STEP 1: START WITH CONTENT

Now that we have our new document, it's time to get typing. A web page is all about content, so that's where we begin our demonstration. EXERCISE 4-1 walks you through entering the raw text content and saving the document in a new folder.

EXERCISE 4-1. Entering content

1. Type the home page content below into the new document in your text editor. Copy it exactly as you see it here, keeping the line breaks the same for the sake of playing along. The raw text for this exercise is also available online at *learningwebdesign.com/5e/materials/*.

```
Black Goose Bistro

The Restaurant
The Black Goose Bistro offers casual lunch and dinner fare in a relaxed
atmosphere. The menu changes regularly to highlight the freshest local
ingredients.

Catering
You have fun. We'll handle the cooking. Black Goose Catering can handle
events from snacks for a meetup to elegant corporate fundraisers.

Location and Hours
Seekonk, Massachusetts;
Monday through Thursday 11am to 9pm; Friday and Saturday, 11am to
midnight
```

2. Select "Save" or "Save as" from the File menu to get the Save As dialog box (FIGURE 4-4). The first thing you need to do is create a new folder (click the New Folder button on both Windows and Mac) that will contain all of the files for the site. The technical name for the folder that contains everything is the local root directory.

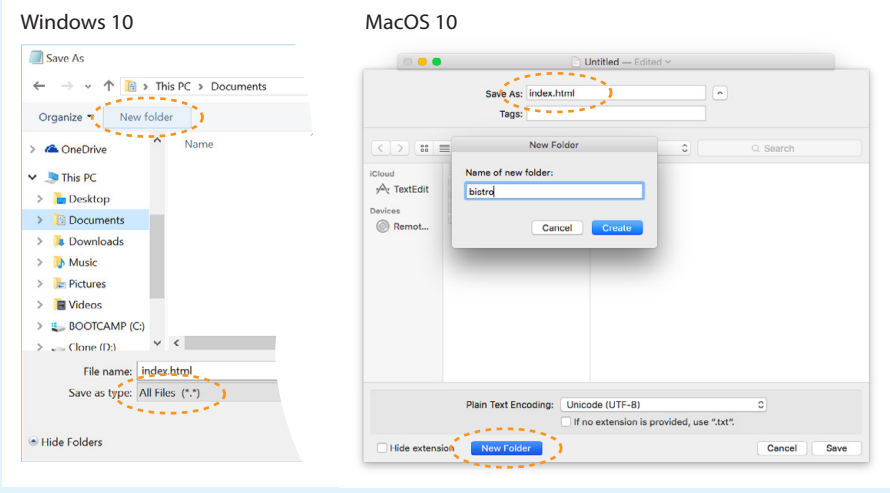

FIGURE 4-4. Saving *index.html* in a new folder called *bistro*.

Name the new folder *bistro*, and save the text file as *index.html* in it. The filename needs to end in *.html* to be recognized by the browser as a web document. See the sidebar **"Naming Conventions"** for more tips on naming files.

3. Just for kicks, let's take a look at *index.html* in a browser.

   Windows users: Double-click the filename in the File Explorer to launch your default browser, or right-click the file for the option to open it in the browser of your choice.

   Mac users: Launch your favorite browser (I'm using Google Chrome) and choose Open or Open File from the File menu. Navigate to *index.html*, and then select the document to open it in the browser.

4. You should see something like the page shown in FIGURE 4-5. We'll talk about the results in the following section.

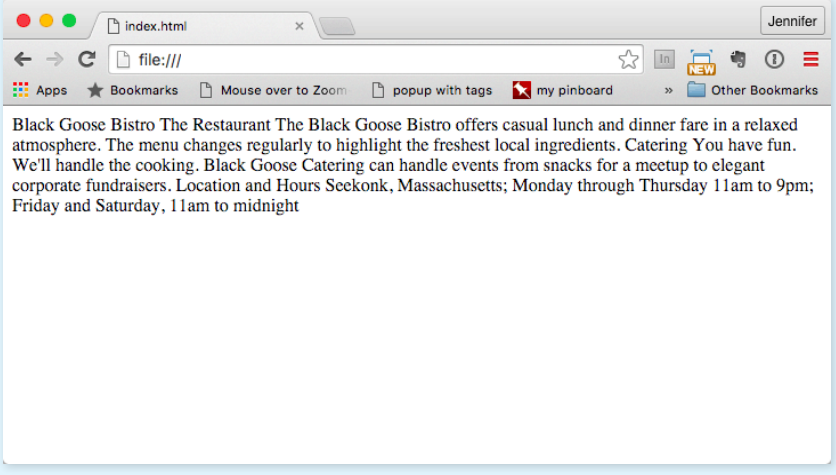

FIGURE 4-5. A first look at the content in a browser.

Naming Conventions

It is important that you follow these rules and conventions when naming your files:

Use proper suffixes for your files. HTML files must end with *.html* or *.htm*. Web graphics must be labeled according to their file format: *.gif*, *.png*, *.jpg* (*.jpeg* is also acceptable, although less common), or *.svg*.

