Web Page Building Blocks

While Web pages have become increasingly complex, their underlying structure remains remarkably simple. The first thing you should know is that it's impossible to create a Web page without HTML. As you will learn, HTML houses your content and describes its meaning. In turn, Web browsers render your HTML-encased content for users.

A Web page is primarily made up of three components:

- *Text content*: The bare text that appears on the page to inform visitors about your business, family vacation, products, or whatever the focus of your page may be.
- *References to other files*: These load items such as images, audio, video, and SVG files, and they link to other HTML pages and assets, as well as to style sheets (which control your page's layout) and JavaScript files (which add behavior to your page).
- *Markup*: The HTML elements that describe your text content and make the references work. (The *m* in HTML stands for *markup*.)

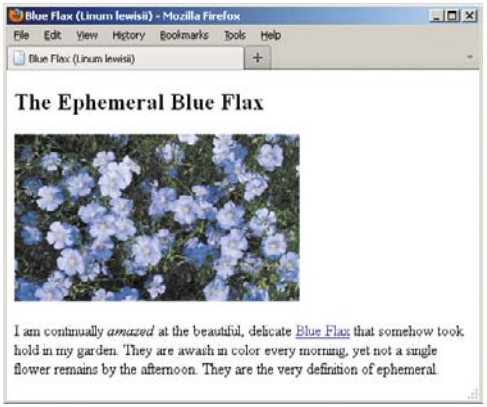
In This Chapter

It's important to note that each of these components in a Web page is made up exclusively of text. This means that pages are saved in text-only format and can be viewed on practically any browser on any platform, whether desktop, mobile, tablet, or otherwise. It guarantees the universality of the Web. A page may look different when viewed on one device versus another, but that's OK. The important thing as a first step is to make content accessible to all users, and HTML affords that.

In addition to the three components that a Web page is primarily made up of, a page also includes HTML that provides information about the page itself, most of which your users don't see explicitly and that is primarily intended for browsers and search engines. This can include information about the content's primary language (English, French, and so on), character encoding (typically UTF-8), and more.

This chapter will walk you through a basic HTML page, discuss some best practices, and explain each of the three important components.

Note: As mentioned in the introduction, I use *HTML* to refer to the language in general. For those instances in which I'm highlighting special characteristics unique to a version of the language, I will use the individual name. For example, "*HTML5* introduces several new elements and redefines or eliminates others that previously existed in *HTML 4* and *XHTML 1.0*." For more details, please consult "How This Book Works" in the introduction.



A A typical default rendering of the page. Although this shows the page in Firefox, the page displays similarly in other browsers.

A Basic HTML Page

Let's take a look at a basic HTML page to give you context for what's to follow in this chapter and beyond. Figure A illustrates how a desktop browser typically renders the HTML code in B. You'll learn some of the basics about the code B, but don't worry if you don't understand it all right now. This is just to give you a taste of HTML. You have the rest of the book to learn more about it.

You can probably guess some of what's going on in the code, especially in the **body** section. First let's look at the part before the **body**.

B Here is the code for a basic HTML page. I've highlighted the HTML portions so you can distinguish them from the page's text content. As demonstrated in A, the HTML surrounding the text content doesn't appear when you view the page in a browser. But, as you will learn, the markup is essential because it describes the content's meaning. Note, too, that each line happens to be separated with a carriage return. This isn't mandatory and does not impact the page's rendering.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>Blue Flax (Linum lewisii)</title>
</head>
<body>
    <article>
      <h1>The Ephemeral Blue Flax</h1>

      <img src="blueflax.jpg" width="300" height="175" alt="Blue Flax (Linum lewisii)" />

      <p>I am continually <em>amazed</em> at the beautiful, delicate <a href="http://
      → en.wikipedia.org/wiki/Linum_lewisii" rel="external" title="Learn more about Blue
      → Flax">Blue Flax</a> that somehow took hold in my garden. They are awash in color every
      → morning, yet not a single flower remains by the afternoon. They are the very definition
      → of ephemeral.</p>
    </article>
</body>
</html>
```

Everything above the **<body>** start tag is the instructional information for browsers and search engines mentioned earlier C. Each page begins with the DOCTYPE declaration, which tells the browser the HTML version of the page.

You should always use HTML5's DOCTYPE, which is **<!DOCTYPE html>**. The case of the text doesn't matter, but it's more common to use DOCTYPE in all uppercase. Regardless, always include the DOCTYPE in your pages. (See the sidebar "HTML5's Improved DOCTYPE" in Chapter 3 for more information.)

The bits that start at **<!DOCTYPE html>** and continue through **</head>** are invisible to users with one exception: the text between **<title>** and **</title>**—Blue Flax (Linum lewisii)—appears as the title at the very top of the browser window and on a browser tab B. Additionally, it's typically the default name of a browser bookmark or favorite and is valuable information for search engines. Chapter 3 explains what the other parts of the top segment of a page do.

C The **title** element text is the only part of the top area of an HTML document that the user sees. The rest is information about the page for browsers and search engines.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>The Ephemeral Blue Flax
    → (Linum lewisii)</title>
</head>
```

D A page's content exists between the start and end tags of the **body** element. The document ends at **</html>**.

```
<!DOCTYPE html>
<html lang="en">
. . . [document head] . . .
<body>
    <article>
        <h1>The Ephemeral Blue Flax</h1>

        <img src="blueflax.jpg" width="300"
        → height="175" alt="Blue Flax (Linum
        → lewisii)" />

        <p>I am continually <em>amazed</em>
        → at the beautiful, delicate
        → <a href="http://en.wikipedia.org/
        → wiki/Linum_lewisii" rel="external"
        → title="Learn more about Blue Flax">
        → Blue Flax</a> that somehow took
        → hold in my garden. They are awash
        → in color every morning, yet not a
        → single flower remains by the
        → afternoon. They are the very
        → definition of ephemeral.</p>
    </article>
</body>
</html>
```

Meanwhile, your page's content—that is, what *is* visible to users—goes between **<body>** and **</body>**. Finally, the **</html>** end tag signals the end of the page D.

