Chapter 1

How to Write XHTML and CSS

XHTML and CSS are two different specifications used to create web pages. Each has a distinct look and purpose. When used together, the combination can produce a useful, information-rich, and highly attractive web page. That's what this book is all about—learning how to use the two specifications seamlessly to build effective web pages.

In the exercises throughout this book you're going to work with examples of XHTML and CSS code. Studying and incorporating code that others have written to solve the same problems you're facing is one of the best ways to learn any web development language. You'll need to type these code examples into your pages exactly as they appear in the book for them to work. The various keywords and symbols are all significant. The rules defining how a language is put together are its *syntax*. In this chapter, you will learn the syntax of XHTML and CSS. You will learn what each of these specifications does and how to write code in its syntax. Basic rules for typing both XHTML and CSS, such as when to use the spacebar, when to type a semicolon, or when to type a bracket, will also be explained in this chapter.

This book assumes that most people starting out with CSS have probably written some HTML. If that's the case, you can treat this chapter as a refresher and an introduction to CSS. If you are already familiar with the basics of both HTML and CSS syntax, you can skip ahead to Chapter 2 without damage to your mental health or your social life.

In this chapter, you will learn to:

◆ Identify what constitutes a website.

◆ Identify what XHTML and HTML are.

◆ Explain similarities and differences in XHTML and HTML.

◆ Describe what CSS is.

◆ Write XHTML syntax.

◆ Write CSS syntax.

Anatomy of a Website

A summary of what goes into a website may help you understand what HTML/XHTML and CSS do and how they can work together to implement your vision. If you've already worked with one of the "visual tools" like Dreamweaver or FrontPage, you're a little further along the learning curve, but this recap will still help to put what you're about to learn into perspective. Understanding the underlying basics can give you better control over your visual tools as well.

A web page may contain text, images, links, sounds, and movies or moving images. You may also be aware that some pages use scripts written in various languages such as JavaScript, PHP, or ASP to create interactivity, to connect the page to a database, or to collect information submitted in a form.

HTML or XHTML is the glue that holds all those pieces and parts together and displays them in a readable or usable manner in a browser such as Internet Explorer, Firefox, Safari, or Netscape Navigator. The browser is your window on the World Wide Web; XHTML is the language used to tell the browser how to format the pieces and parts of a web page.

You'll use XHTML to structure the headings, paragraphs, tables, images, and lists that logically organize your information so that you can convey your message with words, images, and information. XHTML pages without any CSS attached might look plain and simple, but all your information and content still displays in an organized and clearly meaningful way. Without the correctly marked up and semantically organized underpinnings built in XHTML, no amount of CSS can save your web page from mistaken interpretation, inaccessibility, and poor performance.

CSS enters the scene by adding style to the elements on a web page. Whereas XHTML markup identifies each element in the page logically, defining its function within the overall structure, CSS styles tell the browser how each element should look. The style might involve color, placement, images, fonts, or spacing, but it does not change the underlying pieces and parts formatted by the XHTML. CSS is about presentation. CSS determines whether something is blue or green, aligned on the left or on the right, large or small, visible or hidden, has bullets or doesn't have bullets. However, these styles, grouped into style sheets, have to be applied to some kind of content, such as XHTML pages, to have any effect.

Web content is displayed in more ways than on a computer monitor. It appears in various handheld devices such as PDAs and cell phones. It is read aloud by aural screen readers. It is printed. A single page of XHTML content can be styled in various ways by different CSS rules for the most effective presentation in all of the devices mentioned.

The driving force behind the creation of CSS several years ago was to enable the separation of content from appearance (Figure 1.1). One page can be used in many devices with the proper styling. Your XHTML pages become universally usable, accessible on any device.

FIGURE 1.1
Create one semantically sensible XHTML page and style it for many different devices using CSS.
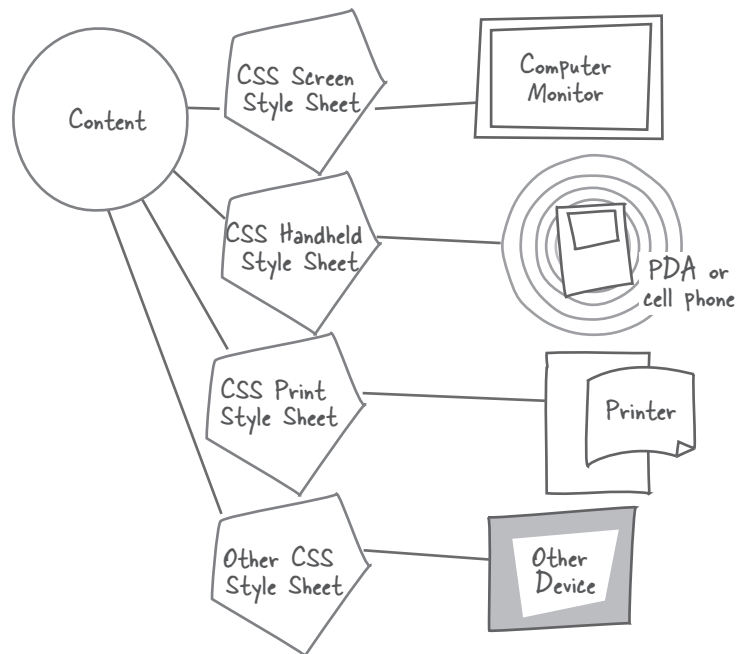
What Are XHTML and HTML?

