MARKING UP TEXT

Once your content is ready to go (you proofread it, right?) and you've added the markup to structure the document (**`html`**, **`head`**, **`title`**, and **`body`**), you are ready to identify the elements in the content. This chapter introduces the elements you have to choose from for marking up text content. There probably aren't as many of them as you might think, and really just a handful that you'll use with regularity. That said, this chapter is a big one and covers a lot of ground.

As we begin our tour of elements, I want to reiterate how important it is to choose elements semantically, that is, in a way that most accurately describes the content's meaning. If you don't like how it looks, change it with a style sheet. A semantically marked up document ensures your content is available and accessible in the widest range of browsing environments, from desktop computers and mobile devices to assistive screen readers. It also allows non-human readers, such as search engine indexing programs, to correctly parse your content and make decisions about the relative importance of elements on the page.

With these principles in mind, it is time to meet the HTML text elements, starting with the most basic element of them all, the humble paragraph.

IMPORTANT NOTE

I will be teaching markup according to the HTML5 standard maintained by the W3C (www.w3.org/TR/html5/) as it appeared as of this writing in mid-2012. There is another "living" (therefore unnumbered) version of HTML maintained by the WHATWG (whatwg.org) that is nearly the same, but usually has some differences. I will be sure to point out elements and attributes that belong to only one spec. Both specs are changing frequently, so I urge you to check online to see whether elements have been added or dropped.

You may have heard that not all browsers support HTML5. That is true. But the vast majority of the elements in HTML5 have been around for decades in earlier HTML versions, so they are supported universally. Elements that are new in HTML5 and may not be well supported will be indicated with this marker: NEW IN HTML5 *. So, unless I explicitly point out a support issue, you can assume that the markup descriptions and examples presented here will work in all browsers.*

IN THIS CHAPTER

Paragraphs

`<p>...</p>`

A paragraph element

Paragraphs are the most rudimentary elements of a text document. You indicate a paragraph with the **p** element by inserting an opening ** ** tag at the beginning of the paragraph and a closing ** ** tag after it, as shown in this example.

```
<p>Serif typefaces have small slabs at the ends of letter strokes. In
general, serif fonts can make large amounts of text easier to read.</p>

<p>Sans-serif fonts do not have serif slabs; their strokes are square
on the end. Helvetica and Arial are examples of sans-serif fonts.
In general, sans-serif fonts appear sleeker and more modern.</p>
```

Visual browsers nearly always display paragraphs on new lines with a bit of space between them by default (to use a term from CSS, they are displayed as a block). Paragraphs may contain text, images, and other inline elements (called phrasing content in the spec), but they may *not* contain headings, lists, sectioning elements, or any element that typically displays as a block by default.

NOTE

You must assign an element to all the text in a document. In other words, all text must be enclosed in some sort of element. Text that is not contained within tags is called "naked" or "anonymous" text, and it will cause a document to be invalid. For more information about checking documents for validity, see Chapter 4, Creating a Simple Page (HTML Overview).

In HTML, it is OK to omit the closing ** ** tag. A browser just assumes it is closed when it encounters the next block element. However, in the stricter XHTML syntax, the closing tag is required (no surprise there). Many web developers, including myself, prefer to close paragraphs and all elements, even in HTML, for the sake of consistency and clarity. I recommend folks who are just learning markup, like yourself, do the same.

Headings

`<h1>...</h1>`
`<h2>...</h2>`
`<h3>...</h3>`
`<h4>...</h4>`
`<h5>...</h5>`
`<h6>...</h6>`

Heading elements

In the last chapter, we used the **h1** and **h2** elements to indicate headings for the Black Goose Bistro page. There are actually six levels of headings, from **h1** to **h6**. When you add headings to content, the browser uses them to create a document outline for the page. Assistive reading devices such as screen readers use the document outline to help users quickly scan and navigate through a page. In addition, search engines look at heading levels as part of their algorithms (information in higher heading levels may be given more weight). For these reasons, it is a best practice to start with the Level 1 heading (**h1**) and work down in numerical order (see note), creating a logical document structure and outline.

NOTE

HTML5 has a new outlining system that looks beyond headings to generate the outline. See the sidebar Sectioning Content later in this chapter for details.

This example shows the markup for four heading levels. Additional heading levels would be marked up in a similar manner.

```
<h1>Type Design</h1>

<h2>Serif Typefaces</h2>
<p>Serif typefaces have small slabs at the ends of letter strokes.
In general, serif fonts can make large amounts of text easier to
read.</p>
```

```
<h3>Baskerville</h3>

<h4>Description</h4>
<p>Description of the Baskerville typeface.</p>

<h4>History</h4>
<p>The history of the Baskerville typeface.</p>

<h3>Georgia</h3>
<p>Description and history of the Georgia typeface.</p>

<h2>Sans-serif Typefaces</h2>
<p>Sans-serif typefaces do not have slabs at the ends of strokes.</p>
```

The markup in this example would create the following document outline:

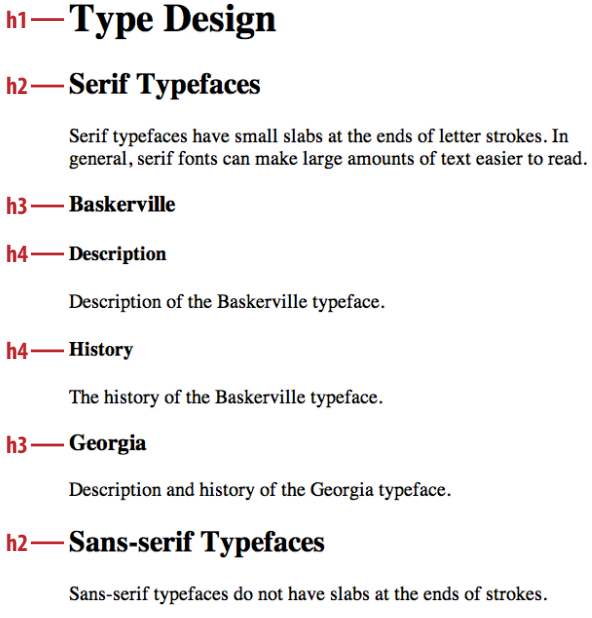
1. Type Design
    1. Serif Typefaces
       + text paragraph
        1. Baskerville
            1. Description
               + text paragraph
            2. History
               + text paragraph
        2. Georgia
           + text paragraph
    2. Sans-Serif Typefaces
       + text paragraph

By default, the headings in our example will be displayed in bold text, starting in very large type for `h1`s, with each consecutive level in smaller text, as shown in Figure 5-1. You can use a style sheet to change their appearance.

NOTE

All screenshots in this book were taken using the Chrome browser on a Mac unless otherwise noted.

Figure 5-1. The default rendering of four heading levels.



Indicating a Shift in Themes

`<hr>`

A horizontal rule

If you want to indicate that one topic or thought has completed and another one is beginning, you can insert what is called in HTML5 a "paragraph-level thematic break" using the **hr** element. It is used as a logical divider between sections of a page or paragraphs of text. The **hr** element adds a logical divider between sections or paragraphs of text without introducing a new heading level.

In HTML versions prior to HTML5, **hr** was defined as a "horizontal rule" because it inserted a horizontal line on the page. Browsers still render **hr** as a 3D shaded rule and put it on a line by itself with some space above and below by default, but it now has a new semantic purpose. If a decorative line is all you're after, it is better to create a rule by specifying a colored border before or after an element with CSS.

hr is an empty element—you just drop it into place where you want the thematic break to occur, as shown in this example and Figure 5-2. Note that in XHTML, the **hr** element must be closed with a slash: **<hr />**.

```
<h3>Times</h3>
<p>Description and history of the Times typeface.</p>
<hr>
<h3>Georgia</h3>
<p>Description and history of the Georgia typeface.</p>
```

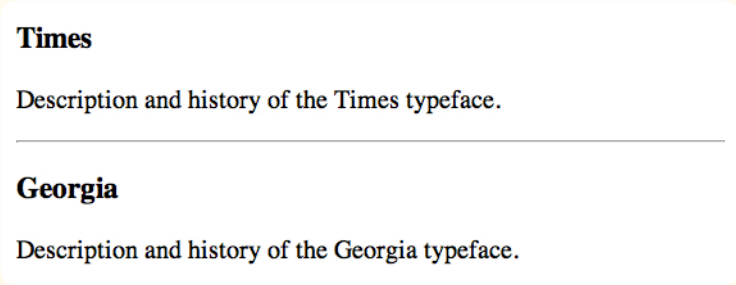


Figure 5-2. The default rendering of a horizontal rule.

Heading groups

`<hgroup>...</hgroup>`

A group of stacked headings

NEW IN HTML5

It is common for headlines to have clarifying subheads or taglines. Take, for example, the title of Chapter 4 in this book:

> **Creating a Simple Page**
> (HTML Overview)

In the past, marking stacks of headings and subheadings was somewhat problematic. The first line, "Creating a Simple Page," is clearly an **h1**, but if you make the second line an **h2**, you may introduce an unintended new level to the document outline. The best you could do was mark it as a paragraph, but that didn't exactly make semantic sense.

For this reason, HTML5 includes the **hgroup** element for identifying a stack of headings as a group.* Browsers that support **hgroup** know to count only the highest-ranked heading in the outline and ignore the rest. Here is how the **hgroup** element could be used to mark up the title of Chapter 4. With this markup, only the **h1**, "Creating a Simple Page," would be represented in the document outline.

```
<hgroup>
  <h1>Creating a Simple Page</h1>
  <h2>(HTML Overview)</h2>
</hgroup>
```

SUPPORT ALERT

The `hgroup` *element is not supported in Internet Explorer versions 8 and earlier (see the sidebar HTML5 Support in Internet Explorer later in this chapter for a workaround). Older versions of Firefox and Safari (prior to 3.6 and 4, respectively) do not support it according to the spec, but they don't ignore it completely, so you can apply styles to it.*

Lists

Humans are natural list makers, and HTML provides elements for marking up three types of lists:

- **Unordered lists.** Collections of items that appear in no particular order.
- **Ordered lists.** Lists in which the sequence of the items is important.
- **Description lists.** Lists that consist of name and value pairs, including but not limited to terms and definitions.