The code's indentation has absolutely no bearing on whether the code is valid HTML. It also doesn't affect how the content displays in the browser (the **pre** element, which you'll learn about in Chapter 4, is the one exception). However, it's customary to indent code that's nested in a parent element to make it easier to glean the hierarchy of elements as you read through the code. You'll learn more about parents and children later in this chapter. You'll also learn in greater detail about the default browser rendering.

First, let's discuss what it means to write semantic HTML and why it is a cornerstone of an effective Web site.

Semantic HTML: Markup with Meaning

HTML is a clever system of including information about the content in a text document. This information, called markup, describes the *meaning* of the content, that is, the *semantics*. You've already seen a few examples in our basic HTML page, such as the **p** element that marks up paragraph content.

HTML does *not* define how the content should appear in a browser; that's the role of CSS (Cascading Style Sheets). HTML5 stresses this distinction more than any prior version of HTML. It's at the core of the language.

You might be wondering why, if that's the case, some text in the basic HTML page A looks larger than other text, or is bold or italicized B.

Great question. The reason is that every Web browser has a built-in CSS file (a *style sheet*) that dictates how each HTML element displays by default, unless you create your own that overwrites it. The default presentation varies slightly from browser to browser, but on the whole it is fairly consistent. More importantly, the content's underlying structure and meaning as defined by your HTML remain the same.

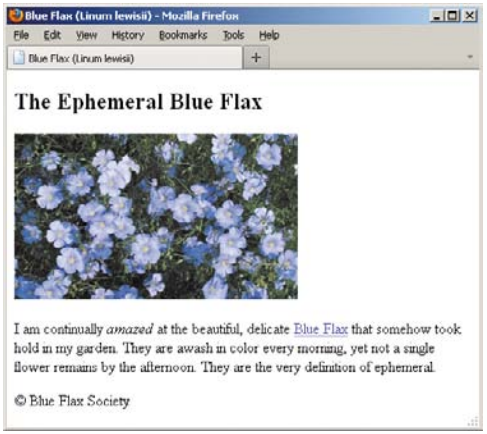
A The content of our basic page plus a second paragraph added at the end. The HTML elements don't dictate how the content should appear, just what they mean. Instead, each browser's built-in style sheet dictates how the content displays by default B.

```
…
<body>
    <article>
        <h1>The Ephemeral Blue Flax</h1>

        <img src="blueflax.jpg" width="300"
        → height="175" alt="Blue Flax (Linum
        → lewisii)" />

        <p>I am continually <em>amazed</em>
        → at the beautiful, delicate
        → <a href="http://en.wikipedia.org/
        → wiki/Linum_lewisii" rel="external"
        → title="Learn more about Blue Flax">
        → Blue Flax</a> that somehow took
        → hold in my garden. They are awash
        → in color every morning, yet not a
        → single flower remains by the
        → afternoon. They are the very
        → definition of ephemeral.</p>

        <p><small>&copy; Blue Flax Society.
        → </small></p>
    </article>
</body>
</html>
```



B A browser's default style sheet renders headings (**h1**–**h6** elements) differently than normal text, italicizes **em** text, and colors and underlines links. Additionally, some elements begin on their own line (**h1** and **p**, for example), and others display within surrounding content (like **a** and **em**). This example includes a second paragraph (the copyright notice) to make it clear that each paragraph occupies its own line. It's simple to overwrite any or all of these presentation rules with your own style sheets.

Block-level, Inline, and HTML5

As you can see, some HTML elements (for example, the **article**, **h1**, and **p**) display on their own line like a paragraph does in a book, while others (for example, the **a** and **em**) render in the same line as other content B. Again, this is a function of the browser's default style rules, not the HTML elements themselves. Allow me to elaborate. Before HTML5, most elements were categorized as either *block-level* (the ones that displayed on their own line) or *inline* (the ones that displayed within a line of text). HTML5 does away with these terms because they associate elements with presentation, which you've learned isn't HTML's role.

Instead, generally speaking, elements that had previously been dubbed inline are categorized in HTML5 as *phrasing content*—that is, elements and their contained text that primarily appear within a paragraph. (Chapter 4 focuses almost exclusively on phrasing content. See the full list at http://dev.w3.org/html5/spec-author-view/content-models.html#phrasing-content-0.)

The old block-level elements also now fall into new HTML5 categories that focus on their semantics. Many of these elements account for the main structural blocks and headings of your content (dig into Chapter 3 to learn more about sectioning content and heading content elements).

With all that said, browsers haven't changed the default display rules for these elements, nor should they. After all, you wouldn't want, say, the two paragraphs (the **p** elements) running into each other, or the **em** text ("amazed") to break the sentence by appearing on its own line (**em** is the element you use for adding emphasis).

So usually headings, paragraphs, and structural elements like **`article`** display on their own line, and phrasing content displays on the same line as surrounding content. And even though HTML5 no longer uses the terms block-level and inline, it helps to know what they mean. It's common for tutorials to use them since they were entrenched in HTML vernacular before HTML5. I might use them occasionally in the book to quickly convey whether an element occupies its own line or shares a line by default.

We'll cover CSS in detail in later chapters, but for now know that a style sheet, like an HTML page, is just text, so you can create one with the same text editor as your HTML.

