1

Structuring Documents for the Web

WHAT YOU WILL LEARN IN THIS CHAPTER

- Creating several example web pages in HTML
- Seeing how a web page describes its structure to a web browser
- Discovering the meaning of some key terms used by web designers, such as elements, attributes, tags, and markup

WROX.COM CODE DOWNLOADS FOR THIS CHAPTER

The wrox.com code downloads for this chapter are found at `www.wrox.com/remtitle.cgi?isbn=9781118340189` on the Download Code tab. The code is in the Chapter 1 download and individually named according to the names throughout the chapter.

In this chapter, you learn the key concept to create any web page: how to give it *structure*. You need to add structure to a document so that web browsers can present the page to people who visit your site in a way they can understand. For example, imagine a news article that contains a headline (or title) and several paragraphs of text; if you want to put this article on the web, you would need to add structure to the words in the document so that the browser knows which words are the headline, and where each paragraph starts and ends. To give a document structure, you need to learn how to create web pages using HTML.

A WEB OF STRUCTURED DOCUMENTS

Every day, you come across all kinds of printed documents—newspapers, train timetables, and insurance forms. You can think of the web as being a sea of documents that all link together and bear a strong similarity to the printed documents that you meet in everyday life.

Take the example of a newspaper. A newspaper consists of several stories or articles (and probably a fair smattering of advertisements, too). Each story has a headline and then some paragraphs, perhaps a subheading, and then some more paragraphs; it may also include a picture or two.

The structure of articles on news websites is similar to the structure of articles in newspapers. Each article consists of headings, paragraphs of text, and some pictures. (Sometimes the pictures might be replaced by a video.) The parallel is quite clear; the only difference is that in a newspaper you may have several stories on a single page, whereas on the web each story tends to get its own page. The news websites also often use homepages that display the headline and a brief summary of the stories.

Consider another example: You're catching a train to see a friend, so you check the schedule or timetable to see what time the train leaves. The main part of the schedule is a *table* telling you what times trains arrive and when they depart from different stations. You can probably think of several types of documents that use tables. From the listings in the financial supplement of your paper to the TV schedule, you come across tables of information every day—and often when this information is put on the web, these tables are re-created.

Another common type of printed document is a *form*. For example, think about a common form from an insurance company. Such a form contains fields to write your name, address, and the amount of coverage, along with check boxes to indicate the number of rooms in the house and what type of lock is on the front door. There are lots of forms on the web, from simple search boxes that ask what you are looking for to the registration forms you are required to fill out before you can place an online order for books or CDs.

As you can see, there are many parallels between the structure of printed documents you come across every day and pages you see on the web. When you are writing web pages, it is the HTML code you start learning in this chapter that tells the web browser how the information you want to display is structured—what text to put in a heading, paragraph, or table, and so on so that the browser can present it properly to the user.

INTRODUCING HTML5

Even if you have never seen any HyperText Markup Language (HTML) code, you may know that it is used to create web pages. There have been five versions of HTML since the web began, and the development of the language is overseen by an organization called the World Wide Web Consortium (W3C).

This book focuses on the latest version of the language, popularly referred to as HTML5. There are two other versions you might encounter. These are HTML 4.01, the last major version of the language from December 1999, and a stricter version from 2000 called Extensible HyperText Markup Language (XHTML). XHTML is still popular in some applications, so important differences between it and HTML5 will be called out in the text.

> ***NOTE*** *Generally, you see just the term HTML used in the rest of this book. The one exception is when there is a feature or convention related to a single version.*

As its name suggests, HTML is a *markup language*, which may sound complicated until you realize that you come across markup every day. When creating a document in a word processor, you can add styles to the text to explain the document's structure. For example, you can distinguish headings from the main body of the text using a heading style (usually with a larger font). You can use the Return (or Enter) key to start a new paragraph. You can insert tables into your document to hold data or create bulleted lists for a series of related points, and so on. Although this does affect the presentation of the document, the key purpose of this kind of markup is to provide a structure that makes the document easier to understand.

When marking up documents for the web, you perform a similar process, except you do it by adding things called *tags* to the text. With HTML, the key thing to remember is that you must add the tags to indicate the structure of the document (not how you want it to be presented); for example, which part of the document is a heading, which parts are paragraphs, what belongs in a table, and so on. Browsers such as Internet Explorer, Firefox, and Google Chrome all use this markup to help present the text in a familiar fashion, similar to that of a word processor—main headings are bigger than the text in paragraphs, there is space above and below each paragraph, and lists of bullet points have a circle in front of them.

> ***NOTE*** *Although earlier versions of HTML enabled you to control the presentation of a document—such as which typefaces and colors a document should use—HTML markup is not supposed to be used to style the document; that is the job of Cascading Style Sheets (CSS), which you meet in Chapter 7, "Cascading Style Sheets."*

Now have a look at a simple web page (`ch01_eg01.html`). You don't need any special programs to write web pages; you can simply use a text editor such as Notepad on Windows or TextEdit on a Mac and save your files with the .html or .htm file extension.

```
<html>
  <head>
    <title>Popular Websites: Google</title>
  </head>
  <body>
    <h1>About Google</h1>
    <p>Google is best known for its search engine, although
       Google now offers a number of other services.</p>
    <p>Google's mission is to organize the world's
       information and make it universally accessible and
       useful.</p>
    <p>Its founders Larry Page and Sergey Brin started
       Google at Stanford University.</p>
  </body>
</html>
```

This may look a bit confusing at first, but it will all make sense soon. As you can see, there are several sets of angle brackets with words or letters between them, such as `<html>`, `<head>`, `</title>`, and `</body>`. These angle brackets and the words inside them are known as *tags*, and these are the markup previously mentioned. Figure 1-1 illustrates what this page would look like in a web browser.

As you can see, this document contains the heading "About Google" and a paragraph of text to introduce the company. Note also that it says "Popular Websites: Google" in the top-left corner of the browser window; this is known as the *title* of the page (to the right it says Mozilla Firefox, which is the browser this page was opened in).

To understand the markup in this first example, you need to look at what is written between the angle brackets and compare that with what you see in the figure, which is what you do next.

FIGURE 1-1

Tags and Elements

If you look at the first and last lines of the code for the previous example, you see pairs of angle brackets containing the letters "html". Starting on the first line, the first angled bracket looks like a less-than sign (<); then there are the letters "html," followed by a second angled bracket, which looks like a greater-than sign (>). The two brackets and all the characters between them are known as a *tag*.

In this example, there are lots of tags, and they are all in pairs; there are *opening tags* and *closing tags*. The closing tag is always slightly different from the opening tag in that it has a forward slash (/) after the first angled bracket: `</html>`.

A pair of tags and the content these include are known as an *element*. In Figure 1-2, you see the heading for the page of the previous example.

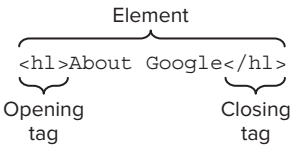

FIGURE 1-2

The opening tag says, "This is the beginning of a heading" and the closing tag says, "This is the end of a heading." Like most tags in HTML, the text inside the angled brackets explains the purpose of the tag—here `h1` indicates that it is a level 1 heading (or top-level heading). As you will see shortly, there are also tags for subheadings (`<h2>`, `<h3>`, `<h4>`, `<h5>`, and `<h6>`). If you don't put tags around the words "About Google," it is just another bit of text; it would not be clear that these words formed the heading.

Now look at the three paragraphs of text about the company; each one is placed between an opening `<p>` tag and a closing `</p>` tag. And you guessed it, the `p` stands for paragraph.

> ***WARNING*** *You must understand the basic distinction between tags and elements: A tag usually consists of left-angle and right-angle brackets and letters and numbers between those brackets, whereas elements are the opening and closing tags plus anything between the two tags.*

> ***WARNING*** *To be precise, there are also tags that consist of just one left-angle bracket and one right-angle bracket, with no content and no closing tag. These are also elements.*

As you can see, the tags throughout this example actually describe what you will find between them, creating the structure of the document. The text between the `<h1>` and `</h1>` tags is a heading, and the text between the opening `<p>` and closing `</p>` tags makes up paragraphs. Indeed, the whole document is contained between opening `<html>` and closing `</html>` tags.

You often find that terms from a family tree are used to describe the relationships between elements. For example, an element that contains another element is known as the *parent*, whereas the element that's between the parent element's opening and closing tags is called a *child* of that element. So, the `<title>` element is a child of the `<head>` element, the `<head>` element is the parent of the `<title>` element, and so on. Furthermore, the `<title>` element can be thought of as a grandchild of the `<html>` element.