All list elements—the lists themselves and the items that go in them—are displayed as block elements by default, which means that they start on a new line and have some space above and below, but that may be altered with CSS. In this section, we'll look at each list type in detail.

Unordered lists

`<ul>...</ul>`
Unordered list

`<li>...</li>`
List item within an unordered list

Just about any list of examples, names, components, thoughts, or options qualify as unordered lists. In fact, most lists fall into this category. By default, unordered lists display with a bullet before each list item, but you can change that with a style sheet, as you'll see in a moment.

To identify an unordered list, mark it up as a **ul** element. The opening **
** tag goes before the first list item, and the closing tag **
** goes after the last item. Then, each item in the list gets marked up as a list item (**li**) by enclosing it in opening and closing **li** tags, as shown in this example. Notice that there are no bullets in the source document. They are added automatically by the browser (Figure 5-3).

```
<ul>
  <li><a href="">Serif</a></li>
  <li><a href="">Sans-serif</a></li>
  <li><a href="">Script</a></li>
  <li><a href="">Display</a></li>
  <li><a href="">Dingbats</a></li>
</ul>
```

NOTE

The only thing that is permitted within an unordered list (that is, between the start and end `ul` *tags) is one or more list items. You can't put other elements in there, and there may not be any untagged text. However, you can put any type of flow element within a list item (*`li`*).*

* Although potentially useful, the future of the **hgroup** element is uncertain. If you are interested in using it for a published site, you should check the HTML5 specification first.

- Serif
- Sans-serif
- Script
- Display
- Dingbats

Figure 5-3. The default rendering of the sample unordered list. The bullets are added automatically by the browser.

Nesting Lists

Any list can be nested within another list; it just has to be placed within a list item. This example shows the structure of an unordered list nested in the second ordered list item.

```
<ol>
  <li></li>
  <li>
    <ul>
      <li></li>
      <li></li>
      <li></li>
    </ul>
  </li>
</ol>
```

When you nest an unordered list within another unordered list, the browser automatically changes the bullet style for the second-level list. Unfortunately, the numbering style is not changed by default when you nest ordered lists. You need to set the numbering styles yourself using style sheets.

But here's the cool part. We can take that same unordered list markup and radically change its appearance by applying different style sheets, as shown in Figure 5-4. In the figure, I've turned off the bullets, added bullets of my own, made the items line up horizontally, even made them look like graphical buttons. The markup stays exactly the same.

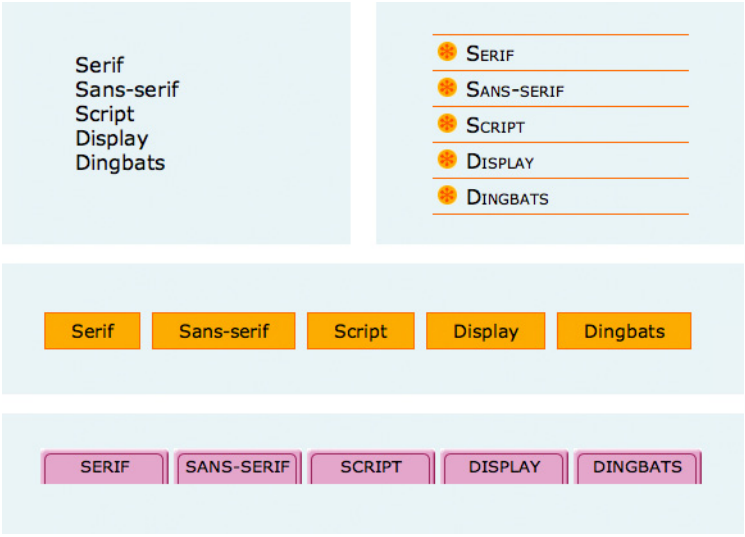


Figure 5-4. With style sheets, you can give the same unordered list many different looks.

Ordered lists

`<ol>...</ol>`
Ordered list

`<li>...</li>`
List item within an ordered list

Ordered lists are for items that occur in a particular order, such as step-by-step instructions or driving directions. They work just like the unordered lists described earlier, except they are defined with the `ol` element (for ordered list, of course). Instead of bullets, the browser automatically inserts numbers before ordered list items, so you don't need to number them in the source document. This makes it easy to rearrange list items without renumbering them.

Ordered list elements must contain one or more list item elements, as shown in this example and in Figure 5-5:

```
<ol>
  <li>Gutenburg develops moveable type (1450s)</li>
  <li>Linotype is introduced (1890s)</li>
  <li>Photocomposition catches on (1950s)</li>
  <li>Type goes digital (1980s)</li>
</ol>
```

1. Gutenburg develops moveable type (1450s)
2. Linotype is introduced (1890s)
3. Photocomposition catches on (1950s)
4. Type goes digital (1980s)

Figure 5-5. The default rendering of an ordered list. The numbers are added automatically by the browser.

If you want a numbered list to start at a number other than "1," you can use the **start** attribute in the **ol** element to specify another starting number, as shown here:

```
<ol start="17">
  <li>Highlight the text with the text tool.</li>
  <li>Select the Character tab.</li>
  <li>Choose a typeface from the pop-up menu.</li>
</ol>
```

The resulting list items would be numbered 17, 18, and 19, consecutively.

Changing Bullets and Numbering

You can use the **list-style-type** style sheet property to change the bullets and numbers for lists. For example, for unordered lists, you can change the shape from the default dot to a square or an open circle, substitute your own image, or remove the bullet altogether. For ordered lists, you can change the numbers to roman numerals (I., II., III. or i., ii., iii.), letters (A., B., C., or a., b., c.), and several other numbering schemes. In fact, as long as the list is marked up semantically, it doesn't need to display with bullets or numbering at all. Changing the style of lists with CSS is covered in Chapter 18, CSS Techniques.

Description lists

Description lists are used for any type of name/value pairs, such as terms and their definitions, questions and answers, or other types of terms and their associated information. Their structure is a bit different from the other two lists that we just discussed. The whole description list is marked up as a **dl** element. The content of a **dl** is some number of **dt** elements indicating the names and **dd** elements for their respective values. I find it helpful to think of them as "terms" (to remember the "t" in **dt**) and "definitions" (for the "d" in **dd**), even though that is only one use of description lists in HTML5.

`<dl>...</dl>`
A description list

`<dt>...</dt>`
A name, such as a term or label

`<dd>...</dd>`
A value, such as a description or definition

Here is an example of a list that associates forms of typesetting with their descriptions (Figure 5-6).

```
<dl>
  <dt>Linotype</dt>
  <dd>Line-casting allowed type to be selected, used, then recirculated
into the machine automatically. This advance increased the speed of
typesetting and printing dramatically.</dd>

  <dt>Photocomposition</dt>
  <dd>Typefaces are stored on film then projected onto photo-sensitive
paper. Lenses adjust the size of the type.</dd>

  <dt>Digital type</dt>
  <dd><p>Digital typefaces store the outline of the font shape in a
format such as Postscript. The outline may be scaled to any size for
output.</p>
      <p>Postscript emerged as a standard due to its support of
```

```
graphics and its early support on the Macintosh computer and Apple
laser printer.</p>
  </dd>
</dl>
```

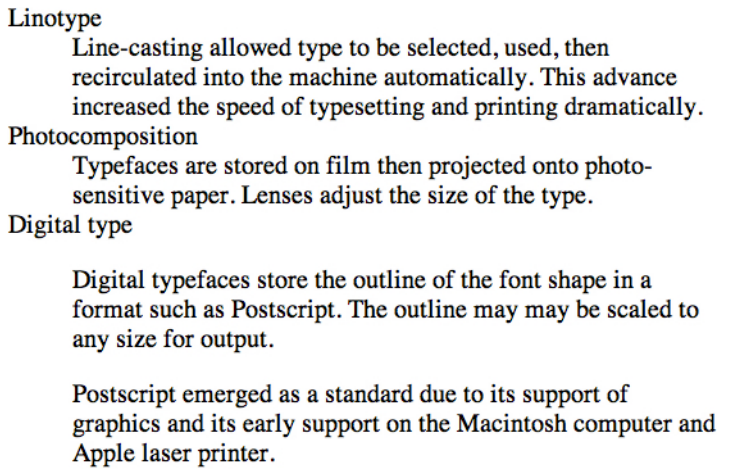


Figure 5-6. The default rendering of a definition list. Definitions are set off from the terms by an indent.

The **dl** element is only allowed to contain **dt** and **dd** elements. It is OK to have multiple definitions with one term and vice versa. You cannot put headings or content-grouping elements (like paragraphs) in names (**dt**), but the value (**dd**) can contain any type of flow content.

Sectioning Roots

The **blockquote** is in a category of elements called sectioning roots. Headings in a sectioning root element will not be included in the main document outline. That means you can have a complex heading hierarchy within a **blockquote** without worrying how it will affect the overall structure of the document. Other sectioning root elements include **figure**, **details**, **fieldset** (for organizing form fields), **td** (a table cell), and **body** (because it has its own outline, which also happens be the outline of the document).

More Content Elements

We've covered paragraphs, headings, and lists, but there are a few more special text elements to add to your HTML toolbox that don't fit into a neat category: long quotations (**blockquote**), preformatted text (**pre**), and figures (**figure** and **figcaption**). One thing these elements do have in common is that they are considered "grouping content" in the HTML5 spec (along with **p**, **hr**, the list elements, and the generic **div**, covered later in this chapter). The other thing they share is that browsers typically display them as block elements by default.

Long quotations

`<blockquote>...</blockquote>`

A lengthy, block-level quotation

If you have a long quotation, a testimonial, or a section of copy from another source, you should mark it up as a **blockquote** element. It is recommended that content within **blockquote** elements be contained in other elements, such as paragraphs, headings, or lists, as shown in this example (see the sidebar Sectioning Roots).

```
<p>Renowned type designer, Matthew Carter, has this to say about his
profession:</p>

<blockquote>
  <p>Our alphabet hasn't changed in eons; there isn't much latitude in
what a designer can do with the individual letters.</p>

  <p>Much like a piece of classical music, the score is written
down - it's not something that is tampered with - and yet, each
conductor interprets that score differently. There is tension in
the interpretation.</p>
</blockquote>
```

NOTE

There is also the inline element, `q`*, for short quotations in the flow of text. We'll talk about it later in this chapter.*

Figure 5-7 shows the default rendering of the **`blockquote`** example. This can be altered with CSS.