HTML5's Focus on Semantics

HTML5 emphasizes HTML semantics, leaving all visual styling to CSS. That wasn't always the case with earlier versions of HTML.

A proper means to style pages didn't exist in the Web's nascent years; HTML was already a few years old by the time CSS1 was formally introduced in December of 1996. To fill that gap in the meantime, HTML included a handful of presentational elements whose purpose was to allow basic styling of text, such as making it bold, italicized, or a different size than surrounding text.

Those elements served their purpose for the time, but they rightfully fell out of favor as best practices evolved for Web development. Central to that thinking was—and still very much is—the notion that HTML is for describing the content's meaning only, not its display.

The presentational HTML elements broke this best practice. As such, HTML 4 deprecated their use, recommending authors use CSS to style text and other page elements instead.

HTML5 goes further; it eliminates some presentational elements and redefines others so they carry only semantic value instead of dictating presentation.

The **`small`** element is one such example. Initially, it was intended to make text smaller than regular text. However, in HTML5 **`small`** represents fine print, such as a legal disclaimer. You can use CSS to make it the largest text on the page if you'd like, but that won't change the meaning of your **`small`** content.

Meanwhile, **`small`**'s old counterpart, the **`big`** element, doesn't exist in HTML5. There are other examples, too, which you'll learn about as you progress through the book.

HTML5 also defines new elements, such as **`header`**, **`footer`**, **`nav`**, **`article`**, **`section`**, and many more that enrich the semantics of your content. You'll learn about those later as well.

However, whether you use an HTML element that's existed since the dawn of the language or one that's new in HTML5, your goal should be the same: Choose the elements that best describe the meaning of your content without regard for their presentation.

C The **body** of our basic page, which contains the **article**, **h1**, **img**, **p**, **em**, and **a** elements to describe the content's meaning. All the content is nested in the **article**.

```
<body>
    <article>
        <h1>The Ephemeral Blue Flax</h1>

        <img src="blueflax.jpg" width="300"
        → height="175" alt="Blue Flax (Linum
        → lewisii)" />

        <p>I am continually <em>amazed</em>
        → at the beautiful, delicate
        → <a href="http://en.wikipedia.org/
        → wiki/Linum_lewisii" rel="external"
        → title="Learn more about Blue Flax">
        → Blue Flax</a> that somehow took
        → hold in my garden. They are awash
        → in color every morning, yet
        → not a single flower remains by
        → the afternoon. They are the very
        → definition of ephemeral.</p>
    </article>
</body>
```

D Headings are critical elements in defining a page's outline. They make a page more accessible to users of screen readers, and search engines use them to determine the focus of a page.

```
<h1>The Ephemeral Blue Flax</h1>
```

E It's easy to add an image to a page with **img**. As defined by the **alt** attribute, "Blue Flax (Linum lewisii)" displays if our image doesn't.

```
<img src="blueflax.jpg" width="300"
→ height="175" alt="Blue Flax (Linum lewisii)"
→ />
```

The Semantics of Our Basic HTML Page

Now that you know HTML's role, let's look a little deeper at the thought process behind marking up sample content. As you'll see, there's no magic to writing semantic HTML. It's mostly common sense once you're familiar with the elements at your disposal. Let's revisit the **body** of our basic page for a taste of some of the most frequently used HTML elements C.

All the content is contained in an **article** element C. In short, **article** defines a distinct piece of content. The **article** element is the appropriate choice to surround the content for our basic page, but not necessarily for every page you'll write. You'll learn more about when to use **article** in Chapter 3.

Next is a heading D. HTML provides you six heading levels, **h1**–**h6**, with **h1** being the most important. An **h2** is a subheading of an **h1**, an **h3** is a subheading of an **h2**, and so on, just like when you type a document with various headings in a word processor.

Every HTML page should have an **h1** (or more, depending on your content), so marking up our heading with **h1** was the obvious choice. The heading elements **h1**–**h6** are covered more in Chapter 3.

Next, you have an image E. The **img** element is the primary choice for displaying an image, so again, there was no debate about which element was appropriate. The **alt** attribute provides text that displays if the image doesn't load or if the page is viewed in a text-only browser. You'll learn more about images in Chapter 5.

The paragraph is marked up with—surprise—the **p** element F. Just as in printed materials, a paragraph can contain a single sentence or several sentences. If our page needed another paragraph, you'd simply add another **p** element after the first one.

There are two elements nested within our paragraph that define the meaning of bits of text: **em** and **a** F. These are examples of the numerous phrasing content elements that HTML5 provides, the majority of which improve the semantics of paragraph text. As mentioned, those, along with **p**, are discussed in Chapter 4.

The **em** element means "stress emphasis." In the case of our page, it emphasizes the amazement the flowers induced F. Remember that because HTML describes the meaning of content, **em** dictates semantic, not visual, emphasis even though it's common to render **em** text in italics.

Finally, the basic page defines a link to another page with the **a** element ("anchor"), which is the most powerful element in all of HTML because it makes the Web, the Web: It links one page to another page or resource, and links one part of a page to another part of a page (either the same page or a different one). In the example, it signifies that the text "Blue Flax" is a link to a page on Wikipedia G.

