FUTURE VISION BIE

One Stop for All Study Materials

& Lab Programs

Future Vision

By K B Hemanth Raj

Scan the QR Code to Visit the Web Page

Or

Visit : https://hemanthrajhemu.github.io

Gain Access to All Study Materials according to VTU,
CSE – Computer Science Engineering,
ISE – Information Science Engineering,
ECE - Electronics and Communication Engineering
& MORE...

Join Telegram to get Instant Updates: https://bit.ly/VTU_TELEGRAM

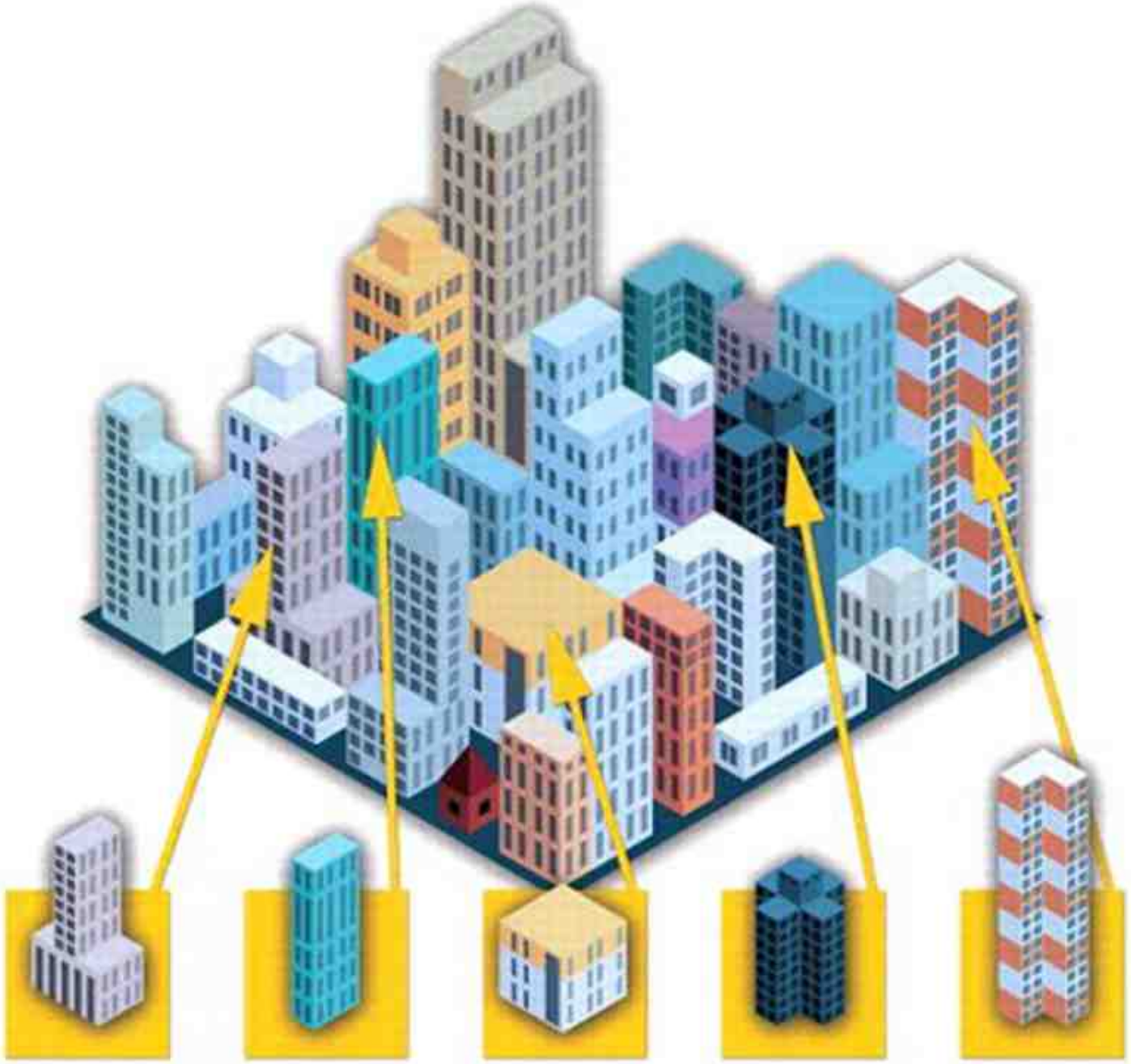
Contact: MAIL: futurevisionbie@gmail.com

INSTAGRAM: www.instagram.com/hemanthraj_hemu/

INSTAGRAM: www.instagram.com/futurevisionbie/

WHATSAPP SHARE: https://bit.ly/FVBIESHARE

FUNDAMENTALS of WEB DEVELOPMENT



RANDY CONNOLLY • RICARDO HOAR

2 Introduction to HTML

CHAPTER OBJECTIVES

In this chapter you will learn . . .

- A very brief history of HTML
- The syntax of HTML
- Why semantic structure is so important for HTML
- How HTML documents are structured
- A tour of the main elements in HTML
- The semantic structure elements in HTML5

This chapter provides an overview of HTML, the building block of all web pages. The massive success and growth of the web has in large part been due to the simplicity of this language. There are many books devoted just to HTML; this book covers HTML in just two chapters. As a consequence, this chapter skips over some details and instead focuses on the key parts of HTML.

2.1 What Is HTML and Where Did It Come from?

Dedicated HTML books invariably begin with a brief history of HTML. Such a history might begin with the ARPANET of the late 1960s, jump quickly to the first public specification of the HTML by Tim Berners-Lee in 1991, and then to HTML's codification by the **World Wide Web Consortium** (better known as the **W3C**) in 1997. Some histories of HTML might also tell stories about the Netscape Navigator and Microsoft Internet Explorer of the early and mid-1990s, a time when intrepid developers working for the two browser manufacturers ignored the W3C and brought forward a variety of essential new tags (such as, for instance, the `<table>` tag), and features such as CSS and JavaScript, all of which have been essential to the growth and popularization of the web.

Perhaps in reaction to these manufacturer innovations, in 1998 the W3C froze the HTML specification at version 4.01. This specification begins by stating:

> *To publish information for global distribution, one needs a universally understood language, a kind of publishing mother tongue that all computers may potentially understand. The publishing language used by the World Wide Web is HTML (from HyperText Markup Language).*

As one can see from the W3C quote, HTML is defined as a **markup language**. A markup language is simply a way of annotating a document in such a way as to make the annotations distinct from the text being annotated. Markup languages such as HTML, Tex, XML, and XHTML allow users to control how text and visual elements will be laid out and displayed. The term comes from the days of print, when editors would write instructions on manuscript pages that might be revision instructions to the author or copy editor. You may very well have been the recipient of markup from caring parents or concerned teachers at various points in your past, as shown in Figure 2.1.

At its simplest, **markup** is a way to indicate *information about the content* that is distinct from the content. This "information about content" in HTML is implemented via **tags** (or more formally, HTML elements, but more on that later). The markup in Figure 2.1 consists of the red text and the various circles and arrows and the little yellow sticky notes. HTML does the same thing but uses textual tags.

In addition to specifying "information about content" many markup languages are able to encode information how to display the content for the end user. These presentation semantics can be as simple as specifying a bold weight font for certain words, and were a part of the earliest HTML specification. Although combining semantic markup with presentation markup is no longer permitted in HTML5, "formatting the content" for display remains a key reason why HTML was widely adopted.

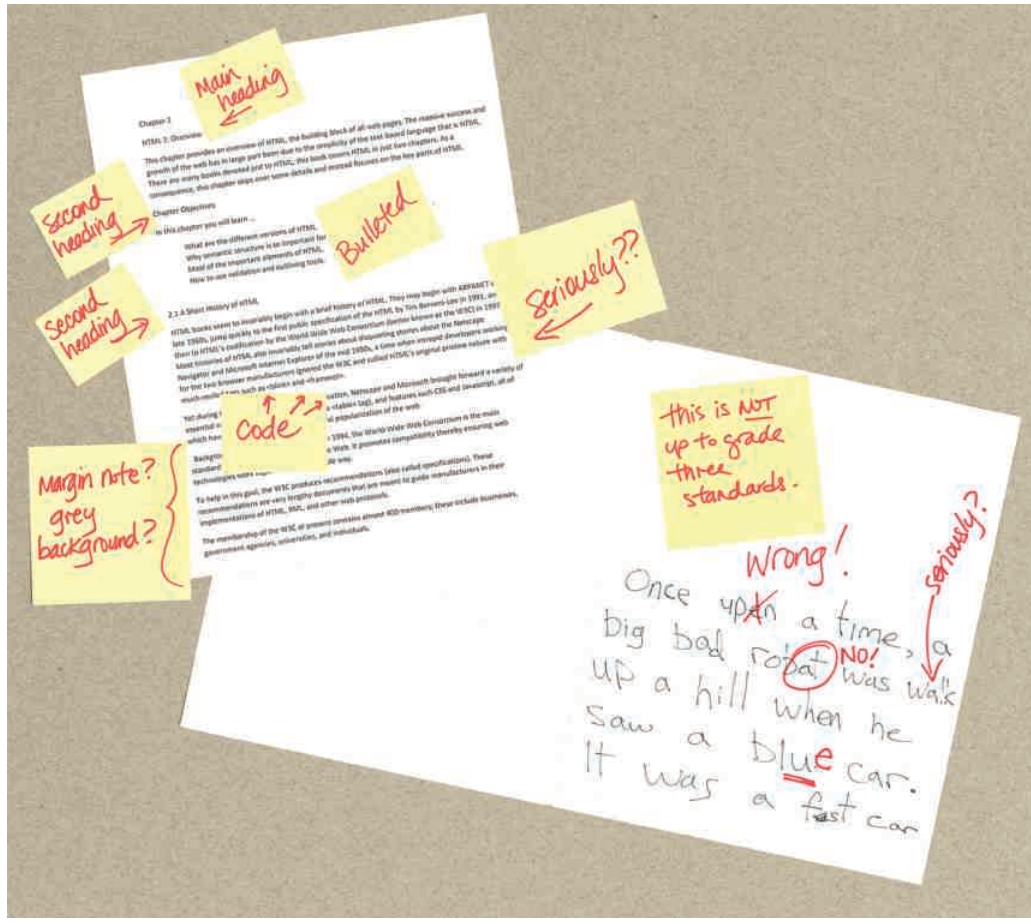


FIGURE 2.1 Sample ad-hoc markup languages

BACKGROUND

Created in 1994, the World Wide Web Consortium (W3C) is the main standards organization for the World Wide Web (WWW). It promotes compatibility, thereby ensuring web technologies work together in a predictable way.

To help in this goal, the W3C produces Recommendations (also called specifications). These Recommendations are very lengthy documents that are meant to guide manufacturers in their implementations of HTML, XML, and other web protocols.

The membership of the W3C at present consists of almost 400 members; these include businesses, government agencies, universities, and individuals.

2.1.1 XHTML

Instead of growing HTML, the W3C turned its attention in the late 1990s to a new specification called XHTML 1.0, which was a version of HTML that used stricter XML (extensible markup language) syntax rules (see Background next).

But why was "stricter" considered a good thing? Perhaps the best analogy might be that of a strict teacher. When one is prone to bad habits and is learning something difficult in school, sometimes a teacher who is more scrupulous about the need to finish daily homework may actually in the long run be more beneficial than a more permissive and lenient teacher.

As the web evolved in the 1990s, web browsers evolved into quite permissive and lenient programs. They could handle sloppy HTML, missing or malformed tags, and other syntax errors. However, it was somewhat unpredictable how each browser would handle such errors. The goal of XHTML with its strict rules was to make page rendering more predictable by forcing web authors to create web pages without syntax errors.

To help web authors, two versions of XHTML were created: XHTML 1.0 Strict and XHTML 1.0 Transitional. The strict version was meant to be rendered

BACKGROUND

Like HTML, XML is a textual markup language. Also like HTML, the formal rules for XML were set by the W3C.