Hypertext Markup Language (HTML) is the programming specification for how web pages can be written so they can be understood and properly displayed by computers or other Internet-capable devices. XHTML is an acronym for Extensible Hypertext Markup Language, a specification that grew out of HTML. You'll see what "extensible" means in a moment, but to understand the role of HTML and XHTML, you need to understand the three terms in HTML.

Hypertext is simply text as it exists in what is called "hyperspace"—the Internet. It is plain text that carries the content of your web page and the programming information needed to display that page and link it to other pages. Hypertext is formatted via a *markup language*—a standardized set of symbols and codes that all browsers can interpret.

WHO MAKES THE RULES?

Oh, yeah? Sez who! In the world of Internet technology, the guys with the "say" are in the W3C. The organization devoted to creating and publishing the standardized rules for various web technologies, including XHTML, is the World Wide Web Consortium, or W3C, at `www.w3.org`. See also the Web Standards Project, a grassroots coalition fighting for the adoption of web standards, at `www.webstandards.org`.

Markup is used to convey two kinds of information about text or other content on a web page: first, it identifies what kind of *structure* the content requires. If you think of a web page as simply a whole lot of words, the HTML is the markup, or framework, that specifies that certain words are headings or lists or paragraphs. The way you mark up the text on the page structures the page into chunks of meaningful information such as headings, subheads, and quotes. I often refer to such chunks of meaningful information as *semantic* structures.

Markup may also define the *presentation* of those elements; for example, the different fonts to be used for headings and subheadings. When it was first developed, HTML was the only tool for defining visual presentation on screen. When the World Wide Web began, the only information transmitted using the *Hypertext Transfer Protocol (HTTP)* was text. As the capability to transfer images, sounds, and other information was added, presentational markup was added to HTML to help format the new information. After a few years of amazing growth, the HTML that was being used to mark up individual elements reached burdensome proportions. It became apparent that markup for presentation was an inefficient way to define what every item of text or graphics on a website should look like, and the web community developed Cascading Style Sheets (CSS) as a better way to handle presentation.

What's the Difference between XHTML and HTML?

The title of this book refers to HTML, because XHTML actually *is* HTML. You'll quickly notice that an XHTML document is saved with a filename ending in `.html`, for example `mypage.html`. XHTML was chosen for the basis of all the code used in this book, but HTML 4.01 is still a perfectly good choice for writing web pages. You will be learning HTML when you learn XHTML. It is a two-for-the-price-of-one bargain. There are a few basic differences in writing XHTML versus writing HTML, and these will be pointed out to you at appropriate times in the book.

Many writers choose to emphasize that XHTML is HTML by using the term (X)HTML to indicate that what they are saying applies to both. Good XHTML is also good HTML. Don't be misled by the book's title into thinking that you won't learn XHTML.

But how are XHTML and HTML different? XHTML is more than HTML, because it is extensible. XHTML uses the syntax rules of the *Extensible Markup Language (XML).* Those syntax rules will be explained later in this chapter. An extensible markup language can be extended with modules that do things such as make math calculations, draw graphical images, or use microformats such as XFN (XHTML Friends Network). Web pages written in XHTML interact with XML easily.

What Is CSS?

CSS is an acronym for Cascading Style Sheets, another programming specification. CSS uses rules called *styles* to determine visual presentation. The style rules are integrated with the content of the web page in several ways. In this book, we will deal with style rules that are embedded in the web page itself, as well as with sets of style rules, known as style sheets, that are linked to or imported into a web page. You will learn to write the style rules and how to import, link, or embed them in the web pages you make.

In HTML, styles can be written into the flow of the HTML, or *inline,* as well. You'll learn more abut linked, embedded and inline styles in Chapter 2.

CSS can also be integrated into web pages in other ways. Sometimes you have no control over these rules. Browsers allow users to set up certain CSS style rules, or user styles, according to their own preferences. The user preferences can override style rules you write. Further, all browsers come with built-in style rules. Generally these built-in styles can be overridden by your CSS style rules. Built-in browser display rules are referred to as *default* presentation rules. Part of what you will learn is what to expect from a browser by default, in order to develop any new CSS rules to override those default display values. You can accept the browser default display, too. If you are happy with what the browser does to style the content, then there's no need to create a rule to over-ride it. It's common for designers to start a style sheet by setting the paddings and margins to 0px to override the browser's default padding and margin settings. Yet other default browser styles, such as the blank line separating one paragraph from another, may be left alone.

Getting Started with XHTML Syntax

The building blocks of XHTML syntax are *tags*, which are used to mark up *elements.* A tag is a code that gives an element its name. For example, the tag used to mark up a paragraph is a p tag, which is called either a "paragraph tag" or "a p tag." When text is marked up with a p tag, it is an instruction to the browser to display the element as a paragraph.

Elements in XHTML, such as paragraphs, can also have attributes and values assigned to them. But before you find out about attributes and values, let's dig into tags just a bit more.

Opening and Closing Tags

Opening and closing tags are used to specify elements. Here is a marked up paragraph element:

```
<p>This is the text of the paragraph.</p>
```

The paragraph is opened with a p tag. Tags are delineated with angle brackets (< and >). So the markup <p> instructs the browser that a paragraph starts now.