F The **p** element may contain other elements that define the semantics of phrases within a paragraph. The **em** and **a** elements are two examples.

```
<p>I am continually <em>amazed</em> at
→ the beautiful, delicate <a href="http://
→ en.wikipedia.org/wiki/Linum_lewisii"
→ rel="external" title="Learn more about
→ Blue Flax">Blue Flax</a> that somehow
→ took hold in my garden. They are awash in
→ color every morning, yet not a single
→ flower remains by the afternoon. They are
→ the very definition of ephemeral.</p>
```

G This **a** element defines a link to the Wikipedia page about Blue Flax. The optional **rel** attribute adds to the semantics by indicating that the link points to another site. The link works without it, though. The optional **title** attribute enhances the semantics of the **a** by providing information about the linked page. It appears in the browser when a user hovers over the link.

```
<a href="http://en.wikipedia.org/wiki/Linum_
→ lewisii" rel="external" title="Learn more
→ about Blue Flax">Blue Flax</a>
```

Pretty easy, right? Once you've learned more about the HTML elements available to you, choosing the right ones for your content is usually a straightforward task. Occasionally, you'll come across a piece of content that reasonably could be marked up in more than one way, and that's OK. There isn't always a right and wrong way, just most of the time.

Lastly, HTML5 doesn't try to provide an element for every type of content imaginable, because the language would become ungainly. Instead, it takes a practical, real-world stance, defining elements that cover the vast majority of cases.

Part of HTML's beauty is that it's simple for anyone to learn the basics, build some pages, and grow their knowledge from there. So, although there are approximately 100 HTML elements, don't let that number scare you. There's a core handful you'll find yourself using time and again, while the remaining ones are reserved for less common cases. You've already learned the basics of several common elements, so you're well on your way.

Why Semantics Matter

Now that you know the importance of semantic HTML and have seen it in action, you need to know the reasons *why* it's important.

Here are some of the most important reasons (this isn't an exhaustive list), some of which we've touched on already:

- Improved accessibility and interoperability (content is available to assistive technologies for visitors with disabilities, and to browsers on desktop, mobile, tablet, and other devices alike)
- Improved search engine optimization (SEO)
- (Typically) lighter code and faster pages
- Easier code maintenance and styling

If you aren't familiar with *accessibility*, it's the practice of making your content available to all users, regardless of their capabilities (see www.w3.org/standards/webdesign/accessibility). Tim Berners-Lee, inventor of the Web, famously said, "The power of the Web is in its universality. Access by everyone regardless of disability is an essential aspect."

Any device with a browser is capable of displaying HTML, since it's just text. The means by which a user accesses content can vary, however. For instance, sighted users view the content, whereas a visually impaired user may increase the page or font size or use a screen reader, software that reads content aloud to them (one example of assistive technology). In some cases, screen readers announce the type of HTML element surrounding content in order to give the user context for what's to follow. For example, the user may be told that a list has been encountered before the individual list items are read aloud. Similarly, users are told when a link is encountered so they can decide whether to follow it.

Screen reader users can navigate a page in a variety of ways, such as jumping from one heading to the next via a keyboard command. This allows them to glean the key topics of a page and listen in more detail to the ones that interest them rather than having to listen to the entire page sequentially.

So you can see why good semantics make a marked difference to users with disabilities.

SEO—that is, your page's ranking in search engine results—can improve, because search engines put an emphasis on the portions of your content that are marked up in a particular way. For instance, the headings tell the search engine spider the primary topics of your page, helping the search engine determine how to index your page's content.

As you progress through the book, you'll learn why good semantics can make your code more efficient and easier to maintain and style.

Markup: Elements, Attributes, and Values

Now that you've seen some HTML, let's take a closer look at what constitutes markup.

HTML has three principal markup components: *elements*, *attributes*, and *values*. You've seen examples of each in our basic page.

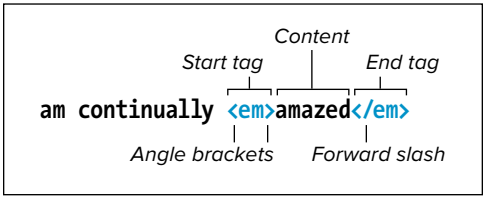
Elements

Elements are like little labels that describe the different parts of a Web page: "This is a heading, that thing over there is a paragraph, and that group of links is navigation." We discussed a few elements in the previous section. Some elements have one or more attributes, which further describe the purpose and content (if any) of the element.

Elements can contain text and other elements, or they can be empty. A non-empty element consists of a *start tag* (the element's name and attributes, if any, enclosed in less-than and greater-than signs), the content, and an *end tag* (a forward slash followed by the element's name, again enclosed in less-than and greater-than signs) A.

An *empty element* (also called a *void element*) looks like a combination start and end tag, with an initial less-than sign, the element's name followed by any attributes it may have, an optional space, an optional forward slash, and the final greater-than sign, which is required B.

The space and forward slash before the end of an empty element are optional in HTML5. It's probably fair to say that those of us who previously coded in XHTML, which requires the forward slash to close an empty element, tend to use it in HTML5 too, though certainly others have dropped it. I include it in my code, but if you choose to omit it from yours, the page will behave



A Here is a typical HTML element. The *start tag* and *end tag* surround the text the element describes. In this case, the word "amazed" is emphasized, thanks to the `em` element. It's customary to type your element tags in lowercase.

```
<img src="blueflax.jpg" width="300" height="175" alt="Blue Flax (Linum lewisii)" />
                                                                    A space and forward slash
```

B Empty elements, like `img` shown here, do not surround any text content (the `alt` attribute text is part of the element, not surrounded by it). They have a single tag which serves both to open and close the element. The space and forward slash at the end are optional in HTML5, but it's common to include them. However, the `>` that completes the element is required.

exactly the same. Whichever way you go, I recommend doing it consistently.

It's customary to type your element names in all lowercase, although HTML5 isn't picky here either, allowing uppercase letters instead. However, it's rare to find someone nowadays who codes in uppercase, so unless the rebel in you just can't resist, I don't recommend it. It's looked upon as a dated practice.