Never use character spaces within filenames. It is common to use an underline character or hyphen to visually separate words within filenames, such as *robbins_bio.html* or *robbins-bio.html*.

Avoid special characters such as **?**, **%**, **#**, **/**, **:**, **;**, **•**, etc. Limit filenames to letters, numbers, underscores, hyphens, and periods. It is also best to avoid international characters, such as the Swedish å.

Filenames may be case-sensitive, depending on your server configuration. Consistently using all lowercase letters in filenames, although not required, is one way to make your filenames easier to manage.

Keep filenames short. Long names are more likely to be misspelled, and short names shave a few extra bytes off the file size. If you really must give the file a long, multiword name, you can separate words with hyphens, such as *a-long-document-title.html*, to improve readability.

Self-imposed conventions. It is helpful to develop a consistent naming scheme for huge sites—for instance, always using lowercase with hyphens between words. This takes some of the guesswork out of remembering what you named a file when you go to link to it later.

Learning from Step 1

Our page isn't looking so good (FIGURE 4-5). The text is all run together into one block—that's not how it looked when we typed it into the original document. There are a couple of lessons to be learned here. The first thing that is apparent is that the browser ignores line breaks in the source document. The sidebar **"What Browsers Ignore"** lists other types of information in the source document that are not displayed in the browser window.

Second, we see that simply typing in some content and naming the document *.html* is not enough. While the browser can display the text from the file, we haven't indicated the *structure* of the content. That's where HTML comes in. We'll use markup to add structure: first to the HTML document itself (coming up in Step 2), then to the page's content (Step 3). Once the browser knows the structure of the content, it can display the page in a more meaningful way.

STEP 2: GIVE THE HTML DOCUMENT STRUCTURE

We have our content saved in an HTML document—now we're ready to start marking it up.

The Anatomy of an HTML Element

Back in **Chapter 2** you saw examples of elements with an opening tag (`<p>` for a paragraph, for example) and a closing tag (`</p>`). Before we start adding tags to our document, let's look at the anatomy of an HTML element (its syntax) and firm up some important terminology. A generic container element is labeled in FIGURE 4-6.

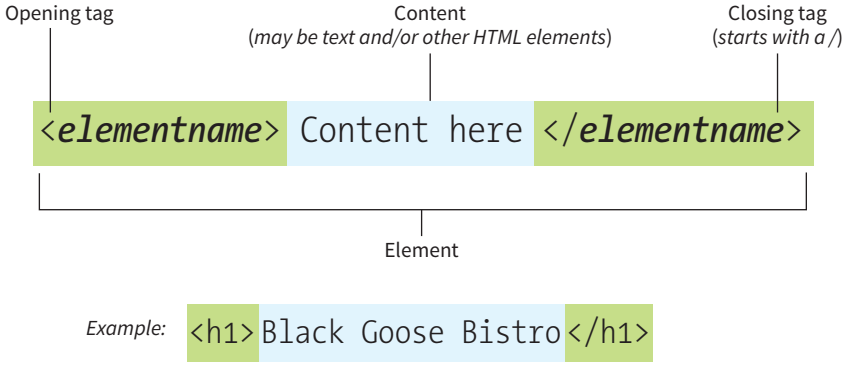

FIGURE 4-6. The parts of an HTML container element.

What Browsers Ignore

The following information in the source document will be ignored when it is viewed in a browser:

Multiple-character (white) spaces

When a browser encounters more than one consecutive blank character space, it displays a single space. So if the document contains

```
long,      long      ago
```

the browser displays:

```
long, long ago
```

Line breaks (carriage returns).

Browsers convert carriage returns to white spaces, so following the earlier "ignore multiple white spaces" rule, line breaks have no effect on formatting the page.

Tabs

Tabs are also converted to character spaces, so guess what? They're useless for indenting text on the web page (although they may make your code more readable).

Unrecognized markup

Browsers are instructed to ignore any tag they don't understand or that was specified incorrectly. Depending on the element and the browser, this can have varied results. The browser may display nothing at all, or it may display the contents of the tag as though it were normal text.

Text in comments

Browsers do not display text between the special `<!--` and `-->` tags used to denote a comment. See the upcoming **"Adding Hidden Comments"** sidebar.

Elements are identified by tags in the text source. A tag consists of the element name (usually an abbreviation of a longer descriptive name) within angle brackets (< >). The browser knows that any text within brackets is hidden and not displayed in the browser window.

The element name appears in the opening tag (also called a start tag) and again in the closing (or end) tag preceded by a slash (/). The closing tag works something like an "off" switch for the element. Be careful not to use the similar backslash character in end tags (see the tip **"Slash Versus Backslash"**).

The tags added around content are referred to as the markup. It is important to note that an element consists of both the content *and* its markup (the start and end tags). Not all elements have content, however. Some are empty by definition, such as the **img** element used to add an image to the page. We'll talk about empty elements a little later in this chapter.