Additionally, if two elements are children of the same parent, they are referred to as *siblings*.

It is worth noting that the tags in this example are all in lowercase characters; you sometimes see web pages written in HTML where tags are uppercase (or a mix of uppercase and lowercase letters). When XHTML was introduced, with its stricter rules, it stated that all tags were written in lowercase. Technically, HTML5 loosens these restrictions to enable mixed case. In practice you generally see lowercase even in HTML5 documents.

> ***NOTE*** *Even though HTML5 enables mixed case tags, lowercase should be used for consistency with XHTML documents, which require lowercase tags.*

Separating Heads from Bodies

Whenever you write a web page in HTML, the whole of the page is contained between the opening `<html>` and closing `</html>` tags, just as it was in the previous example. Inside the `<html>` element, there are two main parts to the page:

- **The `<head>` element:** Often referred to as the head of the page, this contains information *about* the page. (This is not the main content of the page.) For example, it might contain a title and a description of the page or instructions on where a browser can find CSS rules that explain how the document should look. It consists of the opening `<head>` tag, the closing `</head>` tag, and everything in between.
- **The `<body>` element:** Often referred to as the body of the page, this contains the information you actually see in the main browser window. It consists of the opening `<body>` tag, the closing `</body>` tag, and everything in between.

Together, the <html>, <head>, and <body> elements make up the skeleton of an HTML document—they are the foundation upon which every web page is built.

Inside the <head> element of the first example page, you see a <title> element:

```
<head>
  <title>Popular Websites: Google</title>
</head>
```

Between the opening <title> tag and the closing </title> tag are the words "Popular Websites: Google," or the title of this web page. Figure 1-1 shows the words at the top of the browser window, which is where browsers such as Internet Explorer, Firefox, and Chrome display the title of a document. It is also the name they use when you save a page in your Favorites List, and it helps search engines understand what your page is about. The <title> element is mandatory for all web pages.

The real content of your page is held in the <body> element, which is what you want users to read, and this is shown in the main browser window.

> ***WARNING*** *The <head> element contains information about the document, which is not displayed within the main page. The <body> element holds the actual content of the page viewed in your browser.*

You may have noticed that the tags in this example appear in a symmetrical order. If you want to have one element inside another, both the element's opening and closing tags must be inside the containing element. For example, the following is allowed:

```
<p> This paragraph contains some <em>emphasized text.</em></p>
```

whereas the following is wrong because the closing tag is not inside the paragraph element:

```
<p> This paragraph contains some <em>emphasized text. </p></em>
```

In other words, if an element is to contain another element, it must wholly contain that element. This is referred to as *nesting* your elements correctly.

Attributes Tell You about Elements

Attributes in HTML are much like the attributes you experience every day. They are the qualities that describe a person or thing, such as a *tall* man or a *brown* dog. Similarly, HTML elements can be described in ways that web browsers can understand. This section looks at attributes, starting with the most important one that beats at the heart of the web.

What differentiates web documents from standard documents are the *links* (or *hyperlinks*) that take you from one web page to another. Look at a link by adding one to the example you just looked at. Links are created using an <a> element. (The a stands for anchor.)

You can add a link from this page to Google in a new paragraph at the end of the document. There is just one new line in this example (`ch01_eg02.html`) and that line is highlighted:

```
<html>
  <head>
    <title>Popular Websites: Google</title>
  </head>
  <body>
    <h1>About Google</h1>
    <p>Google is best known for its search engine, although Google now offers a
       number of other services.</p>
    <p>Google's mission is to organize the world's information and make it
       universally accessible and useful.</p>
    <p>Its founders Larry Page and Sergey Brin started Google at Stanford
University.</p>
    <p><a href="http://www.Google.com/">Click here to visit Google's Web
    site.</a></p>
  </body>
</html>
```

Inside this new paragraph is the `<a>` element that creates the link. Between the opening `<a>` tag and the closing `</a>` tag is the text that you can click, which says, "Click here to visit Google's Web site." Figure 1-3 shows you what this page looks like in a browser.

FIGURE 1-3

If you look closely at the opening tag of the link, it carries something called an *attribute*. In this case, it's the `href` attribute; this is followed by an equal sign and then a pair of quotation marks, which contain the URL for Google's website. In this case, the `href` attribute tells you where the link should take you. You look at links in greater detail in the Chapter 3, "Links and Navigation," but for the moment this illustrates the purpose of attributes.

- Attributes are used to say something about the element that carries them, and they always appear on the opening tag of the element that carries them. Almost all attributes consist of two parts: a name and a value. The *name* is the property of the element that you want to set. In this example, the `<a>` element carries an attribute whose name is `href`, which you can use to indicate where the link should take you.
- The *value* is what you want the value of the property to be. In this example, the value was the URL of the site that the link should take you to, so the value of the `href` attribute is `http://www.google.com`.

The value of the attribute should always be put in double quotation marks and separated from the name with the equal sign.

There are several attributes in HTML5 that do not consist of a name/value pair but consist of just a name. These are called boolean attributes and you will learn more about those in the section "Attribute Groups."

Another common attribute on anchors is the `title` attribute, which gives a plain language description of the target of the link. You could add one to the example to inform people that Google is a popular search engine.

```
<a href="http://www.Google.com"
   title="Google.com is the world's most popular search engine">
```

This illustrates that elements can carry several attributes; although, an element should never have two attributes of the same name.

Learning from Others by Viewing Their Source Code

When HTML first came out, a lot of people learned how to create pages by using a handy feature that you can find in most common browsers: the ability to look at the source code that made the page.

If you go to the View menu in your browser and then look for an option that says View Source or Page Source, you should see the code that created the page.

If you want to see how the author of a page achieved something on a page, this can be a handy technique. Figure 1-4 shows how to look at the source of the author's homepage. (The window on the right contains the source for the page.)

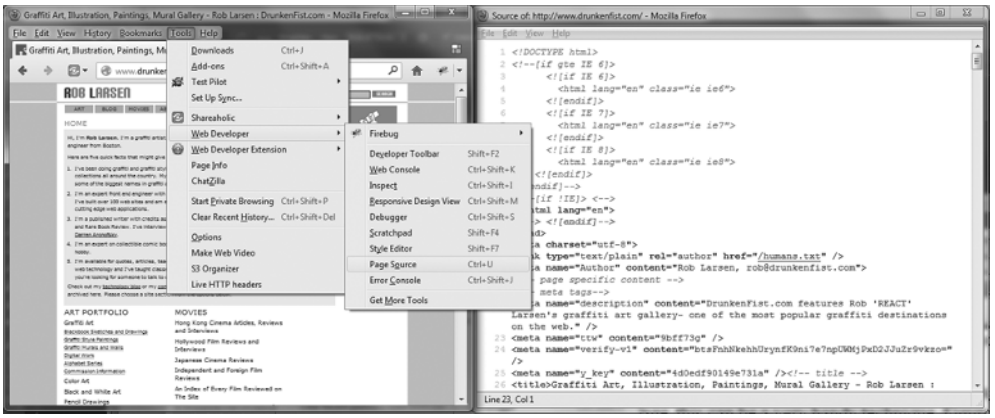

FIGURE 1-4

Elements for Marking Up Text

You now know that an HTML page (also sometimes referred to as an HTML document) consists of elements that describe how its content is structured. Each element describes what you will find between its opening and closing tags. The opening tags can also carry attributes that tell you more about that particular element.

Equipped with this knowledge, you can find that much of learning HTML is a matter of learning what elements you can use, what each of these elements does, and what attributes each can carry.

ATTRIBUTE GROUPS

As you have seen, attributes live on the opening tag of an element and provide extra information about the element that carries them. Many attributes consist of a *name* and a *value*; the name reflects a property of the element the attribute describes, and the value is a value for that property. For example, the `lang` attribute describes the language used within that element; a value such as `EN-US` would indicate that the language used inside the element is U.S. English.

Some attributes consist of only a name, such as `required` or `checked`. These are called boolean attributes. To say something is a boolean (which you learn more about in Chapter 10, "Learning JavaScript") is to indicate that it can be in one of two states: true or false. For HTML attributes the presence of one of the boolean attributes in a tag indicates that the value is true. So, the following are equivalent:

```
<input type="text" required >
<input type="text" required="true">
```

Many of the elements in HTML can carry some or all the attributes you will meet in this section. At first some of them may sound a little abstract; although, they will make more sense as you see them used throughout the book. So don't worry if they do not make much sense at first.