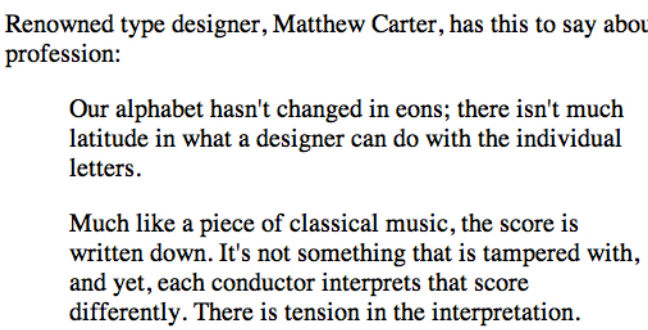


Figure 5-7. *The default rendering of a* `blockquote` *element.*

Preformatted text

`<pre>...</pre>`

Preformatted text

In the previous chapter, you learned that browsers ignore whitespace such as line returns and character spaces in the source document. But in some types of information, such as code examples or poetry, the whitespace is important for conveying meaning. For these purposes, there is the preformatted text (**`pre`**) element. It is a unique element in that it is displayed exactly as it is typed—including all the carriage returns and multiple character spaces. By default, preformatted text is also displayed in a constant-width font (one in which all the characters are the same width, also called monospace), such as Courier.

NOTE

The `white-space:pre` *CSS property can also be used to preserve spaces and returns in the source. Unlike the* `pre` *element, text formatted with the* `white-space` *property is not displayed in a constant-width font.*

The **`pre`** element in this example displays as shown in Figure 5-8. The second part of the figure shows the same content marked up as a paragraph (**`p`**) element for comparison.

```
<pre>
This is                an             example of
      text with a           lot of
                            curious
                            whitespace.
</pre>
```

```
<p>
This is                an             example of
      text with a          lot of
                           curious
                           whitespace.
</p>
```

```
This is                an             example of
      text with a          lot of
                           curious
                           white space.
```

This is an example of text with a lot of curious white space.

Figure 5-8. Preformatted text is unique in that the browser displays the whitespace exactly as it is typed into the source document. Compare it to the paragraph element, in which line returns and character spaces are reduced to a single space.

Figures

`<figure>...</figure>`

Contact information

NEW IN HTML5

`<figcaption>...</figcaption>`

Contact information

NEW IN HTML5

The **figure** element is used for content that illustrates or supports some point in the text. A figure may contain an image, a video, a code snippet, text, or even a table—pretty much anything that can go in the flow of web content—and should be treated and referenced as a self-contained unit. That means if a figure is removed from its original placement in the main flow (to a sidebar or appendix, for example), both the figure and the main flow should continue to make sense.

Although it is possible to simply drop an image into text, wrapping it in **figure** tags makes its purpose explicitly clear. It also allows you to apply special styles to figures but not to other images on the page.

```
<figure>
   <img src="piechart.png" alt="chart showing fonts on mobile devices">
</figure>
```

A caption can be attached to the figure using the optional **figcaption** element above or below the figure content.

```
<figure>
   <pre><code>
    body {
      background-color: #000;
      color: red;
    }
    </code></pre>
   <figcaption>
         Sample CSS rule.
   </figcaption>
</figure>
```

In Exercise 5-1, you'll get a chance to mark up a document yourself and try out the basic text elements we've covered so far.

SUPPORT ALERT

The `figure` *and* `figcaption` *elements are not supported in Internet Explorer versions 8 and earlier (see the sidebar HTML5 Support in Internet Explorer later in this chapter for a workaround). Older versions of Firefox and Safari (prior to 3.6 and 4, respectively) do not support it according to the spec, but allow you to apply styles.*

exercise 5-1 | Marking up a recipe

The owners of the Black Goose Bistro have decided to start a blog to share recipes and announcements. In the exercises in this chapter, we'll assist them with content markup.

Below you will find the raw text of a recipe. It's up to you to decide which element is the best semantic match for each chunk of content. You'll use paragraphs, headings, lists, and at least one special content element.

You can write the tags right on this page. Or, if you want to use a text editor and see the results in a browser, this text file is available online at *www.learningwebdesign.com/4e/materials*. The resulting code appears in Appendix A.

```
Tapenade (Olive Spread)

This is a really simple dish to prepare and it's always a big hit at parties. My father recommends:

"Make this the night before so that the flavors have time to blend. Just bring it up to room temperature
before you serve it. In the winter, try serving it warm."

Ingredients

1 8oz. jar sundried tomatoes
2 large garlic cloves
2/3 c. kalamata olives
1 t. capers

Instructions

Combine tomatoes and garlic in a food processor. Blend until as smooth as possible.

Add capers and olives. Pulse the motor a few times until they are incorporated, but still retain some
texture.

Serve on thin toast rounds with goat cheese and fresh basil garnish (optional).
```

Organizing Page Content

So far, the elements we've covered handle very specific tidbits of content: a paragraph, a heading, a figure, and so on. Prior to HTML5, there was no way to group these bits into larger parts other than wrapping them in a generic division (**`div`**) element (I'll cover **`div`** in more detail later). HTML5 introduced new elements that give semantic meaning to sections of a typical web page or application, including sections (**`section`**), articles (**`article`**), navigation (**`nav`**), tangentially related content (**`aside`**), headers (**`header`**), and footers (**`footer`**). The new element names are based on a Google study that looked at the top 20 names given to generic division elements (*code.google.com/webstats/2005-12/classes.html*). Curiously, the spec lists the old **`address`** element as a section as well, so we'll look at that one here too.

The elements discussed in this section are well supported by current desktop and mobile browsers, but there is a snag with Internet Explorer versions 8 and earlier. See the sidebar HTML5 Support in Internet Explorer for details on a workaround.

HTML5 Support in Internet Explorer

Most browsers today support the new HTML5 semantic elements, and for those that don't, creating a style sheet rule that tells browsers to format each one as a block-level element is all that is needed to make them behave correctly.

```
section, article, nav, aside, header, footer,
hgroup { display: block; }
```

Unfortunately, that fix won't work with Internet Explorer versions 8 and earlier (versions 9 and later are fine). Not only do early IE browsers not recognize the elements, they also ignore any styles applied to them. The solution is to use JavaScript to create each element so IE knows it exists and will allow nesting and styling. Here's what a JavaScript command creating the **section** element looks like:

```
document.createElement("section");
```

Fortunately, Remy Sharp created a script that creates all of the HTML5 elements for IE in one fell swoop. It is called "HTML5 Shiv" (or Shim) and it lives on a Google-run server, so you can just point to it in your documents. To make sure the new HTML5 elements work in IE8 and earlier, copy this code in the **head** of your document and use a style sheet to style the new elements as blocks:

```
<!--[if lt IE 9]>
<script src="http://html5shiv.googlecode.com/svn/
trunk/html5-els.js"></script>
<![endif]-->
```

Find out more about the HTML5 Shiv here: *html5doctor.com/how-to-get-html5-working-in-ie-and-firefox-2/*.

The HTML5 Shiv is also part of the Modernizr polyfill script that adds HTML5 and CSS3 functionality to older non-supporting browsers. Read more about it online at *modernizr.com*. It is also discussed in Chapter 20, Using JavaScript.

Sections and articles

`<section>…</section>`

Thematic group of content

NEW IN HTML5

`<article>…</article>`

Self-contained, reusable composition

NEW IN HTML5

NOTE

The HTML5 spec recommends that if the purpose for grouping the elements is simply to provide a hook for styling, use the generic `div` *element instead.*

Long documents are easier to use when they are divided into smaller parts. For example, books are divided into chapters, and newspapers have sections for local news, sports, comics, and so on. To divide long web documents into thematic sections, use the aptly named **section** element. Sections typically have a heading (inside the **section** element) and any other content that has a meaningful reason to be grouped together.

The **section** element has a broad range of uses, from dividing a whole page into major sections or identifying thematic sections within a single article. In the following example, a document with information about typography resources has been divided into two sections based on resource type.

```
<section>
  <h2>Typography Books</h2>
  <ul>
     <li>…</li>
  </ul>
</section>

<section>
  <h2>Online Tutorials</h2>
  <p>These are the best tutorials on the web.</p>
  <ul>
    <li>…</li>
  </ul>
</section>
```

Use the **article** element for self-contained works that could stand alone or be reused in a different context (such as syndication). It is useful for magazine or newspaper articles, blog posts, comments, or other items that could be extracted for external use. You can think of it as a specialized section element that answers the question "Could this appear on another site and make sense?" with "yes."

To make things interesting, a long **article** could be broken into a number of sections, as shown here:

```
<article>
  <h1>Get to Know Helvetica</h1>
  <section>
    <h2>History of Helvetica</h2>
    <p>…</p>
  </section>

  <section>
    <h2>Helvetica Today</h2>
    <p>…</p>
  </section>
</article>
```

Conversely, a **section** in a web document might be comprised of a number of articles.

```
<section id="essays">
  <article>
    <h1>A Fresh Look at Futura</h1>
    <p>…</p>
  </article>

  <article>
    <h1>Getting Personal with Humanist</h1>
    <p>…</p>
  </article>
</section>
```

The **section** and **article** elements are easily confused, particularly because it is possible to nest one in the other and vice versa. Keep in mind that if the content is self-contained and could appear outside the current context, it is best marked up as an **article**.

Sectioning Elements

Another thing that **section** and **article** have in common "under the hood" is that both are what HTML5 calls sectioning elements. When a browser runs across a sectioning element in the document, it creates a new item in the document's outline automatically. In prior HTML versions, only headings (**h1**, **h2**, etc.) triggered new outline items. The new **nav** (primary navigation) and **aside** (for sidebar-like information) are also sectioning elements.

In the new HTML5 outlining system, a sectioning element may have its own internal heading hierarchy, starting with **h1**, regardless of its position in the document that contains it. That makes it possible to take an **article** element with its internal outline, place it in another document flow, and know that it won't break the host document's outline. The goal of the new outlining algorithm is to make the markup meet the needs of content use and reuse on the modern Web.