XML is a more general markup language than HTML. It is (and has been) used to mark up any type of data. XML-based data formats (called schemas in XML) are almost everywhere. For instance, Microsoft Office products now use compressed XML as the default file format for the documents it creates. RSS data feeds use XML and Web 2.0 sites often use XML data formats to move data back and forth asynchronously between the browser and the server. The following is an example of a simple XML document:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<art>
  <painting id="290">
    <title>Balcony</title>
    <artist>
      <name>Manet</name>
      <nationality>France</nationality>
    </artist>
    <year>1868</year>
    <medium>Oil on canvas</medium>
  </painting>
</art>
```

By and large, the XML-based syntax rules (called "well-formed" in XML lingo) for XHTML are pretty easy to follow. The main rules are:

(*continued*)

- There must be a single root element.
- Element names are composed of any of the valid characters (most punctuation symbols and spaces are not allowed) in XML.
- Element names can't start with a number.
- Element and attribute names are case sensitive.
- Attributes must always be within quotes.
- All elements must have a closing element (or be self-closing).

XML also provides a mechanism for validating its content. It can check, for instance, whether an element name is valid, or elements are in the correct order, or that the elements follow a proper nesting hierarchy. It can also perform data type checks on the text within an element: for instance, whether the text inside an element called `<date>` is actually a valid date, or the text within an element called `<year>` is a valid integer and falls between, say, the numbers 1950 and 2010. Chapter 17 covers XML in more detail.

by a browser using the strict syntax rules and tag support described by the W3C XHTML 1.0 Strict specification; the transitional recommendation is a more forgiving flavor of XHTML, and was meant to act as a temporary transition to the eventual global adoption of XHTML Strict.

The payoff of XHTML Strict was to be predictable and standardized web documents. Indeed, during much of the 2000s, the focus in the professional web development community was on standards: that is, on limiting oneself to the W3C specification for XHTML.

A key part of the standards movement in the web development community of the 2000s was the use of **HTML validators** (see Figure 2.2) as a means of verifying that a web page's markup followed the rules for XHTML Transitional or Strict. Web developers often placed proud images on their sites to tell the world at large that their site followed XHTML rules (and also to communicate their support for web standards).

Yet despite the presence of XHTML validators and the peer pressure from book authors, actual web browsers tried to be forgiving when encountering badly formed HTML so that pages worked more or less how the authors intended regardless of whether a document was XHTML valid or not.

In the mid-2000s, the W3C presented a draft of the XHTML 2.0 specification. It proposed a revolutionary and substantial change to HTML. The most important was that backwards compatibility with HTML and XHTML 1.0 was dropped. Browsers would become significantly less forgiving of invalid markup. The XHTML 2.0 specification also dropped familiar tags such as `<img>`, `<a>`, `<br>`, and numbered headings such as `<h1>`. Development on the XHTML 2.0 specification dragged on

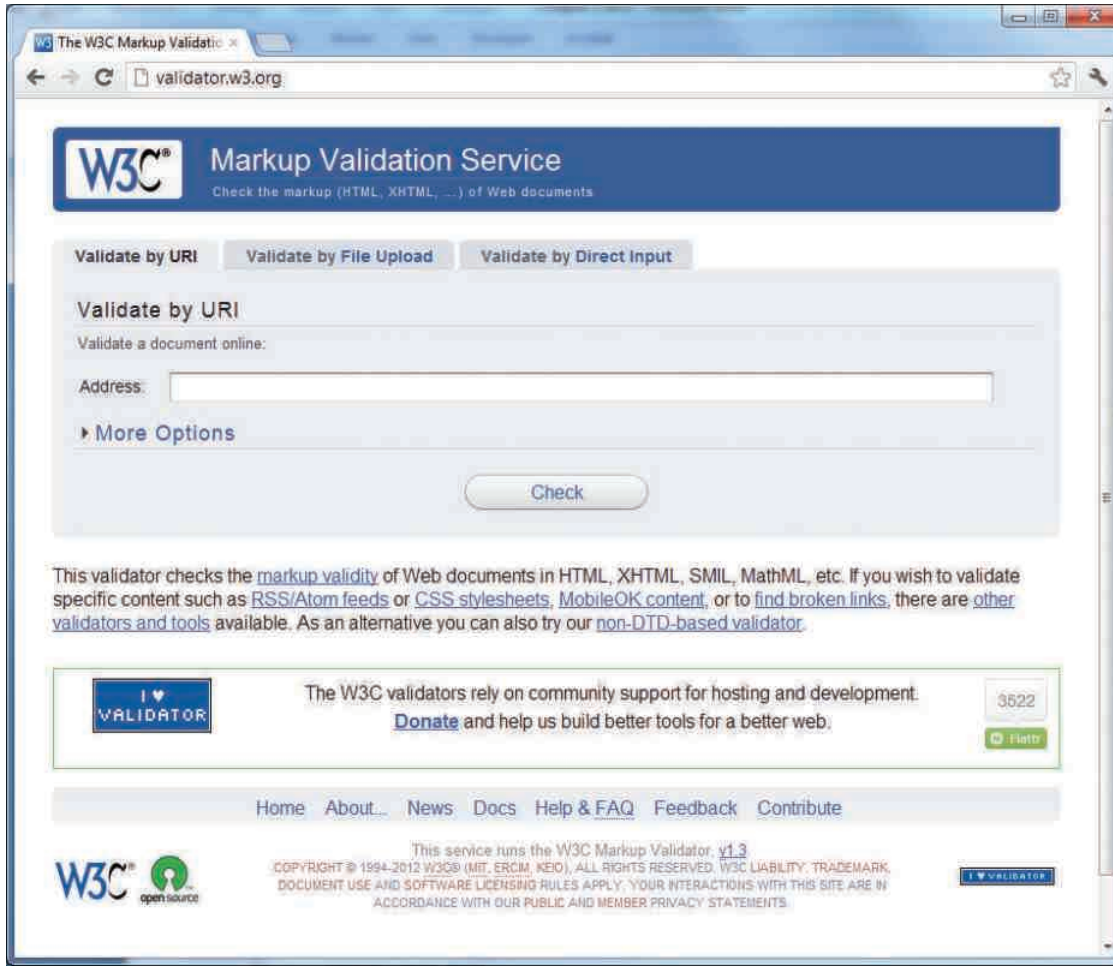


FIGURE 2.2 W3C XHTML validation service

for many years, a result not only of the large W3C committee in charge of the specification, but also of gradual discomfort on the part of the browser manufacturers and the web development community at large, who were faced with making substantial changes to all existing web pages.

2.1.2 HTML5

At around the same time the XHTML 2.0 specification was being developed, a group of developers at Opera and Mozilla formed the WHATWG (Web Hypertext Application Technology Working Group) group within the W3C. This group was not convinced

that the W3C's embrace of XML and its abandonment of backwards-compatibility was the best way forward for the web. Thus the WHATWG charter announced:

> *"The Web Hypertext Applications Technology working group therefore intends to address the need for one coherent development environment for Web applications, through the creation of technical specifications that are intended to be implemented in mass-market Web browsers."*

That is, WHATWG was focused less on semantic purity and more on the web as it actually existed. As well, unlike the large membership of the W3C, the WHATWG group was very small and led by Ian Hickson. As a consequence, the work at WHATWG progressed quickly, and eventually, by 2009, the W3C stopped work on XHTML 2.0 and instead adopted the work done by WHATWG and named it HTML5.

There are three main aims to HTML5:

1. Specify unambiguously how browsers should deal with invalid markup.
2. Provide an open, nonproprietary programming framework (via JavaScript) for creating rich web applications.
3. Be backwards compatible with the existing web.

While parts of the HTML5 are still being finalized, all of the major browser manufacturers have at least partially embraced HTML5. Certainly not all browsers and all versions support every feature of HTML5. This is in fact by design. HTML in HTML5 is now a living language: that is, it is a language that evolves and develops over time. As such, every browser will support a gradually increasing subset of HTML5 capabilities. In late September 2012, the W3C announced that they planned to have the main elements of the HTML5 specification moved to Recommendation status (i.e., the specification would be finalized in terms of features) by late 2014, and the less stable parts of HTML5 moved to HTML5.1 (with a tentative completion date of 2016).

This certainly creates complications for web developers. Does one only use HTML elements that are universally supported by all browsers, or all the newest elements supported only by the most recent browsers, or . . . something in between? This is an interesting question as well for the authors of this textbook. Should we cover only what is supported by the XHTML 1.0 standard or should we cover more of the features in HTML5?

In this text, we have taken the position that HTML5 is not only the future but the present as well. As such, this book assumes that you are using an HTML5 browser. This is not an unreasonable assumption since as of February 2013, a very large majority of web requests are from browsers that have at least partial support of the main features of HTML5 (see Figure 2.3).

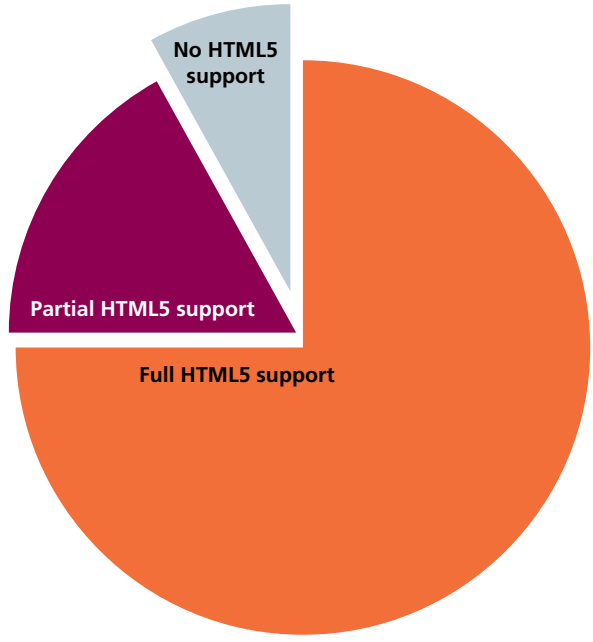


FIGURE 2.3 Browser usage and HTML5 support

2.2 HTML Syntax

The (still) current W3C Recommendation for HTML is the HTML 4.01 specification, which dates back all the way to 1999. In that specification the syntax for marking up documents was defined and centered around using elements and attributes (see Section 2.2.1).

Learning the fundamental concepts and terms that have survived multiple standards is essential in a discipline like web development where specifications, standards, and browsers are constantly evolving.

HANDS-ON EXERCISES

LAB 2 EXERCISE
First Web Page

2.2.1 Elements and Attributes

HTML documents are composed of textual content and HTML elements. The term **HTML element** is often used interchangeably with the term **tag**. However, an HTML element is a more expansive term that encompasses the element name within angle brackets (i.e., the tag) and the content within the tag (though some elements contain no extra content).

An HTML element is identified in the HTML document by tags. A tag consists of the element name within angle brackets. The element name appears in both the beginning tag and the closing tag, which contains a forward slash followed by the

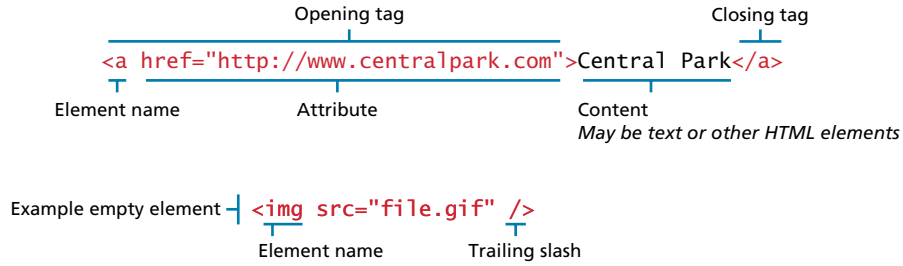


FIGURE 2.4 The parts of an HTML element

element's name, again all enclosed within angle brackets. The closing tag acts like an off-switch for the on-switch that is the start tag.

HTML elements can also contain attributes. An **HTML attribute** is a name=value pair that provides more information about the HTML element. In XHTML, attribute values had to be enclosed in quotes; in HTML5, the quotes are optional, though many web authors still maintain the practice of enclosing attribute values in quotes. Some HTML attributes expect a number for the value. These will just be the numeric value; they will never include the unit.

Figure 2.4 illustrates the different parts of an HTML element, including an example of an empty HTML element. An **empty element** does not contain any text content; instead, it is an instruction to the browser to do something. Perhaps the most common empty element is `<img>`, the image element. In XHTML, empty elements had to be terminated by a trailing slash (as shown in Figure 2.4). In HTML5, the trailing slash in empty elements is optional.

2.2.2 Nesting HTML Elements

Often an HTML element will contain other HTML elements. In such a case, the container element is said to be a parent of the contained, or child, element. Any elements contained within the child are said to be **descendants** of the parent element; likewise, any given child element, may have a variety of **ancestors**.

NOTE

In XHTML, all HTML element names and attribute names had to be lowercase. HTML5 (and HTML 4.01 as well) does not care whether you use upper- or lowercase for element or attribute names. Nonetheless, this book will follow XHTML usage and use lowercase for all HTML names and enclose all attribute values in quotes.

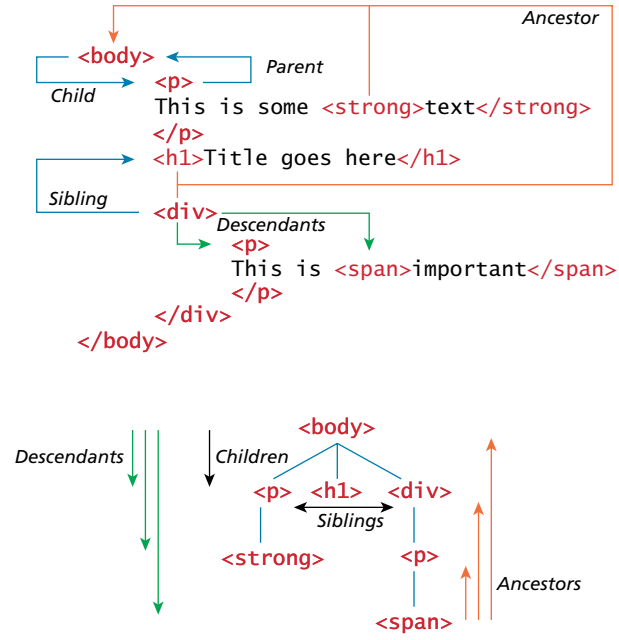


FIGURE 2.5 HTML document outline

This underlying family tree or hierarchy of elements (see Figure 2.5) will be important later in the book when you cover Cascading Style Sheets (CSS) and JavaScript programming and parsing. This concept is called the Document Object Model (DOM) formally, though for now we will only refer to its hierarchical aspects.

In order to properly construct this hierarchy of elements, your browser expects each HTML nested element to be properly nested. That is, a child's ending tag must occur before its parent's ending tag, as shown in Figure 2.6.

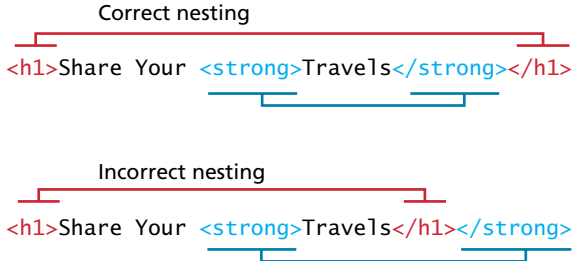


FIGURE 2.6 The proper nesting of HTML elements

2.3 Semantic Markup

In Figure 2.1, some of the yellow sticky note and red ink markup examples are instructions about how the document will be displayed (such as, "main heading" or "bulleted"). You can do the same thing with HTML presentation markup, but this is no longer considered to be a good practice. Instead, over the past decade, a strong and broad consensus has grown around the belief that HTML documents should **only** focus on the structure of the document; information about how the content should look when it is displayed in the browser is best left to CSS (Cascading Style Sheets), a topic introduced in the next chapter, and then covered in more detail in Chapter 5.

As a consequence, beginning HTML authors are often counseled to create semantic HTML documents. That is, an HTML document should not describe how to visually present content, but only describe its content's structural semantics or meaning. This advice might seem mysterious, but it is actually quite straightforward.

Examine the paper documents shown in Figure 2.7. One is a page from the United States IRS explaining the 1040 tax form; another is a page from a textbook

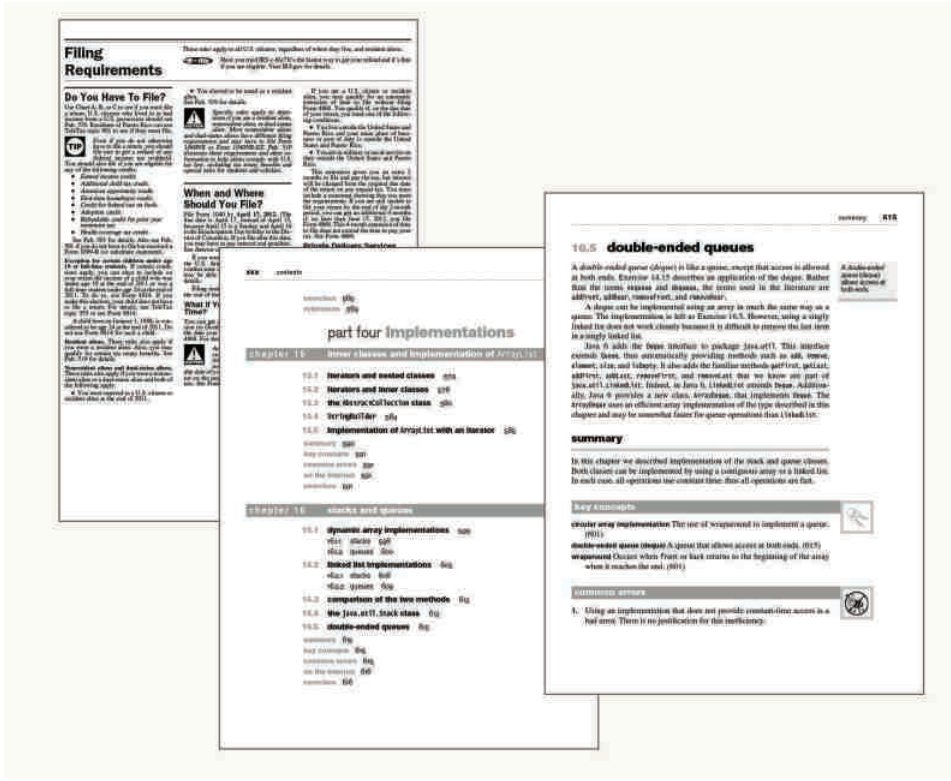


FIGURE 2.7 Visualizing structure

(*Data Structures and Problem Solving Using Java* by Mark Allen Weiss, published by Addison Wesley). In each of them, you will notice that the authors of the two documents use similar means to demonstrate to the reader the structure of the document. That structure (and to be honest the presentation as well) makes it easier for the reader to quickly grasp the hierarchy of importance as well as the broad meaning of the information in the document.

Structure is a vital way of communicating information in paper and electronic documents. All of the tags that we will examine in this chapter are used to describe the basic structural information in a document, such as headings, lists, paragraphs, links, images, navigation, footers, and so on.

Eliminating presentation-oriented markup and writing semantic HTML markup has a variety of important advantages:

- **Maintainability**. Semantic markup is easier to update and change than web pages that contain a great deal of presentation markup. Our students are often surprised when they learn that more time is spent maintaining and modifying existing code than in writing the original code. This is even truer with web projects. From our experience, web projects have a great deal of "requirements drift" due to end user and client feedback than traditional software development projects.
- **Faster**. Semantic web pages are typically quicker to author and faster to download.
- **Accessibility**. Not all web users are able to view the content on web pages. Users with sight disabilities experience the web using voice reading software. Visiting a web page using voice reading software can be a very frustrating experience if the site does not use semantic markup. As well, many governments insist that sites for organizations that receive federal government funding must adhere to certain accessibility guidelines. For instance, the United States government has its own Section 508 Accessibility Guidelines (http://www.section508.gov).

PRO TIP

You can learn about web accessibility by visiting the W3C Web Accessibility initiative website (http://www.w3.org/WAI). The site provides guidelines and resources for making websites more accessible for users with disabilities. These include not just blind users, but users with color blindness, older users with poor eyesight, users with repetitive stress disorders from using the mouse, or even users suffering from ADHD or short-term memory loss. One of the documents produced by the WAI is the Web Content Accessibility Guidelines, which is available via http://www.w3.org/WAI/intro/wcag.php.

- **Search engine optimization**. For many site owners, the most important users of a website are the various search engine crawlers. These crawlers are automated programs that cross the web scanning sites for their content, which is then used for users' search queries. Semantic markup provides better instructions for these crawlers: it tells them what things are important content on the site.

But enough talking about HTML . . . it is time to examine some HTML documents.

2.4 Structure of HTML Documents

Figure 2.8 illustrates one of the simplest *valid* HTML5 documents you can create. As can be seen in the corresponding capture of the document in a browser, such a simple document is hardly an especially exciting visual spectacle. Nonetheless, there is something to note about this example before we move on to a more complicated one.

The `<title>` element (Item 1 in Figure 2.8) is used to provide a broad description of the content. The title is not displayed within the browser window. Instead, the title is typically displayed by the browser in its window and/or tab, as shown in the example in Figure 2.8. The title has some additional uses that are also important to know. The title is used by the browser for its bookmarks and its browser history list. The operating system might also use the page's title, for instance, in the Windows taskbar or in the Mac dock. Perhaps even more important than any of the above reasons, search engines will typically use the page's title as the linked text in their search engine result pages.

For readers with some familiarity with XHTML or HTML 4.01, this listing will appear to be missing some important elements. Indeed, in previous versions, a valid HTML document required additional structure. Figure 2.9 illustrates a more complete HTML5 document that includes these other structural elements as well as some other common HTML elements.

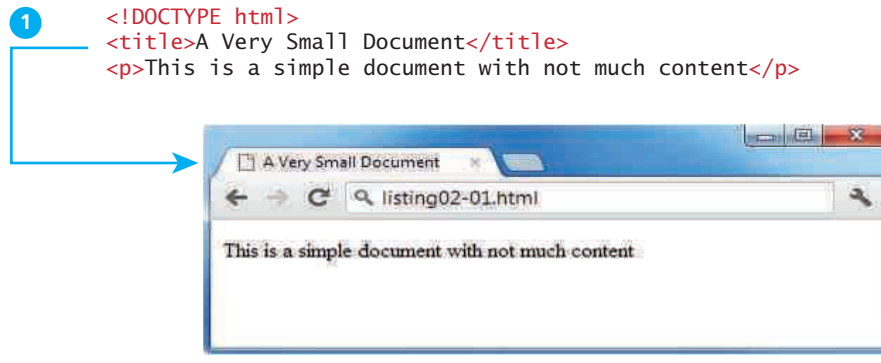


FIGURE 2.8 One of the simplest possible HTML5 documents

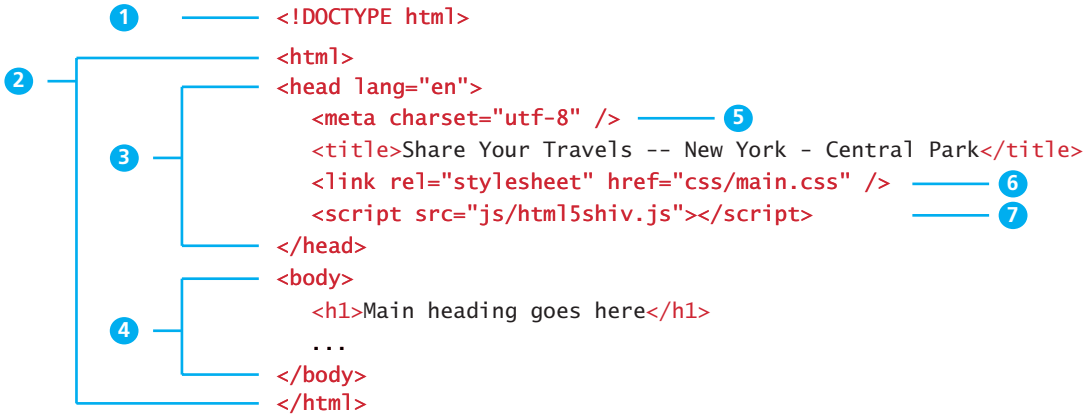


FIGURE 2.9 Structure elements of an HTML5 document

PRO TIP

The `<title>` element plays an important role in search engine optimization (SEO), that is, in improving a page's rank (its position in the results page after a search) in most search engines. While each search engine uses different algorithms for determining a page's rank, the title (and the major headings) provides a key role in determining what a given page is about.

As a result, be sure that a page's title text briefly summarizes the document's content. As well, put the most important content first in the title. Most browsers limit the length of the title that is displayed in the tab or window title to about 60 characters. Chapter 20 goes into far greater detail on SEO.

In comparison to Figure 2.8, the markup in Figure 2.9 is somewhat more complicated. Let's examine the various structural elements in more detail.

2.4.1 DOCTYPE

Item 1 in Figure 2.9 points to the `DOCTYPE` (short for Document Type Definition) element, which tells the browser (or any other client software that is reading this HTML document) what type of document it is about to process. Notice that it does not indicate what version of HTML is contained within the document: it only specifies that it contains HTML. The HTML5 doctype is quite short in comparison to one of the standard doctype specifications for XHTML:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

The XHTML doctype instructed the browser to follow XHTML rules. In the early years of the 2000s, not every browser followed the W3C specifications for HTML and CSS; as support for standards developed in newer browsers, the doctype was used to tell the browser to render an HTML document using the so-called **standards mode** algorithm or render it with the particular browser's older nonstandards algorithm, called **quirks mode**.

Document Type Definitions (DTD) define a document's type for markup languages such as HTML and XML. In both these markup languages, the DTD must appear near the beginning of the document. DTDs have their own syntax that defines allowable element names and their order. The following code from the official XHTML DTD defines the syntax of the `<p>` element:

```
<!ELEMENT p %Inline;>
<!ATTLIST p
 %attrs;
 %TextAlign;
>
```

Within XML, DTDs have largely been replaced by XML schema.

2.4.2 Head and Body

HANDS-ON EXERCISES

LAB 2 EXERCISE
Additional Structure Tags

HTML5 does not require the use of the `<html>`, `<head>`, and `<body>` elements (items 2, 3, and 4 in Figure 2.9). However, in XHTML they were required, and most web authors continue to use them. The `<html>` element is sometimes called the **root element** as it contains all the other HTML elements in the document. Notice that it also has a `lang` attribute. This optional attribute tells the browser the natural language that is being used for textual content in the HTML document, which is English in this example. This doesn't change how the document is rendered in the browser; rather, search engines and screen reader software can use this information.

HTML pages are divided into two sections: the **head** and the **body**, which correspond to the `<head>` and `<body>` elements. The head contains descriptive elements *about* the document, such as its title, any style sheets or JavaScript files it uses, and other types of meta information used by search engines and other programs. The body contains content (both HTML elements and regular text) that will be displayed by the browser. The rest of this chapter and the next chapter will cover the HTML that will appear within the body.

You will notice that the `<head>` element in Figure 2.9 contains a variety of additional elements. The first of these is the `<meta>` element (item 5). The example in Figure 2.9 declares that the character encoding for the document is UTF-8.

NOTE

In HTML5, the use of the `<html>`, `<head>`, and `<body>` elements is optional and even in an older, non-HTML5 browser your page will work fine without them (as the browser inserts them for you). However, for conformity with older standards, this text's examples will continue to use them.

Character encoding refers to which character set standard is being used to encode the characters in the document. As you may know, every character in a standard text document is represented by a standardized bit pattern. The original ASCII standard of the 1950s defined English (or more properly Latin) upper and lowercase letters as well as a variety of common punctuation symbols using 8 bits for each character. UTF-8 is a more complete variable-width encoding system that can encode all 110,000 characters in the Unicode character set (which in itself supports over 100 different language scripts).

Item 6 in Figure 2.9 specifies an external CSS style sheet file that is used with this document. Virtually all commercial web pages created in the last decade make use of style sheets to define the visual look of the HTML elements in the document. Styles can also be defined within an HTML document (using the `<style>` element, which will be covered in Chapter 3); for consistency's sake, most sites place most or all of their style definitions within one or more external style sheet files.

Notice that in this example, the file being referenced (**main.css**) resides within a subfolder called **css**. This is by no means a requirement. It is common practice, however, for web authors to place additional external CSS, JavaScript, and image files into their own subfolders.

Finally, Item 7 in Figure 2.9 references an external JavaScript file. Most modern commercial sites use at least some JavaScript. Like with style definitions, JavaScript code can be written directly within the HTML or contained within an external file. JavaScript will be covered in Chapters 6 and 15 (though JavaScript will be used as well in other chapters).

REMEMBER

Each reference to an external file in an HTML document, whether it be an image, an external style sheet, or a JavaScript file, generates additional HTTP requests resulting in slower load times and degraded performance.

HANDS-ON EXERCISES

LAB 2 EXERCISE
Making Mistakes

2.5 Quick Tour of HTML Elements

HTML5 contains many structural and presentation elements—too many to completely cover in this book. Rather than comprehensively cover all these elements, this chapter will provide a quick overview of the most common elements. Figure 2.10 contains the HTML we will examine in more detail (note that some of the structural tags like `<html>` and `<body>` from the previous section are omitted in this example for brevity's sake). Figure 2.11 illustrates how the markup in Figure 2.10 appears in the browser.

2.5.1 Headings

Item 1 in Figure 2.10 defines two different headings. HTML provides six levels of heading (`h1` through `h6`), with the higher heading number indicating a heading of

```
<body>
   <h1>Share Your Travels</h1>
   <h2>New York - Central Park</h2>
   <p>Photo by Randy Connolly</p>
   <p>This photo of Conservatory Pond in
      <a href="http://www.centralpark.com/">Central Park</a>
      New York City was taken on October 22, 2015 with a
      <strong>Canon EOS 30D</strong> camera.
   </p>
   <img src="images/central-park.jpg" alt="Central Park" />

   <h3>Reviews</h3>
   <div>
      <p>By Ricardo on <time>September 15, 2015</time></p>
      <p>Easy on the HDR buddy.</p>
   </div>

   <div>
      <p>By Susan on <time>October 1, 2015</time></p>
      <p>I love Central Park.</p>
   </div>