In this section, you look at three groups of attributes common to many HTML elements:

- **Core attributes:** Including the `class`, `id`, `style`, and `title` attributes
- **Internationalization attributes:** For example, the `dir` and `lang` attributes
- **Accessibility attributes:** For example, `accesskey` and `tabindex`

> ***WARNING*** *Together, the core attributes and the internationalization attributes are known as universal attributes.*

Core Attributes

The four core attributes that you can use on the majority of HTML elements (although not all) are:

```
id title class style
```

Throughout the rest of the book, these attributes are revisited when they have special meaning for an element that differs from the description given here; otherwise their use can generally be described as you see in the subsections that follow.

The id Attribute

You can use the `id` attribute to uniquely identify any element within a page. You might want to uniquely identify an element so that you can link to that specific part in the document or to specify that a CSS style or piece of JavaScript should apply to the content of just that one element within the document.

The syntax for the `id` attribute is as follows (where *string* is your chosen value for the attribute):

```
id="string"
```

For example, you can use the `id` attribute to distinguish between two paragraph elements, like so:

```
<p id="accounts">This paragraph explains the role of the accounts department.</p>
<p id="sales">This paragraph explains the role of the sales department.</p>
```

Following are some special rules for the value of the `id` attribute:

- Must begin with a letter (A–Z or a–z) and can then be followed by any number of letters, digits (0–9), hyphens, underscores, colons, and periods. (You may not start the value with a digit, hyphen, underscore, colon, or period.)
- Must remain unique within that document; no two `id` attributes may have the same value within one HTML page. This case should be handled by the `class` attribute.

The class Attribute

You can use the `class` attribute to specify that an element belongs to a *class* of elements. For example, you might have a document that contains many paragraphs, and a few of those paragraphs might contain a summary of key points, in which case you could add a `class` attribute whose value is `summary` to the relevant `<p>` elements to differentiate those paragraphs from the rest in the document.

```
<p class="summary">Summary goes here</p>
```

It is commonly used with CSS, so you learn more about the use of the `class` attribute in Chapter 7, which introduces CSS. The syntax of the `class` attribute is as follows:

```
class="className"
```

The value of the attribute may also be a space-separated list of class names, for example:

```
class="className1 className2 className3"
```

The title Attribute

The `title` attribute gives a suggested title for the element. The syntax for the `title` attribute is as follows:

```
title="string"
```

The behavior of this attribute depends upon the element that carries it; although, it is often displayed as a tooltip or while the element loads. Not every element that *can* carry a `title` attribute actually needs one, so when you meet an element that particularly benefits from use of this attribute, you will see the behavior it has when used with that element.

The style Attribute

The `style` attribute enables you to specify CSS rules within the element. You meet CSS in Chapter 7, but for now, here is an example of how it might be used:

```
<p style="font-family:arial; color:#FF0000;">Some text.</p>
```

As a general rule, however, it is best to avoid the use of this attribute. If you want to use CSS rules to govern how an element appears, it is better to use a separate style sheet instead. The only place where this attribute is still commonly used is when it is set with JavaScript. You learn more about that in Chapter 11, "Working with jQuery," when you're introduced to jQuery's powerful tools for manipulating HTML elements.

Internationalization

The web is a worldwide phenomenon. Because of this, there are mechanisms built into the tools that drive the web that allow authors to create documents in different languages. This process is called *internationalization*.

Two common internationalization attributes help users write pages for different languages and character sets:

```
dir lang
```

You look at each next, but it is worth noting that even in current browsers, support for these attributes is still patchy. Therefore where possible you should specify a character set that creates text in the direction you require.

The website of a helpful W3C document that describes internationalization issues in greater detail is found at `www.w3.org/TR/i18n-html-tech-char/`; although, you briefly look at each of these attributes next.

> ***NOTE*** *The internationalization attributes are sometimes referred to as the i18n attributes, an odd name that comes from the draft-ietf-html-i18n specification in which they were first defined.*

The dir Attribute

The `dir` attribute enables you to indicate to the browser the direction in which the text should flow: left to right or right to left. When you want to indicate the directionality of a whole document (or

the majority of the document), use it with the `<html>` element rather than the `<body>` element for two reasons: Its use on the `<html>` element has better support in browsers, and it can apply to the header elements as well as those in the body. You can also use the `dir` attribute on elements within the body of the document if you want to change the direction of a small portion of the document.

The `dir` attribute can take one of two values, as you can see in Table 1-1.

TABLE 1-1: dir Attribute Values

VALUE	MEANING
`ltr`	Left to right (the default value)
`rtl`	Right to left (for languages such as Hebrew or Arabic that are read right to left)

The lang Attribute

The `lang` attribute enables you to indicate the main language used in a document.

The `lang` attribute was designed to offer language-specific display to users; although, it has little effect in the main browsers. The benefits of using the `lang` attribute are for search engines (which can tell the user which language the document is authored in), screen readers (which might need to pronounce different languages in different ways), and applications (which can alert users when either they do not support that language or it is a different language than their default language). When used with the `<html>` element, the attribute applies to the whole document; although, you can use it on other elements, in which case it just applies to the content of those elements.

The values of the `lang` attribute are ISO-639-1 standard two-character language codes. If you want to specify a dialect of the language, you can follow the language code with a dash and a subcode name. Table 1-2 offers some examples.

TABLE 1-2: lang Attribute Values

VALUE	MEANING
`ar`	Arabic
`en`	English
`en-us`	U.S. English
`zh`	Chinese

You can find a list of language codes for most of the main languages in use today in Appendix G, “Language Codes.”

CORE ELEMENTS

Now take a closer look at the four main elements that form the basic structure of every document: `<html>`, `<head>`, `<title>`, and `<body>`. These four elements should appear in every HTML document that you write, and you will see them referred to throughout this book as the *skeleton* of the document.

About DOCTYPEs

Although the four main elements describe the skeleton of a document, one final piece qualifies the document as a whole. The `DOCTYPE` (for DOCument TYPE) tells the browser what rules to follow when showing the document to the user. These rules are called *modes*. This book focuses on the HTML5 `DOCTYPE` that puts the browser into *strict mode*. You can think of strict mode as the browser acknowledging the author wanting to play by the rules. The other common mode, *quirks mode*, tells the browser you will use some funky rules, which have their origins in the late 1990s. You don't want to have anything to do with quirks mode.

So what does the HTML5 `DOCTYPE` look like?

```
<!doctype html>
```

Start all your documents with that `DOCTYPE`, and your pages will always render in the correct mode.

The basic skeleton of an HTML5 page therefore looks like this:

```
<!doctype html>
<html>
  <head>
    <title>The Skeleton of an HTML5 Document</title>
  </head>
  <body>
  </body>
</html>
```

The <html> Element

The `<html>` element is the containing element for the whole HTML document. After the `DOCTYPE` declaration, each HTML document should have an opening `<html>` tag, and each document should end with a closing `</html>` tag.

The `<html>` element can also carry the following attributes, which you learned about in the "Attribute Groups" section:

```
id dir lang
```

The <head> Element

The `<head>` element is just a container for all other header elements. It is the first thing to appear after the opening `<html>` tag.

Each <head> element should contain a <title> element indicating the title of the document; although, it may also contain any combination of the following elements, in any order:

- <base>, which you will meet in Chapter 3, "Links and Navigation"
- <link> to link to an external file, such as a style sheet, which you see in Chapter 7
- <style> to include CSS rules inside the document, covered in Chapter 7
- <script> for including script in the document, which you see in more detail in Chapter 10
- <meta>, which includes information about the document such as a description or the name of the author

The opening <head> tag can carry the following attributes:

```
id dir lang
```

> ***NOTE*** *One meta tag you should be aware of is* `<meta charset=utf-8>`. *This tag and attribute combination tells the browser which character set to use. Character sets are collections of characters used to render written language. For the most part, using utf-8 is going to be the best bet on the web. Utf-8 contains every character in the Unicode character set (over one million characters), which means it can render text in everything from English to Chinese to Russian.*
>
> *For an in-depth discussion of character encoding, see Joel Sposky's article, "The Absolute Minimum Every Software Developer Absolutely, Positively Must Know about Unicode and Character Sets (No Excuses!)," found at* `www.joelonsoftware.com/articles/Unicode.html`.