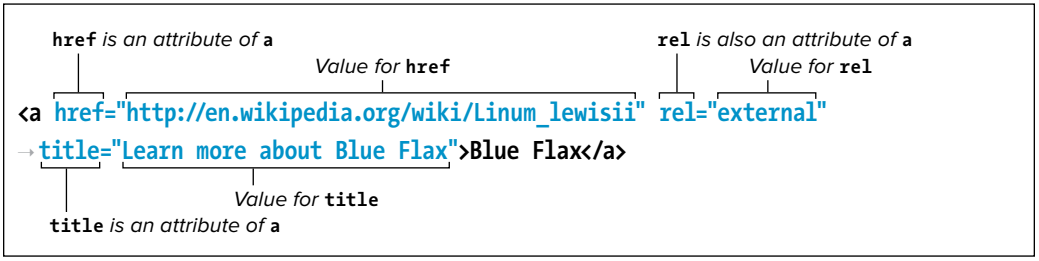
Attributes and Values

Attributes contain information about the content in the document, as opposed to being content itself (C and D). In HTML5, an attribute's value may optionally be enclosed in quotation marks, but it's customary to include them, so I recommend you always do so. And just as with element names, I recommend you type your attribute names in lowercase.

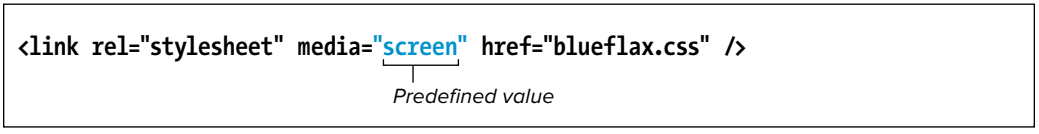
Although you'll find details about acceptable values for most attributes in this book, let me give you an idea of the kinds of values you'll run into as you progress.

Some attributes can accept any value, others are more limited. Perhaps the most common are those that accept enumerated or predefined values. In other words, you must select a value from a standard list of choices E. Be sure to write enumerated values in all lowercase letters.

C Here is a **label** element (which associates a text label with a form field) with a simple attribute-value pair. Attributes are always located inside an element's start tag. It's customary to enclose them in quotation marks.



D Some elements, like **a** shown here, can take one or more attributes, each with its own value. The order is not important. Separate each attribute-value pair from the next with a space.



E Some attributes only accept specific values. For example, the **media** attribute in the **link** element can be set to **all**, **screen**, or **print**, among others, but you can't just make up a value for it like you can with the **title** attribute.

Many attributes require a number for their value, particularly those describing size and length. A numeric value never includes units, just the number. Where units are applicable, as in the width and height of an image or video, they are understood to be pixels.

Some attributes, like **href** and **src**, reference other files and thus must contain values in the form of a URL, or Uniform Resource Locator, a file's unique address on the Web. You'll learn more about URLs in the "URLs" section of this chapter.

Parents and Children

If one element contains another, it is considered to be the parent of the enclosed, or child, element. Any elements contained in the child element are considered descendants of the outer, parent element F. You can actually create a family tree of a Web page that shows the hierarchical relationships between each element on the page and that uniquely identifies each element.

This underlying, family tree-like structure is a key feature of HTML code. It facilitates both styling elements (which you'll begin learning about in Chapter 7) and applying JavaScript behavior to them.

It's important to note that when elements contain other elements, each element must be properly nested, that is, fully contained within its parent. Whenever you use an end tag, it should correspond to the last unclosed start tag. In other words, first open element 1, then open element 2, then close element 2, and then close element 1 G.

```
<article>
   <h1>The Ephemeral Blue Flax</h1>
   <img src="blueflax.jpg"... />
   <p>... continually <em>amazed</em> ... delicate <a ...>Blue Flax</a> ...</p>
</article>
```

F The **article** element is parent to the **h1**, **img**, and **p** elements. Conversely, the **h1**, **img**, and **p** elements are children (and descendants) of the **article**. The **p** element is parent to both the **em** and **a** elements. The **em** and **a** are children of the **p** and also descendants (but not children) of the **article**. In turn, **article** is their ancestor.

Correct (no overlapping lines)

```
<p>... continually <em>amazed</em> ...</p>
<p>... continually <em>amazed ...</p></em>
```

Incorrect (the sets of tags cross over each other)

G Elements must be properly nested. If you open **p** and then **em**, you must close **em** before you close **p**.

A Web Page's Text Content

The text contained within elements is perhaps a Web page's most basic ingredient. If you've ever used a word processor, you've typed some text. Text in an HTML page, however, has some important differences.

First, when a browser renders HTML it collapses extra spaces or tabs into a single space and either converts returns and line feeds into a single space or ignores them altogether (A and B).

Next, HTML used to be restricted to ASCII characters—basically the letters of the English language, numerals, and a few of the most common symbols. Accented characters (common to many languages of Western Europe) and many everyday symbols had to be created with special character references like **é** (for é) or **©** (for ©). See a full list at www.elizabethcastro.com/html/extras/entities.html.

Unicode mitigates a lot of issues with special characters. It's standard practice to encode pages in UTF-8, as in the basic page C, and save HTML files with the same encoding (see "Saving Your Web Page" in Chapter 2). I recommend you do the same.

Because Unicode is a superset of ASCII—it's everything ASCII is, and a lot more—Unicode-encoded documents are compatible with existing browsers and editors, except particularly old ones. Browsers that don't understand Unicode will interpret the ASCII portion of the document properly, while browsers that do understand Unicode will display the non-ASCII portion as well. Even so, it's still common to use character references at times, such as for the copyright symbol since it's easy to both remember and type **©** A.