   <p><small>Copyright &copy; 2015 Share Your Travels</small></p>
</body>
```

FIGURE 2.10 Sample HTML5 document

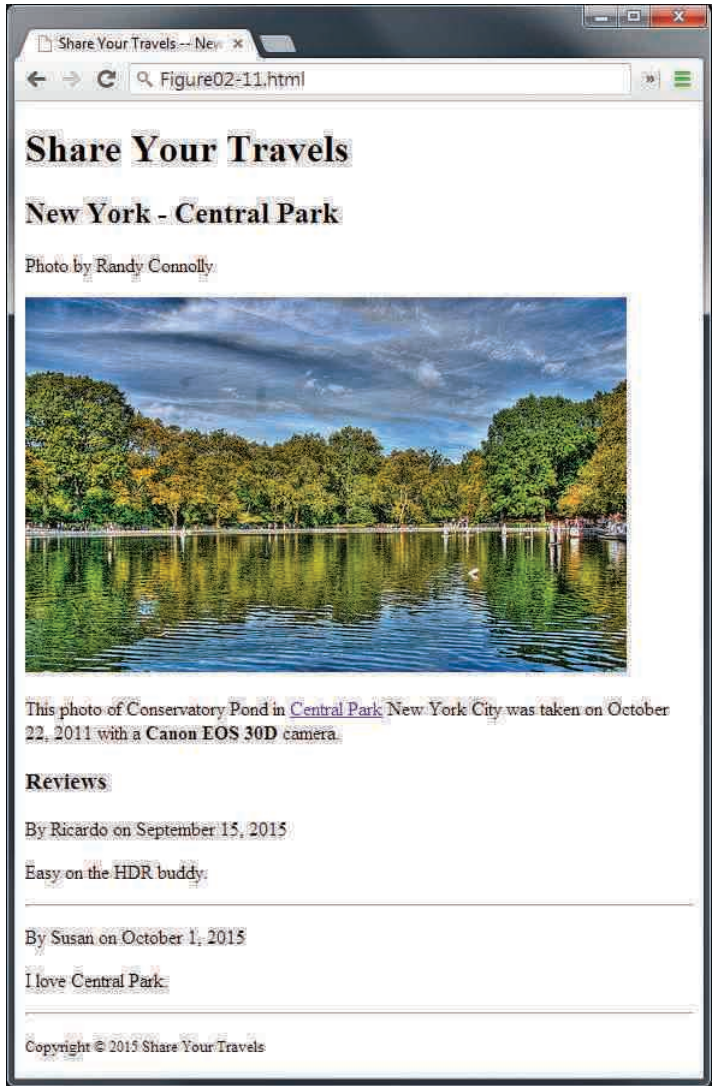


FIGURE 2.11 Figure 2.10 in the browser

less importance. In the real-world documents shown in Figure 2.7, you saw that headings are an essential way for document authors to show their readers the structure of the document.

Headings are also used by the browser to create a **document outline** for the page. Every web page has a document outline. This outline is not something that you see. Rather, it is an internal data representation of the control on the page. This

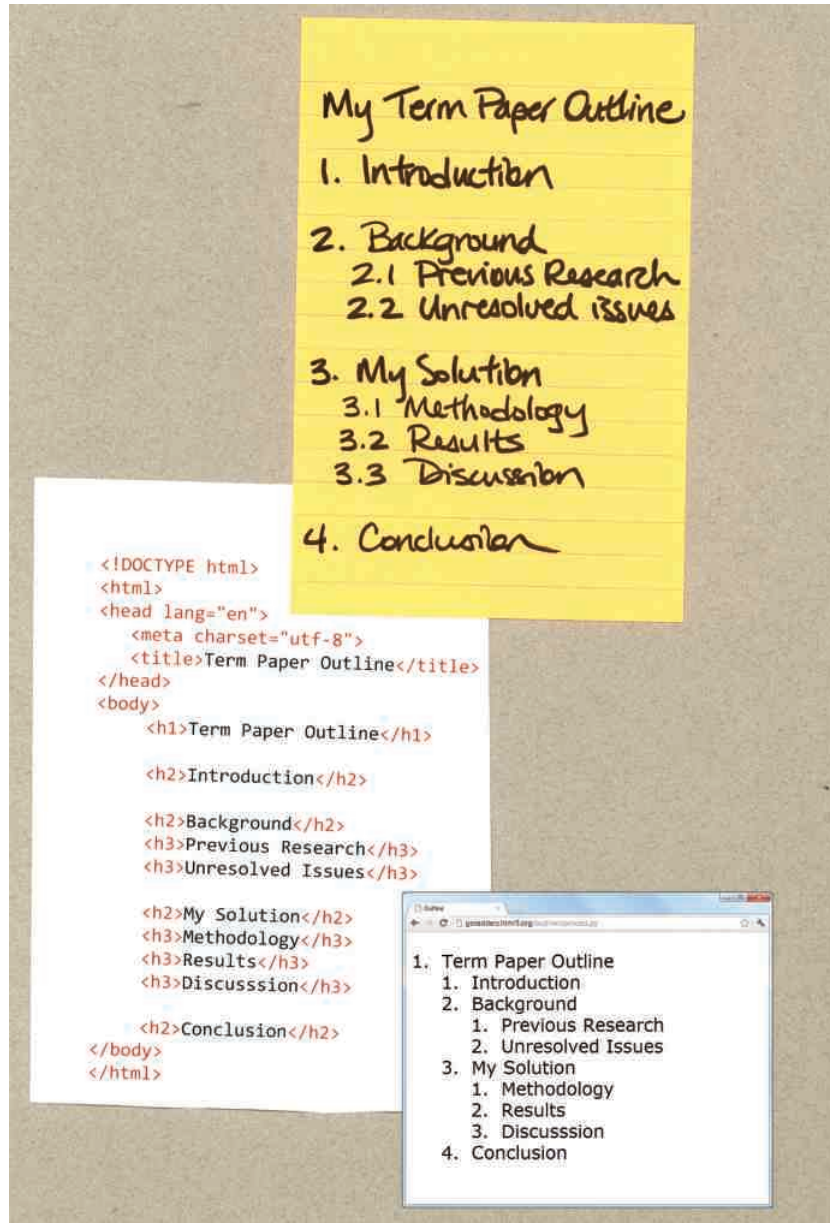


FIGURE 2.12 Example document outlines

document outline is used by the browser to render the page. It is also potentially used by web authors when they write JavaScript to manipulate elements in the document or when they use CSS to style different HTML elements.

FIGURE 2.13 Alternate CSS stylings of the same heading

This document outline is constructed from headings and other structural tags in your content and is analogous to the outlines you may have created for your own term papers in school (see Figure 2.12). There is a variety of web-based tools that can be used to see the document outline. Figure 2.12 illustrates one of these tools; this one is available from http://gsnedders.html5.org/outliner/.

The browser has its own default styling for each heading level. However, these are easily modified and customized via CSS. Figure 2.13 illustrates just some of the possible ways to style a heading.

In practice, specify a heading level that is semantically accurate; do not choose a heading level because of its default presentation (e.g., choosing <h3> because you want your text to be bold and 16pt). Rather, choose the heading level because it is appropriate (e.g., choosing <h3> because it is a third-level heading and not a primary or secondary heading).

PRO TIP

Sometimes it is not obvious what content is a primary heading. For instance, some authors make the site logo an <h1>, the page title an <h2>, and every other heading an <h3> or less. Other authors don't use a heading level for the site logo, but make the page title an <h1>.

There is in fact a website (http://www.h1debate.com) devoted to this debate. At present, about a third of respondents to that site's poll believe the site logo should be an <h1>, while two-thirds believe <h1> should be used for the main heading.

2.5.2 Paragraphs and Divisions

Item 2 in Figure 2.10 defines two paragraphs, the most basic unit of text in an HTML document. Notice that the <p> tag is a container and can contain HTML and other inline HTML elements (the and <a> elements in Figure 2.10). This term refers to HTML elements that do not cause a paragraph break but are part of the regular "flow" of the text and are discussed in more detail in Section 2.5.4.

The indenting on the second paragraph element is optional. Some developers like to use indenting to differentiate a container from its content. It is purely a convention and has no effect on the display of the content.

Don't confuse the <p> element with the line break element (<br/>). The former is a container for text and other inline elements. The line break element forces a line break. It is suitable for text whose content belongs in a single paragraph but which must have specific line breaks: for example, addresses and poems.

Item 6 in Figure 2.10 illustrates the definition of a element. This element is also a container element and is used to create a logical grouping of content (text and other HTML elements, including containers such as <p> and other elements). The element has no intrinsic presentation; it is frequently used in contemporary CSS-based layouts to mark out sections.

HANDS-ON EXERCISES

LAB 2 EXERCISE
Linking

2.5.3 Links

Item 3 in Figure 2.10 defines a hyperlink. Links are an essential feature of all web pages. Links are created using the <a> element (the "a" stands for anchor). A link

PRO TIP

Figure 2.14 shows the book's website (as at the time of writing) and its HTML (as shown in Google's Chrome's Element Inspector, a very handy developer's tool built into the browser).

Notice the many levels of nested `<div>` elements. Some are used by the CSS framework that the site is using to create its basic layout grid (those with `class="grid_##"`); others are given `id` or `class` attributes and are targeted for specific styling in the underlying CSS file.

HTML5 has a variety of new semantic elements (which we will examine later in Section 2.6) that can be used to reduce somewhat the confusing mass of `div` within `div`s within `div`s that is so typical of contemporary web design.

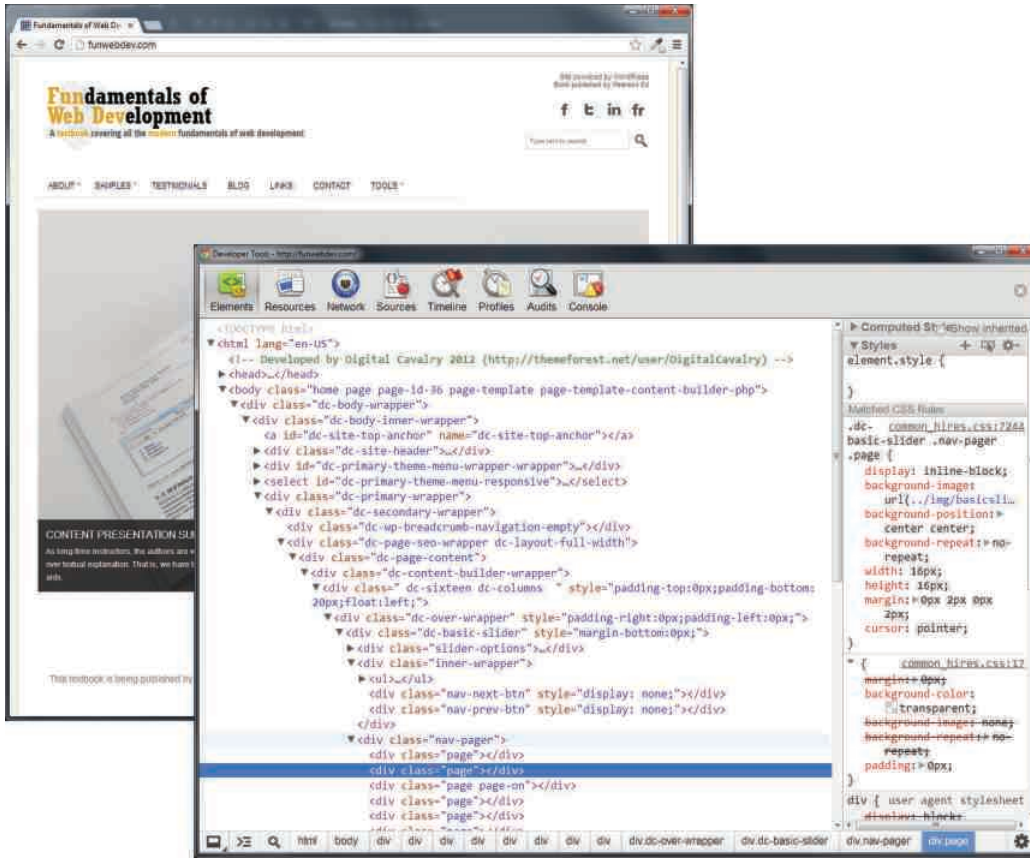


FIGURE 2.14 Using elements to create a complex layout

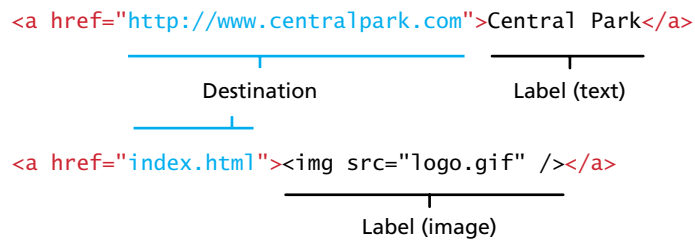


FIGURE 2.15 Two parts of a link

has two main parts: the destination and the label. As can be seen in Figure 2.15, the label of a link can be text or another HTML element such as an image.

You can use the anchor element to create a wide range of links. These include:

- Links to external sites (or to individual resources such as images or movies on an external site).
- Links to other pages or resources within the current site.
- Links to other places within the current page.
- Links to particular locations on another page (whether on the same site or on an external site).
- Links that are instructions to the browser to start the user's email program.
- Links that are instructions to the browser to execute a JavaScript function.
- Links that are instructions to the mobile browser to make a phone call.

Figure 2.16 illustrates the different ways to construct link destinations.

NOTE

Links with the label "Click Here" were once a staple of the web. Today, such links are frowned upon, as they do not provide any information to users as to where the link will take them, are not very accessible, and as a verb "click" is becoming increasingly inaccurate when one takes into account the growth of mobile browsers. Instead, textual link labels should be descriptive. So instead of using the text "Click here to see the race results" simply make the link text "Race Results" or "See Race Results."

2.5.4 URL Relative Referencing

Whether we are constructing links with the <a> element, referencing images with the <img> element, or including external JavaScript or CSS files, we need to be able to successfully reference files within our site. This requires learning the syntax for so-called relative referencing. As you can see from Figure 2.16, when referencing a

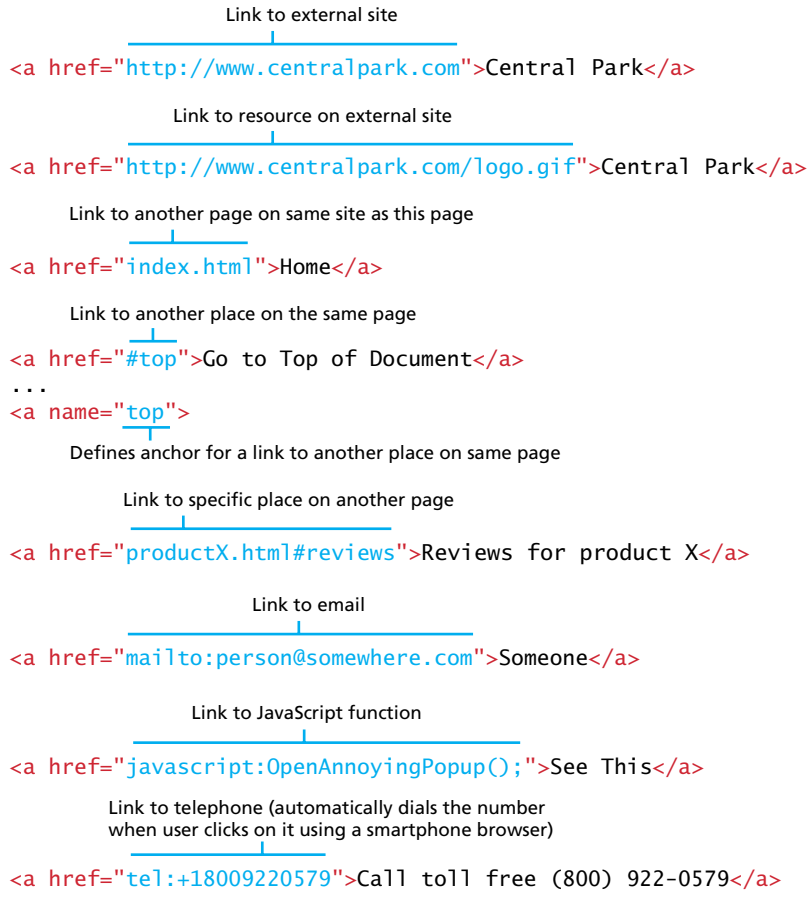


FIGURE 2.16 Different link destinations

page or resource on an external site, a full **absolute reference** is required: that is, a complete URL as described in Chapter 1 with a protocol (typically, `http://`), the domain name, any paths, and then finally the file name of the desired resource.

However, when referencing a resource that is on the same server as your HTML document, you can use briefer relative referencing. If the URL does not include the "**http://**" then the browser will request the current server for the file. If all the resources for the site reside within the same **directory** (also referred to as a **folder**), then you can reference those other resources simply via their file name.

However, most real-world sites contain too many files to put them all within a single directory. For these situations, a relative pathname is required along with the file name. The **pathname** tells the browser where to locate the file on the server.

Pathnames on the web follow Unix conventions. Forward slashes ("/") are used to separate directory names from each other and from file names. Double-periods

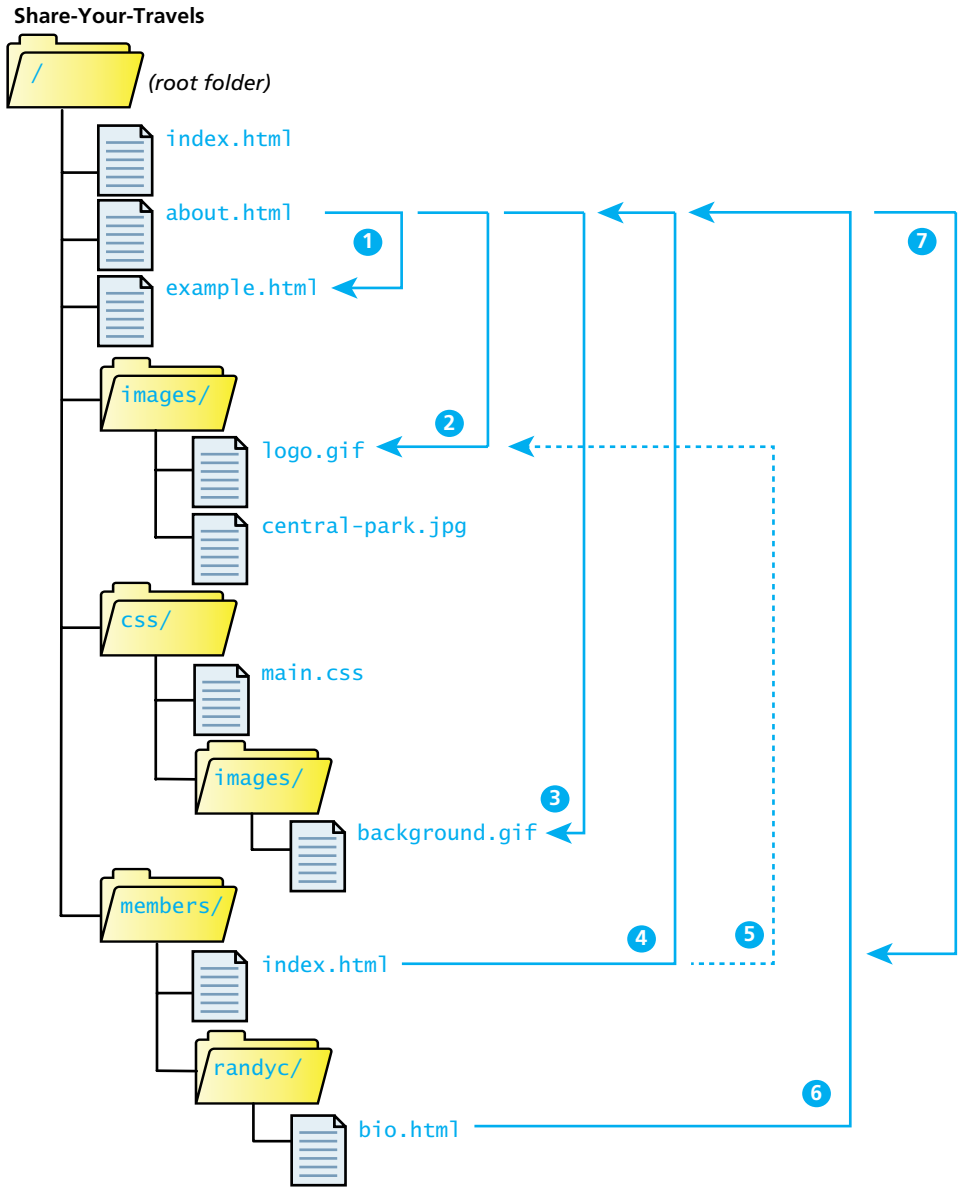


FIGURE 2.17 Example site directory tree

("..") are used to reference a directory "above" the current one in the directory tree. Figure 2.17 illustrates the file structure of an example site. Table 2.1 provides additional explanations and examples of the different types of URL referencing.

Relative Link Type	Example
1 Same Directory To link to a file within the same folder, simply use the file name.	To link to example.html from about.html (in Figure 2.17), use: `<a href="example.html">`
2 Child Directory To link to a file within a subdirectory, use the name of the subdirectory and a slash before the file name.	To link to logo.gif from about.html, use: `<a href="images/logo.gif">`
3 Grandchild/Descendant Directory To link to a file that is multiple subdirectories *below* the current one, construct the full path by including each subdirectory name (separated by slashes) before the file name.	To link to background.gif from about.html, use: `<a href="css/images/background.gif">`

4 Parent/Ancestor Directory
Use "../" to reference a folder *above* the current one. If trying to reference a file several levels above the current one, simply string together multiple "../".
	To link to about.html from index.html in members, use:
	`<a href="../about.html">`
	To link to about.html from bio.html, use:
	`<a href="../../about.html">`

5 Sibling Directory
Use "../" to move up to the appropriate level, and then use the same technique as for child or grandchild directories.
	To link to about.html from index.html in members, use:
	`<a href="../images/about.html">`
	To link to background.gif from bio.html, use:
	`<a href="../../css/images/background.gif">`

6 Root Reference
An alternative approach for ancestor and sibling references is to use the so-called root reference approach. In this approach, begin the reference with the root reference (the "/") and then use the same technique as for child or grandchild directories. **Note that these will only work on the server! That is, they will not work when you test it out on your local machine**.
	To link to about.html from bio.html, use:
	`<a href="/about.html">`
	To link to background.gif from bio.html, use:
	`<a href="/images/background.gif">`

7 Default Document
Web servers allow references to directory names without file names. In such a case, the web server will serve the default document, which is usually a file called index.html (Apache) or default.html (IIS). **Again, this will only generally work on the web server.**
	To link to index.html in members from about.html, use either:
	`<a href="members">`
	Or
	`<a href="/members">`

TABLE 2.1 Sample Relative Referencing

PRO TIP

You can force a link to open in a new browser window by adding the `target="_blank"` attribute to any link.

In general, most web developers believe that forcing a link to open in a new window is not a good practice as it takes control of something (whether a page should be viewed in its own browser window) that rightly belongs to the user away from the user. Nonetheless, some clients will insist that any link to an external site must show up in a new window.

2.5.5 Inline Text Elements

Back in Figure 2.10 the HTML example used three different inline text elements (namely, the `<strong>`, `<time>`, and `<small>` elements). They are called inline elements because they do not disrupt the flow of text (i.e., cause a line break). HTML defines over 30 of these elements. Table 2.2 lists some of the most commonly used of these elements.

2.5.6 Images

HAND-ON EXERCISES

LAB 2 EXERCISE
Adding Images

Item 5 in Figure 2.10 defines an image. While the `<img>` tag is the oldest method for displaying an image, it is not the only way. In fact, it is very common for images to

Element	Description
<a>	Anchor used for hyperlinks.
<abbr>	An abbreviation
** **	Line break
<cite>	Citation (i.e., a reference to another work).
<code>	Used for displaying code, such as markup or programming code.
****	Emphasis
<mark>	For displaying highlighted text
<small>	For displaying the fine-print, i.e., "non-vital" text, such as copyright or legal notices.
****	The inline equivalent of the element. It is generally used to mark text that will receive special formatting using CSS.
****	For content that is strongly important.
<time>	For displaying time and date data

TABLE 2.2 Common Text-Level Semantic Elements

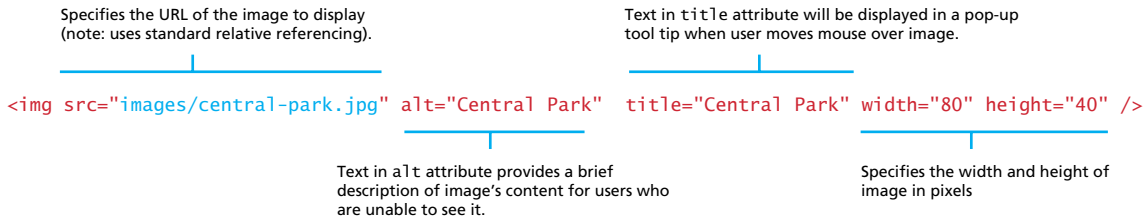


FIGURE 2.18 The <img> element

be added to HTML elements via the `background-image` property in CSS, a technique you will see in Chapter 3. For purely decorative images, such as background gradients and patterns, logos, border art, and so on, it makes semantic sense to keep such images out of the markup and in CSS where they more rightly belong. But when the images are content, such as in the images in a gallery or the image of a product in a product details page, then the <img> tag is the semantically appropriate approach.

Chapter 7 covers images in detail. It examines the different types of graphic file formats, as well as a more detailed examination of the <img> element. Figure 2.18 nonetheless provides a preliminary explanation of the different parts of the <img> element.

2.5.7 Character Entities

Item 9 in Figure 2.10 illustrates the use of a **character entity**. These are special characters for symbols for which there is either no easy way to type them via a keyboard (such as the copyright symbol or accented characters) or which have a reserved meaning in HTML (for instance the "<" or ">" symbols). There are many HTML character entities. They can be used in an HTML document by using the entity name or the entity number. Some of the most common are listed in Table 2.3.

Entity Name	Entity Number	Description
** **		Nonbreakable space. **The browser ignores multiple spaces in the source HTML file. If you need to display multiple spaces, you can do so using the nonbreakable space entity.**
<	&#60;	Less than symbol ("<").
>	&#62;	Greater than symbol (">").
©	&#169;	The © copyright symbol.
€	&#8364;	The € euro symbol.
™	&#8482;	The ™ trademark symbol.
ü	&#252;	The ü—i.e., small u with umlaut mark.

TABLE 2.3 Common Character Entities

2.5.8 Lists

HANDS-ON EXERCISES

LAB 2 EXERCISE
Making a List
Linking with Lists

Figure 2.10 is missing one of the most common block-level elements in HTML, namely, lists. HTML provides three types of lists:

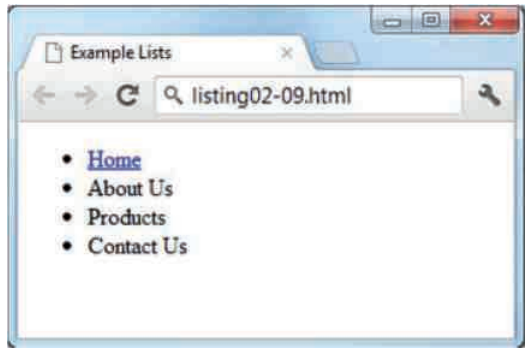
- Unordered lists. Collections of items in no particular order; these are by default rendered by the browser as a bulleted list. However, it is common in CSS to style unordered lists without the bullets. Unordered lists have become the conventional way to markup navigational menus.
- Ordered lists. Collections of items that have a set order; these are by default rendered by the browser as a numbered list.
- Definition lists. Collection of name and definition pairs. These tend to be used infrequently. Perhaps the most common example would be a FAQ list.

As can be seen in Figure 2.19, the various list elements are container elements containing list item elements (`<li>`). Other HTML elements can be included within the `<li>` container, as shown in the first list item of the unordered list in Figure 2.19. Notice as well in the ordered list example in Figure 2.19 that this nesting can include another list.

Notice that the list item element can contain other HTML elements.