A closing tag </p> indicates the end of the paragraph. Notice that the closing tag is the same as the opening tag, with the addition of a forward slash (/) inside the opening bracket of the tag. Tags are like on and off switches: turn on a paragraph here and turn it off there. With a few more sentences added to make the paragraph show up, and a second identical paragraph added, this element would appear in a browser something like Figure 1.2.

FIGURE 1.2
Two paragraph
elements
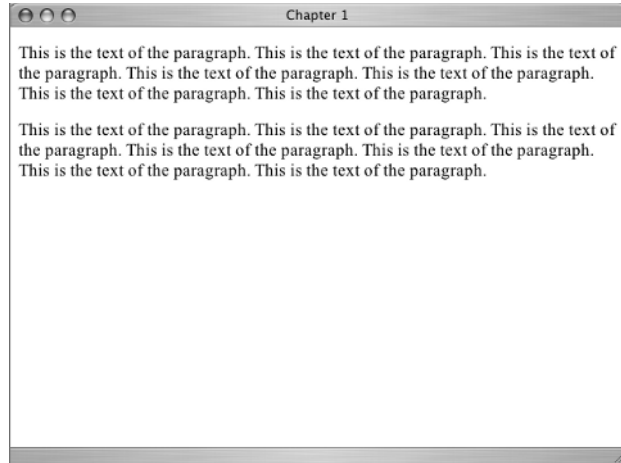
Notice that the browser left a blank line between the two paragraphs and that there is no indent-ing. This is an example of a *default* paragraph. A default display is the browser's built-in interpretation of how the element should be presented. One way to change the browser's default interpretation of an element is to include additional instructions in the form of *attributes* and *values* that further define the element.

DOCTYPEs AND PRESENTATIONAL ATTRIBUTES

In Chapter 3 you will learn about DOCTYPEs, which identify the specific version of XHTML a web document conforms to, and therefore which syntax elements will be allowed.

Keep in mind that while a Transitional HTML or XHTML DOCTYPE allows attribute and value instruc-tions in the element, a Strict DOCTYPE does not allow presentational attributes and values. That's because presentational attributes and values determine *appearance*, an activity better left to the CSS. Separation of content from appearance is one thing a Strict DOCTYPE enforces, well, strictly.

The concept of attributes and values is a widely used way of organizing information. In general, an attribute identifies a category of information about a given type of element, and the value identifies how the attribute is implemented in a specific instance of that element. In XHTML, an example of an attribute that might define a paragraph is alignment. All paragraphs have this characteristic; all are aligned in some way. Text in paragraphs can be left-aligned, right-aligned, centered, or justified. As you can see in Figure 1.2, the browser default for text alignment is left-aligned. In XHTML, the type of alignment you choose is the value. The exact attribute is `align`. The value of this attribute could be `left`, `right`, `center`, or `justify`.

An attribute is written as part of the opening tag. The attribute name is followed by an equal sign (=) and the value in quotation marks (`"`).

Here is a marked up paragraph element with an attribute and value.

```
<p align="right">This is the text of the paragraph.</p>
```

By adding `align="right"` to the first paragraph element in the XHTML that generated Figure 1.2, you can see the alignment change to an appearance similar to Figure 1.3.

FIGURE 1.3
The first paragraph
with align="right"
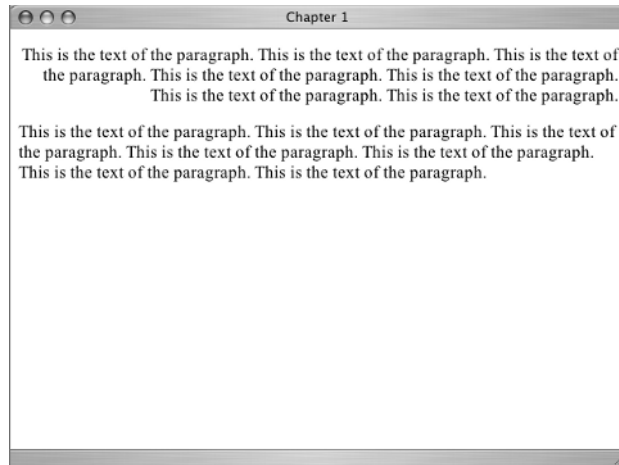
There are two important things to take note of in this example. First, there is a space between the tag name and the attribute name, align. The attribute is followed by an equal sign and the attribute value "right" is enclosed in quotation marks with no surrounding spaces. Attribute/value pairs in HTML/XHTML always follow this syntax. Also notice the closing tag. It does the job of ending the paragraph and the effect of the paragraph's attributes merely by using the forward slash (/) and the p again. When the paragraph ends, all of its attributes and values terminate with it.

One of the distinctions between XHTML and HTML is that closing tags are *required* by XHTML. In HTML, closing tags are not always required.

Empty Elements

An element with text in it, such as the paragraph examples we've seen, is considered nonempty. Some elements you put on a web page do not contain text. Such elements don't need closing tags and are referred to as *empty elements*. An example would be an image or a line break. Empty elements have specific requirements in XHTML, but let's start with an example in HTML:

```
<p>Jingle bells, jingle bells,<br>
Jingle all the way...</p>
```

The HTML tag br (for break) is used for a line break in formatting this paragraph. The line break doesn't have to open and close, it merely has to *be*. A line break moves whatever comes next down to a new line, without any intervening white space, which you would get by default if you put the second line in a new paragraph element.