As of this writing, no browsers support the HTML5 outlining system, so to make your documents accessible and logically structured for all users, it is safest to use headings in descending numerical order, even within sectioning elements.

For more information, I recommend the HTML5 Doctor article "Document Outlines," by Mike Robinson, that tackles HTML5 outlines in more detail than I am able to squeeze in here (*html5doctor.com/outlines/*).

In addition, Roger Johansson's article "HTML5 Sectioning Elements, Headings, and Document Outlines" describes some potential gotchas when working with sectioning elements (*www.456bereastreet.com/archive/201103/html5_sectioning_elements_headings_and_document_outlines/*).

Aside (sidebars)

`<aside>…<aside>`

Tangentially related material

NEW IN HTML5

The **aside** element identifies content that is related but tangential to the surrounding content. In print, its equivalent is a sidebar, but they couldn't call the element **sidebar**, because putting something on the "side" is a presentational description, not semantic. Nonetheless, a sidebar is a good mental model for using the **aside** element. **aside** can be used for pull quotes, background information, lists of links, callouts, or anything else that might be associated with (but not critical to) a document.

In this example, an **aside** element is used for a list of links related to the main article.

```
<h1>Web Typography</h1>
<p>Back in 1997, there were competing font formats and tools for making
them…</p>
<p>We now have a number of methods for using beautiful fonts on web
pages…</p>
<aside>
  <h2>Web Font Resources</h2>
  <ul>
  <li><a href="http://typekit.com/">Typekit</a></li>
  <li><a href="http://www.google.com/webfonts">Google Fonts</a></li>
  </ul>
</aside>
```

The **aside** element has no default rendering, so you will need to make it a block element and adjust its appearance and layout with style sheet rules.

Navigation

`<nav>…</nav>`

Primary navigation links

NEW IN HTML5

The new **nav** element gives developers a semantic way to identify navigation for a site. Earlier in this chapter, we saw an unordered list that might be used as the top-level navigation for a font catalog site. Wrapping that list in a **nav** element makes its purpose explicitly clear.

```
<nav>
<ul>
  <li><a href="">Serif</a>/li>
  <li><a href="">Sans-serif</a></li>
  <li><a href="">Script</a></li>
  <li><a href="">Display</a></li>
  <li><a href="">Dingbats</a>/li>
</ul>
</nav>
```

Not all lists of links should be wrapped in **nav** tags, however. The spec makes it clear that it should be used for links that provide primary navigation around a site or a lengthy section or article.

The **nav** element may be especially helpful from an accessibility perspective. Once screen readers and other devices become HTML5-compatible, users can easily get to or skip navigation sections without a lot of hunting around.

Headers and footers

Because web authors have been labeling header and footer sections in their documents for years, it was kind of a no-brainer that full-fledged **header** and **footer** elements would come in handy. Let's start with headers.

`<header>…</header>`

Introductory material for page, section, or article

NEW IN HTML5

`<footer>…</footer>`

Footer for page, section, or article

NEW IN HTML5

Headers

The **header** element is used for introductory material that typically appears at the beginning of a web page or at the top of a section or article. There is no specified list of what a **header** must or should contain; anything that makes sense as the introduction to a page or section is acceptable. In the following example, the document header includes a logo image, the site title, and navigation.

```
<header>
  <img src="/images/logo.png">
  <hgroup>
  <h1>Nuts about Web Fonts</h1>
  <h2>News from the Web Typography Front</h2>
  </hgroup>
  <nav>
    <ul>
      <li><a href="">Home</a></li>
      <li><a href="">Blog</a></li>
      <li><a href="">Shop</a></li>
    </ul>
  </nav>
</header>

… page content …
```

When used in an individual article, the **header** might include the article title, author, and the publication date, as shown here:

```
<article>
  <header>
    <h1>More about WOFF</h1>
    <p>by Jennifer Robbins, <time datetime="11-11-2011"
    pubdate>November 11, 2011</time></p>
  </header>
  <p>...article content starts here…</p>
</article>
```

WARNING

Neither `header` *nor* `footer` *elements are permitted to contain nested* `header` *or* `footer` *elements.*

Footers

The **footer** element is used to indicate the type of information that typically comes at the end of a page or an article, such as its author, copyright information, related documents, or navigation. The **footer** element may apply to the entire document, or it could be associated with a particular section or article. If the footer is contained directly within the **body** element, either before or after all the other **body** content, then it applies to the entire page or application. If it is contained in a sectioning element (**section**, **article**, **nav**, or **aside**), it is parsed as the footer for just that section. Note that although it is called "footer," there is no requirement that it come last in the docu-

ment or sectioning element. It could also appear at or near the beginning if it makes semantic sense.

NOTE

You can also add headers and footers to sectioning root elements: `body`, `blockquote`, `details`, `figure`, `td`, *and* `fieldset`.

In this simple example we see the typical information listed at the bottom of an article or blog post marked up as a **`footer`**.

```
<article>
  <header>
    <h1>More about WOFF</h1>
    <p>by Jennifer Robbins, <time datetime="11-11-2011"
    pubdate>November 11, 2011</time></p>
  </header>
  <p>...article content starts here…</p>
  <footer>
    <p><small>Copyright &copy;2012 Jennifer Robbins.</small></p>
    <nav>
    <ul>
      <li><a href="">Previous</a></li>
      <li><a href="">Next</a></li>
    </ul>
    </nav>
  </footer>
</article>
```

Addresses

`<address>...</address>`

Contact information

Last, and well, least, is the **`address`** element that is used to create an area for contact information for the author or maintainer of the document. It is generally placed at the end of the document or in a section or article within a document. An **`address`** would be right at home in a **`footer`** element.

It is important to note that the **`address`** element should *not* be used for any old address on a page, such as mailing addresses. It is intended specifically for author contact information (although that could potentially be a mailing address). Following is an example of its intended use. The "a href" parts are the markup for links...we'll get to those in Chapter 6, Adding Links.

NOTE

You'll get a chance to try out the section elements in Exercise 5-3 at the end of this chapter.

```
<address>
Contributed by <a href="../authors/robbins/">Jennifer Robbins</a>,
<a href="http://www.oreilly.com/">O'Reilly Media</a>
</address>
```

The Inline Element Roundup

Now that we've identified the larger chunks of content, we can provide semantic meaning to phrases within the chunks using what HTML5 calls text-level semantic elements. On the street, you are likely to hear them called inline elements because they display in the flow of text by default and do not cause any line breaks. That's also how they were referred to in HTML versions prior to HTML5.

Text-level (inline) elements

Despite all the types of information you could add to a document, there are only a couple dozen text-level semantic elements in HTML5. Table 5-1 lists all of them.

Table 5-1. Text-level semantic elements

Element	Description
a	An anchor or hypertext link (see Chapter 6 for details)
abbr	Abbreviation
b	Added visual attention, such as keywords (bold)
bdi	NEW IN HTML5 Indicates text that may have directional requirements
bdo	Bidirectional override; explicitly indicates text direction (left to right, ltr, or right to left, rtl)
br	Line break
cite	Citation; a reference to the title of a work, such as a book title
code	Computer code sample
data	WHATWG ONLY Machine-readable equivalent dates, time, weights, and other measurable values
del	Deleted text; indicates an edit made to a document
dfn	The defining instance or first occurrence of a term
em	Emphasized text
i	Alternative voice (italic)
ins	Inserted text; indicates an insertion in a document
kbd	Keyboard; text entered by a user (for technical documents)
mark	NEW IN HTML5 Contextually relevant text
q	Short, inline quotation
ruby, rt, rp	NEW IN HTML5 Provides annotations or pronunciation guides under East Asian typography and ideographs
s	Incorrect text (strike-through)
samp	Sample output from programs
small	Small print, such as a copyright or legal notice (displayed in a smaller type size)
span	Generic phrase content
strong	Content of strong importance
sub	Subscript
sup	Superscript
time	NEW IN HTML5 Machine-readable time data
u	Underlined
var	A variable or program argument (for technical documents)
wbr	Word break

The Inline Elements Backstory

Many of the inline elements that have been around since the dawn of the Web were introduced to change the visual formatting of text selections due to the lack of a style sheet system. If you wanted bolded text, you marked it as **b**. Italics? Use the **i** element. In fact, there was once a **font** element used solely to change the font, color, and size of text (the horror!). Not surprisingly, HTML5 kicked the purely presentational **font** element to the curb. However, many of the old-school presentational inline elements (for example, **u** for underline and **s** for strike-through) have been kept in HTML5 and given new semantic definitions (**b** is now for "keywords," **s** for "inaccurate text").

Some inline elements are purely semantic (such as **abbr** or **time**) and don't have default renderings. For these, you'll need to use a CSS rules if you want to change the way they display.

In the element descriptions in this section, I'll provide both the definition of the inline elements and the expected browser default rendering if there is one.

Obsolete HTML 4.01 Text Elements

HTML5 finally retired many elements that were marked as deprecated (phased out and discouraged from use) in HTML 4.01. For the sake of thoroughness, I include them here in case you run across them in legacy markup. But there's no reason to use them—most have analogous style sheet properties or are simply poorly supported.

Element	Description
`acronym`	Indicates an acronym (e.g., NASA); authors should use **`abbr`** instead
`applet`	Inserts a Java applet
`basefont`	Establishes default font settings for a document
`big`	Makes text slightly larger than default text size
`center`	Centers content horizontally
`dir`	Directory list (replaced by unordered lists)
`font`	Font face, color, and size
`isindex`	Inserts a search box
`menu`	Menu list (replaced by unordered lists; however, **`menu`** is now used to provide contextual menu commands)
`strike`	Strike-through text
`tt`	Teletype; displays in constant-width font

Emphasized text

`<em>...</em>`
Stressed emphasis

Use the **`em`** element to indicate which part of a sentence should be stressed or emphasized. The placement of **`em`** elements affects how a sentence's meaning is interpreted. Consider the following sentences that are identical, except for which words are stressed.

```
<p><em>Matt</em> is very smart.</p>
<p>Matt is <em>very</em> smart.</p>
```

The first sentence indicates *who* is very smart. The second example is about *how* smart he is.