```
<ul>
   <li><a href="index.html">Home</a></li>
   <li>About Us</li>
   <li>Products</li>
   <li>Contact Us</li>
</ul>
```

```
<ol>
   <li>Introduction</li>
   <li>Background</li>
   <li>My Solution</li>
   <li>
      <ol>
         <li>Methodology</li>
         <li>Results</li>
         <li>Discussion</li>
      </ol>
   </li>
   <li>Conclusion</li>
</ol>
```



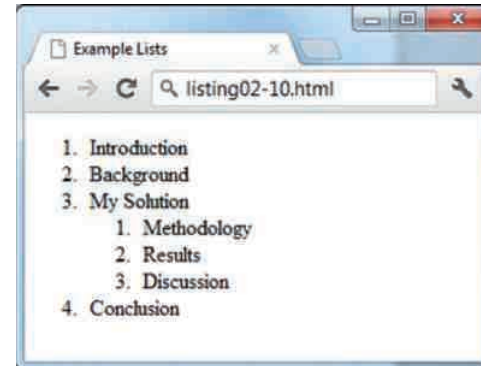


FIGURE 2.19 List elements and their default rendering

2.6 HTML5 Semantic Structure Elements

Section 2.3 discussed the idea of semantic markup and how it improves the maintainability and accessibility of web pages. In the code examples so far, the main semantic elements you have seen are headings, paragraphs, lists, and some inline elements. You also saw the other key semantic block element, namely the division (i.e., `<div>` element).

Figure 2.14 did, however, illustrate one substantial problem with modern, pre-HTML5 semantic markup. Most complex websites are absolutely packed solid with `<div>` elements. Most of these are marked with different `id` or `class` attributes. You will see in Chapter 4 that complex layouts are typically implemented using CSS that targets the various `<div>` elements for CSS styling. Unfortunately, all these `<div>` elements can make the resulting markup confusing and hard to modify. Developers typically try to bring some sense and order to the `<div>` chaos by using `id` or `class` names that provide some clue as to their meaning, as shown in Figure 2.20.

As HTML5 was being developed, researchers at Google and Opera had their search spiders examine millions of pages to see what were the most common `id` and `class` names. Their findings helped standardize the names of the new semantic block structuring elements in HTML5, most of which are shown in Figure 2.20.

The idea behind using these elements is that your markup will be easier to understand because you will be able to replace some of your `<div>` sprawl with cleaner and more self-explanatory HTML5 elements. Figure 2.21 illustrates the simpler version of Figure 2.20, one that uses the new semantic elements in HTML5. Each of these elements is briefly discussed below.

NOTE

In the late spring of 2013, the W3C decided to add a `<main>` element to the HTML5 specification. This change to the specification was made too late to integrate into this chapter. In Figure 2.20, the `<div>` at item 4 could be replaced with a `<main>` element. Do note that support for this new element is not available for all browsers.

2.6.1 Header and Footer

Most website pages have a recognizable header and footer section. Typically the header contains the site logo and title (and perhaps additional subtitles or taglines), horizontal navigation links, and perhaps one or two horizontal banners. The typical footer contains less important material, such as smaller text versions of the navigation, copyright notices, information about the site's privacy policy, and perhaps twitter feeds or links to other social sites.

HANDS-ON EXERCISES

LAB 2 EXERCISE
Header and Footer

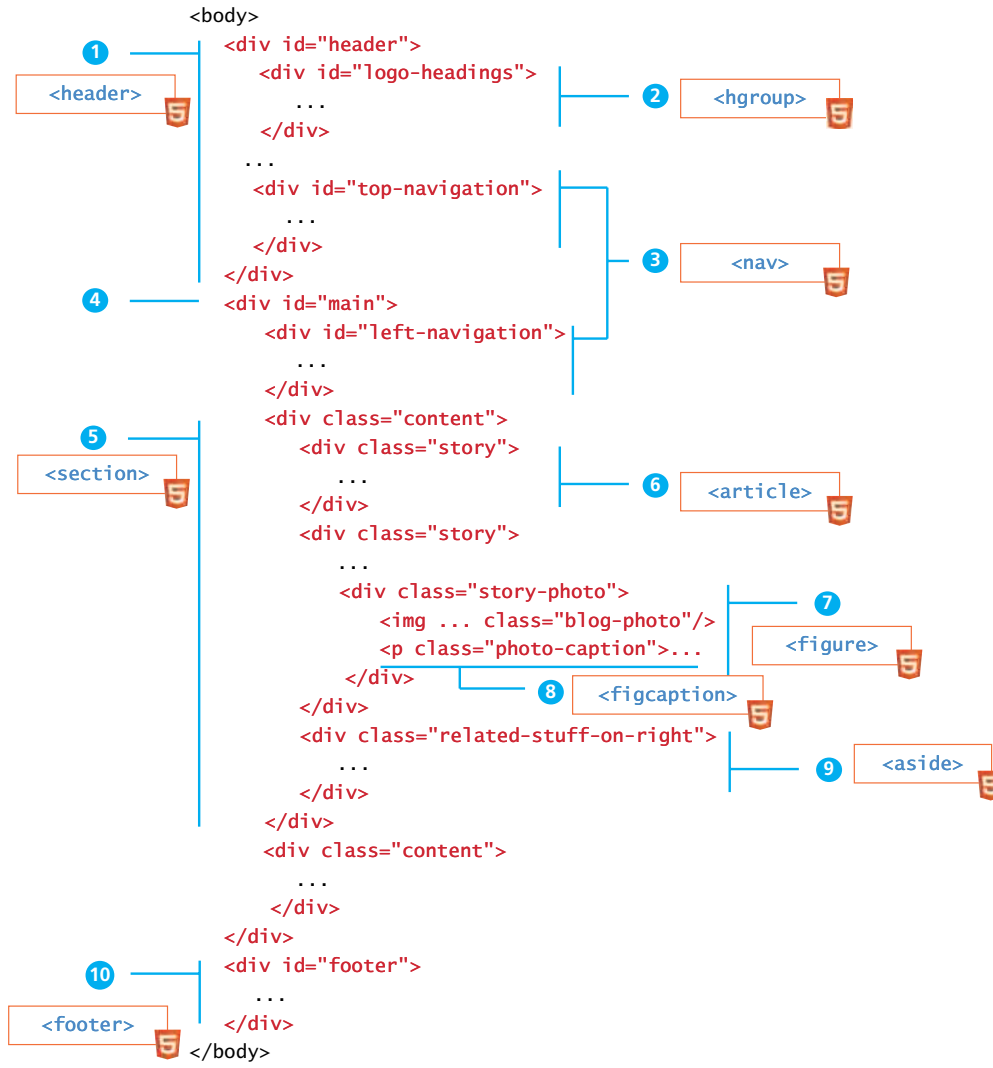


FIGURE 2.20 Sample -based XHTML layout (with HTML5 equivalents)

Both the HTML5 `<header>` and `<footer>` element can be used not only for *page* headers and footers (as shown in items 1 and 9 in Figure 2.21), but also for header and footer elements within other HTML5 containers, such as `<article>` or `<section>`, as indicated by the W3C draft of the HTML5 standard:

> *The header element typically contains the headings for a section (an h1–h6 element or hgroup element), along with content such as introductory material or navigational aids for the section.*
>
> *—W3C Working Draft*

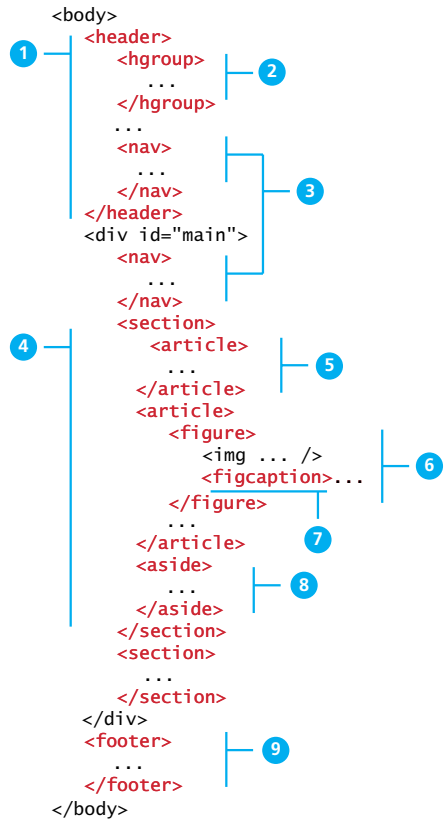


FIGURE 2.21 Sample layout using new HTML5 semantic structure elements

Listing 2.1 demonstrates both uses of the `<heading>` element.

```
<header>
<img src="logo.gif" alt="logo" />
<h1>Fundamentals of Web Development</h1>
...
</header>
<article>
   <header>
      <h2>HTML5 Semantic Structure Elements</h2>
      <p>By <em>Randy Connolly</em></p>
      <p><time>September 30, 2015</time></p>
   </header>
   ...
</article>
```

LISTING 2.1 Heading example

The browser really doesn't care how one uses these HTML5 semantic structure elements. Just like with the element, there is no predefined presentation for these tags.

2.6.2 Heading Groups

As mentioned in the previous section, it is not that unusual for a header to contain multiple headings in close proximity. The <hgroup> element (seen as item 2 in Figure 2.21) can be used in such a circumstance to group them together within one container.

The <hgroup> element can be used in contexts other than a header. For instance, one could also use an <hgroup> within an <article> or a <section> element as well. The <hgroup> element can *only* contain <h1>, <h2>, etc., elements. Listing 2.2 illustrates two example usages of the <hgroup> element.

```
<header>
   <hgroup>
   <h1>Chapter Two: HTML 1</h1>
   <h2>An Introduction</h2>
   </hgroup>
</header>
<article>
   <hgroup>
   <h2>HTML5 Semantic Structure Elements</h2>
   <h3>Overview</h3>
   </hgroup>
</article>
```

LISTING 2.2 hgroup example

NOTE

In April 2013, the W3C decided to drop the <hgroup> element from the W3C specification. While browsers will likely continue to support the <hgroup> element, if you need to group headings, you are encouraged to instead nest them within a element.

HANDS-ON EXERCISES

LAB 2 EXERCISE
Navigation, Articles, and Sections

2.6.3 Navigation

The <nav> element (item 3 in Figure 2.21) represents a section of a page that contains links to other pages or to other parts within the same page. Like the other new HTML5 semantic elements, the browser does not apply any special presentation to the <nav> element. As you can see in the quote from the WHATWG specification for HTML5 (that was used by the W3C in their own Recommendation), the <nav>

element was intended to be used for major navigation blocks, presumably the global and secondary navigation systems as well as perhaps search facilities (see Chapter 17 on usability for more about navigation system names). However, like all the new HTML5 semantic elements in Section 2.6, from the browser's perspective, there is no definite right or wrong way to use the <nav> element. Its sole purpose is to make your markup easier to understand, and by limiting the use of the <nav> element to major elements, your markup will more likely achieve that aim.

> *Not all groups of links on a page need to be in a nav element—the element is* ***primarily intended for sections that consist of major navigation blocks****. In particular, it is common for footers to have a short list of links to various pages of a site, such as the terms of service, the home page, and a copyright page. The footer element alone is sufficient for such cases; while a nav element can be used in such cases, it is usually unnecessary.*
>
> *—WHATWG HTML specification*

Listing 2.3 illustrates a typical example usage of the <nav> element. Note the use of the `role` attribute. It is not required but will improve accessibility. Section 4.5 in Chapter 4 will provide more information about ARIA (Accessible Rich Internet Applications) roles.

```
<header>
  <img src="logo.gif" alt="logo" />
  <h1>Fundamentals of Web Development</h1>
  <nav role="navigation">
    <ul>
       <li><a href="index.html">Home</a></li>
       <li><a href="about.html">About Us</a></li>
       <li><a href="browse.html">Browse</a></li>
    </ul>
  </nav>
</header>
```

LISTING 2.3 nav example

2.6.4 Articles and Sections

The book you are reading is divided into smaller blocks of content called chapters, which make this long book easier to read. Furthermore, each chapter is further divided into sections (and these sections into even smaller subsections), all of which make the content of the book easier to manage for both the reader and the authors. Other types of textual content, such as newspapers, are similarly divided into logical sections. The new HTML5 semantic elements <section> and <article> (items 4 and 5, respectively, in Figure 2.21) play a similar role within web pages.

It might not be clear how to choose between these two elements. The W3C specification provides us with some insight.

> *The article element represents a section of content that forms an independent part of a document or site; for example, a magazine or newspaper article, or a blog entry.*
>
> *The section element represents a section of a document, typically with a title or heading.*
>
> *—W3C Working Draft*

PRO TIP

You may have noticed that the language in these W3C and WHATWG specifications can be rather "dull" and "heavy." While they do try to provide clarity by using consistent terminology throughout the specification, this means that they can also be difficult to understand if one isn't familiar with that terminology. Nonetheless, being able to read and decipher technical documents is a skill that a computing professional eventually does need to master.

According to the W3C, `<section>` is a much broader element, while the `<article>` element is to be used for blocks of content that could potentially be read or consumed independently of the other content on the page. We can gain a further understanding of how to use these two elements by looking at the more expansive WHATWG specification.

> *The section element represents a generic section of a document or application. A section, in this context, is a thematic grouping of content, typically with a heading. Examples of sections would be chapters, the various tabbed pages in a tabbed dialog box, or the numbered sections of a thesis. A Website's home page could be split into sections for an introduction, news items, and contact information.*
>
> *The article element represents a self-contained composition in a document, page, application, or site and that is, in principle, independently distributable or reusable, e.g. in syndication. This could be a forum post, a magazine or newspaper article, a blog entry, a user-submitted comment, an interactive widget or gadget, or any other independent item of content.*
>
> *—WHATWG HTML specification*

The reference to syndication in the WHATWG explanation of the `<article>` element is useful. In the context of the web, **syndication** refers to websites making

their content available to other websites for display. If some block of content could theoretically exist on another website (as if it were syndicated) and still make sense in that new context, then wrap that content within an `<article>` element. If a block of content has some type of heading associated with it, then consider wrapping it within a `<section>` element.

NOTE

The WHATWG specification warns readers that the `<section>` element is **not** a generic container element. HTML already has the `<div>` element for such uses. When an element is needed only for styling purposes or as a convenience for scripting, it makes sense to use the `<div>` element instead. Another way to help you decide whether or not to use the `<section>` element is to ask yourself if it is appropriate for the element's contents to be listed explicitly in the document's outline. If so, then use a `<section>`; otherwise, use a `<div>`.

2.6.5 Figure and Figure Captions

HANDS-ON EXERCISES

LAB 2 EXERCISE
Figures and Captions

Throughout this chapter you have seen screen captures or diagrams or photographs that are separate from the text (but related to it), which are described by a caption, and which are given the generic name of *Figure*. Prior to HTML5, web authors typically wrapped images and their related captions within a nonsemantic `<div>` element. In HTML5 we can instead use the more obvious `<figure>` and `<figcaption>` elements (items 6 and 7 in Figure 2.21).

The W3C Recommendation indicates that the `<figure>` element can be used not just for images but for any type of *essential* content that could be moved to a different location in the page or document and the rest of the document would still make sense.

> *The figure element represents some flow content, optionally with a caption, that is self-contained and is typically referenced as a single unit from the main flow of the document.*
>
> *The element can thus be used to annotate illustrations, diagrams, photos, code listings, etc, that are referred to from the main content of the document, but that could, without affecting the flow of the document, be moved away from that primary content, e.g. to the side of the page, to dedicated pages, or to an appendix.*
>
> *—WHATWG HTML specification*

For instance, as I write this section, I will at some point make reference to one of the figures or code listings. But I cannot write "the illustration above" or "the code listing to the right," even though it is possible that on the page you are looking at right

now, there is an illustration just above these words or the code listing might be just to the right. I cannot do this because at the point of writing these words, the actual page layout is still many months away. But I can make nonspatial references in the text to "Figure 2.22" or to "Listing 2.5"—that is, to the illustration or code samples' captions. The figures and code listings are not optional; they need to be in the text. However, their ultimate position on the page is irrelevant to me as I write the text.

NOTE

The `<figure>` element should not be used to wrap every image. For instance, it makes no sense to wrap the site logo or nonessential images such as banner ads and graphical embellishments within `<figure>` elements. Instead, only use the `<figure>` element for circumstances where the image (or other content) has a caption and where the figure is essential to the content but its position on the page is relatively unimportant.

Figure 2.22 illustrates a sample usage of the `<figure>` and `<figcaption>` element. While this example places the caption below the figure in the markup, this is not required. Similarly, this example shows an image within the `<figure>`, but it could be any content.

Figure could be moved to a different location in document …

But it has to exist in the document (i.e., the figure isn't optional).

```
<p>This photo was taken on October 22, 2011 with a Canon EOS 30D camera.</p>
<figure>
  <img src="images/central-park.jpg" alt="Central Park" /><br/>
  <figcaption>Conservatory Pond in Central Park</figcaption>
</figure>
<p>
It was a wonderfully beautiful autumn Sunday, with strong sunlight and
expressive clouds. I was very fortunate that my one day in New York was
blessed with such weather!
</p>
```

FIGURE 2.22 The figure and figcaption elements in the browser

2.6.6 Aside

The <aside> element (item 8 in Figure 2.21) is similar to the <figure> element in that it is used for marking up content that is separate from the main content on the page. But while the <figure> element was used to indicate important information whose location on the page is somewhat unimportant, the <aside> element "represents a section of a page that consists of content that is tangentially related to the content around the aside element" (from WHATWG specification).

The <aside> element could thus be used for sidebars, pull quotes, groups of advertising images, or any other grouping of non-essential elements.

PRO TIP

Prior to IE 9, CSS styles could not be applied to the semantic elements within HTML5. The most common workaround to this problem was the so-called **HTML5 shiv**, which was a JavaScript-based solution. Some of the examples in later chapters include this shiv, which looks like the following:

```
<!--[if lt IE 9]>
     <script src="html5shiv.js"></script>
<![endif]-->
```

This code makes use of conditional comments, which are supported only by IE. Other browsers will see this code as an HTML comment.

2.7 Chapter Summary

This chapter has provided a relatively fast-paced overview of the significant features of HTML5. Besides covering the details of most of the important HTML elements, an additional focus throughout the chapter has been on the importance of maintaining proper semantic structure when creating an HTML document. To that end, the chapter also covered the new semantic elements defined in HTML5. The next chapter will shift the focus to the visual display of HTML elements and provide the reader with a first introduction to CSS.

2.7.1 Key Terms

absolute referencing
accessibility
ancestors
body
Cascading Style Sheets (CSS)
character entity
definition lists
descendants

directory
document outline
Document Object Model
Document Type Definition
empty element
folder
head
HTML attribute
HTML validators
inline HTML elements
maintainability
markup
markup language
ordered lists
pathname
quirks mode
Recommendations
relative referencing
root element
root reference
schemas
search engine optimization
semantic HTML
specifications
standards mode
syndication
syntax errors
tags
unordered lists
UTF-8
WHATWG
W3C
XHTML 1.0 Strict
XHTML 1.0 Transitional

2.7.2 Review Questions

1. What is the difference between XHTML and HTML5?
2. Why was the XHTML 2.0 standard eventually abandoned?
3. What role do HTML validators play in web development?
4. What are the main syntax rules for XML?
5. What are HTML elements? What are HTML attributes?
6. What is semantic markup? Why is it important?
7. Why is removing presentation-oriented markup from one's HTML documents considered to be a best practice?
8. What is the difference between standards mode and quirks mode? What role does the `doctype` play with these modes?
9. What is the difference between the `<p>` and the `<div>` element? In what contexts should one use the one over the other?
10. Describe the difference between a relative and an absolute reference. When should each be used?
11. What are the advantages of using the new HTML5 semantic elements? Disadvantages?
12. Are you allowed to use more than one `<heading>` element in a web page? Why or why not?

2.7.3 Hands-On Practice

Hands-on practice projects are present in many chapters throughout this textbook and relate the content matter back to a few overarching examples: an art store, a travel website, and a customer relationship management (CRM) portal for a book representative. These projects come with images, databases, and other files, and are included with your purchase of this textbook.

PROJECT 1: Share Your Travel Photos

DIFFICULTY LEVEL: Beginner

Overview

This project is the first step in the creation of a travel photo sharing website. The page you are given is augmented by this project to add a Related Photos section to the page.

HANDS-ON EXERCISES
PROJECT 2.1

Instructions

1. Open chapter02-project01.html in the editor of your choice, so you can start making changes.
2. Open a browser and direct it to the same file (or double click the file in most operating systems). You should see a page like the top part of Figure 2.23

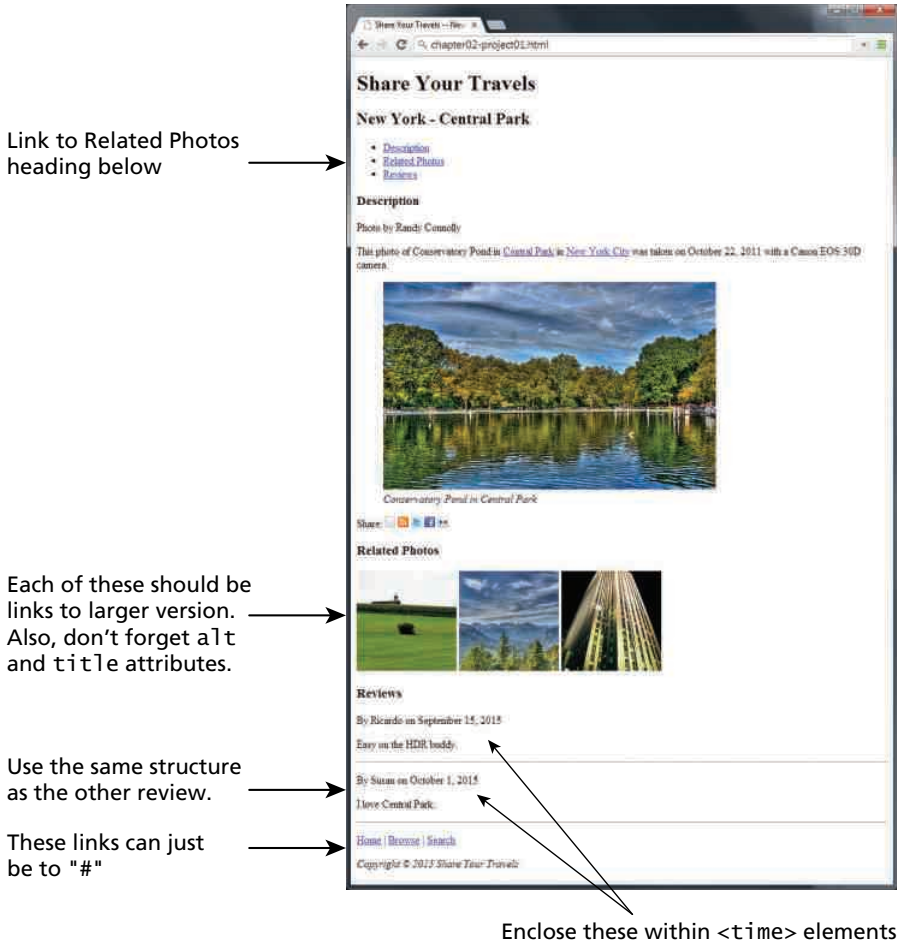


FIGURE 2.23 Completed Project 1

3. Start by adding a link to *Related Photos*, in the unordered list that currently contains *Descriptions* and *Reviews*. (You can make the `href` attribute point to # for the moment.)
4. Now go down to the bottom of the page and add the new Related Photos `<section>`.
5. In the new section add three images from the ones provided in the images folder. Use the small images **related-square1.jpg**, **related-square2.jpg**, and **related-square3.jpg** in the `src` of your `<img>` tag, but link to the large images with almost the same names.

Test

1. Firstly, test your page by seeing if it looks like the one in Figure 2.23.
2. Now check that the link correctly links the Related Photos link to the newly defined section, and that clicking on the related images brings up the larger versions.
3. Validate the page by either using a built-in tool in your editor, or pasting the HTML into **http://validator.w3.org/** and ensure that it displays the message: *This document was successfully checked as HTML5*!

PROJECT 2: Book Rep Customer Relations Management

DIFFICULTY LEVEL: Intermediate

HANDS-ON EXERCISES

PROJECT 2.2

Overview

This project is the first step in the creation of a CRM website. In this project you will be augmenting the provided page to use semantic HTML5 tags.

Instructions

1. Open **chapter02-project02.html** in the editor of your choice, and in a browser. In this project the look of your page will remain unchanged from how it looks at the start as shown in Figure 2.24.
2. Reflect on why adding semantic markup is a worthwhile endeavor, even if the final, rendered page looks identical.
3. Replace and supplement generic HTML tags like `<div>` with semantic tags like `<article>`, `<nav>`, or `<footer>` (for example). Some parts make sense to wrap inside a tag such as `<section>` or `<figure>`.

Test

1. Firstly, test your page side by side with the original in a browser to make sure it looks the same.
2. Validate the page by either using a built-in tool in your editor, or pasting the HTML into **http://validator.w3.org/** and seeing if it passes. You will notice to pass it must do many extra things like have alt attributes on `<img>` tags.

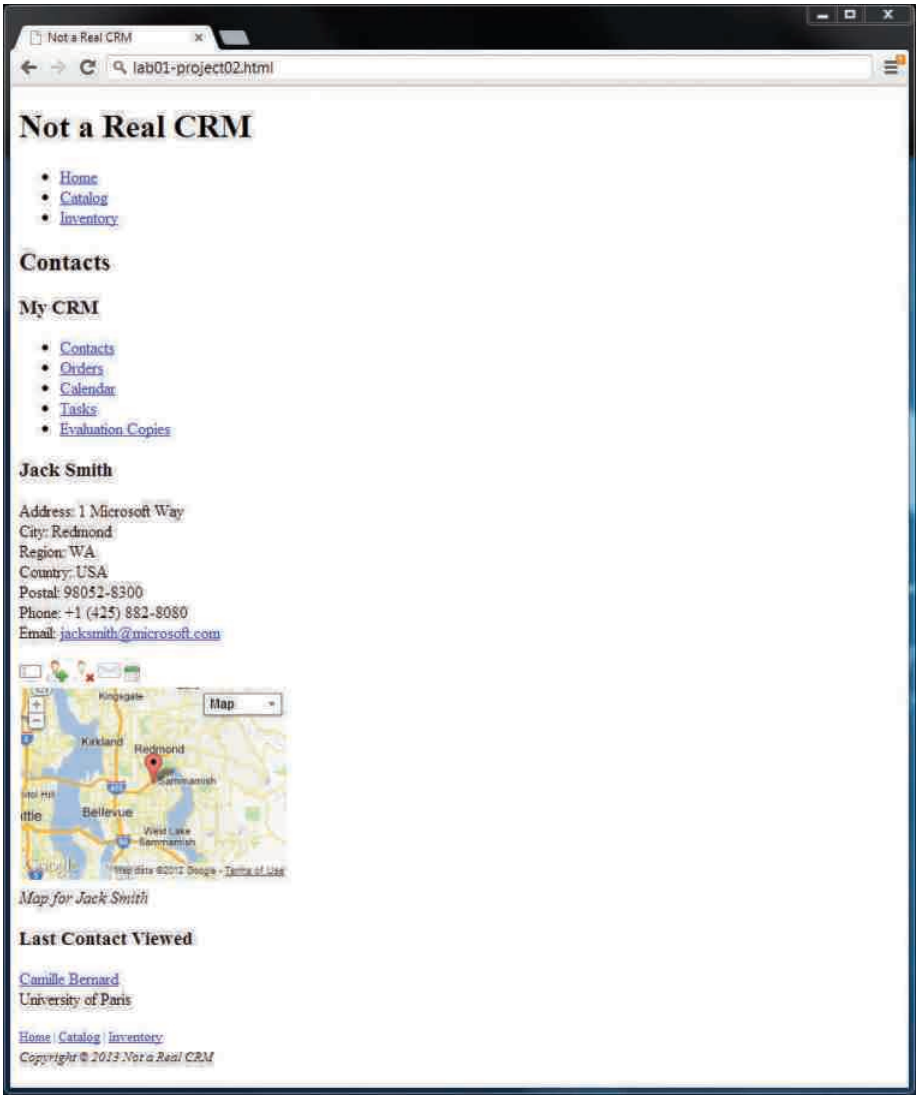


FIGURE 2.24 Completed Project 2

PROJECT 3: Art Store

DIFFICULTY LEVEL: Advanced

Overview

This project is the first step in the creation of an art store website. Unlike the previous exercises, your task is to create an HTML page from scratch based on the image in Figure 2.25.

HANDS-ON EXERCISES
PROJECT 2.3

Instructions

1. Define your own chapter02-project03.html file in the editor of your choice, and open it in a browser.
2. Add markup and content, making best guesses as to what HTML markup to use.
3. Remember to try and get in the habit of using semantic markup, since it adds meaning and has no visual impact.

Test

1. Display your page in a browser, and determine if it looks like Figure 2.25.
2. Validate the page by pasting the HTML into http://validator.w3.org/ and seeing if it passes.

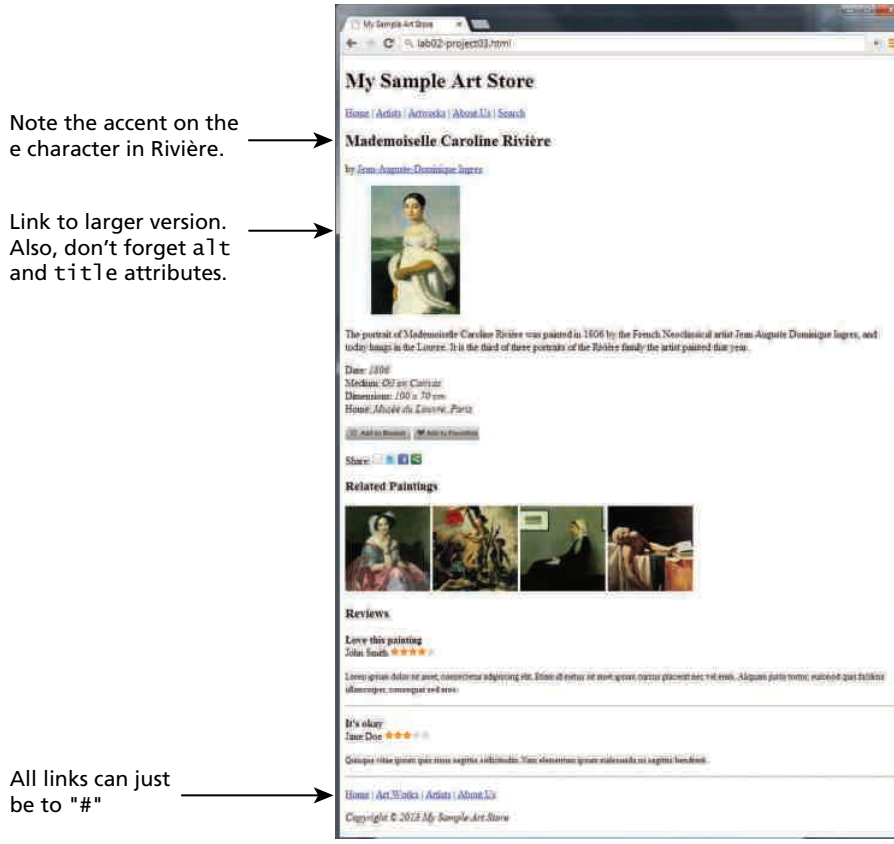


FIGURE 2.25 Completed Project 3

Introduction to CSS 3

CHAPTER OBJECTIVES

In this chapter you will learn . . .

- The rationale for CSS
- The syntax of CSS
- Where CSS styles can be located
- The different types of CSS selectors
- What the CSS cascade is and how it works
- The CSS box model
- CSS text styling

This chapter provides a substantial introduction to CSS (Cascading Style Sheets), the principal mechanism for web authors to modify the visual presentation of their web pages. Just as with HTML, there are many books devoted to CSS.[1,2,3] While simple styling is quite straightforward, more complicated CSS tasks such as layout and positioning can be quite complicated. Since this book covers CSS in just two chapters, it cannot possibly cover all of it. Instead, our intent in this chapter is to cover the foundations necessary for working with contemporary CSS; Chapter 5 will cover CSS layout and positioning.

3.1 What Is CSS?

At various places in the previous chapter on HTML, it was mentioned that in current web development best practices HTML should not describe the formatting or presentation of documents. Instead that presentation task is best performed using Cascading Style Sheets (CSS).

CSS is a W3C standard for describing the appearance of HTML elements. Another common way to describe CSS's function is to say that CSS is used to define the presentation of HTML documents. With CSS, we can assign font properties, colors, sizes, borders, background images, and even position elements on the page.

CSS can be added directly to any HTML element (via the `style` attribute), within the `<head>` element, or, most commonly, in a separate text file that contains only CSS.

3.1.1 Benefits of CSS

Before digging into the syntax of CSS, we should say a few words about why using CSS is a better way of describing appearances than HTML alone. The benefits of CSS include:

- **Improved control over formatting.** The degree of formatting control in CSS is significantly better than that provided in HTML. CSS gives web authors fine-grained control over the appearance of their web content.
- **Improved site maintainability.** Websites become significantly more maintainable because all formatting can be centralized into one CSS file, or a small handful of them. This allows you to make site-wide visual modifications by changing a single file.
- **Improved accessibility.** CSS-driven sites are more accessible. By keeping presentation out of the HTML, screen readers and other accessibility tools work better, thereby providing a significantly enriched experience for those reliant on accessibility tools.
- **Improved page download speed.** A site built using a centralized set of CSS files for all presentation will also be quicker to download because each individual HTML file will contain less style information and markup, and thus be smaller.
- **Improved output flexibility.** CSS can be used to adopt a page for different output media. This approach to CSS page design is often referred to as responsive design. Figure 3.1 illustrates a site that responds to different browser and window sizes.

3.1.2 CSS Versions

Just like with the previous chapter, we should say a few words about the history of CSS. Style sheets as a way to visually format markup predate the web. In the early

FIGURE 3.1 CSS-based responsive design (site by Peerapong Pulpipatnan on ThemeForest.net)

1990s, a variety of different style sheet standards were proposed, including JavaScript style sheets, which was proposed by Netscape in 1996. Netscape's proposal was one that required the use of JavaScript programming to perform style changes. Thankfully for nonprogrammers everywhere, the W3C decided to adopt CSS, and by the end of 1996 the CSS Level 1 Recommendation was published. A year later, the CSS Level 2 Recommendation (also more succinctly labeled simply as CSS2) was published.[4]

Even though work began over a decade ago, an updated version of the Level 2 Recommendation, CSS2.1, did not become an official W3C Recommendation until June 2011. And to complicate matters even more, all through the last decade (and to the present day as well), during the same time the CSS2.1 standard was being worked on, a different group at the W3C was working on a CSS3 draft. To make CSS3 more manageable for both browser manufacturers and web designers, the W3C has subdivided it into a variety of different CSS3 modules. So far the following CSS3 modules have made it to official W3C Recommendations: CSS Selectors, CSS Namespaces, CSS Media Queries, and CSS Color.

3.1.3 Browser Adoption

Perhaps the most important thing to keep in mind with CSS is that the different browsers have not always kept up to the W3C. While Microsoft's Internet Explorer was an early champion of CSS (its IE3, released in 1996, was the first major browser to support CSS, and its IE5 for the Macintosh was the first browser to reach almost 100% CSS1 support in 2000), its later versions (especially IE5, IE6, and IE7) for Windows had uneven support for certain parts of CSS2. However, all browsers have not implemented parts of the CSS2 Recommendation.

For this reason, CSS has a reputation for being a somewhat frustrating language. Based on over a decade of experience teaching university students CSS, this

reputation is well deserved. Since CSS was designed to be a styling language, text styling is quite easy. However, CSS was not really designed to be a layout language, so authors often find it tricky dealing with floating elements, relative positions, inconsistent height handling, overlapping margins, and nonintuitive naming (we're looking at you, `relative` and `!important`). When one adds in the uneven CSS 2.1 support (prior to IE8 and Firefox 2) in browsers for CSS2.1, it becomes quite clear why many software developers developed a certain fear and loathing of CSS.

3.2 CSS Syntax

HANDS-ON EXERCISES

LAB 3 EXERCISE
Adding Styles

A CSS document consists of one or more **style rules**. A rule consists of a **selector** that identifies the HTML element or elements that will be affected, followed by a series of **property:value pairs** (each pair is also called a **declaration**), as shown in Figure 3.2.

The series of declarations is also called the **declaration block**. As one can see in the illustration, a declaration block can be together on a single line, or spread across multiple lines. The browser ignores white space (i.e., spaces, tabs, and returns) between your CSS rules so you can format the CSS however you want. Notice that each declaration is terminated with a semicolon. The semicolon for the last declaration in a block is in fact optional. However, it is sensible practice to also terminate the

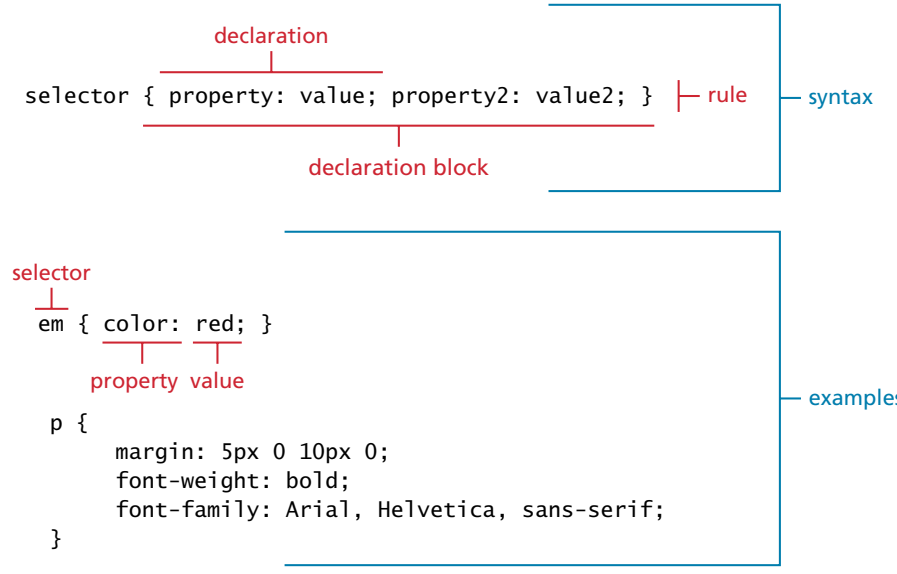


FIGURE 3.2 CSS syntax

last declaration with a semicolon as well; that way, if you add rules to the end later, you will reduce the chance of introducing a rather subtle and hard-to-discover bug.

3.2.1 Selectors

Every CSS rule begins with a selector. The selector identifies which element or elements in the HTML document will be affected by the declarations in the rule. Another way of thinking of selectors is that they are a pattern that is used by the browser to select the HTML elements that will receive the style. As you will see later in this chapter, there are a variety of ways to write selectors.

3.2.2 Properties

Each individual CSS declaration must contain a property. These property names are predefined by the CSS standard. The CSS2.1 recommendation defines over a hundred different property names, so some type of reference guide, whether in a book, online, or within your web development software, can be helpful.[5] This chapter and the next one on CSS (Chapter 5) will only be able to cover most of the common CSS properties. Table 3.1 lists many of the most commonly used CSS properties.

Property Type	Property

Fonts	`font`
	`font-family`
	`font-size`
	`font-style`
	`font-weight`
	`@font-face`

Text	`letter-spacing`
	`line-height`
	`text-align`
	`text-decoration`
	`text-indent`

Color and background	`background`
	`background-color`
	`background-image`
	`background-position`
	`background-repeat`
	`color`

Borders	`border`
	`border-color`
	`border-width`
	`border-style`
	`border-top`
	`border-top-color`
	`border-top-width`
	`etc.`

(*continued*)

Property Type	Property

Spacing	`padding`
	`padding-bottom, padding-left, padding-right, padding-top`
	`margin`
	`margin-bottom, margin-left, margin-right, margin-top`

Sizing	`height`
	`max-height`
	`max-width`
	`min-height`
	`min-width`
	`width`

Layout	`bottom, left, right, top`
	`clear`
	`display`
	`float`
	`overflow`
	`position`
	`visibility`
	`z-index`

Lists	`list-style`
	`list-style-image`
	`list-style-type`

TABLE 3.1 Common CSS Properties

3.2.3 Values

Each CSS declaration also contains a value for a property. The unit of any given value is dependent upon the property. Some property values are from a predefined list of keywords. Others are values such as length measurements, percentages, numbers without units, color values, and URLs.

Colors would seem at first glance to be the most clear of these units. But as we will see in more detail in Chapter 7, color can be a complicated thing to describe. CSS supports a variety of different ways of describing color; Table 3.2 lists the

Method	Description	Example
Name	Use one of 17 standard color names. CSS3 has 140 standard names.	`color: red;` `color: hotpink; /* CSS3 only */`
RGB	Uses three different numbers between 0 and 255 to describe the red, green, and blue values of the color.	`color: rgb(255,0,0);` `color: rgb(255,105,180);`

Hexadecimal	Uses a six-digit hexadecimal number to describe the red, green, and blue value of the color; each of the three RGB values is between 0 and FF (which is 255 in decimal). Notice that the hexadecimal number is preceded by a hash or pound symbol (#).	`color: #FF0000;` `color: #FF69B4;`
RGBa	This defines a partially transparent background color. The "a" stands for "alpha", which is a term used to identify a transparency that is a value between 0.0 (fully transparent) and 1.0 (fully opaque).	`color: rgb(255,0,0, 0.5);`
HSL	Allows you to specify a color using Hue Saturation and Light values. This is available only in CSS3. HSLA is also available as well.	`color: hsl(0,100%,100%);` `color: hsl(330,59%,100%);`

TABLE 3.2 Color Values

different ways you can describe a color value in CSS. Note that we will learn more about web color in Chapter 7.

Just as there are multiple ways of specifying color in CSS, so too there are multiple ways of specifying a unit of measurement. As we will see later in Section 3.7, these units can sometimes be complicated to work with. When working with print design, we generally make use of straightforward absolute units such as inches or centimeters and picas or points. However, because different devices have differing physical sizes as well as different pixel resolutions and because the user is able to change the browser size or its zoom mode, these absolute units don't always make sense with web element measures.

Table 3.3 lists the different units of measure in CSS. Some of these are **relative units**, in that they are based on the value of something else, such as the size of a

Unit	Description	Type
px	Pixel. In CSS2 this is a relative measure, while in CSS3 it is absolute (1/96 of an inch).	Relative (CSS2) Absolute (CSS3)
em	Equal to the computed value of the font-size property of the element on which it is used. When used for font sizes, the em unit is in relation to the font size of the parent.	Relative
%	A measure that is always relative to another value. The precise meaning of % varies depending upon the property in which it is being used.	Relative

(*continued*)

Unit	Description	Type
ex	A rarely used relative measure that expresses size in relation to the x-height of an element's font.	Relative
ch	Another rarely used relative measure; this one expresses size in relation to the width of the zero ("0") character of an element's font.	Relative (CSS3 only)
rem	Stands for root `em`, which is the font size of the root element. Unlike `em`, which may be different for each element, the `rem` is constant throughout the document.	Relative (CSS3 only)
vw, vh	Stands for viewport width and viewport height. Both are percentage values (between 0 and 100) of the viewport (browser window). This allows an item to change size when the viewport is resized.	Relative (CSS3 only)
in	Inches	Absolute
cm	Centimeters	Absolute
mm	Millimeters	Absolute
pt	Points (equal to 1/72 of an inch)	Absolute
Pc	Pica (equal to 1/6 of an inch)	Absolute

TABLE 3.3 Units of Measure Values

NOTE

It is often helpful to add comments to your style sheets. Comments take the form:

```
/* comment goes here */
```

Real-world CSS files can quickly become quite long and complicated. It is a common practice to locate style rules that are related together, and then indicate that they are related via a comment. For instance:

```
/* main navigation */
nav#main {  … }
nav#main ul { … }
nav#main ul li { … }
```

```
/* header */
header { … }
h1 { … }
```

Comments can also be a helpful way to temporarily hide any number of rules, which can make debugging your CSS just a tiny bit less tedious.

parent element. Others are **absolute units**, in that they have a real-world size. Unless you are defining a style sheet for printing, it is recommended you avoid using absolute units. Pixels are perhaps the one popular exception (though, as we shall see later, there are also good reasons for avoiding the pixel unit). In general, most of the CSS that you will see uses either `px`, `em`, or `%` as a measure unit.

3.3 Location of Styles

As mentioned earlier, CSS style rules can be located in three different locations. These three are not mutually exclusive, in that you could place your style rules in all three. In practice, however, web authors tend to place all of their style definitions in one (or more) external style sheet files.

3.3.1 Inline Styles

Inline styles are style rules placed within an HTML element via the `style` attribute, as shown in Listing 3.1. An inline style only affects the element it is defined within and overrides any other style definitions for properties used in the inline style (more about this below in Section 3.5.2). Notice that a selector is not necessary with inline styles and that semicolons are only required for separating multiple rules.

Using inline styles is generally discouraged since they increase bandwidth and decrease maintainability (because presentation and content are intermixed and because it can be difficult to make consistent inline style changes across multiple files). Inline styles can, however, be handy for quickly testing out a style change.

```
<h1>Share Your Travels</h1>
<h2>style="font-size: 24pt"Description</h2>
...
<h2>style="font-size: 24pt; font-weight: bold;">Reviews</h2>
```

LISTING 3.1 Internal styles example

3.3.2 Embedded Style Sheet

HANDS-ON EXERCISES

LAB 3 EXERCISE
Embedded Style Sheets

Embedded style sheets (also called internal styles) are style rules placed within the `<style>` element (inside the `<head>` element of an HTML document), as shown in Listing 3.2. While better than inline styles, using embedded styles is also by and large discouraged. Since each HTML document has its own `<style>` element, it is more difficult to consistently style multiple documents when using embedded styles. Just as with inline styles, embedded styles can, however, be helpful when quickly testing out a style that is used in multiple places within a single HTML document. We sometimes use embedded styles in the book or in lab materials for that reason.

```
<head lang="en">
   <meta charset="utf-8">
   <title>Share Your Travels -- New York - Central Park</title>
   <style>
      h1 { font-size: 24pt; }
      h2 {
       font-size: 18pt;
       font-weight: bold;
       }
   </style>
</head>
<body>
   <h1>Share Your Travels</h1>
   <h2>New York - Central Park</h2>
   ...
```

LISTING 3.2 Embedded styles example

3.3.3 External Style Sheet

HANDS-ON EXERCISES

LAB 3 EXERCISE
External Style Sheets

External style sheets are style rules placed within a external text file with the .css extension. This is by far the most common place to locate style rules because it provides the best maintainability. When you make a change to an external style sheet, all HTML documents that reference that style sheet will automatically use the updated version. The browser is able to cache the external style sheet, which can improve the performance of the site as well.

To reference an external style sheet, you must use a `<link>` element (within the `<head>` element), as shown in Listing 3.3. You can link to several style sheets at a time; each linked style sheet will require its own `<link>` element.