Formatted with a break in the first line, this paragraph would display like Figure 1.4 in a browser. Look again at Figure 1.2 to note the difference between a line break and a new paragraph.

However, XHTML uses syntax based on the rules of Extensible Markup Language (XML) to write HTML. One of the requirements of XML is that every element must be terminated even if it's empty. That means every opening tag must have a corresponding closing tag. This concept is known as *well-formedness*. A well-formed document has no unclosed tags. You may be asking how an element such as br can be "terminated" if there is no closing tag. The solution in XML and XHTML is to add the closing forward slash to the empty tag itself. Empty tags in XHTML look like this example.

```
<p>Jingle bells, jingle bells,<br />
Jingle all the way...</p>
```

FIGURE 1.4
The line break moves
the text down to the
next line.
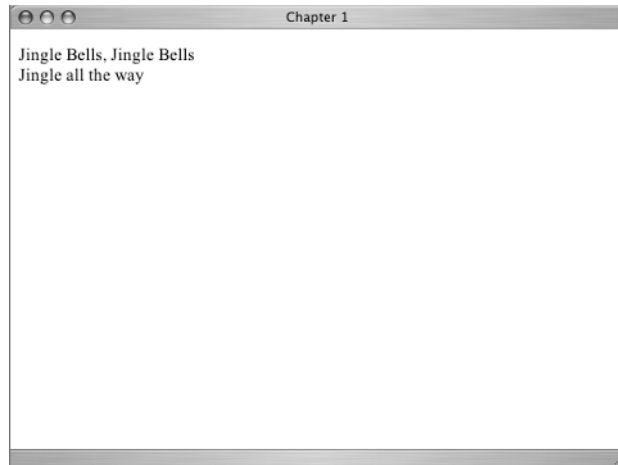
Even empty tags with attributes and values can be closed in this way. Notice the space between the br and the forward slash (/). That's the way it will be shown in the examples in this book, but the space doesn't *have to* be there.

The img (for image) tag is another empty element. Look at this example:

```
<img src="photo.gif" />
```

This empty element places an image on the page, and the source (src) of the image is given as an attribute of the img tag. The space and forward slash at the end give the empty element the required XHTML closing.

There are not many empty elements in XHTML. Others include horizontal rules (hr), the link element, and meta elements. Most of the time you will mark up text with both opening and closing tags. Even if you are writing HTML, where closing tags are not always required, it is considered good practice to include closing tags whenever possible.

SAY, HONEY, JUST HOW OLD IS YOUR BROWSER, ANYWAY?

The space before the closing forward slash is not required by XHTML. In other words, <br/> with no space before the forward slash, is correct. However, inserting a space enables older browsers to correctly display the document, so I will use the space here. (Older browsers are those venerable antiques earlier than version 5, such as Netscape 4.7.) As of this writing Microsoft Corp. is very close to releasing version 7 of Internet Explorer, which most web designers fervently hope will quickly send all older versions of Internet Explorer to the land of the Model T and the Studebaker along with creaky veterans like Netscape 4.*x*.

When Internet Explorer 7 is released, it will join the ranks of other modern browsers such as Firefox, Opera, and Safari in rendering web pages using standards developed by the W3C. If examples used in this book do not render according to current standards in older browsers, I will explain how to work around that problem.

Required vs. presentational attributes

Isn't `src="photo.gif"` an attribute and value set? Earlier I said that XHTML had a DOCTYPE that didn't allow attributes and values, and now your alarm bells are all clanging. Yes, I did say that, you catch on fast. But `src` is *required*, even when using an XHTML Strict DOCTYPE. Without it, the browser doesn't know which image to display. Only attributes and values that have to do with how an element *looks* are not allowed in HTML and XHTML Strict.

XHTML: Specific Requirements

As mentioned previously, XHTML uses XML syntax rules. There are several specifics about writing XHTML syntax:

- Every element must be terminated. (In HTML you can sometimes get away with only using an opening tag, although it isn't a good practice.)

- Specific DOCTYPE declarations are required, which you will learn about in Chapter 3. (In HTML you can just have `<html>` with no DOCTYPE at the beginning of the document. Not a good practice, but possible.)

- All elements, attributes, and values must be in lowercase. (In HTML you can put them in caps if you want.)

- All values must be enclosed in quotation marks. Values can be quoted with single or double quotation marks, but you must be consistent about using the same type each time. The examples in this book will consistently be in double quotation marks ("). (In HTML you can leave out the quotes on the values. Again, not a good practice.)

- Every attribute must be given an explicit value. (In HTML, some attributes don't need explicit values.)

- Comments are not valid within a tag.

- Comments may not contain two hyphens in a row, other than at the beginning and end of the comment.

Leave a comment

An XHTML comment is not displayed on a web page. It's used to leave notes to yourself. The syntax is `<!--put the comment here-->`. Anything enclosed in the `<!-- -->` comment marks will be ignored by the browser. They're human readable. You, being human, can read comments any time you look at the code. Just be aware of how many hyphens you type in a row.

