," but there is little likelihood that support for them will disappear from browsers in the foreseeable future. A course based on this book does not require the use of cascading style sheets unless it is asked for specifically.

2.7 Another Example

Documents 2.5a and 2.5b show how to use a style sheet file to specify different background and text colors for different sections of text.

Document 2.5a (`rwb.css`)

```
p.red {background:red;color:blue;font:20pt Times}
div.white {background:white;color:red;font:20pt Times}
span.blue {background:blue;color:white;font:20pt Times}
```

DOCUMENT 2.5b (`rwb.htm`)

```
<html>
<head>
<title>A Red, White, and Blue Document</title>
<link href="rwb.css" rel="stylesheet" type="text/css" />
</head>
<body>
<img src="stars.jpg" height="150" width="250" />
<p class="red">
This text should be blue on a red background.
</p><p><div class="white" style="font-style: italic;">
This text should be red on a white background.
</div></p>
<p><span class="blue">This text should be white on a blue
background.</span>
</p>
</body>
</html>
```

This text should be blue on a red background

This text should be red on a white background

This text should be white on a blue background

(See Color Example 4 for full-color output.)

The stars (which are supposed to be red, silver, and blue) have been drawn using the Windows Paint program.

http://www.springer.com/978-1-84628-656-8

An Introduction to HTML and JavaScript
for Scientists and Engineers
Brooks, D.R.
2007, XI, 200 p. 6 illus. in color. With online
files/update., Softcover
ISBN: 978-1-84628-656-8