Emphasized text (**`em`**) elements nearly always display in italics by default (Figure 5-9), but of course you can make them display any way you like with a style sheet. Screen readers may use a different tone of voice to convey stressed content, which is why you should use an **`em`** element only when it makes sense semantically, not just to achieve italic text.

Important text

`<strong>...</strong>`
Strong importance

The **`strong`** element indicates that a word or phrase as important. In the following example, the **`strong`** element identifies the portion of instructions that requires extra attention.

```
<p>When checking out of the hotel, <strong>drop the keys in the red box
by the front desk</strong>.</p>
```

Visual browsers typically display **strong** text elements in bold text by default. Screen readers may use a distinct tone of voice for important content, so mark text as **strong** only when it makes sense semantically, not just to make text bold.

The following is a brief example of our **em** and **strong** text examples. Figure 5-9 should hold no surprises.

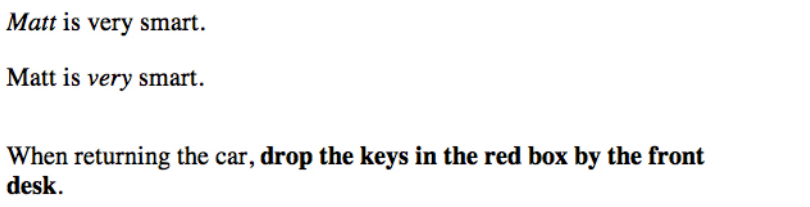


Figure 5-9. The default rendering of emphasized and strong text.

The previously presentational elements that are sticking around in HTML5 with fancy new semantic definitions

As long as we're talking about bold and italic text, let's see what the old **b** and **i** elements are up to now. The elements **b**, **i**, **u**, **s**, and **small** were introduced in the old days of the Web as a way to provide typesetting instructions (bold, italic, underline, strikethrough, and smaller text, respectively). Despite their original presentational purposes, these elements have been included in HTML5 and given updated, semantic definitions based on patterns of how they've been used. Browsers still render them by default as you'd expect (Figure 5-10). However, if a type style change is all you're after, using a style sheet rule is the appropriate solution. Save these for when they are semantically appropriate.

Let's look at these elements and their correct usage, as well as the style sheet alternatives.

`<b>...</b>`
Keywords or visually emphasized text (bold)

`<i>...</i>`
Alternative voice (italic)

`<s>...</s>`
Incorrect text (strike-through)

`<u>...</u>`
Annotated text (underline)

`<small>...</small>`
Legal text; small print (smaller type size)

b

HTML 4.01 definition: Bold

HTML5 definition: Keywords, product names, and other phrases that need to stand out from the surrounding text without conveying added importance or emphasis.

CSS alternative: For bold text, use **font-weight**. Example: **font-weight: bold**

Example: `<p>The slabs at the ends of letter strokes are called <b>serifs</b>.</p>`

NOTE

It helps me to think about how a screen reader would read the text. If I don't want the word read in a loud, emphatic tone of voice, but it really should be bold, then **b** *may be more appropriate than* **strong**.

i

HTML 4.01 definition: Italic

HTML5 definition: Indicates text that is in a different voice or mood than the surrounding text, such as a phrase from another language, a technical term, or thought.

CSS alternative: For italic text, use **font-style**. Example: **font-style: italic**

Example:
```
<p>Simply change the font and <i>Voila!</i>, a new personality.</p>
```

s

HTML 4.01 definition: Strike-through text

HTML5 definition: Indicates text that is incorrect.

CSS Property: To put a line through a text selection, use **text-decoration**. Example: **text-decoration: line-through;**

Example:
```
<p>Scala Sans was designed by <s>Eric Gill</s> Martin Majoor.</p>
```

u

HTML 4.01 definition: Underline

HTML5 definition: There are a few instances when underlining has semantic significance, such as underlining a formal name in Chinese or indicating a misspelled word after a spell check. Note that underlined text is easily confused as a link and should generally be avoided except for a few niche cases.

CSS Property: For underlined text, use **text-decoration**. Example: **text-decoration: underline**

Example:
```
<p>New York subway signage is set in <u>Halvetica</u>.<p>
```

small

HTML 4.01 definition: Renders in font smaller than the surrounding text

HTML5 definition: Indicates an addendum or side note to the main text, such as the legal "small print" at the bottom of a document.

CSS Property: To make text smaller, use **font-size**. Example: **font-size: 80%**

Example:
```
<p><a href="">Download <b>Jenville Handwriting Font</b></a></p>

<p><small>This font is free for commercial use.</small></p>
```

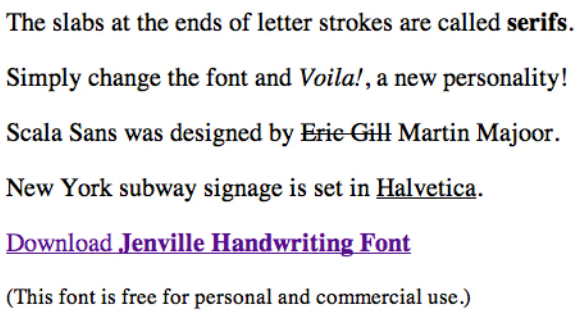


Figure 5-10. The default rendering of `b`, `i`, `u`, `s`, *and* `small` *elements.*

Short quotations

Use the quotation (`q`) element to mark up short quotations, such as "To be or not to be," in the flow of text, as shown in this example (Figure 5-11).

```
Matthew Carter says, <q>Our alphabet hasn't changed in eons.</q>
```

`<q>...</q>`

Short inline quotation

According to the HTML spec, browsers should add quotation marks around `q` elements automatically, so you don't need to include them in the source document. And for the most part they do, with the exception of Internet Explorer versions 7 and earlier. Fortunately, as of this writing, those browsers make up only 5–8% of browser usage, and it's sure to be significantly less by the time you read this. If you are concerned about a small percentage of users seeing quotations without their marks, stick with using quotation marks in your source, a fine alternative.

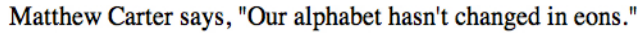


Figure 5-11. Nearly all browsers add quotation marks automatically around `q` *elements.*

REMINDER

Nesting Elements

You can apply two elements to a string of text (for example, a phrase that is both a quote and in another language), but be sure they are nested properly. That means the inner element, including its closing tag, must be completely contained within the outer element, and not overlap.

```
<q><i>Je ne sais pas.</i></q>
```

Abbreviations and acronyms

Marking up acronyms and abbreviations with the `abbr` element provides useful information for search engines, screen readers, and other devices. Abbreviations are shortened versions of a word ending in a period (Conn. for Connecticut, for example). Acronyms are abbreviations formed by the first letters of the words in a phrase (such as WWW or USA). The `title` attribute provides the long version of the shortened term, as shown in this example:

```
<abbr title="Points">pts.</abbr>
<abbr title="American Type Founders">ATF</abbr>
```

`<abbr>...</abbr>`

Abbreviation or acronym

NOTE

In HTML 4.01, there was an `acronym` *element especially for acronyms, but it has been made obsolete in HTML5 in favor of using the* `abbr` *for both.*

Citations

`<cite>...</cite>`
Citation

The **cite** element is used to identify a reference to another document, such as a book, magazine, article title, and so on. Citations are typically rendered in italic text by default. Here's an example:

```
<p>Passages of this article were inspired by <cite>The Complete Manual
of Typography</cite> by James Felici.</p>
```

Defining terms

`<dfn>...</dfn>`
Defining term

It is common to point out the first and defining instance of a word in a document in some fashion. In this book, defining terms are set in blue text. In HTML, you can identify them with the **dfn** element and format them visually using style sheets.

```
<p><dfn>Script typefaces</dfn> are based on handwriting.</p>
```

Program code elements

`<code>...</code>`
Code

`<var>...</var>`
Variable

`<samp>...</samp>`
Program sample

`<kbd>...</kbd>`
User-entered keyboard strokes

A number of inline elements are used for describing the parts of technical documents, such as code (**code**), variables (**var**), program samples (**samp**), and user-entered keyboard strokes (**kbd**). For me, it's a quaint reminder of HTML's origins in the scientific world (Tim Berners-Lee developed HTML to share documents at the CERN particle physics lab in 1989).

Code, sample, and keyboard elements typically render in a constant-width (also called monospace) font such as Courier by default. Variables usually render in italics.

Subscript and superscript

`<sub>...</sub>`
Subscript

`<sup>...</sup>`
Superscript

The subscript (**sub**) and superscript (**sup**) elements cause the selected text to display in a smaller size, positioned slightly below (**sub**) or above (**sup**) the baseline. These elements may be helpful for indicating chemical formulas or mathematical equations.

Figure 5-12 shows how these examples of subscript and superscript typically render in a browser.

```
<p>H<sub>2</sub>O</p>

<p>E=MC<sup>2</sup></p>
```

H_2O $E=MC^2$

Figure 5-12. Subscript and superscript

Highlighted text

The new **mark** element indicates a word that may be considered especially relevant to the reader. One might use it to call out a search term in a page of results, to manually call attention to a passage of text, indicate the current page in a series. Some designers (and browsers) give marked text a light colored background as though it was marked with a highlighter marker, as shown in Figure 5-13.

```
<p> ... PART I. ADMINISTRATION OF THE GOVERNMENT. TITLE IX.
TAXATION. CHAPTER 65C. MASS. <mark>ESTATE TAX</mark>. Chapter 65C:
Sect. 2. Computation of <mark>estate tax</mark>.</p>
```

... PART I. ADMINISTRATION OF THE GOVERNMENT. TITLE IX. TAXATION. CHAPTER 65C. MASS. ESTATE TAX. Chapter 65C: Sect. 2. Computation of estate tax.

Figure 5-13. Search terms are marked as mark elements and given a yellow background with a style sheet so they are easier for the reader to find.