One last thing: capitalization. In HTML, the capitalization of element names is not important (it is not case-sensitive). So **<img>**, **<Img>**, and **<IMG>** are all the same as far as the browser is concerned. However, most developers prefer the consistency of writing element names in all lowercase (see **Note**), as I will be doing throughout this book.

Basic Document Structure

FIGURE 4-8 shows the recommended minimal skeleton of an HTML document. I say "recommended" because the only element that is *required* in HTML is the **title**. But I feel it is better, particularly for beginners, to explicitly organize documents into metadata (**head**) and content (**body**) areas. Let's take a look at what's going on in this minimal markup example.

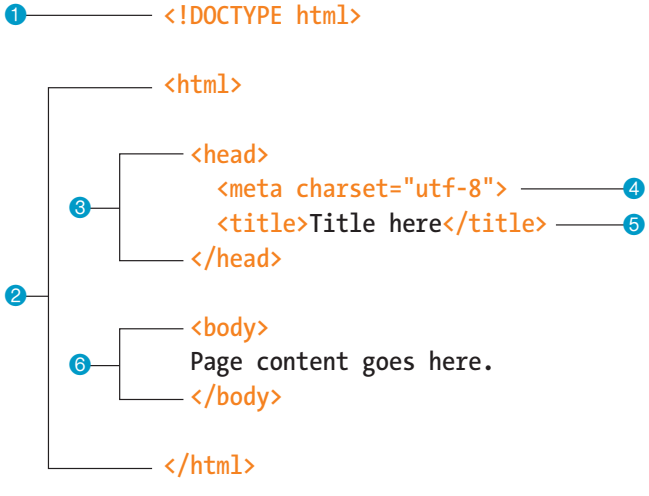

FIGURE 4-8. The minimal structure of an HTML document includes **head** and **body** contained within the **html** root element.

■ MARKUP TIP

Slash Versus Backslash

HTML tags and URLs use the slash character (/). The slash character is found under the question mark (?) on the English QWERTY keyboard (key placement on keyboards in other countries may vary).

It is easy to confuse the slash with the backslash character (\), which is found under the bar character (|); see FIGURE 4-7. The backslash key will not work in tags or URLs, so be careful not to use it.

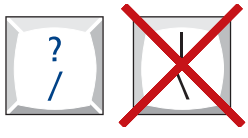

FIGURE 4-7. Slash versus backslash keys.

NOTE

There is a stricter version of HTML called XHTML that requires all element and attribute names to appear in lowercase. HTML5 has made XHTML all but obsolete except for certain use cases when it is combined with other XML languages, but the preference for all lowercase element names has persisted.

❶ I don't want to confuse things, but the first line in the example isn't an element at all. It is a document type declaration (also called DOCTYPE declaration) that lets modern browsers know which HTML specification to use to interpret the document. This DOCTYPE identifies the document as written in HTML5.

❷ The entire document is contained within an **`html`** element. The **`html`** element is called the root element because it contains all the elements in the document, and it may not be contained within any other element.

❸ Within the **`html`** element, the document is divided into a head and a body. The **`head`** element contains elements that pertain to the document that are not rendered as part of the content, such as its title, style sheets, scripts, and metadata.

❹ **`meta`** elements provide document metadata, information about the document. In this case, it specifies the character encoding (a standardized collection of letters, numbers, and symbols) used in the document as Unicode version UTF-8 (see the sidebar **"Introducing Unicode"**). I don't want to go into too much detail on this right now, but know that there are many good reasons for specifying the **`charset`** in every document, so I have included it as part of the minimal document markup. Other types of metadata provided by the **`meta`** element are the author, keywords, publishing status, and a description that can be used by search engines.

❺ Also in the **`head`** is the mandatory **`title`** element. According to the HTML specification, every document must contain a descriptive title.

❻ Finally, the **`body`** element contains everything that we want to show up in the browser window.

Are you ready to start marking up the Black Goose Bistro home page? Open the *index.html* document in your text editor and move on to EXERCISE 4-2.

Introducing Unicode

All the characters that make up languages are stored in computers as numbers. A standardized collection of characters with their reference numbers (code points) is called a coded character set, and the way in which those characters are converted to bytes for use by computers is the character encoding. In the early days of computing, computers used limited character sets such as ASCII that contained 128 characters (letters from Latin languages, numbers, and common symbols). The early web used the Latin-1 (ISO 8859-1) character encoding that included 256 Latin characters from most Western languages. But given the web was "worldwide," it was clearly not sufficient.

Enter Unicode. Unicode (also called the Universal Character Set) is a super-character set that contains over 136,000 characters (letters, numbers, symbols, ideograms, logograms, etc.) from all active modern languages. You can read all about it at *unicode.org*. Unicode has three standard encodings—UTF-8, UTF-16, and UTF-32—that differ in the number of bytes used to represent the characters (1, 2, or 3, respectively).