As you proceed through the book you'll see examples of comments used to provide information about what certain bits of code are about. The syntax for writing comments in CSS is different. You will learn about CSS comments later in the book.

Although the preceding rules are not required when writing HTML, they all work just fine in HTML. The only XHTML syntax rule that does not produce valid HTML is using the forward slash to terminate an empty element. Should you decide to use valid HTML instead of XHTML, you will

need to make only the minor adjustment to your coding habits to leave out the terminal forward slash in an empty element. I just mentioned it, but it bears repeating: there is nothing wrong with using valid HTML 4.01 to write your web pages. A choice had to be made for the examples in the book, and XHTML was selected. That choice does not diminish the value of value HTML 4.01. One of the reasons XHTML was selected for this book is that learning the rules of XHTML provides you with a set of good habits (for example, including a closing tag for elements like paragraphs and list items) that are good practices when writing HTML.

I threw the word *valid* around a couple of times in the last paragraph. You'll find out what that is in Chapter 3 when you learn about DOCTYPEs.

KEEP IT SIMPLE: SWITCHING FROM XHTML TO HTML

Q: I don't want to write XHTML, I want to write HTML this time. What do I have to remember?

A: Not much. Change the DOCTYPE and don't use a forward slash in empty elements.

SO MANY TAGS, SO LITTLE TIME

Throughout this book you'll learn the most important XHTML elements and attributes, but as you begin working on your own you'll find it valuable to have a complete reference to the language. You can find that reference in *HTML Complete* (Sybex, 2003), a compilation of useful information that also contains a command reference for CSS.

You can learn everything about every single element in HTML or XHTML from the W3C. For example, the specification for XHTML 1 is at www.w3.org/TR/xhtml1.

Getting Started with CSS Syntax

Cascading Style Sheets (CSS) are used to add presentational features to elements within your markup. CSS can set colors, fonts, backgrounds, borders, margins, and even the placement of elements on the page.

A style sheet is simply a text document containing one or more style rules. Style rules can be either placed directly in an XHTML document or linked to it as a completely separate file. In Chapter 2 you'll explore both these approaches, but most of the time CSS is linked to the XHTML page. In one document, you'll have your XHTML page, which you will learn to plan in a clean, logical structure of the headings, paragraphs, links, and images needed to present your ideas. In another file, you'll have your style sheet, which gives color, layout, emphasis, and pizzazz to your display. This way, you can change the way your web page looks simply by changing the styles in your style sheet or attaching a different style sheet, without changing the content at all.

The power to change a site's complete appearance by changing the style rules gives you great flexibility in its appearance. It also saves enormous amounts of time on maintenance and upkeep, since style rules are in a file that is apart from the content. Any number of web pages can be linked to a single style sheet, so it becomes merely a matter of minutes to make sweeping changes to the appearance of all those pages. When the style sheet is downloaded by a browser, it is saved in a special folder called *cache*. The next time the browser downloads a page using that style sheet, there is no wait for the user while it downloads because the browser already has it in cache. So every page that uses that style sheet will download very quickly, saving waiting time and bandwidth charges.

A further benefit of the separation of content from presentation is that your web pages are more accessible to all sorts of devices. That clean and logical structure in your XHTML that I mentioned makes your pages easier to index by search engines, too.

Styles and style sheets look very different from XHTML, and a different set of syntax rules is used for writing styles.

CSS TIPPING POINT

Visit www.csszengarden.com to see inspirational examples of the same content styled in many different ways using CSS. This website created the tipping point in the wide adoption of CSS for styling web pages by visibly proving that a well-structured page of HTML could have literally thousands of different presentations with CSS. Seeing is believing.

Selectors and Declarations

Style rules are written with *selectors* and *declarations*. Selectors identify which elements of an XHTML page the style will apply to. The most basic selector is the *element selector*, sometimes called the *type selector* because it selects a particular type of element. For example, the selector p selects all the paragraph elements on a page and is therefore an element or type selector.

For each selector, you write a set of declarations that govern how the selected element will display. Together the selector and declarations make up a style rule or, more simply, a style. Here is a style rule for the selector p:

```
p {
font-family: Arial, Helvetica, sans-serif;
font-size: small;
color: blue;
}
```

Let's examine that bit by bit. You already know that the p is the selector. Everything that comes between the two curly braces ({}) is the declaration block, which contains three different declarations in this example.

A declaration consists of a *property* followed by a colon, a space, and then the *value*. A semicolon follows the value. As you can see in this example, a property in CSS is similar to an attribute in XHTML. They both identify a characteristic of the element you are formatting. The first property declared in this example is the font family to be used for text in the paragraphs. Arial is the first choice, if the user's computer has it. If not, Helvetica will do, and if neither is available, the system's default sans-serif font will have to do.

FONT FAMILIES AND TYPEFACES

Font family is the slightly fussy typographical term for what we usually call a *typeface* or just a *font*. Strictly speaking, every variation in size and weight within a typeface is considered a separate font, and the whole set of these variations is considered the font family.

You'll learn more about fonts and font families in Chapter 4. Typography and fonts are the topic of many books, including *The Non-Designer's Design Book* by Robin Williams (Peachpit Press, 2003). The Web Style Guide at www.webstyleguide.com/type/index.html provides a good introduction to the topic.