`<mark>...</mark>`

Contextually relevant text

NEW IN HTML5

SUPPORT ALERT

The **mark** *element is not supported in Internet Explorer versions 8 and earlier (see the sidebar HTML5 Support in Internet Explorer earlier in this chapter for a workaround). Older versions of Firefox and Safari (prior to 3.6 and 4, respectively) do not support it according to the spec, but do allow you to apply styles to it.*

Times and machine-readable information

When we look at the phrase "noon on November 4," we know that it is a date and a time. But the context might not be so obvious to a computer program. The **time** element allows us to mark up dates and times in a way that is comfortable for a human to read, but also encoded in a standardized way that computers can use. The content of the element presents the information to people, and the **datetime** attribute presents the same information in a machine-readable way.

The **time** element indicates dates, times, or date-time combos. It might be used to pass the date and time information to an application, such as saving an event to a personal calendar. It might be used by search engines to find the most recently published articles. Or it could be used to restyle time information into an alternate format (e.g., changing 18:00 to 6 p.m.).

The **datetime** attribute specifies the date and/or time information in a standardized time format illustrated in Figure 5-14. It begins with the date (year, month, day), followed by the letter T to indicate time, listed in hours, minutes, seconds (optional), and milliseconds (also optional). Finally, the time zone is indicated by the number of hours behind (–) or ahead (+) of Greenwich Mean Time (GMT). For example, "–05:00" indicates the Eastern Standard time zone, which is five hours behind GMT.

`<time>...</time>`

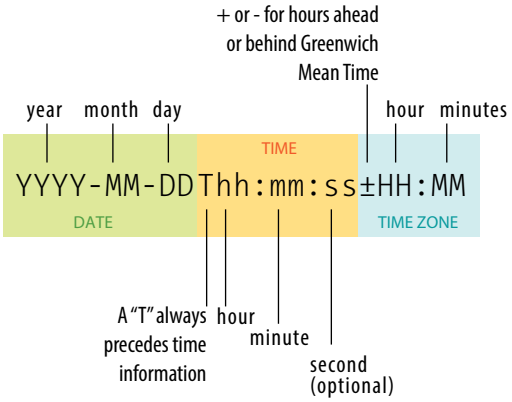
Time data

NEW IN HTML5

NOTE

The **time** *element is not intended for marking up times for which a precise time or date cannot be established, such as "the end of last year" or "the turn of the century."*

Figure 5-14. Standardized date and time syntax.



Example:

3pm PST on December 25, 2012

```
2012-12-25T15:00-8:00
```

The WHATWG HTML specification includes a **pubdate** attribute for indicating that the time is the publication date of a document, as shown in this example. The **pubdate** attribute is not included in the W3C HTML5 spec as of this writing, but it may be included at a later date if it becomes widely used.

```
Written by Jennifer Robbins (<time datetime="2012-09-01T 20:00-05:00"
pubdate>September 1, 2012, 8pm EST</time>)
```

`<data>...</data>`

Machine-readable data

WHATWG ONLY

The WHATWG also includes the **data** element for helping computers make sense of content, which can be used for all sorts of data, including dates, times, measurements, weights, and so on. It uses the **value** attribute for the machine-readable information. Here are a couple of examples:

```
<data value="12">Twelve</data>
<data value="2011-11-12">Last Saturday</data>
```

SUPPORT ALERT

Both `time` *and* `data` *are new elements and are not universally supported as of this writing. However, you can apply styles to them and they will be recognized by browsers other than IE8 and earlier.*

I'm not going to go into more detail on the **data** element, because as of this writing, the powers that be are still discussing exactly how it should work, and the W3C has not adopted it for the HTML5 spec. Also, as a beginner, you are unlikely to be dealing with machine-readable data yet anyway. But still, it is interesting to see how markup can be used to provide usable information to computer programs and scripts as well as to your fellow humans.

Inserted and deleted text

`<ins>...</ins>`

Inserted text

`<del>...</del>`

Deleted text

The **ins** and **del** elements are used to mark up edits indicating parts of a document that have been inserted or deleted (respectively). These elements rely on style rules for presentation (i.e., there is no dependable browser default). Both the **ins** and **del** elements can contain either inline or block elements, depending on what type of content they contain.

```
Chief Executive Officer: <del title="retired">Peter Pan</del><ins>Pippi
Longstockings</ins>
```

Adding Breaks

Line breaks

`<br>`

Line break

Occasionally, you may need to add a line break within the flow of text. We've seen how browsers ignore line breaks in the source document, so we need a specific directive to tell the browser to "add a line break here."

The inline line break element (**br**) does exactly that. The **br** element could be used to break up lines of addresses or poetry. It is an empty element, which means it does not have content. Just add the **br** element (**
** in XHTML) in the flow of text where you want a break to occur, as shown in here and in Figure 5-15.

```
<p>So much depends <br>upon <br><br>a red wheel <br>barrow</p>
```

So much depends
upon

a red wheel
barrow

Figure 5-15. Line breaks are inserted at each `br` *element.*

Accommodating Non-Western Languages

Because the Web is "world-wide," there are a few elements designed to address the needs of non-western languages.

Changing direction

The **bdo** (bidirectional override) element allows a phrase in a right-to-left (**rtl**) reading language (such as Hebrew or Arabic) to be included in a left-to-right (**ltr**) reading flow, or vice versa.

```
This is how you write Shalom: <bdo dir="rtl">&#x0
5E9;&#x05DC;&#x05D5;&#x05DD;</bdo>
```

The **bdi** (bidirectional isolation) element is similar, but it is used to isolate a selection that *might* read in a different direction, such as a name or comment added by a user.

Hints for East Asian languages

HTML5 also includes the **ruby**, **rt**, and **rp** elements used to add ruby annotation to East Asian languages. Ruby annotations are little notes that typically appear above ideographs and provide pronunciation clues or translations. Within the **ruby** element, the **rt** element indicates the helpful ruby text. Browsers that support ruby text typically display it in a smaller font above the main text. As a backup for browsers that don't support ruby, you can put the ruby text in parentheses, each marked with the **rp** element. Non-supporting browsers display all the text on the same line, with the ruby in parentheses. Supporting browsers ignore the content of the **rp** elements and display only the **rt** text above the glyphs. The Ruby system has spotty browser support as of this writing.

```
<ruby>
 字<rp>(</rp><rt>han</rt><rp>)<rp>
 汉<rp>(</rp><rt>zi</rt><rp>)<rp>
</ruby>
```

This example was taken from the HTML5 Working Draft at whatwg.com, used with permission under an MIT License.

Unfortunately, the **br** element is easily abused (see the following warning). Consider whether using the CSS **white-space** property (introduced in Chapter 12, Formatting Text) might be a better alternative for maintaining line breaks from your source without extra markup.

Word breaks

<wbr>

Word break

The word break (**wbr**) element lets you mark the place where a word should break if it needs to (a "line break opportunity" according to the spec). It takes some of the guesswork away from the browser and allows authors to specify the best spot for the word to be split over two lines. Keep in mind that the word breaks at the **wbr** element only if it needs to (Figure 5-16). If there is enough room, the word stays in one piece. Browsers have supported this element for a long time, but it has recently been incorporated into the HTML standard.

```
<p>The biggest word you've ever heard and this is how it goes:
<em>supercali<wbr>fragilistic<wbr>expialidocious</em>!</p>
```

The biggest word you've ever heard and this is how it goes: *supercalifragilistic expialidocious*!

Figure 5-16. When there is not enough room for a word to fit on a line, it will break at the location of the **wbr** *element.*

WARNING

Be careful that you aren't using **br** *elements to force breaks into text that really ought to be a list. For example, don't do this:*

```
<p>Times<br>
Georgia<br>
Garamond
</p>
```

If it's a list, use the semantically correct unordered list element instead, and turn off the bullets with style sheets.

```
<ul>
  <li>Times</li>
  <li>Georgia</li>
  <li>Garamond</li>
</ul>
```

exercise 5-2 | Identifying inline elements

This little post for the Black Goose Bistro blog will give you an opportunity to identify and mark up a variety of inline elements. See if you can find phrases to mark up accurately with the following elements:

`b` `br` `cite` `dfn` `em` `i` `q` `small` `time`

Because markup is always somewhat subjective, your resulting markup may not look exactly like the example in Appendix A, but there is an opportunity to use all of the elements listed above in the article. For extra credit, there is a phrase that should have two elements applied to it (remember to nest them properly by closing the inner element before you close the outer one).

You can write the tags right on this page. Or, if you want to use a text editor and see the results in a browser, this text file is available online at *www.learningwebdesign.com/4e/materials*. The resulting code appears in Appendix A.

```
<article>
   <header>
   <p>posted by BGB, November 15, 2012</p>
   </header>
<h1>Low and Slow</h1>
<p>This week I am extremely excited about a new cooking technique
called sous vide. In sous vide cooking, you submerge the food
(usually vacuum-sealed in plastic) into a water bath that is
precisely set to the target temperature you want the food to be
cooked to. In his book, Cooking for Geeks, Jeff Potter describes
it as ultra-low-temperature poaching.</p>
<p>Next month, we will be serving Sous Vide Salmon with Dill
Hollandaise. To reserve a seat at the chef table, contact us
before November 30.</p>

<p>blackgoose@example.com
555-336-1800</p>

<p> Warning: Sous vide cooked salmon is not pasteurized. Avoid it
if you are pregnant or have immunity issues.</p>
</article>
```

Generic Elements (div and span)

`<div>...</div>`
Generic block-level element

`<span>...</span>`
Generic inline element

What if none of the elements we've talked about so far accurately describes your content? After all, there are endless types of information in the world, but as you've seen, not all that many semantic elements. Fortunately, HTML provides two generic elements that can be customized to describe your content perfectly. The **div** element indicates a division of content, and **span** indicates a word or phrase for which no text-level element currently exists. The generic elements are given meaning and context with the **id** and **class** attributes, which we'll discuss in a moment.

The **div** and **span** elements have no inherent presentation qualities of their own, but you can use style sheets to format them however you like. In fact, generic elements are a primary tool in standards-based web design because they enable authors to accurately describe content and offer plenty of "hooks" for adding style rules. They also allow elements on the page to be accessed and manipulated by JavaScript.

We're going to spend a little time on **div** and **span** (as well as the **id** and **class** attributes) and learn how authors use them to structure content.