The <title> Element

You should specify a title for every page that you write using the <title> element (which, as you saw earlier in the chapter, is a child of the <head> element). It is presented and used in several ways:

- At the top of a browser window (as you saw in the first example and Figure 1-1)
- As the default name for a bookmark in browsers such as IE, Firefox, and Chrome
- By search engines that use its content to help index pages

Therefore, you must use a title that describes the content of your site. For example, the homepage of this book should not just say "Homepage"; rather it should describe what your site is about. Rather than just saying "Wrox Homepage," it is more helpful to write:

```
<title>Wrox: Programming Books, Learn HTML, CSS, ASP.Net, PHP</title>
```

The test for a good title is whether visitors can tell what they will find on that page just by reading the title, without looking at the actual content of the page, and whether it uses words that people would use if they were going to search for this kind of information.

The `<title>` element should contain only the text for the title; it may not contain any other elements. The `<title>` element can carry the following attributes:

```
id dir lang
```

Links and Style Sheets

Although you'll learn more about CSS and JavaScript later in the book, they are going to be a common element on every page you look at and build, so it's important that we quickly cover how to include them in your web pages.

Adding a style sheet relies on the `<link>` element. The `<link>` element also uses the `href` attribute, which you learned about already, to point to a resource on the web. In this case, instead of pointing to a new page or website to visit when a link is clicked, it points to the location of a file containing style information for the current page. The `rel` (for relation) attribute indicates that the linked document is a style sheet and should be handled accordingly.

```
<link rel="stylesheet" href="css/main.css">
```

Adding a script to the page is even easier. You add a `<script>` element to the page and add a `src` attribute pointing to the location of the JavaScript file you want to use.

```
<script src="js/main.js"></script>
```

> ***WARNING*** *Always include the closing* `</script>` *tag when inserting a script element, even if, as in this case, there's no content between the opening and closing tags. If you don't, strange things can happen.*

You need to start using one piece of JavaScript. Now that you know how to add a script to the page, you're ready to learn about the HTML5 Shiv and Modernizr.

Ensuring Backward Compatibility for HTML5 Tags

There's one final element you see in many HTML5 documents. Because of a wrinkle in the way that Internet Explorer deals with unknown HTML elements, you need to include a small piece of JavaScript in the head of your document. If you don't, things look wrong when you view your pages in Internet Explorer 8 or less. It's called the HTML5 Shiv, and including it in your pages looks something like this:

```
<script src="http://cdnjs.cloudflare.com/ajax/libs/html5shiv/3.6/html5shiv.min.js">
</script>
```

As for what it does, for the time being, just know that it needs to be there, or you'll get unpredictable results in IE8 and older.

If you're interested in the history of this small but vitally important script, see Paul Irish's article, "The History of the HTML5 Shiv," at `http://paulirish.com/2011/the-history-of-the-html5-shiv/`.

Building on the HTML5 Shiv, there's another library you'll encounter in this book. It's called Modernizr and at its core includes the HTML5 Shiv for backward compatibility; in addition, it adds in tests for emerging web features that you can use when building sites. That way you can ensure that you're not trying to serve something to a browser that can't actually handle it. You'll learn about Modernizr throughout the book, but for now the following code sample shows how to include Modernizr:

```
<script
src="http://cdnjs.cloudflare.com/ajax/libs/modernizr/2.6.1/modernizr.min.js">
</script>
```

The power of Modernizr will become more apparent throughout the book. For now the examples in the book use the simpler HTML5 Shiv to ensure compatibility with Internet Explorer 8 and below.

And now, with that trip through the head, you're ready to start adding in the star of the show, the content of your page.

The <body> Element

The `<body>` element appears after the `<head>` element, and as you have already seen, it contains the part of the web page that you actually see in the main browser window, which is sometimes referred to as *body content*. The `<body>` element can carry all the attributes from the *attribute groups*.

Common Content Elements

You spend most of the remaining part of this chapter learning the different elements you can use to describe the structure of text. These include:

- The six levels of headings: `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, and `<h6>`
- Paragraphs `<p>`, preformatted sections `<pre>`, line breaks `<br />`, and addresses `<address>`
- Grouping elements: `<div>`, `<header>`, `<hgroup>`, `<nav>`, `<section>`, `<article>`, and `<hr>`
- Presentational elements: `<b>`, `<i>`, `<sup>`, and `<sub>`
- Phrase elements: `<em>`, `<strong>`, `<abbr>`, `<dfn>`, `<blockquote>`, `<q>`, `<cite>`, `<code>`, `<kbd>`, `<var>`, and `<samp>`
- Lists such as unordered lists using `<ul>` and `<li>`; ordered lists using `<ol>` and `<li>`; and definition lists using `<dl>`, `<dt>`, and `<dd>`
- Editing elements: `<ins>` and `<del>`

That may sound like a lot of elements, but you might be surprised at how quickly you can move through them.

BASIC TEXT FORMATTING

Because almost every document you create contains some form of text, the elements you are about to meet are likely to feature in most pages that you will build. In this section, you learn how to use *basic text formatting elements*:

- `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, and `<h6>`
- `<p>`, `<br>`, and `<pre>`

As you read through this section, one browser might display each of these elements in a certain way, and another browser might display the same page in a slightly different way. For example, the typefaces used, the font sizes, and the spaces around these elements may differ between browsers. (And therefore the amount of space a section of text takes up can vary, too.)

Before you look at the elements, it helps to know how text displays by default without any elements. This helps demonstrate the importance to use markup to tell the browser if you want it to treat text differently.

White Space and Flow

Before you start to mark up your text, it's best to understand what HTML does when it comes across spaces and how browsers treat long sentences and paragraphs of text.

You might think that if you put several consecutive spaces between two words, the spaces would appear between those words onscreen, but this is not the case; by default, only one space displays. This is known as *white space collapsing*. Similarly, if you start a new line in your source document, or you have consecutive empty lines, these will be ignored and simply treated as one space, as will tab characters. For example, consider the following paragraph (taken from `ch01_eg03.html` in the code samples):

```
<p>This   paragraph shows how   multiple spaces     between     words are
treated as a single space. This is known as white space collapsing, and
the big spaces between    some of the    words will not appear   in the
browser.

It also demonstrates how the browser will treat multiple carriage returns
(new lines) as a single space, too.</p>
```

In Figure 1-5 the browser treats the multiple spaces and several carriage returns (where text appears on a new line) as if there were only one single space. It also enables the line to take up the full width of the browser window.

Now look at the code for this example again, and compare where each new line starts in the code with where each new line starts onscreen. Unless told otherwise, when a browser displays text, it automatically takes up the full width of the screen and *wraps* the text onto new lines when it runs out of space (refer to Figure 1-5). You can see the effect of this better if you open this example in a browser and try resizing the browser window (making it smaller and larger) and notice how the text wraps at new places on the screen. (This example is available with the rest of the download code for this book at `www.wrox.com`.)

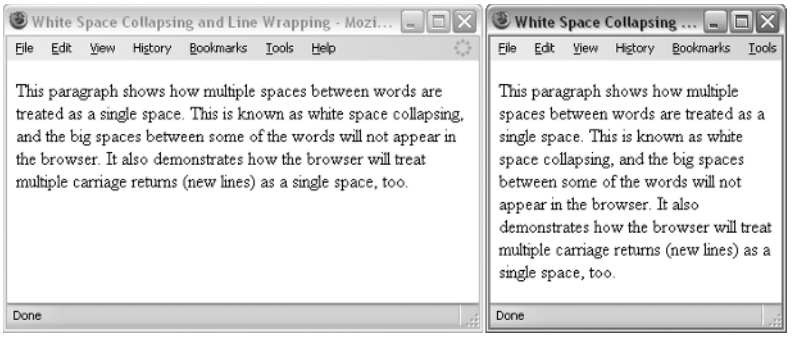

FIGURE 1-5

White space collapsing can be particularly helpful because it enables you to add extra spaces into your HTML that do not show up when viewed in a browser. You can use these spaces to indent your code, which makes it easier to read. The first two examples in this chapter demonstrated indented code, where child elements are indented from the left to distinguish them from their parent elements, which is used this throughout this book to make the code more readable. (If you want to preserve the spaces in a document, you need to use either the `<pre>` element, which you learn about later in the chapter, or an entity reference such as ` `, a nonbreaking space, which you learn about in Appendix F, "Special Characters.")

Now that you know how multiple spaces and line breaks are collapsed, you can see why you must learn how to use the elements in the rest of this chapter to break up and control the presentation of your text.

Creating Headings Using <hn> Elements

No matter what sort of document you create, most documents have headings in one form or another. Newspapers use headlines; a heading on a form tells you the purpose of the form; the title of a table of sports results tells you the league or division the teams play in; and so on.

In longer pieces of text, headings can also help structure a document. If you look at the table of contents for this book, you can see how different levels of headings have been arranged to add structure to the book, with subheadings under the main headings.