It is considered good practice to include more than one font family in a declaration because not all computer systems come equipped with the same set of fonts. As in this example, the fonts are normally listed in the order of preference.

Generally, if no font family is specified, a browser will use Times as the default. Times is a serif font.

The second declaration in the preceding rule is `font-size: small`. You will learn more about the various options in font sizes in Chapters 4 and 5, but I'm sure you can guess that this declaration sets the font for all the `p` elements to a small size. The final declaration sets the color to `blue`. Color values are most often expressed with a code, but there are few colors that can be given by name. You'll learn more about the way to code color choices in Chapter 3. The CSS property `color` refers to elements in the foreground, in this example color of the paragraph text. To set a background color for a paragraph (or any other selector being styled), use the property `background-color`.

BROWSER DEFAULT FONT SIZES

Unless a user has changed the browser default settings, the default `font-size` setting in most browsers is `medium`. There are several keywords used for `font-size`: `xx-small, x-small, small, medium, large, x-large, xx-large, smaller, larger`. These keywords are considered relative measures, since font sizes specified by the designer will be sized in relation to whatever default font size is set. You'll find out more about this in Chapter 2.

The style rule example above has the effect of making every paragraph on that page appear in a font that is slightly smaller than normal, blue, and Arial.

In the examples in this book, each style declaration is written on a separate line, and the closing curly brace is on its own line as well. This makes the style easier for humans to read. However, style rules don't have to be typed in exactly that form for browsers to read them. For example, you could write the rule like this:

```
p {font-family: Arial, Helvetica, sans-serif; font-size: small; color: blue;}
```

If you do put more than one declaration on a line, be sure to leave a space after the semicolon separating one property declaration from the next.

Some style properties can be written in shorthand form. For example, `font` can be used as shorthand for all the font properties including `font-style`, `font-variant`, `font-weight`, `font-size`, `line-height`, and `font-family`. That allows you to combine the two previous declarations about fonts into one shorthand declaration like this:

```
p {
font: small Arial, Helvetica, sans-serif;
color: blue;
}
```

To expand the paragraph example a bit with more shorthand, here's another example:

```
p {
font: bold small/150% Arial, Helvetica, sans-serif;
color: blue;
}
```

This shorthand rule sets the `font-weight` to `bold`. The `small/150%` represents `font-size` and `line-height` in shorthand. The rest of the rule is already familiar to you.

SELECTORS GET SPECIFIC

Often you'll need to be more explicit in styling elements in the XHTML than the first example shows. CSS does allow for more specific selectors than the general element selector already described. Since the selector distinguishes what element in the document will be affected by the style rule, an element selector such as the p selector in the example above will affect all the p elements. There are times when you want particular (or what CSS terms *specific*) instances of paragraphs to follow different rules from those assigned to all p elements in general. Two of those types of selectors are the *ID selector* and the *class selector*. These two selectors allow you to write style rules for elements in a particular context. For example, instead of styling all the paragraphs on a page, you can style only the paragraphs of a certain class or ID.

SELECTORS IN THIS BOOK

You'll learn to use the following selectors in this book: adjacent-sibling selectors, attribute selectors, child selectors, element selectors, class selectors, ID selectors, descendant selectors, pseudo class selectors, pseudo-element selectors, the universal selector, and group selectors.

ID Selectors

IDs can only be used once per XHTML page. They are usually used to identify content that you style as a structural unit, such as a header, footer, content block, or menu. You will be working with this concept in almost every chapter of this book, but for now, you will simply see how ID selectors look in a style sheet.

ID selectors are preceded by this symbol: #. The correct term for this symbol is "octothorpe," but most people in the United States call it a pound sign or hash sign. In this book, we will use the term hash sign for this symbol. An id rule in a style sheet looks like this:

```
#footer {
font-size: x-small;
}
```

This rule makes everything in the section (or division) of the page identified as footer extra small. Notice that there is no space between the hash sign and the id name. Other page elements in addition to the div (or division) can be identified with an id. You'll see examples of various ways to use ID selectors throughout the book.

But a footer division may contain other elements, such as an address, that you don't want to style as x-small. Or suppose you want only the address in the footer to be extra small?

You could accomplish this using a *descendant selector* that applies to only an address in the page division identified as footer:

```
#footer address {
font-size: small;
}
```

Notice the space between the #footer and the address. This font-size value won't apply to other elements on the page, only to address elements placed within the context of the footer division. This use of the ID selector followed by the element selector is very specific to only particular elements in particular parts of the page.

The selector #footer address is a contextual (or descendant) selector. You can build contextual selectors into your XHTML with named IDs that allow for finely drawn CSS selectors, but contextual

selectors are not just used with IDs. In the upcoming chapters, you will use descendant selectors to create many CSS styles. Chapter 2 explains more about the hierarchical relationship of HTML elements that creates a descendant element.

You create the names for the `id` selectors yourself. They don't have to relate to any XHTML tag. It is good practice to create a simple and meaningful name that reflects the structural purpose of the content of the section identified with an `id`. The `id` selector `#footer` is a good example of a name that reflects some meaningful purpose on the page. If you worked on a style sheet and went back to it after several months, the id selector `#footer` would still make sense to you as you reviewed and changed the style sheet.