Divide it up with a div

The **div** element is used to create a logical grouping of content or elements on the page. It indicates that they belong together in some sort of conceptual unit or should be treated as a unit by CSS or JavaScript. By marking related content as a **div** and giving it a unique **id** identifier or indicating that it is part of a **class**, you give context to the elements in the grouping. Let's look at a few examples of **div** elements.

MARKUP TIP

It is possible to nest **div** elements within other **div** elements, but don't go overboard. You should always strive to keep your markup as simple as possible, so add a **div** element only if it is necessary for logical structure, styling, or scripting.

In this example, a **div** element is used as a container to group an image and two paragraphs into a product "listing."

```
<div class="listing">
  <img src="felici-cover.gif" alt="">
  <p><cite>The Complete Manual of Typography</cite>, James Felici</p>
  <p>A combination of type history and examples of good and bad type
design.</p>
</div>
```

By putting those elements in a **div**, I've made it clear that they are conceptually related. It will also allow me to style two **p** elements within listings differently than other paragraphs on the page.

Here is another common use of a **div** used to break a page into sections for layout purposes. In this example, a heading and several paragraphs are enclosed in a **div** and identified as the "news" division.

```
<div id="news">
  <h1>New This Week</h1>
  <p>We've been working on...</p>
  <p>And last but not least,... </p>
</div>
```

Now that I have an element known as "news," I could use a style sheet to position it as a column to the right or left of the page. You might be thinking, "Hey Jen, couldn't you use a **section** element for that?" You could! In fact, authors may turn to generic **div**s less now that we have better semantic grouping elements in HTML5.

Get inline with span

A **span** offers the same benefits as the **div** element, except it is used for phrase elements and does not introduce line breaks. Because spans are inline elements, they can only contain text and other inline elements (in other words, you cannot put headings, lists, content-grouping elements, and so on, in a **span**). Let's get right to some examples.

There is no **telephone** element, but we can use a **span** to give meaning to telephone numbers. In this example, each telephone number is marked up as a **span** and classified as "tel":

```
<ul>
  <li>John: <span class="tel">999.8282</span></li>
  <li>Paul: <span class="tel">888.4889</span></li>
  <li>George: <span class="tel">888.1628</span></li>
  <li>Ringo: <span class="tel">999.3220</span></li>
</ul>
```

You can see how the classified spans add meaning to what otherwise might be a random string of digits. As a bonus, the **span** element enables us to apply the same style to phone numbers throughout the site (for example, ensuring line breaks never happen within them, using a CSS **white-space: nowrap** declaration). It makes the information recognizable not only to humans but to computer programs that know that "tel" is telephone number information. In fact, some values—including "tel"—have been standardized in a markup system known as Microformats that makes web content more useful to software (see the Microformats and Metadata sidebar).

id and class attributes

In the previous examples, we saw the **id** and **class** attributes used to provide context to generic **div** and **span** elements. **id** and **class** have different purposes, however, and it's important to know the difference.

Identification with id

The **id** attribute is used to assign a *unique* identifier to an element in the document. In other words, the value of **id** must be used only once in the document. This makes it useful for assigning a name to a particular element, as though it were a piece of data. See the sidebar id and class Values for information on providing values for the **id** attribute.

This example uses the books' ISBN numbers to uniquely identify each listing. No two book listings may share the same **id**.

id and class Values

The values for **id** and **class** attributes should start with a letter (A–Z or a–z) or underscore (although Internet Explorer 6 and earlier have trouble with underscores, so they are generally avoided). They should not contain any character spaces or special characters. Letters, numbers, hyphens, underscores, colons, and periods are OK. Also, the values are case-sensitive, so "sectionB" is not interchangeable with "Sectionb."

```
<div id="ISBN0321127307">
  <img src="felici-cover.gif" alt="">
  <p><cite>The Complete Manual of Typography</cite>, James Felici</p>
  <p>A combination of type history and examples of good and bad type.
  </p>
</div>

<div id="ISBN0881792063">
  <img src="bringhurst-cover.gif" alt="">
  <p><cite>The Elements of Typographic Style</cite>, Robert Bringhurst
  </p>
  <p>This lovely, well-written book is concerned foremost
   with creating beautiful typography.</p>
</div>
```

Web authors also use **id** when identifying the various sections of a page. In the following example, there may not be more than one element with the **id** of "main," "links," or "news" in the document.

```
<section id="main">
  <!-- main content elements here -->
</section>

<section id="news">
  <!-- news items here -->
</section>

<aside id="links">
  <!-- list of links here -->
</aside>
```

Not Just for divs

The **id** and **class** attributes may be used with all elements in HTML5, not just **div** and **span**. For example, you could identify an ordered list as "directions" instead of wrapping it in a **div**.

```
<ol id="directions">
  <li>...</li>
  <li>...</li>
  <li>...</li>
</ol>
```

Note that in HTML 4.01, **id** and **class** may be used with all elements except **base**, **basefont**, **head**, **html**, **meta**, **param**, **script**, **style**, and **title**.

Microformats and Metadata

As you've seen, the elements in HTML fall short in describing every type of information. A group of developers decided that if **class** names could be standardized (for example, always using "tel" for telephone numbers), they could establish systems for describing data to make it more useful. This system is called Microformats. Microformats extend the semantics of HTML markup by establishing standard *values* for **id**, **class**, and **rel** attributes rather than creating whole new elements.

There are several Microformat "vocabularies" used to identify things such as contact information (hCard) or calendar items (hCalendar). The Microformats.org site is a good place to learn about them. To give you the general idea, the following example describes the parts of an event using the hCalendar Microformat vocabulary so the browser can automatically add it to your calendar program.

```
<section class="vevent">
 <span class="summary">O'Reilly Emerging
 Technology Conference</span>,
 <time class="dtstart" datetime="20110306">Mar 6
 </time> -
 <time class="dtend" datetime="20110310">10,
 2011</time>
 <div class="location">Manchester Grand Hyatt,
 San Diego, CA</div>
 <a class="url" href="http://events.example.com
 pub/e/403">Permalink</a>
</section>
```

The hCard vocabulary identifies components of typical contact information (stored in vCard format), including: address (**adr**), postal code (**postal-code**), states (**region**), and telephone numbers (**tel**), to name a few. The browser can then use a service to grab the information from the web page and automatically add it to an address book.

There is a lot more to say about Microformats than I can fit in this book. And not only that, but there are two additional, more complex systems for adding metadata to web pages in development at the W3C: RDFa and Microdata. It's not clear how they are all going to shake out in the long run, and I'm thinking that this metadata stuff is more than you want to take on right now anyway. But when you are ready to learn more, WebSitesMadeRight.com has assembled a great big list of introductory articles and tutorials on all three options: *websitesmaderight.com/2011/05/html5-microdata-microformats-and-rdfa-tutorials-and-resources/*.

Classification with class

The **class** attribute classifies elements into conceptual groups; therefore, unlike the **id** attribute, multiple elements may share a **class** name. By making elements part of the same class, you can apply styles to all of the labeled elements at once with a single style rule or manipulate them all with a script. Let's start by classifying some elements in the earlier book example. In this first example, I've added **class** attributes to classify each **div** as a "listing" and to classified paragraphs as "descriptions."

```
<div id="ISBN0321127307" class="listing">
  <header>
  <img src="felici-cover.gif" alt="">
  <p><cite>The Complete Manual of Typography</cite>, James Felici</p>
  </header>
  <p class="description">A combination of type history and examples of
good and bad type.</p>
</div>

<div id="ISBN0881792063" class="listing">
  <header>
  <img src="bringhurst-cover.gif" alt="">
  <p><cite>The Elements of Typographic Style</cite>, Robert Bringhurst
  </p>
  </header>
  <p class="description">This lovely, well-written book is concerned
foremost with creating beautiful typography.</p>
</div>
```

TIP

The **id** attribute is used to *identify.*
The **class** attribute is used to *classify.*

Notice how the same element may have both a **class** and an **id**. It is also possible for elements to belong to multiple classes. When there is a list of **class** values, simply separate them with character spaces. In this example, I've classified each **div** as a "book" to set them apart from possible "cd" or "dvd" listings elsewhere in the document.

```
<div id="ISBN0321127307" class="listing book">
  <img src="felici-cover.gif" alt="CMT cover">
  <p><cite>The Complete Manual of Typography</cite>, James Felici</p>
  <p class="description">A combination of type history and examples of
good and bad type.</p>
</div>

<div id="ISBN0881792063" class="listing book">
  <img src="bringhurst-cover.gif" alt="ETS cover">
  <p><cite>The Elements of Typographic Style</cite>, Robert Bringhurst
  </p>
  <p class="description">This lovely, well-written book is concerned
  foremost with creating beautiful typography.</p>
</div>
```

This should have given you a good introduction to how **div** and **span** elements with **class** and **id** attributes are used to add meaning and organization to documents. We'll work with them even more in the style sheet chapters in Part III.

Some Special Characters

There's just one more text-related topic before we close this chapter out.

Some common characters, such as the copyright symbol ©, are not part of the standard set of ASCII characters, which contains only letters, numbers, and a few basic symbols. Other characters, such as the less-than symbol (<), are available, but if you put one in an HTML document, the browser will interpret it as the beginning of a tag.

Characters such as these must be escaped in the source document. Escaping means that instead of typing in the character itself, you represent it by its numeric or named character reference. When the browser sees the character reference, it substitutes the proper character in that spot when the page is displayed.

There are two ways of referring to a specific character: by an assigned numeric value (numeric entity) or using a predefined abbreviated name for the character (called a named entity). All character references begin with an "&" and end with a ";".

Some examples will make this clear. I'd like to add a copyright symbol to my page. The typical Mac keyboard command, Option-G, which works in my word processing program, may not be understood properly by a browser or other software. Instead, I must use the named entity **©** (or its numeric equivalent, **©**) where I want the symbol to appear (Figure 5-17).

```
<p>All content copyright &copy; 2012, Jennifer Robbins</p>
```

or:

```
<p>All content copyright &#169; 2012, Jennifer Robbins</p>
```

HTML defines hundreds of named entities as part of the markup language, which is to say you can't make up your own entity. Table 5-2 lists some commonly used character references. If you'd like to see them all, the complete list of character references has been assembled online by the nice folks at the Web Standards Project at *www.webstandards.org/learn/reference/charts/entities/*.