HTML offers six levels of headings, which use the elements `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, and `<h6>`. Browsers display the `<h1>` element as the largest of the six and `<h6>` as the smallest. (Although you can see in Chapter 7 that you can use CSS to override the size and style of any of the elements.) The levels of headings would look something like those in Figure 1-6 (`ch01_eg04.html`).

> ***WARNING*** *Most browsers display the contents of the* `<h1>`*,* `<h2>`*, and* `<h3>` *elements larger than the default size of text in the document. The content of the* `<h4>` *element would be the same size as the default text, and the content of the* `<h5>` *and* `<h6>` *elements would be smaller unless you instruct them otherwise using CSS.*

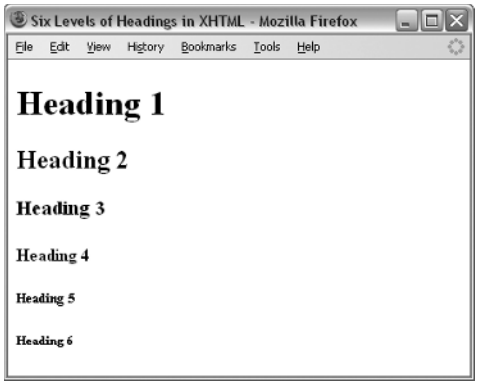

FIGURE 1-6

Here is another example of how you might use headings to structure a document (`ch01_eg05.html`), where the `<h2>` elements are subheadings of the `<h1>` element. (This actually models the structure of this section of the chapter.)

```
<h1>Basic Text Formatting</h1>
<p> This section is going to address the way in which you mark up text.
Almost every document you create will contain some form of text, so this
will be a very important section. </p>
<h2>White Space and Flow</h2>
<p> Before you start to mark up your text, it is best to understand what HTML does
when it comes across spaces and how browsers treat long sentences and
paragraphs of text.</p>
<h2>Creating Headings</h2>
<p> No matter what sort of document you are creating, most documents have
 headings in some form or other...</p>
```

Figure 1-7 shows how this will look.

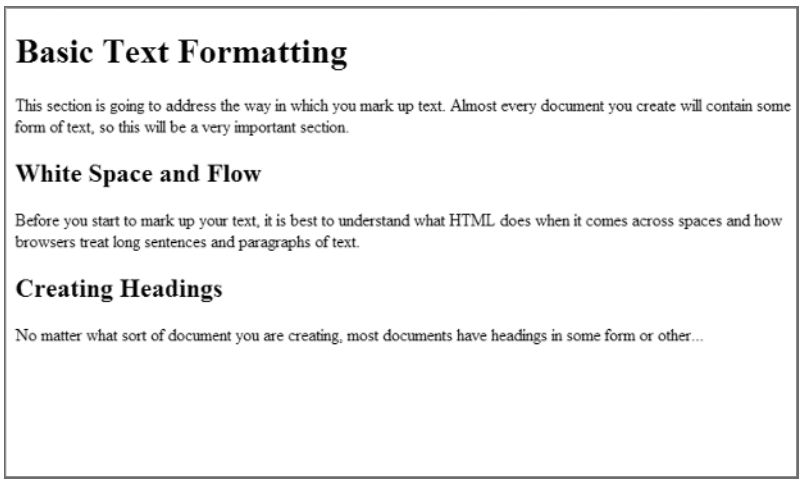

FIGURE 1-7

The six heading elements can all carry the universal attributes:

```
class id style title dir lang
```

Creating Paragraphs Using the Element

The element offers another way to structure your text. Each paragraph of text should go in between an opening and closing tag, as in this example (ch01_eg06.html):

```
<p>Here is a paragraph of text.</p>
<p>Here is a second paragraph of text.</p>
<p>Here is a third paragraph of text.</p>
```

When a browser displays a paragraph, it usually inserts a new line before the next paragraph and adds a little bit of extra vertical space, as shown in Figure 1-8.

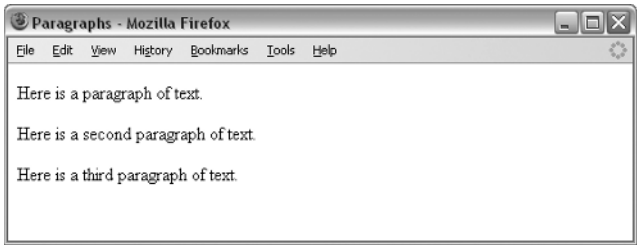

FIGURE 1-8

The element can carry all the universal attributes:

```
class id style title dir lang
```

Creating Line Breaks Using the
 Element

Whenever you use the
 element, anything following it starts on the next line. The
 element is an example of an *empty element*; you don't need opening *and* closing tags, because there is nothing to go in between them.

You can use multiple
 elements to push text down several lines, and many designers use two line breaks between paragraphs of text rather than using the element to structure text, as follows:

```
Paragraph one<br><br>
Paragraph two<br><br>
Paragraph three<br><br>
```

Although two
 elements look similar to using a element, remember that HTML markup is supposed to describe the structure of the content. So if you use two
 elements between paragraphs, you are not describing the document structure.

> ***NOTE*** *Strictly speaking, do not use* `<br>` *elements to position text; use them only within a block-level element. The* `<p>` *element is a block-level element; you learn more about these in the "Understanding Block and Inline Elements" section.*

Here you can see an example of the `<br>` element in use within a paragraph (`ch01_eg07.html`):

```
<p>When you want to start a new line you can use the line break element.
So, the next<br />word will appear on a new line.</p>
```

Figure 1-9 shows you how the line breaks look after the words "next" and "do."

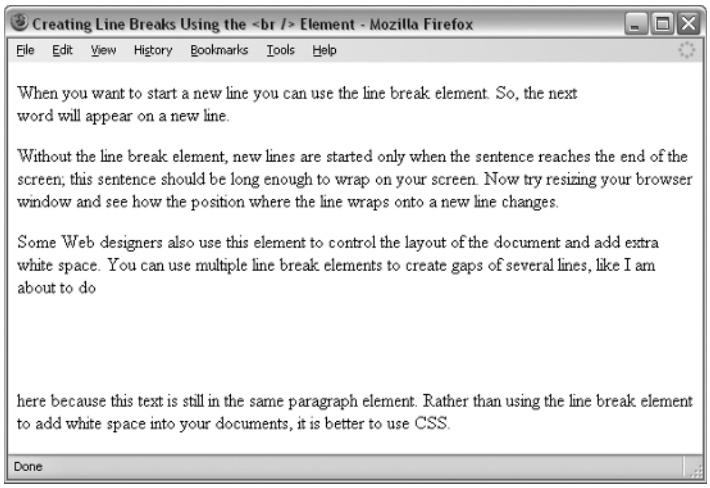

FIGURE 1-9

The `<br>` element can carry the following attributes:

```
class id style title
```

Creating Preformatted Text Using the <pre> Element

Sometimes you want your text to follow the exact format of how it is written in the HTML document; you don't want the text to wrap onto a new line when it reaches the edge of the browser. You also don't want it to ignore multiple spaces, and you want the line breaks where you put them.

Any text between the opening `<pre>` tag and the closing `</pre>` tag preserves the formatting of the source document. You should be aware, however, that most browsers display this text in a monospaced font by default. (Courier is an example of a monospaced font because each letter of the alphabet takes up the same width. Compare this to a nonmonospaced font, where an *i* is usually narrower than an *m*.)

The most common uses of the <pre> element are to represent computer source code. For example, the following shows some JavaScript inside a <pre> element (ch01_eg08.html):

```
<pre>
function testFunction( strText ){
   console.log( strText )
}
</pre>
```

Figure 1-10 shows how the content of the <pre> element displays in the monospaced font. More important, you can see how it follows the formatting shown inside the <pre> element—the white space is preserved.

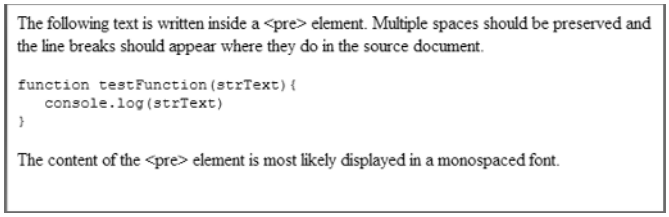

FIGURE 1-10

Although tab characters can have an effect inside a <pre> element, and a tab is supposed to represent eight spaces, the implementation of tabs varies across browsers, so it is advisable to use spaces instead.

You will come across more elements that you can use to represent code in the next chapter, "Fine-tuning Your Text," which covers the <code>, <kbd>, and <var> elements.