HTML5 uses the UTF-8 encoding by default, which allows wide-ranging languages to be mixed within a single document. It is always a good idea to declare the character encoding for a document with the **`meta`** element, as shown in the previous example. Your server also needs to be configured to identify HTML documents as UTF-8 in the HTTP header (information about the document that the server sends to the user agent). You can ask your server administrator to confirm the encoding of the HTML documents.

EXERCISE 4-2. Adding minimal structure

1. Open the new *index.html* document if it isn't open already and add the DOCTYPE declaration:

```
<!DOCTYPE html>
```

2. Put the entire document in an HTML root element by adding an **<html>** start tag after the DOCTYPE and an **</html>** end tag at the very end of the text.
3. Next, create the document head that contains the title for the page. Insert **<head>** and **</head>** tags before the content. Within the **head** element, add information about the character encoding **<meta charset="utf-8">**, and the title, "Black Goose Bistro", surrounded by opening and closing **<title>** tags.
4. Finally, define the body of the document by wrapping the text content in **<body>** and **</body>** tags. When you are done, the source document should look like this (the markup is shown in color to make it stand out):

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <title>Black Goose Bistro</title>
</head>

<body>
Black Goose Bistro

The Restaurant
The Black Goose Bistro offers casual lunch and
dinner fare in a relaxed atmosphere. The menu
changes regularly to highlight the freshest local
ingredients.

Catering
You have fun. We'll handle the cooking. Black
Goose Catering can handle events from snacks for a
meetup to elegant corporate fundraisers.

Location and Hours
Seekonk, Massachusetts;
Monday through Thursday 11am to 9pm; Friday and
Saturday, 11am to midnight
</body>
</html>
```

5. Save the document in the *bistro* directory, so that it overwrites the old version. Open the file in the browser or hit Refresh or Reload if it is open already. FIGURE 4-9 shows how it should look now.

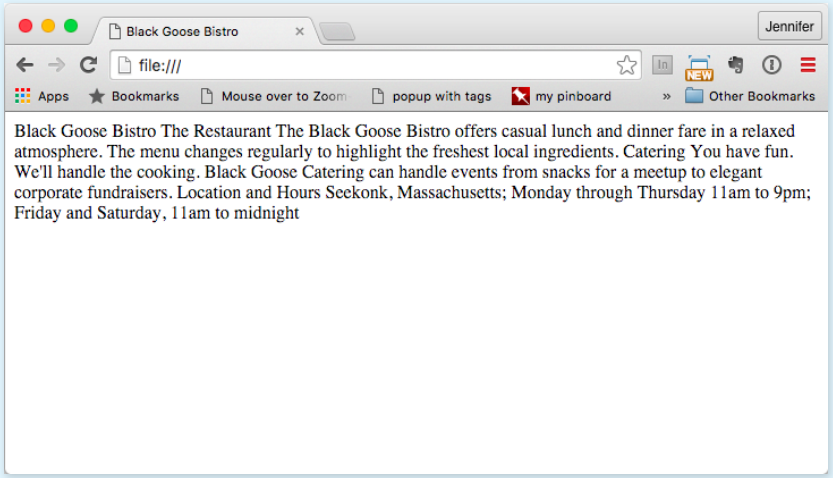

FIGURE 4-9. The page in a browser after the document structure elements have been defined.

Not much has changed in the bistro page after setting up the document, except that the browser now displays the title of the document in the top bar or tab (FIGURE 4-9). If someone were to bookmark this page, that title would be added to their Bookmarks or Favorites list as well (see the sidebar **"Don't Forget a Good Title"**). But the content still runs together because we haven't given the browser any indication of how it should be structured. We'll take care of that next.

STEP 3: IDENTIFY TEXT ELEMENTS

With a little markup experience under your belt, it should be a no-brainer to add the markup for headings and subheads (`h1` and `h2`), paragraphs (`p`), and emphasized text (`em`) to our content, as we'll do in EXERCISE 4-3. However, before we begin, I want to take a moment to talk about what we're doing and not doing when marking up content with HTML.

Mark It Up Semantically

The purpose of HTML is to add meaning and structure to the content. It is *not* intended to describe how the content should look (its presentation).

Your job when marking up content is to choose the HTML element that provides the most meaningful description of the content at hand. In the biz, we call this semantic markup. For example, the most important heading at the beginning of the document should be marked up as an `h1` because it is the most important heading on the page. Don't worry about what it looks like... you can easily change that with a style sheet. The important thing is that you choose elements based on what makes the most sense for the content.

In addition to adding meaning to content, the markup gives the document structure. The way elements follow each other or nest within one another creates relationships between them. You can think of this structure as an outline (its technical name is the DOM, for Document Object Model). The underlying document hierarchy gives browsers cues on how to handle the content. It is also the foundation upon which we add presentation instructions with style sheets and behaviors with JavaScript.

Although HTML was intended to be used strictly for meaning and structure since its creation, that mission was somewhat thwarted in the early years of the web. With no style sheet system in place, HTML was extended to give authors ways to change the appearance of fonts, colors, and alignment using markup alone. Those presentational extras are still out there, so you may run across them if you view the source of older sites or a site made with old tools. In this book, however, I'll focus on using HTML the right way, in keeping with the contemporary standards-based, semantic approach to web design.

OK, enough lecturing. It's time to get to work on that content in EXERCISE 4-3.

Don't Forget a Good Title

A `title` element is not only required for every document, but it is also quite useful. The title is what is displayed in a user's Bookmarks or Favorites list and on tabs in desktop browsers. Descriptive titles are also a key tool for improving accessibility, as they are the first things a person hears when using a screen reader (an assistive device that reads the content of a page aloud for users with impaired sight). Search engines rely heavily on document titles as well.

For these reasons, it's important to provide thoughtful and descriptive titles for all your documents and avoid vague titles, such as "Welcome" or "My Page." You may also want to keep the length of your titles in check so they are able to display in the browser's title area. Knowing that users typically have a number of tabs open or a long list of Bookmarks, put your most uniquely identifying information in the first 20 or so characters.

The purpose of HTML is to add meaning and structure to the content.

EXERCISE 4-3. Defining text elements

1. Open the document *index.html* in your text editor, if it isn't open already.
2. The first line of text, "Black Goose Bistro," is the main heading for the page, so we'll mark it up as a Heading Level 1 (**h1**) element. Put the opening tag, **<h1>**, at the beginning of the line and the closing tag, **</h1>**, after it, like this:

```
<h1>Black Goose Bistro</h1>
```

3. Our page also has three subheads. Mark them up as Heading Level 2 (**h2**) elements in a similar manner. I'll do the first one here; you do the same for "Catering" and "Location and Hours."

```
<h2>The Restaurant</h2>
```

4. Each **h2** element is followed by a brief paragraph of text, so let's mark those up as paragraph (**p**) elements in a similar manner. Here's the first one; you do the rest:

```
<p>The Black Goose Bistro offers casual lunch and dinner fare in
a relaxed atmosphere. The menu changes regularly to highlight the
freshest local ingredients.</p>
```

5. Finally, in the Catering section, I want to emphasize that visitors should just leave the cooking to us. To make text emphasized, mark it up in an emphasis element (**em**) element, as shown here:

```
<p>You have fun. <em>We'll handle the cooking.</em> Black Goose
Catering can handle events from snacks for a meetup to elegant
corporate fundraisers.</p>
```

6. Now that we've marked up the document, let's save it as we did before, and open (or reload) the page in the browser. You should see a page that looks much like the one in FIGURE 4-10. If it doesn't, check your markup to be sure that you aren't missing any angle brackets or a slash in a closing tag.

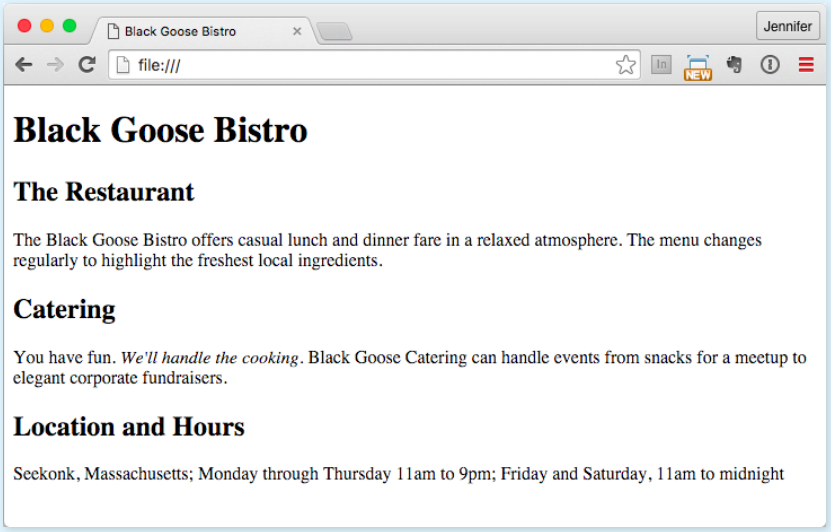

FIGURE 4-10. The home page after the content has been marked up with HTML elements.

Now we're getting somewhere. With the elements properly identified, the browser can now display the text in a more meaningful manner. There are a few significant things to note about what's happening in FIGURE 4-10.

Block and Inline Elements

Although it may seem like stating the obvious, it's worth pointing out that the heading and paragraph elements start on new lines and do not run together as they did before. That is because by default, headings and paragraphs display as block elements. Browsers treat block elements as though they are in little rectangular boxes, stacked up in the page. Each block element begins on a new line, and some space is also usually added above and below the entire element by default. In FIGURE 4-11, the edges of the block elements are outlined in red.

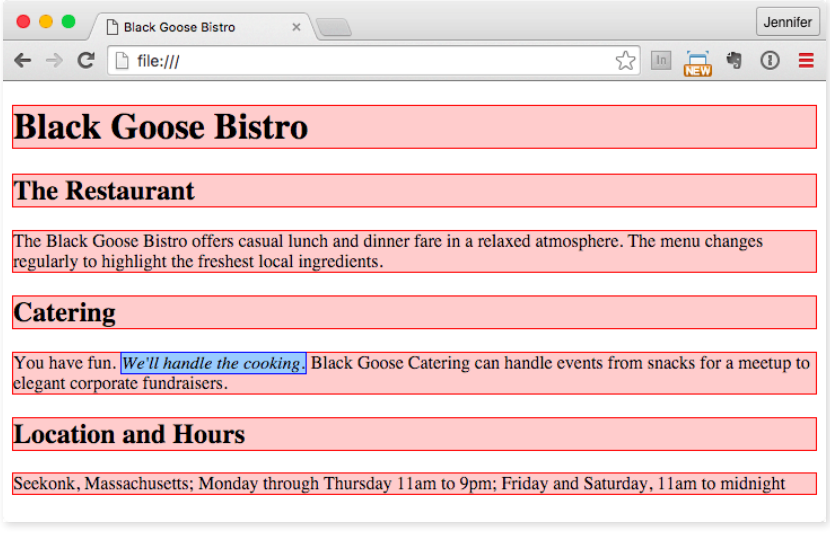

FIGURE 4-11. The outlines show the structure of the elements in the home page.

By contrast, look at the text we marked up as emphasized (`em`, outlined in blue in FIGURE 4-11). It does not start a new line, but rather stays in the flow of the paragraph. That is because the `em` element is an inline element (also called a text-level semantic element or phrasing element). Inline elements do not start new lines; they just go with the flow.

Default Styles

The other thing that you will notice about the marked-up page in FIGURES 4-10 and 4-11 is that the browser makes an attempt to give the page some

Adding Hidden Comments

You can leave notes in the source document for yourself and others by marking them up as comments. Anything you put between comment tags (`<!-- -->`) will not display in the browser and will not have any effect on the rest of the source:

```
<!-- This is a comment -->
<!-- This is a
  multiple-line comment
  that ends here. -->
```

Comments are useful for labeling and organizing long documents, particularly when they are shared by a team of developers. In this example, comments are used to point out the section of the source that contains the navigation:

```
<!-- start global nav -->
<ul>
  …
</ul>
<!-- end global nav -->
```

Bear in mind that although the browser will not display comments in the web page, readers can see them if they "view source," so be sure that the comments you leave are appropriate for everyone.

visual hierarchy by making the first-level heading the biggest and boldest thing on the page, with the second-level headings slightly smaller, and so on.

How does the browser determine what an `h1` should look like? It uses a style sheet! All browsers have their own built-in style sheets (called user agent style sheets in the spec) that describe the default rendering of elements. The default rendering is similar from browser to browser (for example, `h1`s are always big and bold), but there are some variations (the `blockquote` element for long quotes may or may not be indented).

If you think the `h1` is too big and clunky as the browser renders it, just change it with your own style sheet rule. Resist the urge to mark up the heading with another element just to get it to look better—for example, using an `h3` instead of an `h1` so it isn't as large. In the days before ubiquitous style sheet support, elements were abused in just that way. You should always choose elements based on how accurately they describe the content, and don't worry about the browser's default rendering.

We'll fix the presentation of the page with style sheets in a moment, but first, let's add an image to the page.

STEP 4: ADD AN IMAGE

What fun is a web page with no images? In EXERCISE 4-4, we'll add an image to the page with the `img` element. Images will be discussed in more detail in **Chapter 7, Adding Images**, but for now, they give us an opportunity to introduce two more basic markup concepts: empty elements and attributes.

Empty Elements

So far, nearly all of the elements we've used in the Black Goose Bistro home page have followed the syntax shown in FIGURE 4-6: a bit of text content surrounded by start and end tags.

A handful of elements, however, do not have content because they are used to provide a simple directive. These elements are said to be empty. The image element (`img`) is an example of an empty element. It tells the browser to get an image file from the server and insert it at that spot in the flow of the text. Other empty elements include the line break (`br`), thematic breaks (`hr`, a.k.a. "horizontal rules"), and elements that provide information about a document but don't affect its displayed content, such as the `meta` element that we used earlier.

FIGURE 4-12 shows the very simple syntax of an empty element (compare it to FIGURE 4-6).

```
<element-name>
```

Example: The **br** element inserts a line break.