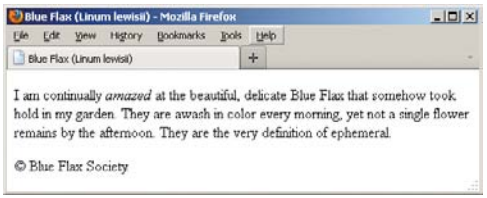
A A page's text content (highlighted) is mostly anything besides the markup. In this example, note that each sentence is separated by at least one carriage return, and some words are separated by several spaces (just to emphasize the point about collapsing returns and spaces). Also, it includes a special character reference (**©**) for the copyright symbol to ensure that it is properly displayed no matter the encoding in which you save this document.

```
<p>I am continually <em>amazed</em> at the
→ beautiful,    delicate Blue Flax that
→ somehow took hold in my garden.

They are awash in       color every
→ morning, yet not a single flower
→ remains by the afternoon.



They are the very definition of
→ ephemeral.</p>
<p>&copy; Blue Flax Society.</p>
```



B Note that when you view the document with a browser, the extra returns and spaces are ignored and the character reference is replaced by the corresponding symbol (©).

C Specify your document's character encoding directly after the **head** start tag. The **charset** attribute sets the encoding type.

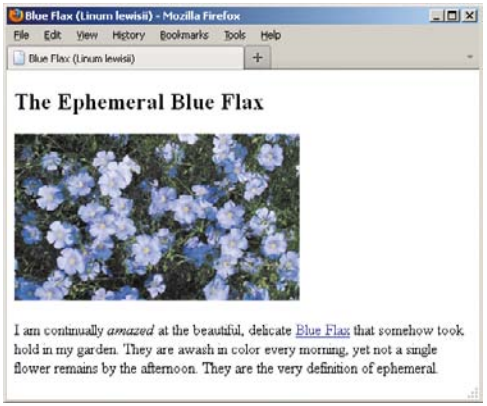
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>Blue Flax (Linum lewisii)</title>
</head>
<body>
…
</body>
</html>
```

A In our basic HTML document, there is a reference to an image file called **blueflax.jpg**, which the browser will request, load, and display when it loads the rest of the page. The page also includes a link to another page about Blue Flax.

```
...
<article>
    <h1>The Ephemeral Blue Flax</h1>

    <img src="blueflax.jpg" width="300"
    → height="175" alt="Blue Flax (Linum
    → lewisii)" />