Class Selectors

Class selectors can be used as many times as you want per XHTML page. Class selectors are preceded by a period (`.`). As with `id` selectors, you create the `class` name yourself. If you want a style that will highlight certain terms on your page, you can create a `class` and name it `term`.

```
.term {
background-color: silver;
}
```

This style rule would put a silver background behind any words or phrases that were identified as being in the class `term`. Notice that there is no space between the preceding period and the name of the class. In Chapter 2, you'll learn how to apply both class and id attributes to your XHTML elements.

THERE'S AN ART TO NAMING CLASSES AND IDS

One of the reasons CSS is popular is because a page's whole look can change almost instantly. It is good practice to choose class names that express purpose rather than some momentary choice such as a color, which might be changed later. So a class named `.term` is a better choice here than a class named `.silver` because the name `term` will continue to make sense no matter what color is used.

Supposed you were creating divs and IDs for a layout with navigation in a column on the left side. You would need to think up an ID name for the navigation column. An ID named `#nav` would be a better choice than `#leftcol` because the navigation will always be navigation, but the left column might be positioned somewhere else in a later redesign.

So use functional or semantic terms when dreaming up class and ID names. It will help you keep your CSS selectors meaningful over time.

You can use complex combinations of selectors, IDs, and classes to style specific sections of your pages. The following selector would apply only to paragraphs of a class called `term` in a division of the page called `footer`.

```
#footer p.term {
background-color: gray;
}
```

Notice that when writing a style declaration for an XHTML element that is assigned to a particular class, such as the `p` element, there is no space between the element and the period and class name: `p.term`.

GET YOUR SELECTORS TRANSLATED INTO ENGLISH

It's often useful to study and adapt the code behind websites that use features you want to try, but CSS selectors can look dauntingly complex. SelectORacle is a free tool at `gallery.theopalgroup.com/selectoracle/` that will translate complex CSS selectors into plain English to help you understand exactly what elements CSS rules are selecting. For example, SelectORacle helpfully explains that the selector `#content .feature a` selects any a element that is a descendant of any element with a class attribute that contains the word `feature` that is a descendant of any element with an `id` attribute that equals `content`.

GROUPING SELECTORS

You may want to use the same style for several elements on your page. Perhaps you want all the paragraphs, lists, and block quotes on the page to have the same font size. To achieve this effect, list the selectors, separated by commas, and give the `font-size` declaration:

```
p, li, blockquote {
font-size: medium;
}
```

This rule makes the text in any paragraph, list, or block quote have the font size `medium`. Notice that there is a space between each item in the comma-separated list of selectors.

DESCENDANT SELECTORS

In the previous example, the comma sets up a rule for every element in the comma-separated list. Here is a similar rule with no comma:

```
blockquote em {
font-size: medium;
}
```

Without the comma, it looks a whole lot like the preceding comma-free `#footer p` rule, doesn't it? The selector `blockquote em`, with no comma, styles only an `em` (emphasized) element that's part of a `blockquote`, not every `em` element.

The comma (or the lack of a comma) is an important distinction between grouped selectors and descendant selectors. Grouped selectors use commas. Descendant selectors do not.

SPACES, COMMAS, SEMI-COLONS, BRACKETS: IT ALL MATTERS

XHTML and CSS are highly detail-oriented pursuits. You have to pay attention to the details such as whether you see a comma, a colon, a semicolon, a space, quotation marks, a bracket or brace. Type carefully. Do this and you will have greater success with both the XHTML and the CSS examples in the book. If you do an exercise and it doesn't seem to work as I say it should, check carefully for typos. The syntax is exacting, and a missing semicolon or quotation mark can cause a whole page to render as gibberish. And I don't mean the kind of gibberish you get when you use one of those "lorem ipsum" generators to fill a page with a lot of meaningless text so you can see how a layout works.

Quotation Marks

The last difference between the syntax rules for writing XHTML and writing CSS that you'll look at before moving on to Chapter 2 involves quotation marks.

You recall that in XHTML, all attribute values in a tag must be enclosed in quotation marks. In CSS style declarations, however, property values do not appear in quotes. Most of the time, you don't see quotation marks in style sheets except when listing a font whose name contains two or three words.

For example, you looked at this style rule earlier, in which no quotes were used:

```
p {
font-family: Arial, Helvetica, sans-serif;
font-size: small;
color: blue;
}
```

The fonts listed here are one-word font names. Sans-serif is hyphenated and doesn't have a space, so it is considered one word. However, if you want to list a font name such as Times New Roman, which is more than one word and includes spaces, you wrap the name in quotation marks, like this:

```
p {
font-family: "Times New Roman", Times, Georgia, serif;
}
```

Notice that the comma separating Times New Roman from Times is *after* the ending quotation mark. Also note that although specific font-family names such as Times are capitalized, generic font names such as serif are not.

You may see quotation marks in the URL to an image in a style sheet. Quotation marks are optional, not required, in URLs. Here's an example, written three different ways, all correct, with various examples of quotation mark use (or no quotation marks).

```
p {
background-image: url(bg.gif);
}
p {
background-image: url('bg.gif');
}
p {
background-image: url("bg.gif");
}
```

Although the W3C considers single quotation marks legal, they aren't supported by Mac versions of Internet Explorer, so it's wise to stick with either no quotation marks or double quotation marks in the style sheet.