```
<p>1005 Gravenstein Highway North<br>Sebastopol, CA 95472</p>
```

FIGURE 4-12. Empty element structure.

Attributes

Let's get back to adding an image with the empty **img** element. Obviously, an **<img>** tag is not very useful by itself—it doesn't indicate which image to use. That's where attributes come in. Attributes are instructions that clarify or modify an element. For the **img** element, the **src** (short for "source") attribute is required, and specifies the location (URL) of the image file.

The syntax for an attribute is as follows:

```
attributename="value"
```

Attributes go after the element name, separated by a space. In non-empty elements, attributes go in the opening tag only:

```
<element attributename="value">
<element attributename="value">Content</element>
```

You can also put more than one attribute in an element in any order. Just keep them separated with spaces:

```
<element attribute1="value" attribute2="value">
```

FIGURE 4-13 shows an **img** element with its required attributes labeled.

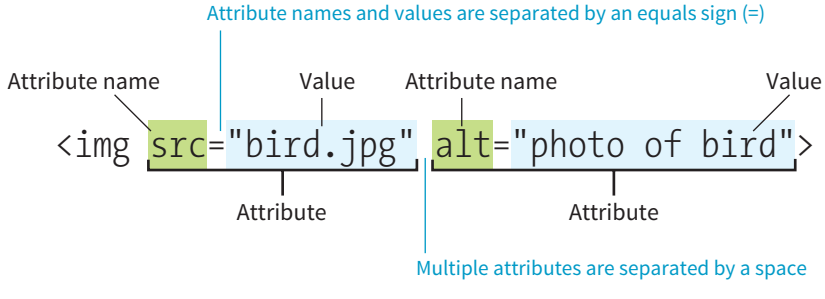

FIGURE 4-13. An **img** element with two attributes.

What Is That Extra Slash?

If you poke around in source documents for existing web pages, you may see empty elements with extra slashes at the end, like so: **<img />**, **
, **<meta />, and **<hr />**. That indicates the document was written according to the stricter rules of XHTML. In XHTML, all elements, including empty elements, must be closed (or terminated, to use the proper term). You terminate empty elements by adding a trailing slash before the closing bracket. The preceding character space is not required but was used for backward compatibility with browsers that did not have XHTML parsers, so **<img/>**, **
**, and so on are valid.

Attributes are instructions that clarify or modify an element.

Here's what you need to know about attributes:

- Attributes go after the element name in the opening tag only, never in the closing tag.
- There may be several attributes applied to an element, separated by spaces in the opening tag. Their order is not important.
- Most attributes take values, which follow an equals sign (=). In HTML, some attribute values are single descriptive words. For example, the **`checked`** attribute, which makes a form checkbox checked when the form loads, is equivalent to **`checked="checked"`**. You may hear this type of attribute called a Boolean attribute because it describes a feature that is either on or off.
- A value might be a number, a word, a string of text, a URL, or a measurement, depending on the purpose of the attribute. You'll see examples of all of these throughout this book.
- Wrapping attribute values in double quotation marks is a strong convention, but note that quotation marks are not required and may be omitted. In addition, either single or double quotation marks are acceptable as long as the opening and closing marks match. Note that quotation marks in HTML files need to be straight ("), not curly (").
- The attribute names and values available for each element are defined in the HTML specifications; in other words, you can't make up an attribute for an element.
- Some attributes are required, such as the **`src`** and **`alt`** attributes in the **`img`** element. The HTML specification also defines which attributes are required in order for the document to be valid.

Now you should be more than ready to try your hand at adding the **`img`** element with its attributes to the Black Goose Bistro page in EXERCISE 4-4. We'll throw a few line breaks in there as well.

EXERCISE 4-4. Adding an image

1. If you're working along, the first thing you'll need to do is get a copy of the image file on your hard drive so you can see it in place when you open the file locally. The image file is provided in the materials for this chapter (*learningwebdesign.com/5e/materials*). You can also get the image file by saving it right from the sample web page online at *learningwebdesign.com/5e/materials/ch04/bistro*. Right-click (or Control-click on a Mac) the goose image and select "Save to disk" (or similar) from the pop-up menu, as shown in FIGURE 4-14. Name the file *blackgoose.png*. Be sure to save it in the *bistro* folder with *index.html*.
2. Once you have the image, insert it at the beginning of the first-level heading by typing in the **`img`** element and its attributes as shown here:

```
<h1><img src="blackgoose.png" alt="logo">Black Goose Bistro</h1>
```

Windows: Right-click on the image to access the pop-up menu.

Mac: Control-click on the image to access the pop-up menu. The options may vary by browser.

FIGURE 4-14. Saving an image file from a page on the web.

The **src** attribute provides the name of the image file that should be inserted, and the **alt** attribute provides text that should be displayed if the image is not available. Both of these attributes are required in every **img** element.

3. I'd like the image to appear above the title, so add a line break (**br**) after the **img** element to start the headline text on a new line.