    <p>I am continually <em>amazed</em> at
    → the beautiful, delicate <a href=
    → "http://en.wikipedia.org/wiki/Linum_
    → lewisii" rel="external" title="Learn
    → more about the Blue Flax">Blue Flax
    → </a> that somehow took hold in my
    → garden. They are awash in color every
    → morning, yet not a single flower
    → remains by the afternoon. They are the
    → very definition of ephemeral.</p>
</article>
...
```



B Images and other non-text content are referenced from a Web page, and the browser displays them together with the text.

Links, Images, and Other Non-Text Content

Of course, part of what makes the Web so vibrant are the links from one page to another, and the images, videos, music, animations, and more. Instead of actually enclosing the external files, such as videos, in the HTML file, these files are saved independently and are simply referenced from within the page A. Since the reference is nothing more than text, the HTML file remains nearly universally accessible.

Browsers can handle links and images (except in text-only browsers) without skipping a beat B. However, they can't necessarily handle every other kind of file. If you reference a file that your visitor's browser doesn't understand, the browser will often try to find a plugin or helper application—some appropriate program on the visitor's computer—that is capable of opening that kind of file.

You can also give browsers extra information about how to render content with a plugin if it requires it, or how to download the plugin if the visitor doesn't already have it on their computer.

All this business about downloading and installing plugins disrupts a user's experience on your site, assuming they stick around. Plugins can also introduce performance problems because they aren't a native part of the browser.

Flash, for instance, has been the most widespread plugin for years. No doubt you've watched an online video played through Flash at some point and experienced your computer slow down or the occasional browser crash (or both).

HTML5 attempts to mitigate many of these issues by introducing native media playback in the browser via the **`audio`** and **`video`** elements. Unfortunately, there's been debate among the browser vendors about which media formats to support, so you can't always do away with plugins altogether yet. But it's a start.

You'll learn more about images in Chapter 5, and go over plugins, HTML5's media elements, and more in Chapter 17.

File Names

Like any other text document, a Web page has a file name that identifies itself to you, your visitors, and your visitors' Web browsers. There are a few tips to keep in mind when assigning file names to your Web pages that will help you organize your files, make it easier for your visitors to find and access your pages, ensure that their browsers view the pages correctly, and improve SEO (A and B).

Use Lowercase File Names

Since the file name you choose for your Web page determines what your visitors will have to type in order to get to your page, you can save your visitors from inadvertent typos (and headaches) by using only lowercase letters in your file names. It's also a big help when you create links between your pages yourself. If all your file names have only small letters, it's just one less thing you'll have to worry about.

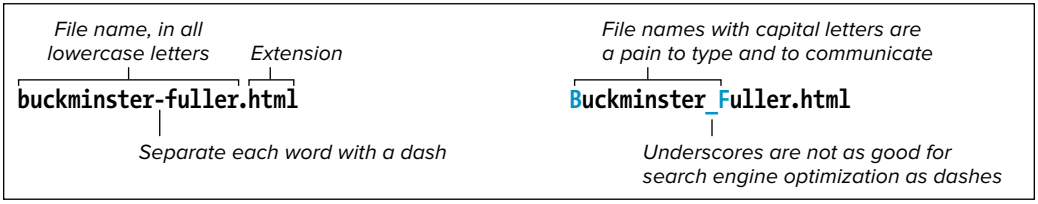
Separate Words with a Dash

Never include spaces between words in your file names. Instead, use a dash, for example, **`company-history.html`** and **`my-favorite-movies.html`**. You'll come across the occasional site that uses underscores ("_") instead, but they aren't recommended, because dashes are preferred by search engines.

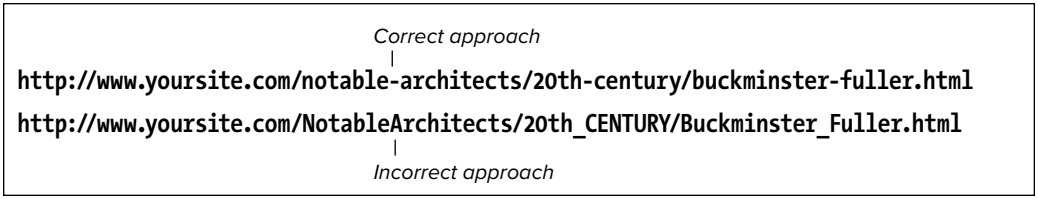
Use the Proper Extension

The principal way a browser knows that it should read a text document as a Web page is by looking at its extension. Although **`.htm`** also works, **`.html`** is customary, so I recommend you use that as your extension. If the page has some other extension, such as **`.txt`**, the browser will treat it as text and show all your nice code to the visitor.

TIP Be aware that neither Mac OS nor Windows always reveals a document's real extension. Change your folder options, if necessary, so you can see extensions.



A Remember to use all lowercase letters for your file names, separate words with a dash, and add the .html extension. Mixing upper- and lowercase letters makes it harder for your visitors to type the proper address and find your page.



B Use all lowercase letters and dashes for your directories and folders as well. The key is consistency. If you don't use uppercase letters, your visitors (and you) don't have to waste time wondering "Now, was that a capital B or a small one?"

URLs

Uniform Resource Locator, or URL, is a fancy name for address. It contains information about where a file is and what a browser should do with it. Each file on the Internet has a unique URL.

The first part of the URL is called the *scheme*. It tells the browser how to deal with the file that it is about to open. The most common scheme you will see is HTTP, or Hypertext Transfer Protocol. It is used to access Web pages A.

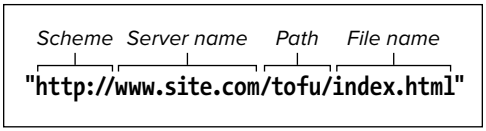
The second part of the URL is the name of the server where the file is located, followed by the path that leads to the file, and the file's name itself. Sometimes, a URL omits a file name and ends with a path, which may or may not include a trailing forward slash B. In this case, the URL refers to the default file in the last directory in the path, typically called **`index.html`**.

Other common schemes are **`https`**, for secure Web pages; **`ftp`** (File Transfer Protocol), for downloading files C; **`mailto`**, for sending email D; and **`file`**, for accessing files on a local hard disk or local file sharing networks (you won't have occasion to use the **`file`** scheme very often, if at all) E.

A scheme is generally followed by a colon and two forward slashes. **`mailto`** and **`news`** are exceptions; these take only a colon.

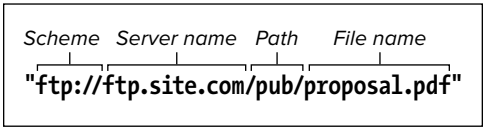
Notice that the **`file`** scheme is followed by a colon and three slashes. That's because the host, which in other schemes goes between the second and third slashes, is assumed to be the local computer. Always type schemes in lowercase letters.

Of these schemes, you will use **`http`** and **`mailto`** most frequently. The others are for specialized cases.



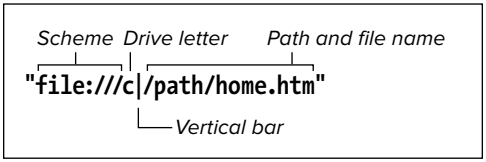
A Your basic URL contains a scheme, server name, path, and file name.

B A URL with a trailing forward slash and no file name points to the default file in the last directory named (in this case, the **`tofu`** directory). The most common default file name is **`index.html`**. So, this URL and the one from the previous example point to the same page.