TRY IT OUT Basic Text Formatting

Now that you've seen the basic elements that you can use to format your text—headings and paragraphs—it's time to put that information to work.

In this example, you create a page for a fictional company called Example Café. You will work on this example throughout the book to build up an entire site. This page is going to be the homepage for the site, introducing people to the café:

1. Add the skeleton of the document: the DOCtype declaration and the <html>, <head>, <title>, and <body> elements.

```
<!doctype html>
<html>
  <head>
    <title>Example Cafe - community cafe in Newquay, Cornwall, UK </title>
  </head>
  <body>
  </body>
</html>
```

The entire page is contained in the <html> element. The <html> element can contain only two child elements: the <head> element and <body> element. The <head> element contains the title for the page, and you can tell from the title of the page the type of information the page contains.

Meanwhile, the <body> element contains the main part of the web page, the part that viewers actually see in the main part of the web browser.

2. Add to your page a main heading and some level 2 headings to add structure to the information on the page:

```
<body>
  <h1>EXAMPLE CAFE</h1>
  <h2>A community cafe serving home cooked, locally sourced, organic food</h2>
  <h2>This weekend's special brunch</h2>
</body>
```

3. Fill out the page with some paragraphs that follow the headings:

```
 <body>
  <h1>EXAMPLE CAFE</h1>
    <p>Welcome to example cafe. We will be developing this site throughout
    the book.</p>
  <h2>A community cafe serving home cooked, locally sourced, organic food</h2>
    <p>With stunning views of the ocean, Example Cafe offers the perfect
environment to unwind and recharge the batteries.</p>
    <p>Our menu offers a wide range of breakfasts, brunches and lunches,
    including a range of vegetarian options.</p>
    <p>Whether you sip on a fresh, hot coffee or a cooling smoothie, you never
    need to feel rushed - relax with friends or just watch the world go by.</p>
    <h2>This weekend's special brunch</h2>
    <p>This weekend, our season of special brunches continues with scrambled egg
    on an English muffin. Not for the faint-hearted, the secret to these eggs is
    that they are made with half cream and cooked in butter, with no more than
    four eggs in the pan at a time.</p>
</body>
```

4. Save the file as index.html and then open it in a web browser. The result should look something like Figure 1-11. The name index.html is often given to the homepage of sites built in HTML.

How It Works

The basic skeleton, the <head> and <body> elements, combined with a mix of <p> elements and headings might be simple, but it represents a complete web page. The default rendering of a <p> element and the <h1> and <h2> elements give this page solid structure even without any style information.

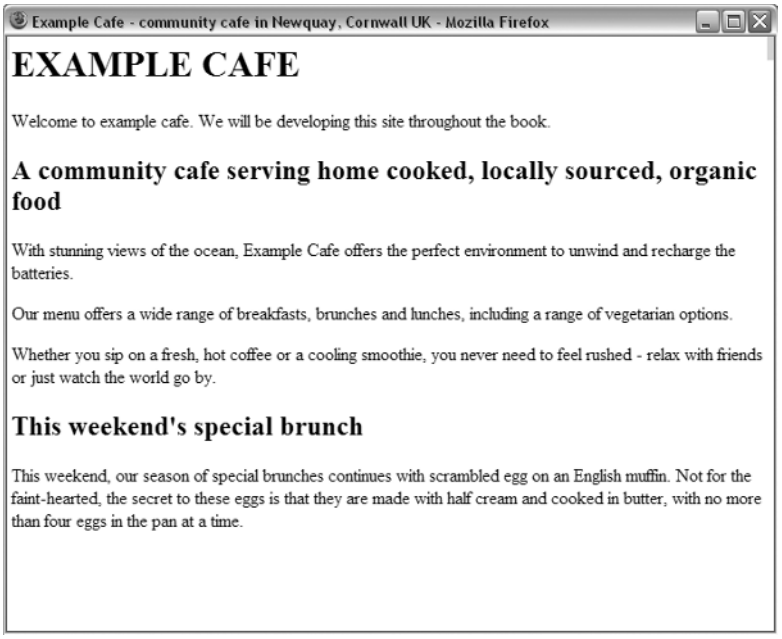

FIGURE 1-11

UNDERSTANDING BLOCK AND INLINE ELEMENTS

Now that you have seen many of the elements that you can use to mark up text, you must consider all the elements that live inside the `<body>` element because each can fall into one of two categories:

- Block-level elements
- Inline elements

This is quite a conceptual distinction, but it has important ramifications for other features of HTML.

Block-level elements appear on the screen as if they have a carriage return or line break before and after them. For example, the `<p>`, `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`, `<ul>`, `<ol>`, `<dl>`, `<pre>`, `<hr />`, `<blockquote>`, and `<address>` elements are all block-level elements. They all start on their own new lines, and anything that follows them appears on its own new line, too.

Inline elements, on the other hand, can appear within sentences and do not need to appear on new lines of their own. The `<b>`, `<i>`, `<u>`, `<em>`, `<strong>`, `<sup>`, `<sub>`, `<small>`, `<ins>`, `<del>`, `<code>`, `<cite>`, `<dfn>`, `<kbd>`, and `<var>` elements are all inline elements.

For example, look at the following heading and paragraph; both of these elements start on new lines, and anything that follows them goes on a new line, too. Meanwhile, the inline elements in the paragraph are not placed on their own new lines. Here is the code (`ch01_eg09.html`):

```
<h1>Block-Level Elements</h1>
  <p><strong>Block-level elements</strong> always start on a new line. The
  <code>&lt;h1&gt;</code> and <code>&lt;p&gt;</code> elements will not sit
  on the same line, whereas the inline elements flow with the rest of the
  text.</p>
```

You can see what this looks like in Figure 1-12.

FIGURE 1-12

Strictly speaking, inline elements may not contain block-level elements and can appear only within block-level elements. (So you should not have a `<b>` element outside a block-level element.) Block-level elements, meanwhile, can contain other block-level elements and inline elements.

GROUPING CONTENT

You can find one of the most interesting new additions to the HTML5 specification in a number of new elements added to aid in grouping content. As you just learned, elements represent the different parts of an article with different levels of headings and paragraphs. As you'll learn throughout the chapter, there are many other elements left to mark up to describe text.

One thing that was lacking was the ability to group all that well-described content into meaningful ways. So, although you could write an article and mark up each paragraph meaningfully, it wasn't possible to indicate that the entire block of text was an article. HTML5 changes that with the addition of several new elements designed to enable you to more accurately group content.

This section introduces the various ways to group content in HTML.

The New Outline Algorithm in HTML5

In adding these new grouping elements, the authors of the specification made a change to the way headings are used in HTML. Before HTML5, the general standard was to have one `<h1>` element per page. The basic idea was that the entire HTML document was by itself a standalone element and should therefore have just one overall outline with a single `<h1>` at the top of the tree.

As the web evolved to contain more complicated, compound documents, this concept no longer held true for many of the documents on the web. With a blog, you can easily have several separate articles all contained in a single HTML document. Each of those articles could contain a logical outline of its own, independent of the rest of the page. Because of this, the ability to add more than one `<h1>` to the page inside different sectioning elements was added.

As you'll learn throughout this section and as you continue to work with HTML5, there's much greater flexibility in the ways you can structure pages using the new markup.

As an aid to work with HTML5 documents, there's a handy bookmarklet and Google Chrome extension called the HTML5 Outliner (h5o) that can help to analyze the structure of your pages. Even experienced authors find it useful, so it's definitely good to have on hand to make sure your pages make sense at the outline level. You can download it from Google code: `http://code.google.com/p/h5o/`.

The Element

Before the extension of the grouping elements in HTML5, the most common container for groups of HTML elements was the `<div>`. It represents a generic block of content and is designed to be used with `classes` and `ids` to give structure to documents. Taking the markup from Example Café, you could mark it up with `<div>` elements representing different content sections.

For example, if you want to set the header apart in some way, you can mark it up like the following example. Using a `<div>` with a class of `header` you can encapsulate the site title and tagline into a single structure:

```
<div class="header">
  <h1>EXAMPLE CAFE</h1>
  <p>Welcome to example cafe. We will be developing this site throughout
the book.</p>
</div>
```

Although this is actually quite useful for styling and scripting, there's no semantic meaning behind the `<div>`, no matter how well chosen the `classes` or `ids` are. As you see throughout this section, the new sectioning elements in HTML5 enhance the utility of the `<div>` with (mostly) straightforward semantic meaning.