```
<h1><img src="blackgoose.png" alt="logo"><br>Black Goose Bistro</h1>
```

4. Let's break up the last paragraph into three lines for better clarity. Drop a **
** tag at the spots you'd like the line breaks to occur. Try to match the screenshot in FIGURE 4-15.
5. Now save *index.html* and open or refresh it in the browser window. The page should look like the one shown in FIGURE 4-15. If it doesn't, check to make sure that the image file, *blackgoose.png*, is in the same directory as *index.html*. If it is, then check to make sure that you aren't missing any characters, such as a closing quote or bracket, in the **img** element markup.

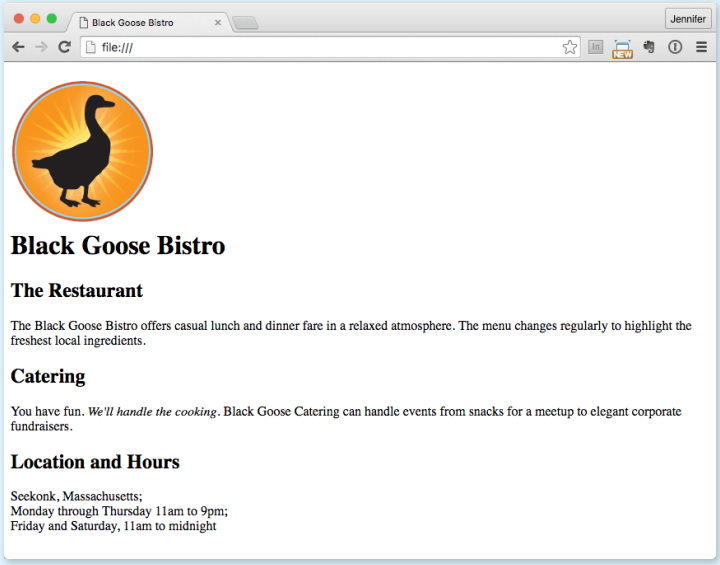

FIGURE 4-15. The Black Goose Bistro page with the logo image.

STEP 5: CHANGE THE LOOK WITH A STYLE SHEET

Depending on the content and purpose of your website, you may decide that the browser's default rendering of your document is perfectly adequate. However, I think I'd like to pretty up the Black Goose Bistro home page a bit to make a good first impression on potential patrons. "Prettying up" is just my way of saying that I'd like to change its presentation, which is the job of Cascading Style Sheets (CSS).

In EXERCISE 4-5, we'll change the appearance of the text elements and the page background by using some simple style sheet rules. Don't worry about understanding them all right now. We'll get into CSS in more detail in **Part III**. But I want to at least give you a taste of what it means to add a "layer" of presentation onto the structure we've created with our markup.

EXERCISE 4-5. Adding a style sheet

1. Open *index.html* if it isn't open already. We're going to use the **style** element to apply a very simple embedded style sheet to the page. This is just one of the ways to add a style sheet; the others are covered in **Chapter 11, Introducing Cascading Style Sheets**.
2. The **style** element is placed inside the document **head**. Start by adding the **style** element to the document as shown here:

```
<head>
  <meta charset="utf-8">
  <title>Black Goose Bistro</title>
  <style>

  </style>
</head>
```

3. Next, type the following style rules within the **style** element just as you see them here. Don't worry if you don't know exactly what's going on (although it's fairly intuitive). You'll learn all about style rules in **Part III**.

```
<style>
body {
  background-color: #faf2e4;
  margin: 0 10%;
  font-family: sans-serif;
  }
h1 {
  text-align: center;
  font-family: serif;
  font-weight: normal;
  text-transform: uppercase;
  border-bottom: 1px solid #57b1dc;
  margin-top: 30px;
}
h2 {
  color: #d1633c;
  font-size: 1em;
}
</style>
```

4. Now it's time to save the file and take a look at it in the browser. It should look like the page in FIGURE 4-16. If it doesn't, go over the style sheet to make sure you didn't miss a semicolon or a curly bracket. Look at the way the page looks with our styles compared to the browser's default styles (FIGURE 4-15).

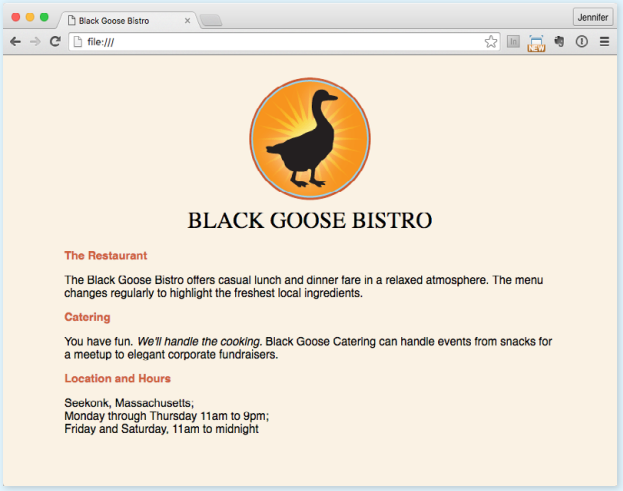

FIGURE 4-16. The Black Goose Bistro page after CSS style rules have been applied.