```
<head lang="en">
   <meta charset="utf-8">
   <title>Share Your Travels -- New York - Central Park</title>
   <link rel="stylesheet" href="styles.css" />
</head>
```

LISTING 3.3 Referencing an external style sheet

NOTE

There are in fact three different types of style sheet:

1. Author-created style sheets (what you are learning in this chapter)
2. User style sheets
3. Browser style sheets

User style sheets allow the individual user to tell the browser to display pages using that individual's own custom style sheet. This option can usually be found in a browser's accessibility options.

The browser style sheet defines the default styles the browser uses for each HTML element. Some browsers allow you to view this stylesheet. For instance, in Firefox, you can view this default browser style sheet via the following URL: resource://gre-resources/forms.css. The browser stylesheet for WebKit browsers such as Chrome and Safari can be found (for now) at: http://trac.webkit.org/browser/trunk/Source/WebCore/css/html.css.

3.4 Selectors

As teachers, we often need to be able to relay a message or instruction to either individual students or groups of students in our classrooms. In spoken language, we have a variety of different approaches we can use. We can identify those students by saying things like: "all of you talking in the last row," or "all of you sitting in an aisle seat," or "all of you whose name begins with 'C'," or "all first year students," or "John Smith." Each of these statements identifies or selects a different (but possibly overlapping) set of students. Once we have used our student selector, we can then provide some type of message or instruction, such as "talk more quietly," "hand in your exams," or "stop texting while I am speaking."

HANDS-ON EXERCISES

LAB 3 EXERCISE
Element, Class, and Id Selectors

NOTE

Figure 2.6 back in Chapter 2 illustrated some of the familial terminologies (such as descendants, ancestors, siblings, etc.) that are used to describe the relationships between elements in an HTML document. The Document Object Model (DOM) is how a browser represents an HTML page internally. This DOM is akin to a tree representing the overall hierarchical structure of the document.

As you progress through these chapters on CSS you will at times have to think about the elements in your HTML document in terms of their position within the hierarchy. Figure 3.3 illustrates a sample document structure as a hierarchical tree.

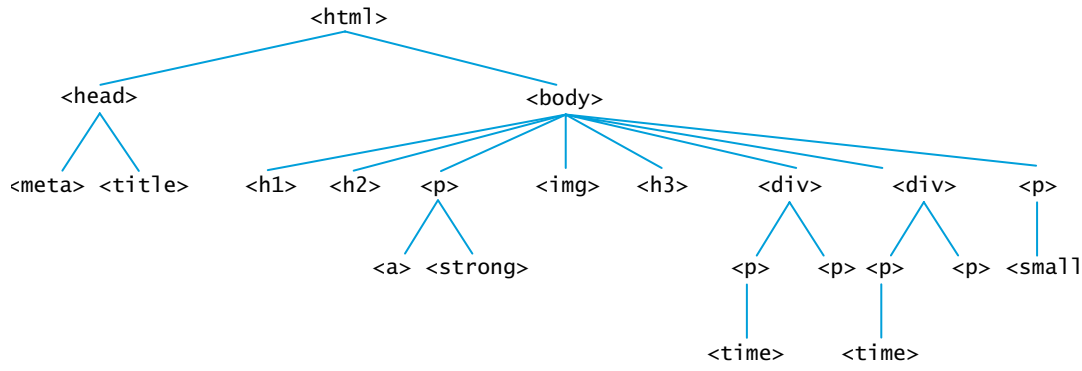


FIGURE 3.3 Document outline/tree

In a similar way, when defining CSS rules, you will need to first use a selector to tell the browser which elements will be affected by the property values. CSS selectors allow you to select individual or multiple HTML elements.

The topic of selectors has become more complicated than it was when we started teaching CSS in the late 1990s. There are now a variety of new selectors that are supported by most modern browsers. Before we get to those, let us look at the three basic selector types that have been around since the earliest CSS2 specification.

3.4.1 Element Selectors

Element selectors select all instances of a given HTML element. The example CSS rules in Figure 3.2 illustrate two element selectors. You can select all elements by using the universal element selector, which is the * (asterisk) character.

You can select a group of elements by separating the different element names with commas. This is a sensible way to reduce the size and complexity of your CSS files, by combining multiple identical rules into a single rule. An example grouped selector is shown in Listing 3.4, along with its equivalent as three separate rules.

3.4.2 Class Selectors

A class selector allows you to simultaneously target different HTML elements regardless of their position in the document tree. If a series of HTML elements have been labeled with the same `class` attribute value, then you can target them for styling by using a class selector, which takes the form: period (.) followed by the class name.

Listing 3.5 illustrates an example of styling using a class selector. The result in the browser is shown in Figure 3.4.

```
/* commas allow you to group selectors */
p, div, aside {
   margin: 0;
   padding: 0;
}
/* the above single grouped selector is equivalent to the
   following: */
p {
   margin: 0;
   padding: 0;
}
div {
   margin: 0;
   padding: 0;
}
aside {
   margin: 0;
   padding: 0;
}
```

LISTING 3.4 Sample grouped selector

PRO TIP

Grouped selectors are often used as a way to quickly **reset** or remove browser defaults. The goal of doing so is to reduce browser inconsistencies with things such as margins, line heights, and font sizes. These reset styles can be placed in their own CSS file (perhaps called reset.css) and linked to the page **before** any other external style sheets. An example of a simplified reset is shown below:

```
html, body, div, span, h1, h2, h3, h4, h5, h6, p {
  margin: 0;
  padding: 0;
  border: 0;
  font-size: 100%;
  vertical-align: baseline;
}
```

3.4.3 Id Selectors

An id selector allows you to target a specific element by its `id` attribute regardless of its type or position. If an HTML element has been labeled with an `id` attribute,

```
<head>
   <title>Share Your Travels </title>
  <style>
          .first {
           font-style: italic;
           color: red;
          }
    </style>
</head>
<body>
   <h1 class="first">Reviews</h1>
   <div>
      <p class="first">By Ricardo on <time>September 15, 2015</time></p>
      <p>Easy on the HDR buddy.</p>
   </div>
   <hr/>

   <div>
      <p class="first">By Susan on <time>October 1, 2015</time></p>
      <p>I love Central Park.</p>
   </div>
   <hr/>
</body>
```

LISTING 3.5 Class selector example

then you can target it for styling by using an id selector, which takes the form: pound/hash (#) followed by the id name.

Listing 3.6 illustrates an example of styling using an id selector. The result in the browser is shown in Figure 3.5.

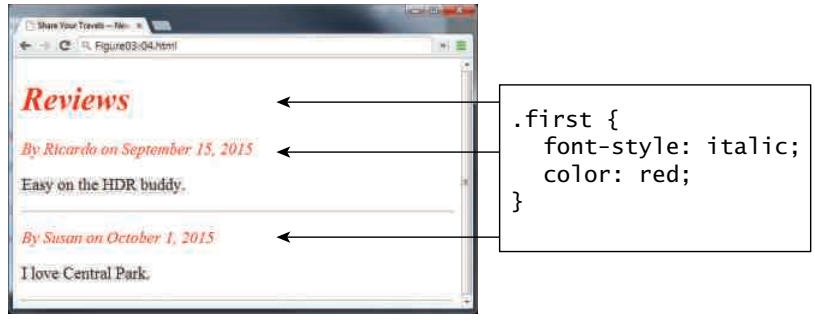


FIGURE 3.4 Class selector example in browser

```
<head lang="en">
   <meta charset="utf-8">
   <title>Share Your Travels -- New York - Central Park</title>
  <style>
        #latestComment {
         font-style: italic;
         color: red;
        }
</style>
</head>
<body>
   <h1>Reviews</h1>
   <div id="latestComment">
      <p>By Ricardo on <time>September 15, 2015</time></p>
      <p>Easy on the HDR buddy.</p>
   </div>
   <hr/>
   <div>
      <p>By Susan on <time>October 1, 2015</time></p>
      <p>I love Central Park.</p>
   </div>
   <hr/>
</body>
```

LISTING 3.6 Id selector example

NOTE

Id selectors should only be used when referencing a single HTML element since an `id` attribute can only be assigned to a single HTML element. Class selectors should be used when (potentially) referencing several related elements.

It is worth noting, however, that the browser is quite forgiving when it comes to id selectors. While you should only use a given id attribute once in the markup, the browser is happy to let you use it multiple times!

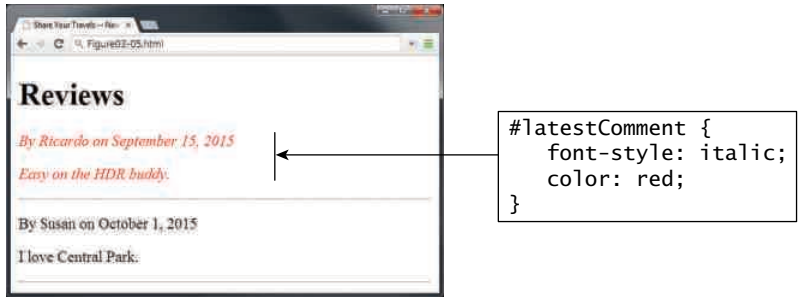


FIGURE 3.5 Id selector example in browser

3.4.4 Attribute Selectors

HANDS-ON EXERCISES

LAB 3 EXERCISE
Attribute Selectors

An attribute selector provides a way to select HTML elements either by the presence of an element attribute or by the value of an attribute. This can be a very powerful technique, but because of uneven support by some of the browsers, not all web authors have used them.

Attribute selectors can be a very helpful technique in the styling of hyperlinks and images. For instance, perhaps we want to make it more obvious to the user when a pop-up tooltip is available for a link or image. We can do this by using the following attribute selector:

```
[title] { … }
```

This will match any element in the document that has a `title` attribute. We can see this at work in Listing 3.7, with the results in the browser shown in Figure 3.6.

```
<head lang="en">
   <meta charset="utf-8">
   <title>Share Your Travels</title>
     <style>
          [title] {
          cursor: help;
          padding-bottom: 3px;
          border-bottom: 2px dotted blue;
          text-decoration: none;
        }
   </style>
</head>
<body>
   <div>
      <img src="images/flags/CA.png" title="Canada Flag" />
      <h2><a href="countries.php?id=CA" title="see posts from Canada">
          Canada</a>
      </h2>
      <p>Canada is a North American country consisting of ... </p>
      <div>
          <img src="images/square/6114907897.jpg"
               title="At top of Sulphur Mountain" />
          <img src="images/square/6592317633.jpg"
               title="Grace Presbyterian Church" />
          <img src="images/square/6592914823.jpg"
               title="Calgary Downtown" />
     </div>
   </div>
</body>
```

LISTING 3.7 Attribute selector example

```
[title] {
   cursor: help;
   padding-bottom: 3px;
   border-bottom: 2px dotted blue;
   text-decoration: none;
}
```

FIGURE 3.6 Attribute selector example in browser

Table 3.4 summarizes some of the most common ways one can construct attribute selectors in CSS3.

Selector	Matches	Example
[]	A specific attribute.	`[title]` Matches any element with a title attribute
[=]	A specific attribute with a specific value.	`a[title="posts from this country"]` Matches any <a> element whose title attribute is exactly `"posts from this country"`
[~=]	A specific attribute whose value matches at least one of the words in a space-delimited list of words.	`[title~="Countries"]` Matches any `title` attribute that contains the word `"Countries"`
[^=]	A specific attribute whose value begins with a specified value.	`a[href^="mailto"]` Matches any <a> element whose `href` attribute begins with `"mailto"`
[*=]	A specific attribute whose value contains a substring.	`img[src*="flag"]` Matches any <img> element whose `src` attribute contains somewhere within it the text `"flag"`
[$=]	A specific attribute whose value ends with a specified value.	`a[href$=".pdf"]` Matches any <a> element whose `href` attribute ends with the text `".pdf"`

TABLE 3.4 Attribute Selectors

3.4.5 Pseudo-Element and Pseudo-Class Selectors

HANDS-ON EXERCISES

LAB 3 EXERCISE
Pseudo-selectors

A pseudo-element selector is a way to select something that does not exist explicitly as an element in the HTML document tree but which is still a recognizable selectable object. For instance, you can select the first line or first letter of any HTML element using a pseudo-element selector. A pseudo-class selector does apply to an HTML element, but targets either a particular state or, in CSS3, a variety of family relationships. Table 3.5 lists some of the more common pseudo-class and pseudo-element selectors.

The most common use of this type of selectors is for targeting link states. By default, the browser displays link text blue and visited text links purple. Listing 3.8 illustrates the use of pseudo-class selectors to style not only the visited and unvisited link colors, but also the hover color, which is the color of the link when the mouse is over the link. Do be aware that this state does not occur on touch screen devices. Note the syntax of pseudo-class selectors: the colon (`:`) followed by the pseudo-class selector name. Do be aware that a space is *not* allowed after the colon.

Believe it or not, the order of these pseudo-class elements is important. The `:link` and `:visited` pseudo-classes should appear before the others. Some developers use a mnemonic to help them remember the order. My favorite is "Lord Vader, Former Handle Anakin" for Link, Visited, Focus, Hover, Active.

Selector	Type	Description
`a:link`	pseudo-class	Selects links that have not been visited
`a:visited`	pseudo-class	Selects links that have been visited
`:focus`	pseudo-class	Selects elements (such as text boxes or list boxes) that have the input focus.
`:hover`	pseudo-class	Selects elements that the mouse pointer is currently above.
`:active`	pseudo-class	Selects an element that is being activated by the user. A typical example is a link that is being clicked.
`:checked`	pseudo-class	Selects a form element that is currently checked. A typical example might be a radio button or a check box.
`:first-child`	pseudo-class	Selects an element that is the first child of its parent. A common use is to provide different styling to the first element in a list.
`:first-letter`	pseudo-element	Selects the first letter of an element. Useful for adding drop-caps to a paragraph.
`:first-line`	pseudo-element	Selects the first line of an element.

TABLE 3.5 Common Pseudo-Class and Pseudo-Element Selectors

```
<head>
   <title>Share Your Travels</title>
  <style>
       a:link {
       text-decoration: underline;
       color: blue;
     }
       a:visited {
       text-decoration: underline;
       color: purple;
     }
       a:hover {
       text-decoration: none;
       font-weight: bold;
     }
       a:active {
       background-color: yellow;
     }
  </style>
</head>
<body>
     <p>Links are an important part of any web page. To learn more about
        links visit the <a href="#">W3C</a> website.</p>
   <nav>
     <ul>
       <li><a href="#">Canada</a></li>
       <li><a href="#">Germany</a></li>
       <li><a href="#">United States</a></li>
     </ul>
   </nav>
</body>
```

LISTING 3.8 Styling a link using pseudo-class selectors

NOTE

Notice the use of the "#" url used in the <a> elements in Listing 3.8. This is a common practice used by developers when they are first testing a design. The designer might know that there is a link somewhere, but the precise URL might still be unknown. In such a case, using the "#" url is helpful: the browser will recognize them as links, but nothing will happen when they are clicked. Later, using the source code editor's search functionality will make it easy to find links that need to be finalized.

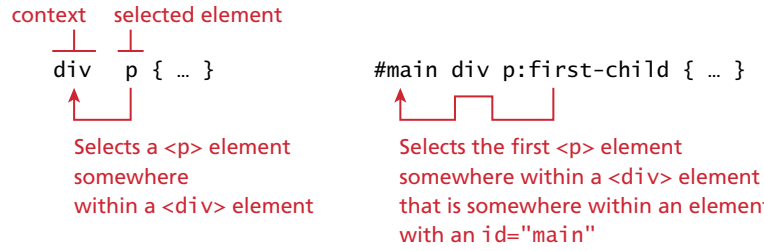


FIGURE 3.7 Syntax of a descendant selection

3.4.6 Contextual Selectors

HANDS-ON EXERCISES

LAB 3 EXERCISE
Contextual Selectors

A **contextual selector** (in CSS3 also called **combinators**) allows you to select elements based on their *ancestors*, *descendants*, or *siblings*. That is, it selects elements based on their context or their relation to other elements in the document tree. While some of these contextual selectors are used relatively infrequently, almost all web authors find themselves using descendant selectors.

A **descendant selector** matches all elements that are contained within another element. The character used to indicate descendant selection is the space character. Figure 3.7 illustrates the syntax and usage of the syntax of the descendant selector.

Table 3.6 describes the other contextual selectors.

Selector	Matches	Example
Descendant	A specified element that is contained somewhere within another specified element.	`div p` Selects a <p> element that is contained somewhere within a element. That is, the <p> can be any descendant, not just a child.
Child	A specified element that is a direct child of the specified element.	`div>h2` Selects an <h2> element that is a child of a element.
Adjacent sibling	A specified element that is the next sibling (i.e., comes directly after) of the specified element.	`h3+p` Selects the first <p> after any <h3>.
General sibling	A specified element that shares the same parent as the specified element.	`h3~p` Selects all the <p> elements that share the same parent as the <h3>.

TABLE 3.6 Contextual Selectors

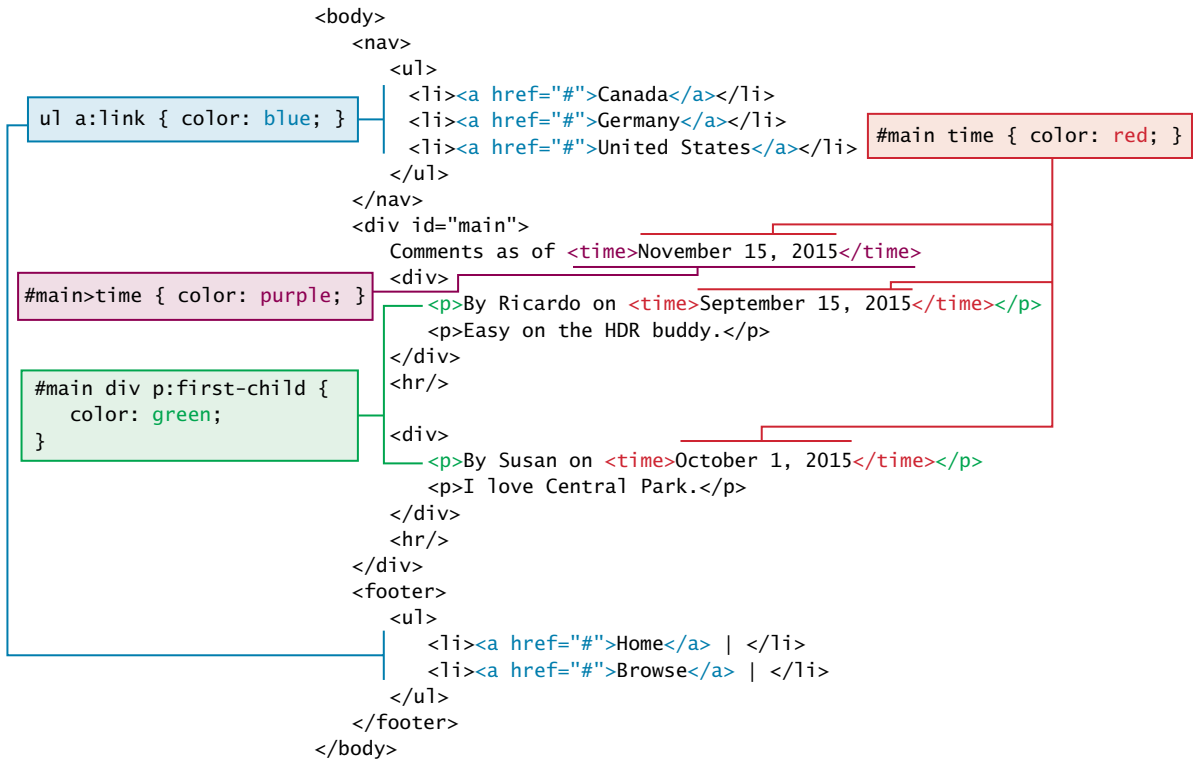


FIGURE 3.8 Contextual selectors in action

Figure 3.8 illustrates some sample uses of a variety of different contextual selectors.

NOTE

You can combine contextual selectors with grouped selectors. The comma is like the logical OR operator. Thus the grouped selector:

```
div#main div time, footer ul li { color: red; }
```

is equivalent to:

```
div#main div time { color: red; }
footer ul li { color: red; }
```

3.5 The Cascade: How Styles Interact

HANDS-ON EXERCISES

LAB 3 EXERCISE
The CSS Cascade

In an earlier Pro Tip in this chapter, it was mentioned that in fact there are three different types of style sheets: author-created, user-defined, and the default browser style sheet. As well, it is possible within an author-created stylesheet to define multiple rules for the same HTML element. For these reasons, CSS has a system to help the browser determine how to display elements when different style rules conflict.

The "Cascade" in CSS refers to how conflicting rules are handled. The visual metaphor behind the term **cascade** is that of a mountain stream progressing downstream over rocks (and not that of a popular dishwashing detergent). The downward movement of water down a cascade is meant to be analogous to how a given style rule will continue to take precedence with child elements (i.e., elements "below" in a document outline as shown in Figure 3.3).

CSS uses the following cascade principles to help it deal with conflicts: inheritance, specificity, and location.

3.5.1 Inheritance

Inheritance is the first of these cascading principles. Many (but not all) CSS properties affect not only themselves but their descendants as well. Font, color, list, and text properties (from Table 3.1) are inheritable; layout, sizing, border, background, and spacing properties are not.

Figures 3.9 and 3.10 illustrate CSS inheritance. In the first example, only some of the property rules are inherited for the `<body>` element. That is, only the body element (thankfully!) will have a thick green border and the 100-px margin; however, all the text in the other elements in the document will be in the Arial font and colored red.

In the second example in Figure 3.10, you can assume there is no longer the body styling but instead we have a single style rule that styles *all* the `<div>` elements. The `<p>` and `<time>` elements within the `<div>` inherit the bold font-weight property but not the margin or border styles.

However, it is possible to tell elements to inherit properties that are normally not inheritable, as shown in Figure 3.11. In comparison to Figure 3.10, notice how the `<p>` elements nested within the `<div>` elements now inherit the border and margins of their parent.

3.5.2 Specificity

Specificity is how the browser determines which style rule takes precedence when more than one style rule could be applied to the same element. In CSS, the more specific the selector, the more it takes precedence (i.e., overrides the previous definition).

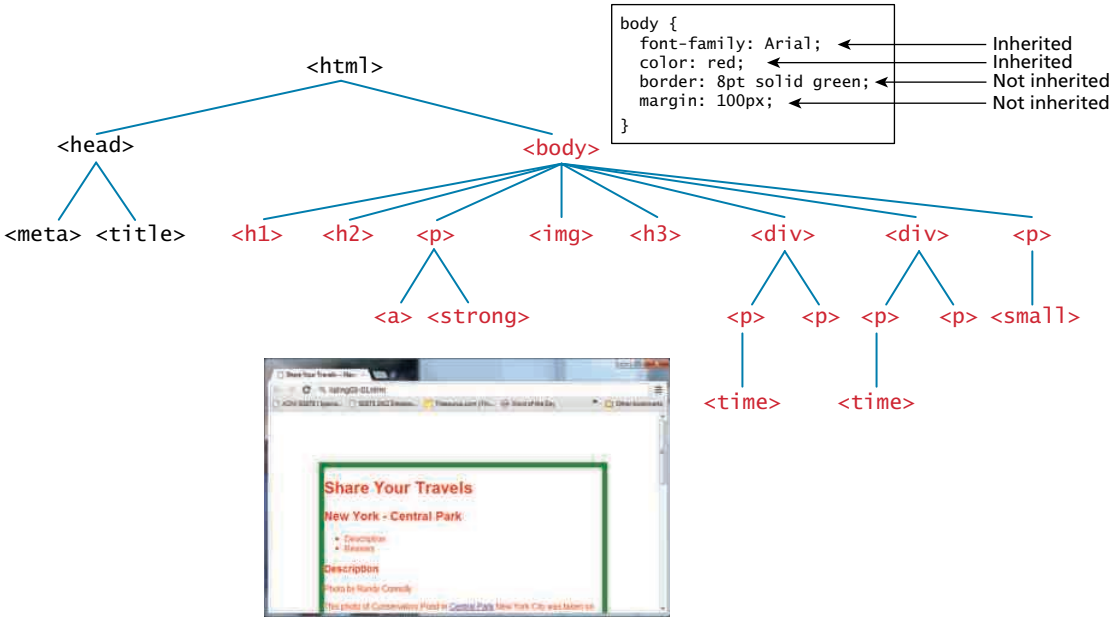


FIGURE 3.9 Inheritance

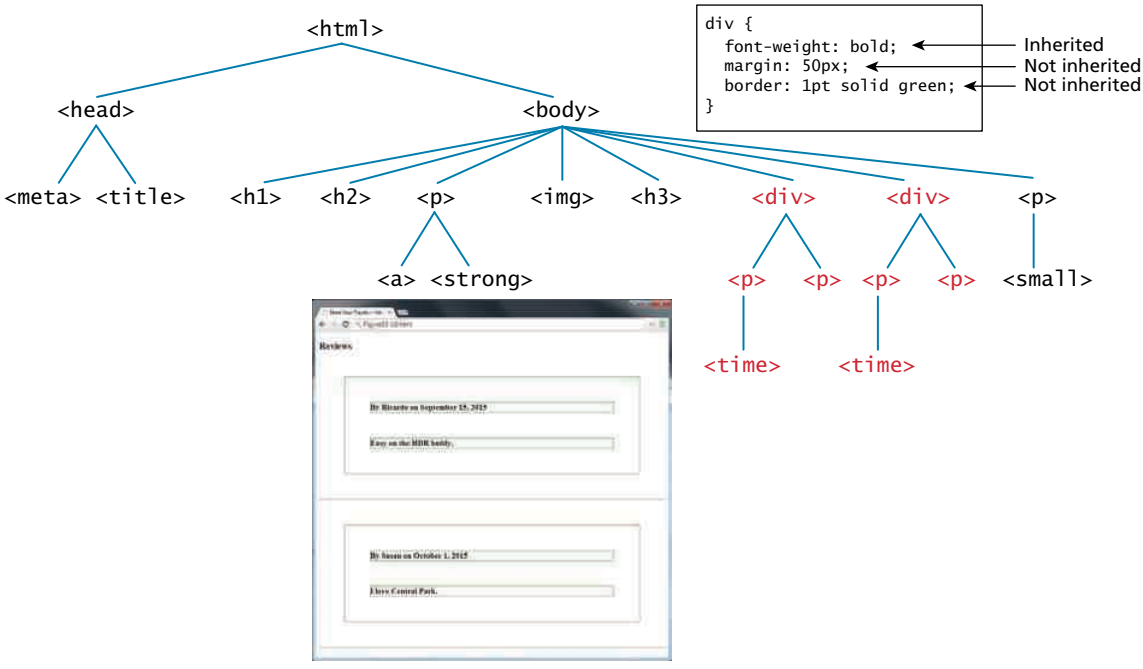
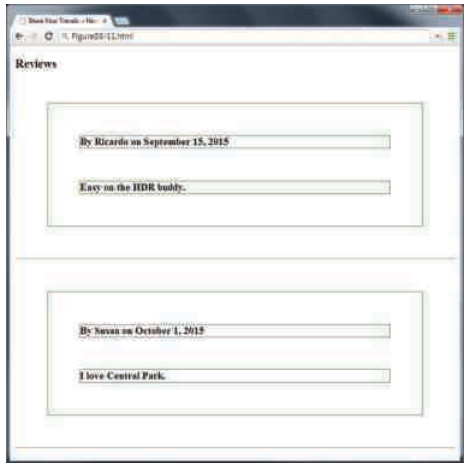


FIGURE 3.10 More Inheritance




```
div {
  font-weight: bold;
  margin: 50px;
  border: 1pt solid green;
}
p {
  border: inherit;
  margin: inherit;
}
```

```
<h3>Reviews</h3>
<div>
   <p>By Ricardo on <time>September 15, 2015</time></p>
   <p>Easy on the HDR buddy.</p>
</div>
<hr/>

<div>
   <p>By Susan on <time>October 1, 2015</time></p>
   <p>I love Central Park.</p>
</div>
<hr/>
```

FIGURE 3.11 Using the `inherit` value

NOTE

Most CSS designers tend to avoid using the `inherit` property since it can usually be replaced with clear and obvious rules. For instance, in Figure 3.11, the use of inherit can be replaced with the more verbose, but clearer, set of rules:

```
div {
  font-weight: bold;
}
p, div {
  margin: 50px;
  border: 1pt solid green;
}
```

Another way to define specificity is by telling you how it works. The way that specificity works in the browser is that the browser assigns a weight to each style rule; when several rules apply, the one with the greatest weight takes precedence.

In the example shown in Figure 3.12, the color and font-weight properties defined in the `<body>` element are inheritable and thus potentially applicable to all the child elements contained within it. However, because the `<div>` and `<p>` elements also have the same properties set, they *override* the value defined for the `<body>` element because their selectors (`<div>` and `<p>`) are more specific. As a consequence, their font-weight is normal and their text is colored either green or magenta.

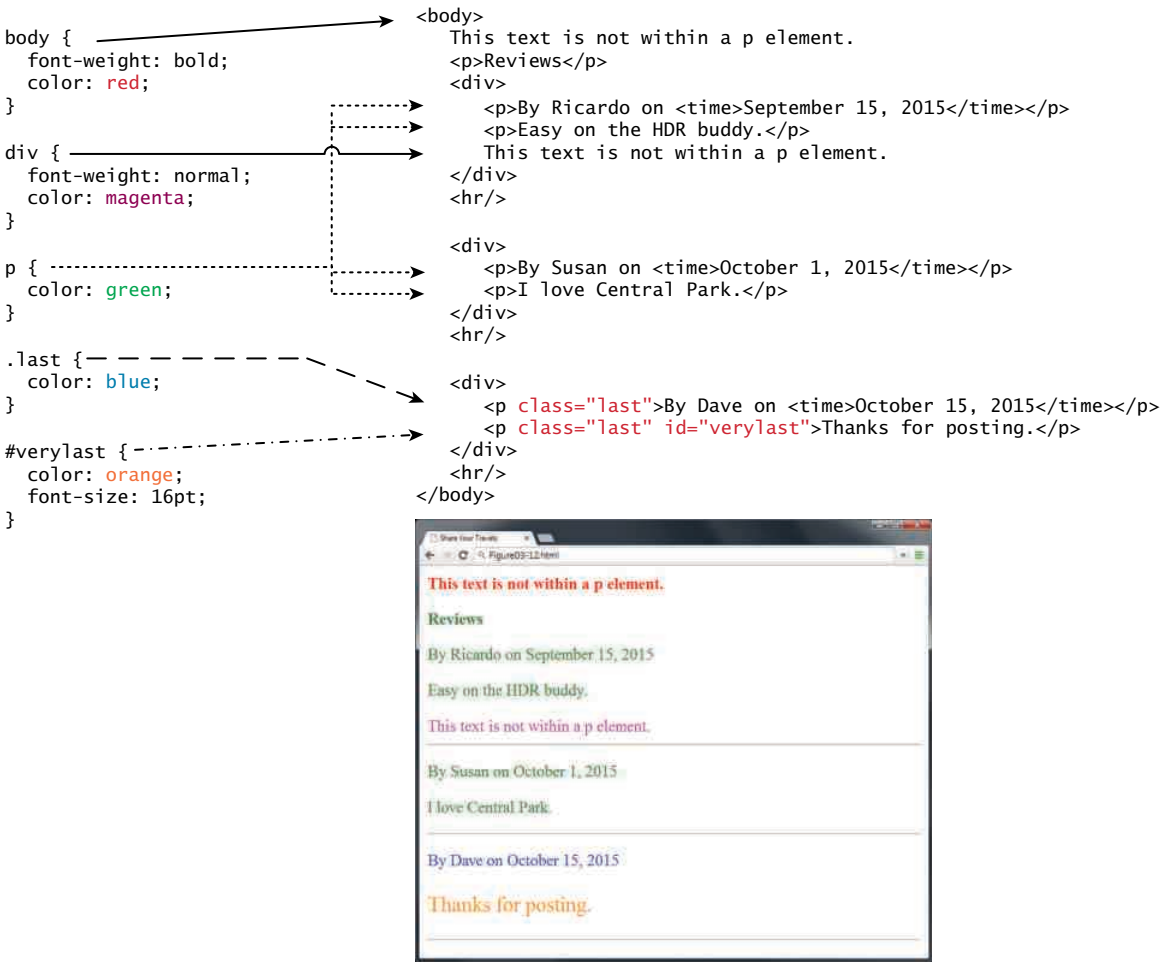


FIGURE 3.12 Specificity

As you can see in Figure 3.12, class selectors take precedence over element selectors, and id selectors take precedence over class selectors. The precise algorithm the browser is supposed to use to determine specificity is quite complex.[6] A simplified version is shown in Figure 3.13.

3.5.3 Location

Finally, when inheritance and specificity cannot determine style precedence, the principle of location will be used. The principle of location is that when rules have the same specificity, then the latest are given more weight. For instance, an inline style will override one defined in an external author style sheet or an embedded style sheet.

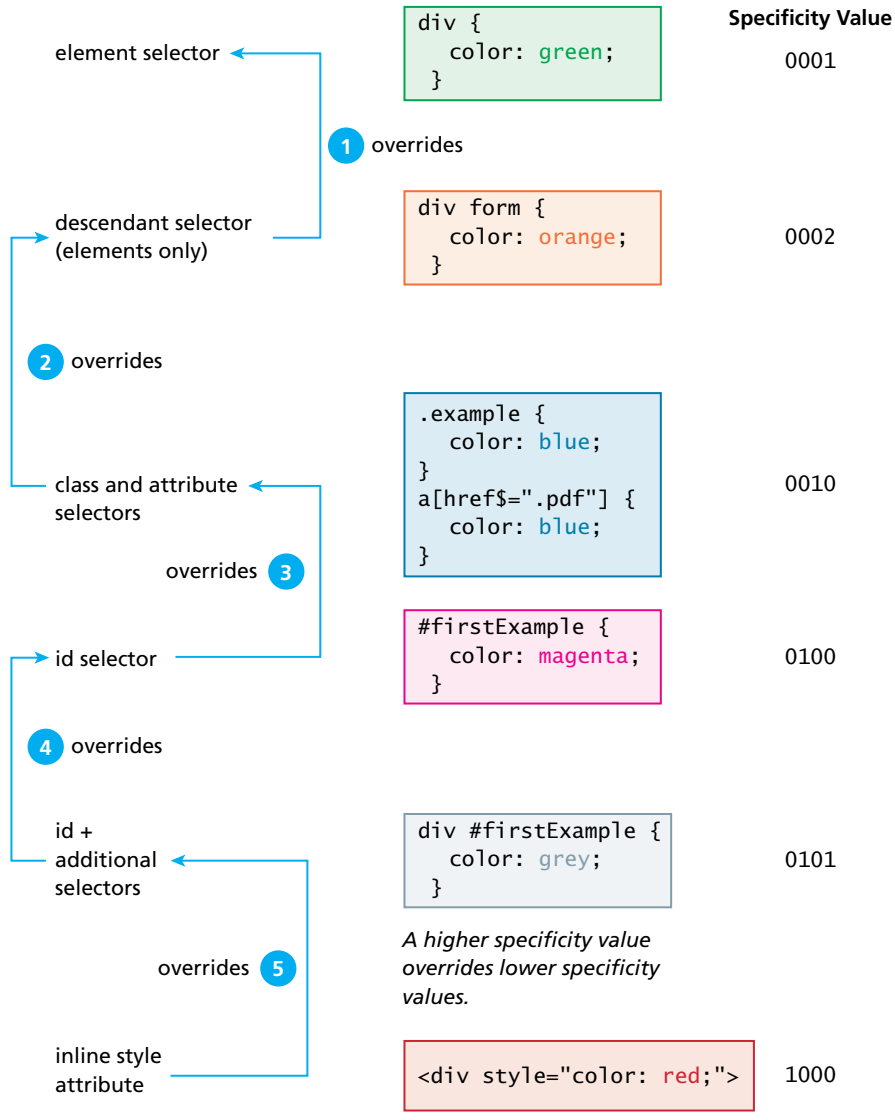


FIGURE 3.13 Specificity algorithm

Similarly, an embedded style will override an equally specific rule defined in an external author style sheet if it appears after the external sheet's `<link>` element. Styles defined in external author style sheet X will override styles in external author style sheet Y if X's `<link>` element is after Y's in the HTML document. Similarly,

PRO TIP

The algorithm that is used to determine specificity of any given element is defined by the W3C as follows.

- First count 1 if the declaration is from a "style" attribute in the HTML, 0 otherwise (let that value = a).
- Count the number of ID attributes in the selector (let that value = b).
- Count the number of class selectors, attribute selectors, and pseudo-classes in the selector (let that value = c).
- Count the number of element names and pseudo-elements in the selector (let that value = d).
- Finally, concatenate the four numbers a+b+c+d together to calculate the selector's specificity.

The following sample selectors are given along with their specificity value.

```
<tag style="color: red">                  1000
body .example                             0011
body .example strong                      0012
div#first                                 0101
div#first .error                          0111
#footer .twitter a                        0111
#footer .twitter a:hover                  0121
body aside#left div#cart strong.price     0214
```

It should be noted that in general you don't really need to know the specificity algorithm in order to work with CSS. However, knowing it can be invaluable when one is trying to debug a CSS problem. During such a time, you might find yourself asking the question, "Why isn't my CSS rule doing anything? Why is the browser ignoring it?" Quite often the answer to that question is that a rule with a higher specificity is taking precedence.

when the same style property is defined multiple times within a single declaration block, the last one will take precedence.

Figure 3.14 illustrates how location affects precedence. Can you guess what will be the color of the sample text in Figure 3.14?

The answer to the question is: The color of the sample text in Figure 3.14 will be red. What would be the color of the sample text if there wasn't an inline style definition?

It would be magenta.

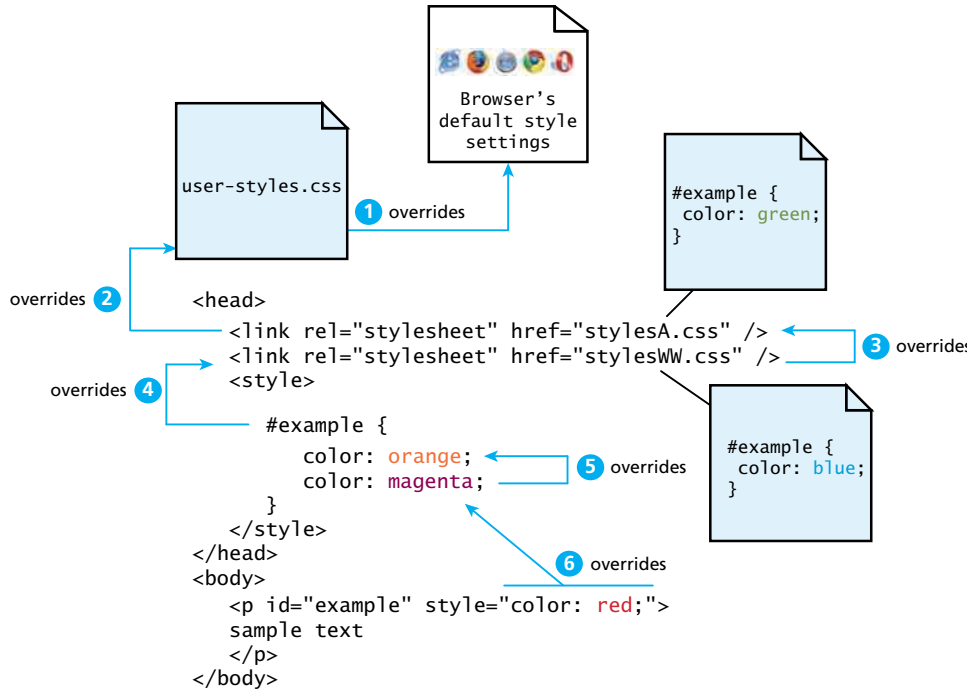


FIGURE 3.14 Location

PRO TIP

There is one exception to the principle of location. If a property is marked with `!important` (which does *not* mean *NOT* important, but instead means *VERY* important) in an author-created style rule, then it will override any other author-created style regardless of its location. The only exception is a style marked with `!important` in a user style sheet: such a rule will override all others. Of course very few users know how to do this, so it is a pretty uncommon scenario.

3.6 The Box Model

In CSS, all HTML elements exist within an element box shown in Figure 3.15. In order to become proficient with CSS, you must become familiar with the element box.

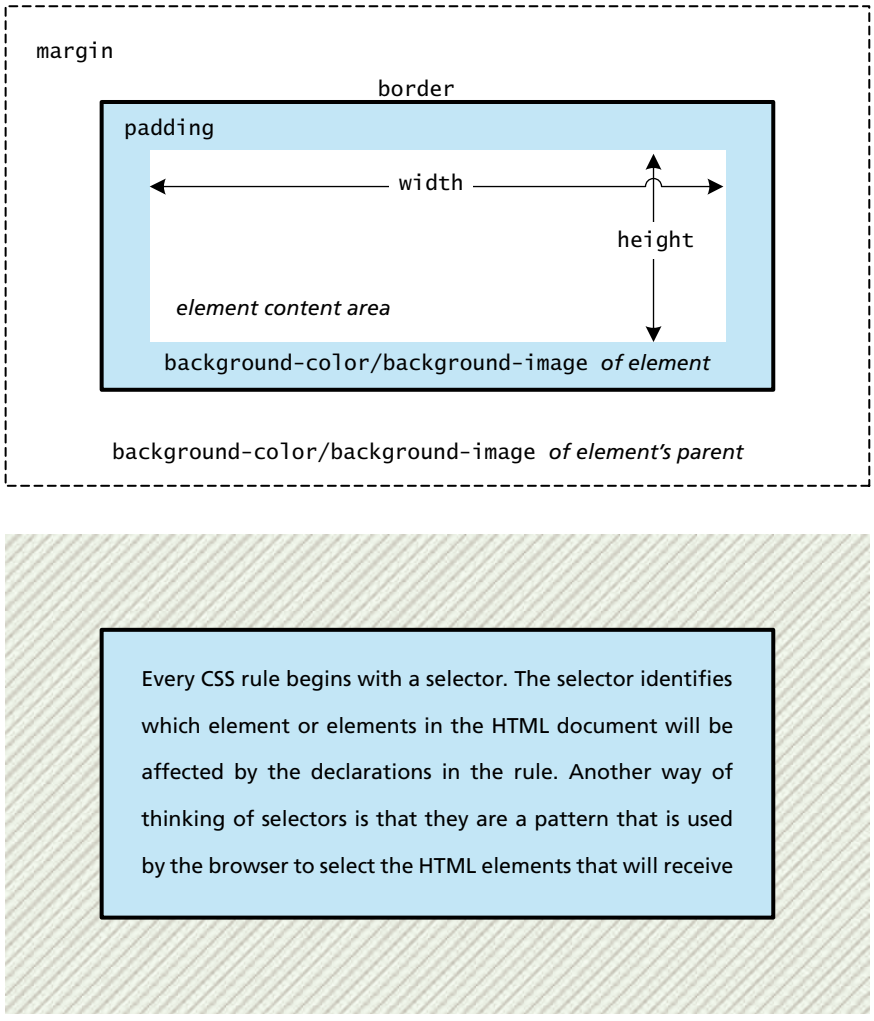


FIGURE 3.15 CSS box model

3.6.1 Background

As can be seen in Figure 3.15, the background color or image of an element fills an element out to its border (if it has one, that is). In contemporary web design, it has become extremely common to use CSS to display purely presentational images (such as background gradients and patterns, decorative images, etc.) rather than using the `<img>` element. Table 3.7 lists the most common background properties.

HANDS-ON EXERCISES

LAB 3 EXERCISE
Background Style

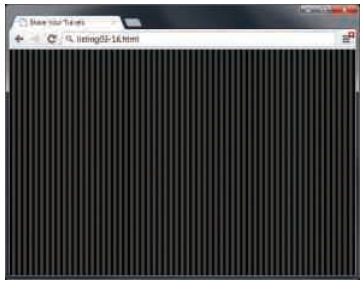
Property	Description
background	A combined shorthand property that allows you to set multiple background values in one property. While you can omit properties with the shorthand, do remember that any omitted properties will be set to their default value.
background-attachment	Specifies whether the background image scrolls with the document (default) or remains fixed. Possible values are: `fixed`, `scroll`.
background-color	Sets the background color of the element. You can use any of the techniques shown in Table 3.2 for specifying the color.
background-image	Specifies the background image (which is generally a jpeg, gif, or png file) for the element. Note that the URL is relative to the CSS file and not the HTML. CSS3 introduced the ability to specify multiple background images.
background-position	Specifies where on the element the background image will be placed. Some possible values include: `bottom`, `center`, `left`, and `right`. You can also supply a pixel or percentage numeric position value as well. When supplying a numeric value, you must supply a horizontal/vertical pair; this value indicates its distance from the top left corner of the element, as shown in Figure 3.16.
background-repeat	Determines whether the background image will be repeated. This is a common technique for creating a tiled background (it is in fact the default behavior), as shown in Figure 3.17. Possible values are: `repeat`, `repeat-x`, `repeat-y`, and `no-repeat`.
background-size	New to CSS3, this property lets you modify the size of the background image.

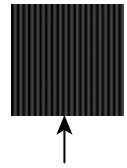
TABLE 3.7 Common Background Properties

3.6.2 Borders

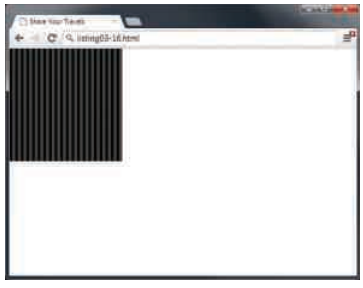
Borders provide a way to visually separate elements. You can put borders around all four sides of an element, or just one, two, or three of the sides. Table 3.8 lists the various border properties.

Border widths are perhaps the one exception to the general advice against using the pixel measure. Using `em` units or percentages for border widths can result in unpredictable widths as the different browsers use different algorithms (some round up, some round down) as the zoom level increases or decreases. For this reason, border widths are almost always set to pixel units.

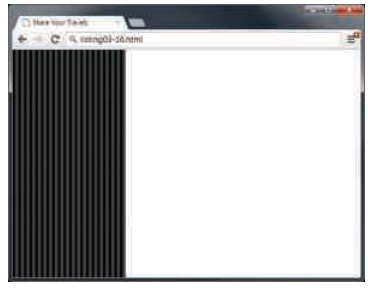




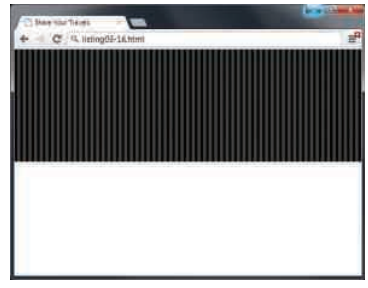
```
background-image: url(../images/backgrounds/body-background-tile.gif);
background-repeat: repeat;
```



```
background-repeat: no-repeat;
```

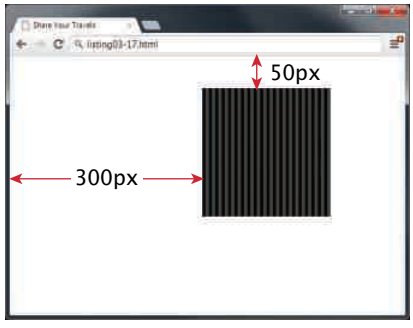


```
background-repeat: repeat-y;
```



```
background-repeat: repeat-x;
```

FIGURE 3.16 Background repeat




```
body {
        background: white url(../images/backgrounds/body-background-tile.gif) no-repeat;
        background-position: 300px 50px;
}
```

FIGURE 3.17 Background position

3.6.3 Margins and Padding

Margins and padding are essential properties for adding white space to a web page, which can help differentiate one element from another. Figure 3.19 illustrates how these two properties can be used to provide spacing and element differentiation.

HANDS-ON EXERCISES

LAB 3 EXERCISE
Borders, Margins, and Padding

Property	Description
border	A combined shorthand property that allows you to set the style, width, and color of a border in one property. The order is important and must be: `border-style border-width border-color`
border-style	Specifies the line type of the border. Possible values are: `solid, dotted, dashed, double, groove, ridge, inset, and outset.`
border-width	The width of the border in a unit (but not percents). A variety of keywords (`thin`, `medium`, etc.) are also supported.
border-color	The color of the border in a color unit.
border-radius	The radius of a rounded corner.
border-image	The URL of an image to use as a border.

TABLE 3.8 Border Properties

NOTE

With border, margin, and padding properties, it is possible to set the properties for one or more sides of the element box in a single property, or to set them individually using separate properties. For instance, we can set the side properties individually:

```
border-top-color: red;          /* sets just the top side */
border-right-color: green;      /* sets just the right side */
border-bottom-color: yellow;    /* sets just the bottom side */
border-left-color: blue;        /* sets just the left side */
```

Alternately, we can set all four sides to a single value via:

```
border-color: red;              /* sets all four sides to red */
```

Or we can set all four sides to different values via:

```
border-color: red green orange blue;
```

When using this multiple values shortcut, they are applied in clockwise order starting at the top. Thus the order is: top right bottom left as shown in Figure 3.18. The mnemonic TRouBLe might help you memorize this order.

FIGURE 3.18 CSS TRBL (Trouble) shortcut

Another shortcut is to use just two values; in this case the first value sets top and bottom, while the second sets the right and left.