The <header> Element

As part of the development of the new specification, the editor of the HTML standard, Ian Hickson, did a survey of the web and identified common markup patterns. Some of these he captured as new HTML elements. This is one such element.

As you saw in the previous example, the concept of a "header" for common introductory or navigation content is a useful one and was repeated over and over on the web. Marking up the previous example with a `<header>` simplifies the markup and imparts more semantic meaning to the page.

```
<header>
  <h1>EXAMPLE CAFE</h1>
  <p>Welcome to example cafe. We will be developing this site throughout
the book.</p>
</header>
```

Although this example shows a header generated at the page level, you can also use headers within other grouping elements. For example, an `<article>` element, which you'll learn about shortly, can have its own header containing information about the author, the data published, and the title of the article.

The <hgroup> Element

The `<hgroup>` element is designed to group together multiple levels of headings that have some logical connection, for example, subheadings, alternative titles, or taglines.

Adding an `<hgroup>` element, and a silly tagline, to the previous example illustrates how to use the `<hgroup>` element:

```
<header>
  <hgroup>
    <h1>EXAMPLE CAFE</h1>
    <h2>Serving Home Style Example Markup since 2012</h2>
  </hgroup>
</header>
```

The <nav> Element

The `<nav>` element represents a navigation section of the page, containing a list of links to other pages or site sections within the site or application. Because the Example Café site is going to have more than one page, look at one way to mark up a simple menu linking to other pages on the site. In this example you can see a series of `<p>` tags, containing links to the site's other pages. Later you learn about working with lists and learn a more common pattern for marking up navigational elements. For now, just focus on the use of the `<nav>` element.

```
<nav>
  <p><a href="recipes.html">Recipes</p>
  <p><a href="menu.html">Menu</a></p>
  <p><a href="opening_times.html">Opening Times</a></p>
  <p><a href-"contact.html">Contact</a></p>
</nav>
```

The <section> Element

The `<section>` element is used to represent a section of a document or application. A `<section>` differs from a `<div>`, the most generic content grouping element, by the idea that content contained in a `<section>` is designed to be part of the document's outline.

Although the Example Café site is going to have separate pages, it can conceivably be one page broken into several sections. A simplified example, just using the headings for each section and no content, would look something like this:

```
<section>
  <h1>Introduction</h1>
</section>
<section>
```

```
  <h1>Recipes</h1>
</section>
<section>
  <h1>Menu</h1>
</section>
<section>
  <h1>Opening Times</h1>
</section>
<section>
  <h1>Contact</h1>
</section>
```

The <article> Element

In the words of the specification, you can use an `<article>` element to mark up "independent content." That's not the friendliest description, so it's useful to think of common examples such as a blog post, a forum post, a movie review, a news article, or an interactive widget. The basic rule of thumb is that if the content can be syndicated or shared without the rest of the site context, it should be marked up as an article. For example, a short, glowing review of the Example Café might be marked up like the following example:

```
<article>
  <h1>Example Cafe, Great Food Delivered with Superior Markup and Style</h1>
  <p>It's rare to find a restaurant that combines as many exemplary elements as
the example café. From the superior markup of the café's website to the
delicious dishes served with care nothing about the Example Café is
left to chance.</p>
</article>
```

The <hr> Element

The `<hr>` element creates a horizontal rule across the page. It is an empty element, rather like the `<br>` element.

```
<hr>
```

This is frequently used to separate distinct sections of a page where a new heading is not appropriate.

The <blockquote> Element

When you want to quote a passage from another source, you should use the `<blockquote>` element. Note that there is a separate `<q>` element for use with smaller quotations, as discussed in the next section. Here's `ch01_eg10.html`:

```
<p>The following description of the blockquote element is taken from
the WHATWG site:</p>
<blockquote> The blockquote element represents a section that is quoted from
another source. Content inside a blockquote must be quoted from another
source, whose address, if it has one, may be cited in the
cite attribute.</blockquote>
```

Text inside a <blockquote> element is usually indented from the left and right edges of the surrounding text, and some browsers use an italicized font. (Use it only for quotes—if you simply want an indented paragraph of text, use CSS.) You can see what this looks like in Figure 1-13.

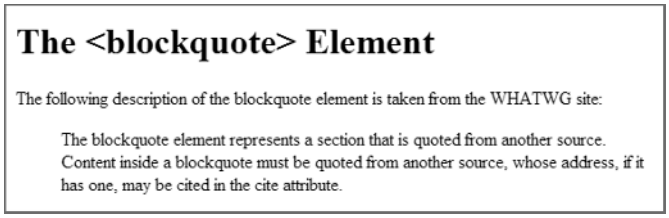

FIGURE 1-13

Using the cite Attribute with the <blockquote> Element

You can use the `cite` attribute on the <blockquote> element to indicate the source of the quote. The value of this attribute should be a URL pointing to an online document; if possible, the exact place in that document. Browsers do not currently do anything with this attribute, but it means the source of the quote is there should you need it in the future (`ch01_eg11.html`).

```
<blockquote cite=
"http://developers.whatwg.org/grouping-content.html#the-blockquote-element">
The blockquote element represents a section that is quoted from another source.
Content inside a blockquote must be quoted from another source, whose address,
if it has one, may be cited in the cite attribute.</blockquote>
```

The <aside> Element

The <aside> element is used to mark up related content such as pull quotes, sidebars, and ads. The content in the aside should be related to the surrounding content.

Pulling the first glowing sentence of the previous review illustrates a simple usage of the <aside> element.

```
<article>
  <h1>Example Cafe, Great Food Delivered with Superior Markup and Style</h1>
  <p>It's rare to find a restaurant that combines as many exemplary elements as
the example café. From the superior markup of the café's website to the
delicious dishes served with care nothing about the Example Café is left
to chance.</p>
  <aside> It's rare to find a restaurant that combines as many exemplary
elements as the example café.</aside>
</article>
```

The <footer> Element

Like the <header>, the <footer> was an extremely common `class` and `id` name found during Ian Hickson's survey of the web. A common usage of the footer is for legal copy. On some sites, such as those of pharmaceutical companies or financial services firms, this can be quite a large block of text,

sometimes stretching to several screen's worth of text. Thankfully Example Café doesn't have quite the same legal and regulatory overhead, so you can just add a simple copyright notice into your `<footer>`.

```
<footer>
  <p>All content copyright Example Café 2012</p>
</footer>
```

The <address> Element

The `<address>` element is used to mark up contact information for an article element or for the document as a whole. In this example you add it as contact information in the footer.

```
<footer>
  <address>For more information contact <a href="mailto:examplecafe@example.com">
Example Café via email</address>
  <p>All content copyright Example Café 2012</p>
</footer>
```

WORKING WITH LISTS

There are many reasons you might want to add a list to your pages, from putting your five favorite albums on your homepage to including a numbered set of instructions for visitors to follow (such as the steps you follow in the Try It Out examples in this book).

You can create three types of lists in HTML:

- **Unordered:** Like lists of bullet points
- **Ordered:** Use a sequence of numbers or letters instead of bullet points
- **Definition:** Enable you to specify a term and its definition

You can think of more uses for the lists as you meet them and start using them.

Using the <ul> Element to Create Unordered Lists

If you want to make a list of bullet points, write the list within the `<ul>` element (which stands for *unordered list*). Each bullet point or line you want to write should then be contained between opening `<li>` tags and closing `</li>` tags. (The `li` stands for *list item*.)

You should always close the `<li>` element. Even though you might see some HTML pages that leave off the closing tag, this is a bad habit you should avoid.

If you want to create a bulleted list, you can do so like this (`ch01_eg12.html`):

```
<ul>
  <li>Bullet point number one</li>
  <li>Bullet point number two</li>
  <li>Bullet point number three</li>
</ul>
```

In a browser, this list would look something like Figure 1-14.

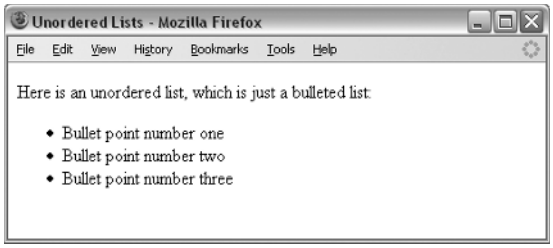

FIGURE 1-14

As promised, the following shows the list of links from the illustration of the `<nav>` element reworked to use an unordered list. Navigation elements on the web are commonly marked up using lists, so it's useful to get used to that pattern as soon as possible.

```
<nav>
  <ul>
    <li><a href="recipes.html">Recipes</li>
    <li><a href="menu.html">Menu</a></li>
    <li><a href="opening_times.html">Opening Times</a></li>
    <li><a href-"contact.html">Contact</a></li>
  </ul>
</nav>
```

The `<ul>` and `<li>` elements can carry all the universal attributes and UI event attributes.