In Chapter 2, you will build XHTML documents and CSS style sheets. While you're doing that, you'll learn where to write styles and how to link to styles. Chapter 2 will explain the meaning of the *cascade* in Cascading Style Sheets.

Real-World Scenario

A strong thread emphasized throughout this book is that the use of the standard specifications recommended by the W3C by both web professionals and browser manufacturers makes writing XHTML and CSS easier, faster, and more universal.

There is a grassroots group working hard for the implementation of web standards called The Web Standards Project (WaSP) at `www.webstandards.org` (Figure 1.5).

The Web Standards Project site provides opportunities for you to take action in favor of web standards and offers information to help you learn to use those standards.

Exemplary attention to standards, valid code using a strict DOCTYPE, progressive examples of CSS and other web technologies are to be expected from this website. It delivers. Use the browser's View Source option to look at this site's clearly structured and valid XHTML.

It will be helpful when viewing the real-world examples in this book, and in testing your own work as you progress through the book, if you download the Mozilla Firefox browser at `www.mozilla.com/firefox`. After you have installed the Firefox browser, download and install the Firefox Web Developer Add-on, from `https://addons.mozilla.org/firefox/60`. Using the Web Developer add-on toolbar gives you easy-to-reach tools to view HTML and CSS code for any web page.

Be aware that any site given as a "Real-World Scenario"—in fact, any site referred to in this book—is protected by copyright law from being copied. Copyright law protects both the images and the text in a website. I encourage you to look and learn but not to take. There may be a few exceptions—for example, some CSS layout sites say in very clear terms that you have permission to take the material for your own use—but in general, it is best to assume that you do not have permission to "borrow" material from any website.

FIGURE 1.5
The Web Standards Project home page contains news in a section called Recent Buzz—listen to that WaSP buzz!
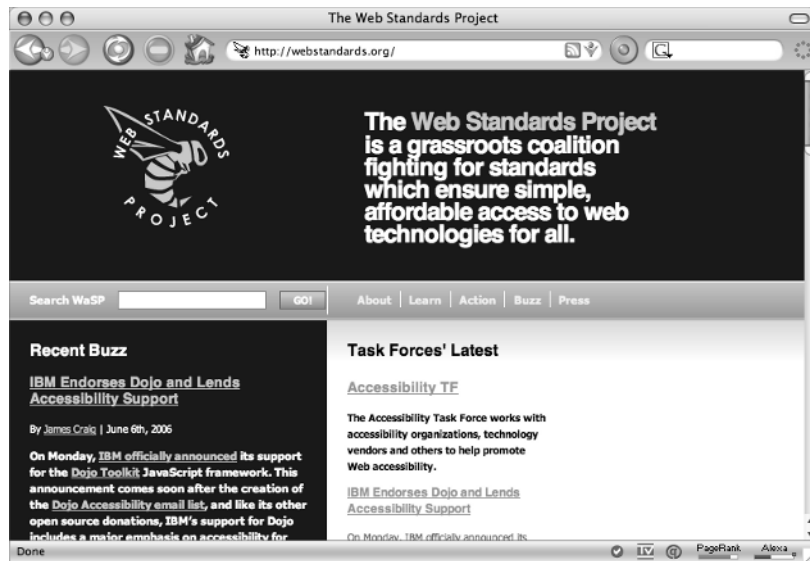
The Bottom Line

Identify what constitutes a website   Websites are built with many interconnected technologies and applications. You should be familiar with how XHTML and CSS fit into that puzzle.

> **Master It**   Describe the purpose of XHTML and CSS in building a web page.

Identify what XHTML and HTML are   XHTML and HTML are markup languages. XHTML is an extension of HTML based on XML, itself a markup language.

> **Master It**   What do the acronyms HTML and XHTML mean?

Explain similarities and differences in XHTML and HTML   Since XHTML is based on HTML, there are many similarities between the two. The important differences arise from the distinction that XHTML uses XML syntax, while HTML is not required to follow XML syntax rules.

> **Master It**   List XML syntax rules that XHTML must follow. Make note of any that do not apply to HTML.

Describe what CSS is   CSS stands for Cascading Style Sheets, a specification that sets out properties and values that may be applied to the presentation of HTML or XHTML elements.

> **Master It**   Explain how CSS rules are applied to a web page.

Write XHTML Syntax   Meaningful elements in the content of a web page are marked up with XHTML tags, which may or may not have attributes and values giving more information about the element.

> **Master It**   To demonstrate XHTML syntax, write two complete XHTML elements demonstrating the difference between an element with attributes and values and one without.

Write CSS Syntax   Style rules are written with *selectors* and *declarations*. The most basic selector is the element selector. Other types of selectors include adjacent-sibling selectors, attribute selectors, child selectors, class selectors, ID selectors, descendant selectors, pseudo class selectors, pseudo-element selectors, the universal selector, and group selectors.

For each selector, the declarations of properties and values for that selector govern how the element will display. Together the selector and declarations make up a style rule or, more simply, a style.

> **Master It**   Write a style rule for the selector h1 that sets the `font-family` to `Arial, Helvetica, sans-serif` and sets the `font-size` to `1.5em`. Write the rule in full and then write it again in shorthand.