All content copyright © 2007, Jennifer Robbins

Figure 5-17. The special character is substituted for the character reference when the document is displayed in the browser.

NOTE

In XHTML, every instance of an ampersand must be escaped so that it is not interpreted as the beginning of a character entity, even when it appears in the value of an attribute. For example:

```
<img src="sno.jpg" alt="Sifl
&amp; Olly Show" />
```

Non-breaking Spaces

One interesting character to know about is the non-breaking space (**` `**). Its purpose is to ensure that a line doesn't break between two words. So, for instance, if I mark up my name like this:

```
Jennifer Robbins
```

I can be sure that my first and last names will always stay together on a line.

Table 5-2. Common special characters and their character references

Character	Description	Name	Number
	Character space (nonbreaking space)	` `	` `
&	Ampersand	`&`	`&#038;`
‘	Apostrophe	`&apos;`	`&#039;`
<	Less-than symbol (useful for displaying markup on a web page)	`<`	`&#060;`
>	Greater-than symbol (useful for displaying markup on a web page)	`>`	`&#062;`
©	Copyright	`&copy;`	`&#169;`
®	Registered trademark	`&reg;`	`&#174;`
™	Trademark	`&trade;`	`&#8482;`
£	Pound	`&pound;`	`&#163;`
¥	Yen	`&yen;`	`&#165;`
€	Euro	`&euro;`	`&#8364;`
–	En-dash	`&ndash;`	`&#8211;`
—	Em-dash	`&mdash;`	`&#8212;`
‘	Left curly single quote	`&lsquo;`	`&#8216;`
’	Right curly single quote	`&rsquo;`	`&#8217;`
“	Left curly double quote	`&ldquo;`	`&#8220;`
”	Right curly double quote	`&rdquo;`	`&#8221;`
•	Bullet	`&bull;`	`&#8226;`
…	Horizontal ellipsis	`&hellip;`	`&#8230;`

Putting It All Together

TIP

Remember that indenting each hierarchical level in your HTML source consistently makes the document easier to scan and update later.

So far, you've learned how to mark up elements, and you've met all of the HTML elements for adding structure and meaning to text content. Now it's just a matter of practice. Exercise 5-3 gives you an opportunity to try out everything we've covered so far: document structure elements, block elements, inline elements, sectioning elements, and character entities. Have fun!

exercise 5-3 | The Black Goose Blog page

Now that you've been introduced to all of the text elements, you can put them to work by marking up the Blog page for the Black Goose Bistro site. The content is shown below (the second post is already marked up with the inline elements from Exercise 5-2). Get the starter text file online at *www.learningwebdesign.com/4e/materials*. The resulting markup is in Appendix A and included in the ***materials*** folder.

Once you have the text file, follow the instructions listed after the copy. The resulting page is shown in Figure 5-18.

```
The Black Goose Blog

Home
Menu
Blog
Contact

Summer Menu Items
posted by BGB, June 15, 2013
Our chef has been busy putting together the
perfect menu for the summer months. Stop by to
try these appetizers and main courses while the
days are still long.

Appetizers
Black bean purses
Spicy black bean and a blend of mexican cheeses
wrapped in sheets of phyllo and baked until
golden. $3.95

Southwestern napoleons with lump crab -- new
item!
Layers of light lump crab meat, bean and corn
salsa, and our handmade flour tortillas. $7.95

Main courses

Shrimp sate kebabs with peanut sauce
Skewers of shrimp marinated in lemongrass, garlic,
and fish sauce then grilled to perfection. Served
with spicy peanut sauce and jasmine rice. $12.95

Jerk rotisserie chicken with fried plantains --
new item!
Tender chicken slow-roasted on the rotisserie,
flavored with spicy and fragrant jerk sauce and
served with fried plantains and fresh mango.
$12.95

Low and Slow
posted by BGB, November 15, 2012
<p>This week I am <em>extremely</em> excited
about a new cooking technique called <dfn><i>sous
vide</i></dfn>. In <i>sous vide</i> cooking,
you submerge the food (usually vacuum-sealed in
plastic) into a water bath that is precisely
set to the target temperature of the food. In
his book, <cite>Cooking for Geeks</cite>, Jeff
Potter describes it as <q>ultra-low-temperature
poaching</q>.</p>

<p>Next month, we will be serving <b>Sous Vide
Salmon with Dill Hollandaise</b>. To reserve
a seat at the chef table, contact us before
November 30.</p>


Location: Baker's Corner, Seekonk, MA
Hours: Tuesday to Saturday, 11am to midnight

All content copyright &copy; 2012, Black Goose
Bistro and Jennifer Robbins
```

Figure 5-18. The finished menu page.

1. Add all the document structure elements first (**`html`**, **`head`**, **`meta`**, **`title`**, and **`body`**). Give the document the title "Black Goose Bistro: Blog."

2. The first thing we'll do is identify the top-level heading and the list of links as the header for the document by wrapping them in a **header** element (don't forget the closing tag). Within the header, the headline should be an **h1** and the list of links should be an unordered list (**ul**). Don't worry about making the list items links; we'll get to linking in the next chapter. Give the list more meaning by identifying it as the primary navigation for the site (**nav**).
3. This blog page has two posts titled "Summer Menu Items" and "Low and Slow." Mark each one up as an **article**.
4. Now we'll get the first article into shape! Let's create a **header** for this article that contains the heading (**h2** this time because we've moved down in the document hierarchy) and the publication information (**p**). Identify the publication date for the article with the **time** element, just as you did in Exercise 5-2.
5. The content after the header is clearly a simple paragraph. However, the menu has some interesting things going on. It is divided into two conceptual sections (Appetizers and Main Courses), so mark those up as **section** elements. Be careful that the closing section tag (**</section>**) appears before the closing article tag (**</article>**) so the elements are nested correctly and don't overlap. Finally, let's identify the sections with **id** attributes. Name the first one "appetizers" and the second "maincourses."
6. With our sections in place, now we can mark up the content. We're down to **h3** for the headings in each section. Choose the most appropriate list elements to describe the menu item names and their descriptions. Mark up the lists and each item within the lists.
7. Now we can add a few fine details. *Classify* each price as "price" using **span** elements.
8. Two of the dishes are new items. Change the double hyphens to an em-dash character and mark up "new items!" as "strongly important." Classify the title of each new dish as "newitem" (hint, use the existing **dt** element; there is no need to add a **span** this time). This allows us to target menu titles with the "newitem" class and style them differently than other menu items.
9. That takes care of the first article. The second article is already mostly marked up from the previous exercise, but you should mark up the header with the appropriate heading and publication information.
10. So far so good, right? Now make the remaining content that applies to the whole page a **footer**. Mark each line of content within the footer as a paragraph.
11. Let's give the location and hours information some context by putting them in a **div** named "about." Make the labels "Location" and "Hours" appear on a line by themselves by adding line breaks after them. If you'd like, you could also mark up the hours with the **time** element.
12. Finally, copyright information is typically "small print" on a document, so mark it up accordingly. As the final touch, add a copyright symbol after the word "copyright."

Save the file, name it ***bistro_blog.html***, and check your page in a modern browser (remember that IE 8 and earlier won't know what to do with those new HTML5 sectioning elements). How did you do?

Markup tips:

- Choose the element that best fits the meaning of the selected text.
- Don't forget to close elements with closing tags.
- Put all attribute values in quotation marks for clarity
- "Copy and paste" is your friend when adding the same markup to multiple elements. Just be sure what you copied is correct before you paste it throughout the document.

Test Yourself

Were you paying attention? Here is a rapid-fire set of questions to find out.

1. Add the markup to add a thematic break between these paragraphs.

```
<p>People who know me know that I love to cook.</p>

<p>I've created this site to share some of my favorite
recipes.</p>
```

2. What's the difference between a **blockquote** and a **q** element?

3. Which element displays whitespace exactly as it is typed into the source document?

4. What is the difference between a `ul` and an `ol`?

5. How do you remove the bullets from an unordered list? (Be general, not specific.)

6. What element would you use to provide the full name of the W3C (World Wide Web Consortium) in the document? Can you write out the complete markup?

7. What is the difference between a `dl` and a `dt`?

8. What is the difference between `id` and `class`?

9. What is the difference between an `article` and a `section`?

10. Name and write the characters generated by these character entities:

`&mdash;`	_____	`&`	_____
` `	_____	`&copy;`	_____
`&bull;`	_____	`&trade;`	_____

Want More Practice?

Try marking up your own résumé. Start with the raw text, and then add document structure elements, content grouping elements, then inline elements as we've done in Exercise 5-3. If you don't see an element that matches your information just right, try creating one using a `div` or a `span`.

Element Review: Text

The following is a summary of the elements we covered in this chapter. New HTML5 elements are indicated by "(5)." The **data** element is included only in the WHATWG HTML version as of this writing.

Page sections

address	author contact information
article (5)	self-contained content
aside (5)	tangential content (sidebar)
footer (5)	related content
header (5)	introductory content
nav (5)	primary navigation
section (5)	conceptually related group of content

Heading content

h1...h6	headings, levels 1 through 6
hgroup	heading group

Grouping content

blockquote	blockquote
div	generic division
figure (5)	related image or resource
figcaption (5)	text description of a figure
hr	paragraph-level thematic break (horizontal rule)
p	paragraph
pre	preformatted text

List elements

dd	definition
dl	definition list
dt	term
li	list item (for ul and ol)
ol	ordered list
ul	unordered list

Breaks

br	line break
wbr (5)	word break

Phrasing elements

abbr	abbreviation
b	added visual attention (bold)
bdi (5)	possible direction change
bdo	bidirectional override
cite	citation
code	code sample
data (WHATWG)	machine-readable equivalent
del	deleted text
dfn	defining term
em	stress emphasis
i	alternate voice (italic)
ins	inserted text
kbd	keyboard text
mark (5)	highlighted text
q	short inline quotation
ruby (5)	section containing ruby text
rp (5)	parentheses in ruby text
rt (5)	ruby annotations
s	strike-through; incorrect text
samp	sample output
small	annotation; "small print"
span	generic phrase of text
strong	strong importance
sub	subscript
sub	superscript
time (5)	machine-readable time data
u	added attention (underline)
var	variable