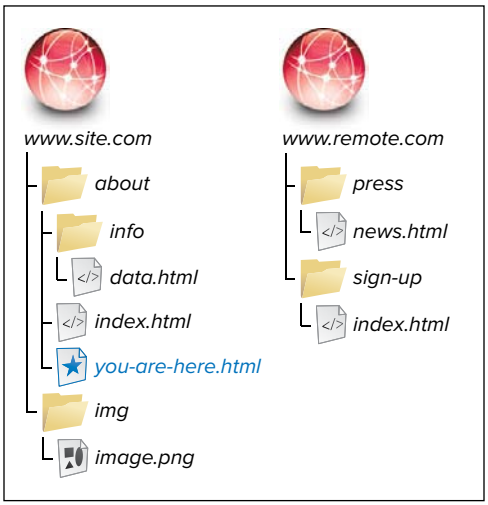


C When the user clicks this URL, the browser will begin an FTP transfer of the file **`proposal.pdf`**.

D A URL for an email address includes the **`mailto`** scheme followed by a colon but no forward slashes, and then the email address itself.



E To reference a file on a local Windows machine, use the **`file`** scheme. For Macintosh, use **`file:///Harddisk/path/filename`**. No vertical bar is required. (This sometimes works for Windows as well.)



F The document that contains the URLs (`you-are-here.html` in this case) is the reference point for relative URLs. In other words, relative URLs are relative to that file's location on the server. Absolute URLs will work no matter where they are located, because they always contain the full URL to a resource.

Absolute URLs

URLs can be either absolute or relative. An *absolute URL* shows the entire path to the file, including the scheme, the server name, the complete path, and the file name itself F. An absolute URL is analogous to a complete street address, including name, street and number, city, state, zip code, and country. No matter where a letter is sent from, the post office will be able to find the recipient. In terms of URLs, this means that the location of the absolute URL itself has no bearing on the location of the actual file referenced—whether it is in a Web page on your server or another server, an absolute URL to a particular file will look exactly the same.

When you're referencing a file from someone else's server, you'll always use an absolute URL. You'll also need to use absolute URLs for FTP sites or, generally, any kind of URL that doesn't use an HTTP protocol.

Table 1.1 describes how you would access various files from `you-are-here.html`—both those on the same site (site.com) as the page and on another site (remote.com)—as a way of illustrating the difference between relative and absolute URLs.

TABLE 1.1 Absolute URLs vs. Relative URLs

File name	Absolute URL (can be used anywhere)	Relative URL (only works in you-are-here.html)
index.html	http://www.site.com/about/index.html	index.html
data.html	http://www.site.com/about/info/data.html	/info/data.html
image.png	http://www.site.com/img/image.png	../img/image.png
news.html	http://www.remote.com/press/news.html	(none: use absolute)
index.html	http://www.remote.com/sign-up/index.html	(none: use absolute)

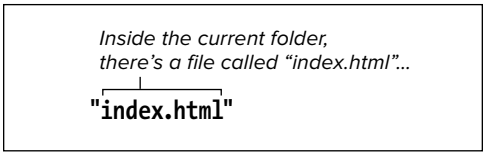
Relative URLs

To give you directions to my neighbor's house, instead of giving her complete address I might just say, "it's three doors down on the right." This is a relative address—where it points to depends on where the information originates. With the same information in a different city, you'd never find my neighbor.

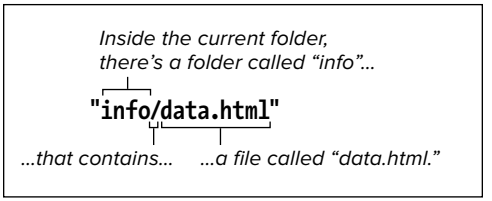
In the same way, a relative URL describes the location of the desired file with reference to the location of the file that contains the URL reference itself. So, you might have the URL say something like "link to the xyz page that's in the same directory as this page."

The relative URL for a file that is in the same directory as the current page (that is, the one containing the URL in question) is simply the file name and extension G. You create the URL for a file in a subdirectory of the current directory by typing the name of the subdirectory followed by a forward slash and then the name and extension of the desired file H.

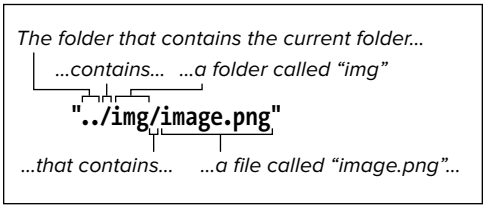
To reference a file in a directory at a higher level of the file hierarchy, use two periods and a forward slash I. You can combine and repeat the two periods and forward slash to reference any file on the same server or drive as the current file.



G The relative URL to link to a file in the same folder (see F). Only the file's name and extension are required in the URL, rather than preceding those with **`http://www.site.com/about/`** (the folder in which both files live).



H To reference a file (**`data.html`**, in this example) that is within a folder inside the current folder (see F), add the sub-folder's name and a forward slash in front of the file name.



I This file, as you can see in F, is in a folder (*img*) that sits alongside the current folder (*about*) in the site's root directory. In that case, you use two periods and a forward slash to go up a level, and then note the subdirectory, followed by a forward slash, followed by the file name. (In normal practice, you'd choose a more descriptive image file name than **`image.png`**, which is deliberately generic for the example.)

Alternatively, if your files are on a Web server, you can avoid cumbersome file paths such as **`../../img/family/vacation.jpg`** by first jumping straight to your site's root and then drilling down from there to the targeted file. A single forward slash at the beginning achieves this, so the *root relative* URL in this case would be **`/img/family/vacation.jpg`** (assuming the **`img`** folder sits in the site's root folder, which is customary). Again, this only works on a Web server, like at the hosting provider that serves your site or one you're running locally on your machine (Apache is the most popular choice for that).

If you aren't developing your site locally on a server, then generally you'll want to use relative URLs (except when pointing to files on someone else's server, of course). They'll make it easy to move your pages from a local system to a server. As long as the relative position of each file remains constant, you won't have to change any of the paths, so the links will work correctly.

Key Takeaways

The basics of HTML and some key best practices provide the foundation for building effective Web sites. Let's revisit the key takeaways:

- A Web page is primarily made up of three components: text content, references to other files, and markup.
- HTML markup is composed of elements, attributes, and values.
- It's customary to write your HTML in all lowercase (DOCTYPE is an exception), surround your attribute values with quotes, and close empty elements with a space and a forward slash (/).
- Always begin your HTML documents with the DOCTYPE declaration:

  ```
  <!DOCTYPE html>
  ```

- A page's content goes in the **body** element. Instructions primarily intended for the browser and search engines are before that, mostly in the **head**.
- Mark up your content with semantic HTML and without regard for how it should appear in a browser.
- Semantic HTML improves accessibility and can make your site more efficient, and easier to maintain and style.
- CSS controls the presentation of HTML content.
- Each browser's own style sheet dictates the default presentation of HTML. You can overwrite these rules with your own CSS.
- Create file and folder names in all lowercase, and separate words with a dash instead of a space or underscore.

Next you'll learn about how to work with Web page files.