Ordered Lists

Sometimes, you want your lists to be ordered. In an ordered list, rather than prefixing each point with a bullet point, you can use either numbers (1, 2, 3), letters (A, B, C), or Roman numerals (i, ii, iii) to prefix the list item.

An ordered list is contained inside the `<ol>` element. Each item in the list should then be nested inside the `<ol>` element and contained between opening `<li>` and closing `</li>` tags (`ch01_eg13.html`).

```
<ol>
  <li>Point number one</li>
  <li>Point number two</li>
  <li>Point number three</li>
</ol>
```

The result should be similar to what you see in Figure 1-15.

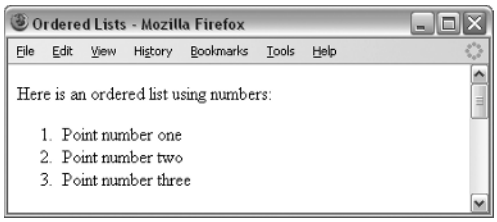

FIGURE 1-15

In Chapter 7, in the section on styling lists with CSS, you learn about customizing the type of glyph used for your ordered lists.

Using the start Attribute to Change the Starting Number in Ordered Lists

If you want to specify the number that a numbered list should start at, you can use the `start` attribute on the `<ol>` element. The value of this attribute should be the numeric representation of that point in the list (`ch01_eg14.html`).

```
<ol start="4">
  <li>Point number one</li>
  <li>Point number two</li>
  <li>Point number three</li>
</ol>
```

You can see the result in Figure 1-16.

FIGURE 1-16

Count Down in Your Ordered Lists with the reversed Attribute

The boolean `reversed` attribute allows you to reverse the order of ordered lists, counting down from the highest number (supported only in Chrome). The following code sample shows this in action (`ch01_eg15.html`).

```
<ol reversed>
  <li>Point number one</li>
  <li>Point number two</li>
  <li>Point number three</li>
</ol>
```

You can see the count begin with "3" in Figure 1-17.

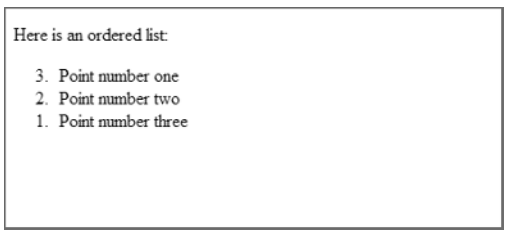

FIGURE 1-17

Specify a Marker with the type Attribute

The `type` attribute allows you to specify the class of markers to use with ordered lists. Table 1-3 shows the available options. If you use this attribute, keep in mind that the values are case-sensitive.

TABLE 1-3: type Attribute Values

KEYWORD	STATE	DESCRIPTION
1	decimal	Decimal number (default)
a	lower-alpha	Lowercase Latin alphabet
A	upper-alpha	Uppercase Latin alphabet
i	lower-roman	Lowercase Roman numerals
I	upper-roman	Uppercase Roman numerals

The following code sample (`ch01_eg16.html`) shows a list using the lowercase Latin alphabet.

```
<ol type="a">
  <li>Point number one</li>
  <li>Point number two</li>
  <li>Point number three</li>
</ol>
```

You can see the count begin with "a" in Figure 1-18.

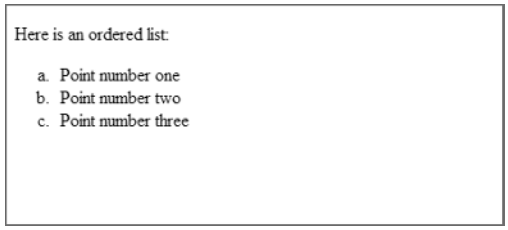

FIGURE 1-18

Definition Lists

The HTML5 spec states `<dl>` is for description lists, which have a slightly wider remit than term and definition. "The `<dl>` element represents a description list, which consists of zero or more term-description (name/value) groupings; each grouping associates one or more terms/names (the contents of `<dt>` elements) with one or more descriptions/values (the contents of `<dd>` elements)."

The definition list is a special kind of list for providing terms followed by a short text definition or description for them. Definition lists are contained inside the `<dl>` element. The `<dl>` element then contains alternating `<dt>` and `<dd>` elements. The content of the `<dt>` element is the term you define. The `<dd>` element contains the definition of the previous `<dt>` element. For example, here is a definition list that describes the different types of lists in HTML (`ch01_eg17.html`):

```
<dl>
  <dt>Unordered List</dt>
  <dd>A list of bullet points.</dd>
  <dt>Ordered List</dt>
  <dd>An ordered list of points, such as a numbered set of steps.</dd>
  <dt>Definition List</dt>
  <dd>A list of terms and definitions.</dd>
</dl>
```

In a browser, this looks something like Figure 1-19.

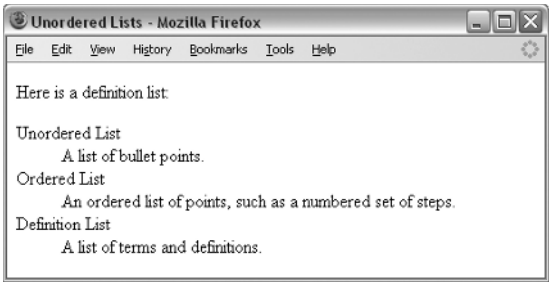

FIGURE 1-19

Each of these elements can carry the universal attributes and UI event attributes.

Nesting Lists

You can nest lists inside other lists. For example, you might want a numbered list with separate points corresponding to one of the list items. Number each nested list separately, unless you specify otherwise using the `start` attribute. And you should place each new list inside a `<li>` element (`ch01_eg18.html`):

```
<ol type="I">
  <li>Item one</li>
  <li>Item two</li>
  <li>Item three</li>
  <li>Item four
    <ol type="i">
      <li>Item 4.1</li>
      <li>Item 4.2</li>
```

```
        <li>Item 4.3</li>
      </ol>
    </li>
    <li>Item Five</li>
  </ol>
```

In a browser, this looks something like Figure 1-20.

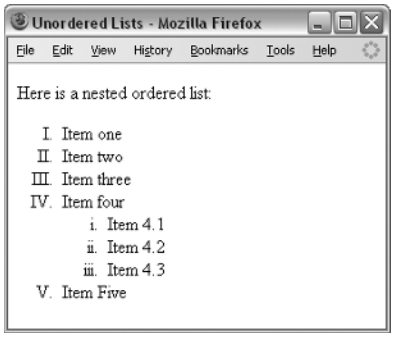

FIGURE 1-20

SUMMARY

In this chapter, you learned about the similarities between web pages and print documents; for example, a news story in print or on the web consists of a headline, some paragraphs of text, maybe some subheadings, and one or more pictures. On the web you need to explain the structure of these documents, and you can do that using HTML.

You know that HTML5 is the latest version of the HTML5 specification, and you know the special DOCTYPE used to put HTML documents into standards mode.

You have learned that the content of a web page is marked up using elements that describe the structure of the document. These elements consist of an opening tag, a closing tag, and some content between the opening and closing tags. To alter some properties of elements, the opening tag may carry attributes, and attributes are commonly written as name/value pairs. They can also be empty elements.

You also learned a lot of new elements and the attributes they can carry. You've seen how every HTML document should contain at least the `<html>`, `<head>`, `<title>`, and `<body>` elements. In the next chapter you learn about fine-tuning your text by adding meaning for both humans and computers, more specific structure and other features that will greatly enrich your web pages.

EXERCISES

1. Mark up the following list, with inserted and deleted content:

Ricotta pancake ingredients:

- 1 ~~1/2~~ 3/4 cups ricotta
- 3/4 cup milk
- 4 eggs
- 1 cup plain white flour
- 1 teaspoon baking powder
- ~~75g~~ 50g butter
- pinch of salt

The answers to all the exercises are in Appendix A, "Answers to Exercises."

▶ WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY TAKEAWAY
HTML	HTML is the foundation that the World Wide Web is built upon.
Tags, elements, and attributes	Tags, elements, and attributes combine to describe content for the web.
Paragraphs and headers	HTML offers tools to create basic document outlines, including different levels of headers and plain paragraphs.
Grouping content	Starting with HTML5, there are several ways to group content in HTML, including elements that describe quotes, headers, footers, and articles.
Lists	HTML has several tools for marking up lists.