```
border-color: red yellow;    /* top+bottom=red, right+left=yellow */
```

As you can see in Figure 3.15 and Figure 3.19, margins add spacing around an element's content, while padding adds spacing within elements. Borders divide the margin area from the padding area.

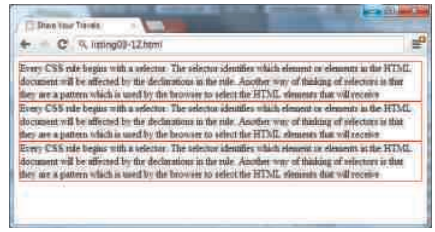
There is a very important thing to notice about the margins in Figure 3.19. Did you notice that the space between paragraphs one and two and between two and three is the same as the space before paragraph one and after paragraph three? This is due to the fact that adjoining vertical margins collapse.

Figure 3.20 illustrates how adjoining vertical margins collapse in the browser. If overlapping margins did not collapse, then margin space for 2 would be 180 px (90 px for the bottom margin of the first `<div>` + 90 px for the top margin of the second `<div>`), while the margins for 4 and 5 would be 100 px. However, as you can see in Figure 3.20, this is not the case.

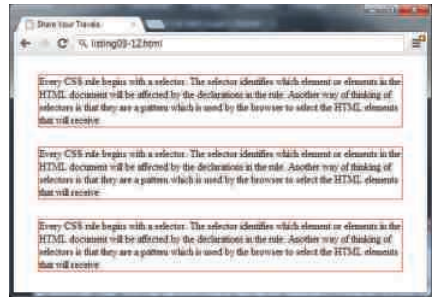
The W3C specification defines this behavior as **collapsing margins**:

> *In CSS, the adjoining margins of two or more boxes (which might or might not be siblings) can combine to form a single margin. Margins that combine this way are said to collapse, and the resulting combined margin is called a collapsed margin.*

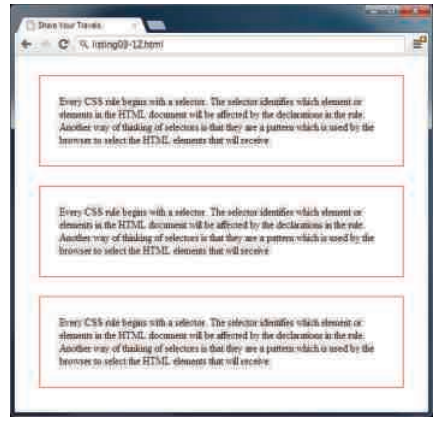
What this means is that when the **vertical** margins of two elements touch, only the largest margin value of the elements will be displayed, while the smaller margin value will be collapsed to zero. Horizontal margins, on the other hand, **never** collapse.



```
p {
    border: solid 1pt red;
    margin: 0;
    padding: 0;
}
```



```
p {
    border: solid 1pt red;
    margin: 30px;
    padding: 0;
}
```



```
p {
    border: solid 1pt red;
    margin: 30px;
    padding: 30px;
}
```

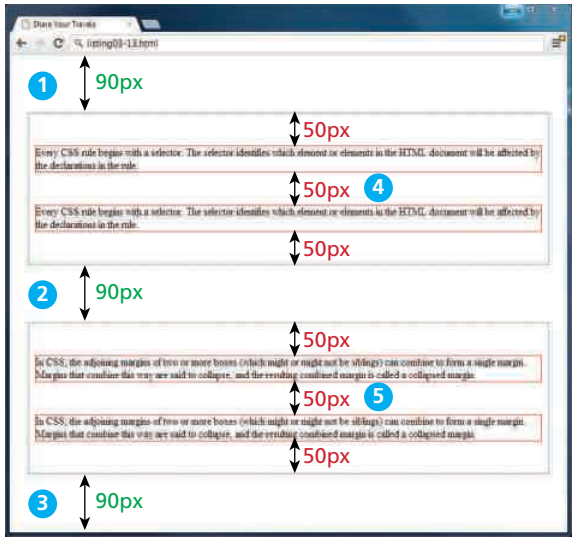
FIGURE 3.19 Borders, margins, and padding provide element spacing and differentiation

To complicate matters even further, there are a large number of special cases in which adjoining vertical margins do **not** collapse (see the W3C Specification for more detail).

From our experience, collapsing (or not collapsing) margins are one of the main problems (or frustrations) that our students face when working with CSS.

3.6.4 Box Dimensions

Box dimensions (i.e., the `width` and `height` properties) also frequently trouble new CSS authors. Why is this the case?




```
<div>
  <p>Every CSS rule ...</p>
  <p>Every CSS rule ...</p>
</div>
<div>
  <p>In CSS, the adjoining ... </p>
  <p>In CSS, the adjoining ... </p>
</div>
```

```
div {
   border: dotted 1pt green;
   padding: 0;
   margin: 90px 20px;
}
```

```
p {
   border: solid 1pt red;
   padding: 0;
   margin: 50px 20px;
}
```

FIGURE 3.20 Collapsing vertical margins

One reason is that only block-level elements and nontext inline elements such as images have a width and height that you can specify. By default (in CSS this is the `auto` value), the width of and height of elements is the actual size of the content. For text, this is determined by the font size and font face; for images, the width and height of the actual image in pixels.

Since the width and the height only refer to the size of the content area, the total size of an element is equal to the size of its content area plus the sum of its padding, borders, and margins. This is something that tends to give beginning CSS students trouble. Figure 3.21 illustrates the default `content-box` element sizing behavior. It also shows the newer alternative `border-box` approach, which is perhaps more intuitive, but which requires vendor prefixes for it to work on all recent browsers.

For block-level elements such as `<p>` and `<div>` elements, there are limits to what the `width` and `height` properties can actually do. You can shrink the width, but the content still needs to be displayed, so the browser may very well ignore the height that you set. As you can see in Figure 3.22, the default width is the browser viewport. But in the second screen capture in the image, with the changed width and height, there is not enough space for the browser to display all the content within the element. So while the browser will display a background color of 200×100 px (i.e., the size of the element as set by the `width` and `height` properties), the height of the actual textual content is much larger (depending on the font size).

It is possible to control what happens with the content if the box's width and height are not large enough to display the content using the `overflow` property, as shown in Figure 3.23.

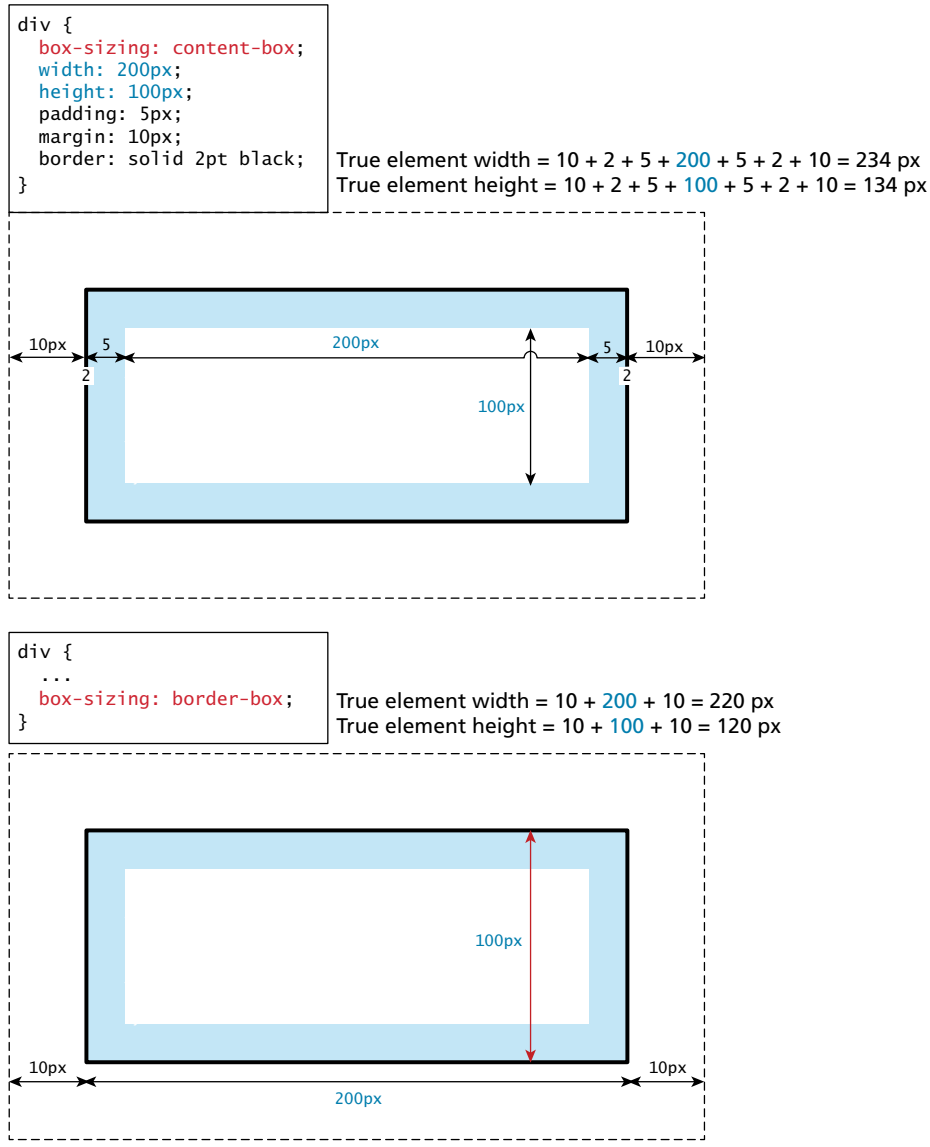


FIGURE 3.21 Calculating an element's true size

While the example CSS in Figure 3.22 uses pixels for its measurement, many contemporary designers prefer to use percentages or em units for widths and heights. When you use percentages, the size is relative to the size of the parent element, while using ems makes the size of the box relative to the size of the text

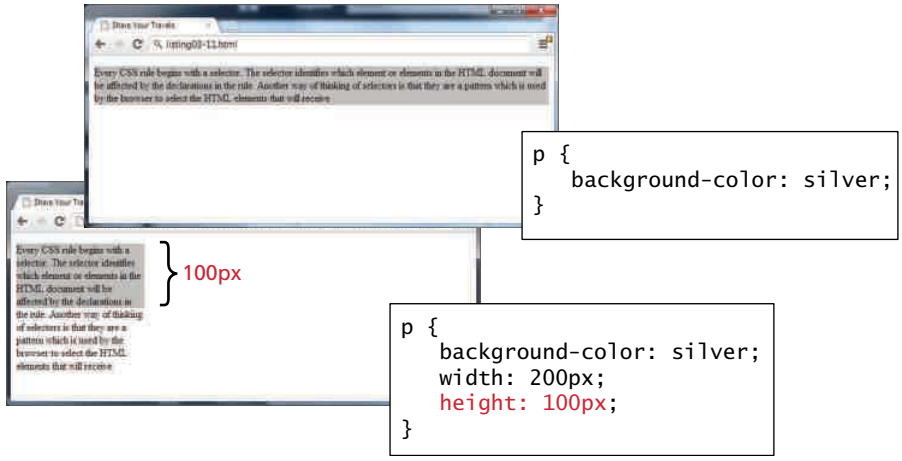


FIGURE 3.22 Limitations of height property

NOTE

Vendor prefixes are a way for browser manufacturers to add new CSS properties that might **not** be part of the formal CSS specification. The prefix for Chrome and Safari is –webkit-, for Firefox it is -moz-, for Internet Explorer it is -ms-, and for Opera –o-. Thus, to set the box-sizing property to border-box, we would have to write something like this:

```
-webkit-box-sizing: border-box;
-moz-box-sizing: border-box;
/* Opera and IE support this property without prefix */
box-sizing: border-box;
```

There is currently a fair degree of controversy about vendor prefixes. On the one hand, they let web authors take advantage of a single browser's support for a new CSS feature (whether part of the W3C standard or not) without waiting for it to become standard across all browsers. But on the other hand, the proliferation of vendor prefixes has made contemporary CSS files significantly more complicated.

More seriously, there has been a great deal of concern in the browser community that many developers are only adding webkit vendor prefixes; as a consequence, a site on Chrome and Safari (i.e., the main webkit browsers) may look better than competing browsers.

In the spring of 2012, developers at Mozilla and Microsoft announced that their browsers were going to support the -webkit- prefix. This has many developers worried that this turns Apple and Google and not the W3C into the de facto CSS standard maker moving forward.

FIGURE 3.23 Overflow property

within it. The rationale behind using these relative measures is to make one's design scalable to the size of the browser or device that is viewing it. Figure 3.24 illustrates how percentages will make elements respond to the current size of the browser.

One of the problems with using percentages as the unit for sizes is that as the browser window shrinks too small or expands too large (for instance on a wide-screen monitor), elements might become too small or too large. You can put absolute pixel constraints on the minimum and maximum sizes via the `min-width`, `min-height`, `max-width`, and `max-height` properties.

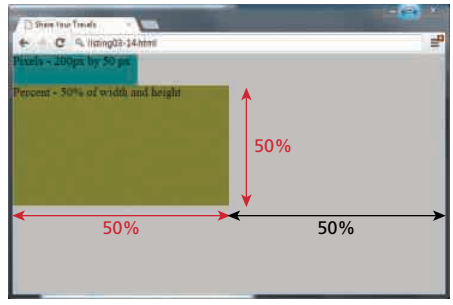
PRO TIP

Developer tools in current browsers make it significantly easier to examine and troubleshoot CSS than was the case a decade ago. Figure 3.25 illustrates how you can use the various browsers' CSS inspection tools to examine, for instance, the box values for a selected element.

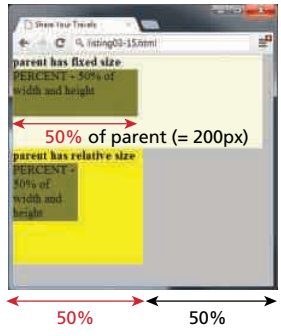
Another way to experiment and learn CSS is to use an online CSS "playground," such as cssdesk.com or dabblet.comole. These sites allow you to type in CSS and see its effect immediately.

```
<style>
  html,body {
     margin:0;
     width:100%;
     height:100%;
     background: silver;
  }
  .pixels {
     width:200px;
     height:50px;
     background: teal;
  }
  .percent {
     width:50%;
     height:50%;
     background: olive;
  }
```


```
<body>
   <div class="pixels">
     Pixels - 200px by 50 px
   </div>
   <div class="percent">
     Percent - 50% of width and height
   </div>
</body>
```




```
  .parentFixed {
     width:400px;
     height:150px;
     background: beige;
  }
  .parentRelative {
     width:50%;
     height:50%;
     background: yellow;
  }
</style>
```




```
<body>
<div class="parentFixed">
   <strong>parent has fixed size</strong>
   <div class="percent">
      PERCENT - 50% of width and height
   </div>
</div>
<div class="parentRelative">
   <strong>parent has relative size</strong>
   <div class="percent">
      PERCENT - 50% of width and height
   </div>
</div>
</body>
```

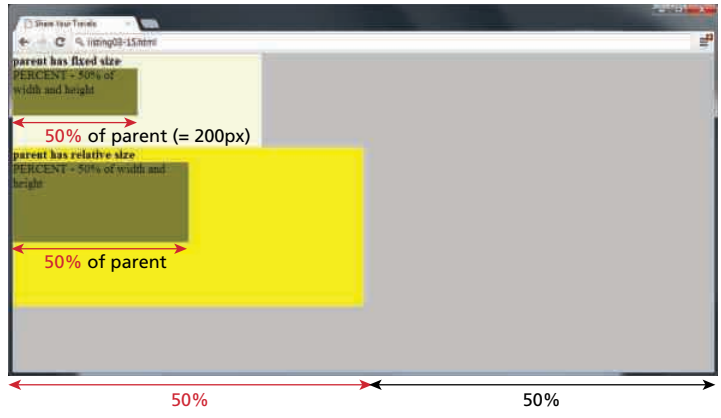


FIGURE 3.24 Box sizing via percents

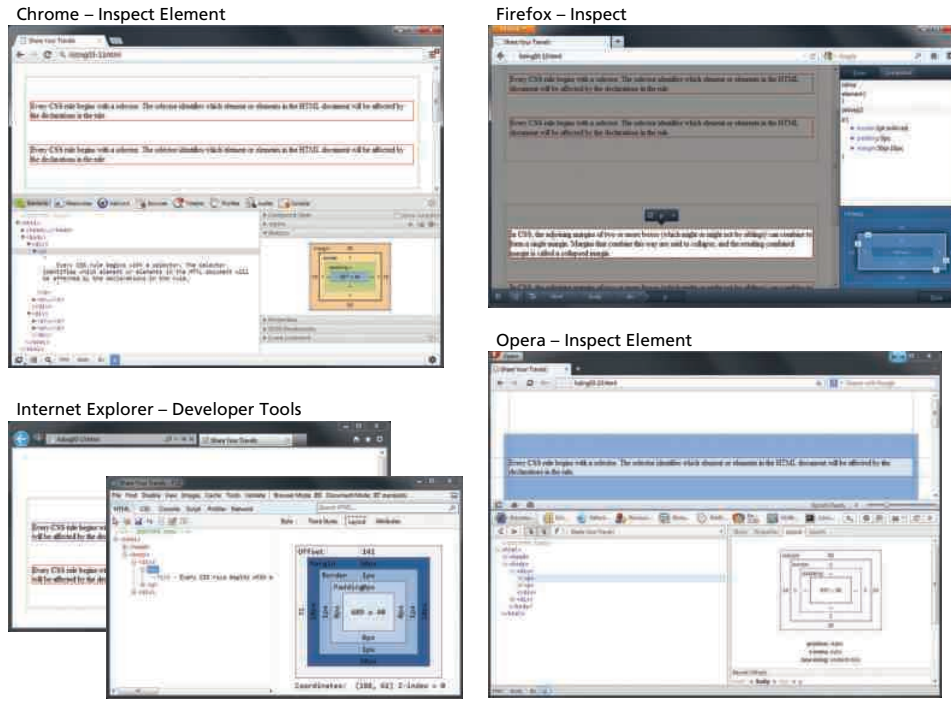


FIGURE 3.25 Inspecting CSS using developer tools within modern browsers

3.7 CSS Text Styling

CSS provides two types of properties that affect text. The first we call font properties because they affect the font and its appearance. The second type of CSS text properties are referred to here as paragraph properties since they affect the text in a similar way no matter which font is being used.

Many of the most common font properties as shown in Table 3.9 will at first glance be familiar to anyone who has used a word processor. There are, however, a range of interesting potential problems when working with fonts on the web (as compared to a word processor).

3.7.1 Font Family

The first of these problems involves specifying the font family. A word processor on a desktop machine can make use of any font that is installed on the computer; browsers are no different. However, just because a given font is available on the web developer's computer, it does not mean that that same font will be available for all users who view the site. For this reason, it is conventional to supply a so-called web

HANDS-ON EXERCISES

LAB 3 EXERCISE
CSS Fonts

Property	Description
font	A combined shorthand property that allows you to set the family, style, size, variant, and weight in one property. While you do not have to specify each property, you must include at a minimum the font size and font family. In addtion, the order is important and must be: `style weight variant size font-family`
font-family	Specifies the typeface/font (or generic font family) to use. More than one can be specified.
font-size	The size of the font in one of the measurement units.
font-style	Specifies whether `italic`, `oblique` (i.e., skewed by the browser rather than a true italic), or `normal`.
font-variant	Specifies either `small-caps` text or none (i.e., regular text).
font-weight	Specifies either `normal`, `bold`, `bolder`, `lighter`, or a value between 100 and 900 in multiples of 100, where larger number represents weightier (i.e., bolder) text.

TABLE 3.9 Font Properties

font stack, that is, a series of alternate fonts to use in case the original font choice is not on the user's computer. As you can see in Figure 3.26, the alternatives are separated by commas; as well, if the font name has multiple words, then the entire name must be enclosed in quotes.

Notice the final generic font family choice in Figure 3.26. The `font-family` property supports five different generic families; the browser supports a typeface from each family. The different generic font families are shown in Figure 3.27.

While there is no real limit to the number of fonts that one can specify with the `font-family` property, in practice, most developers will typically choose three or four stylistically similar fonts.

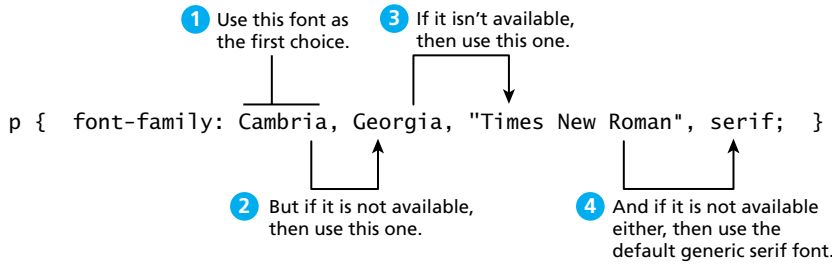


FIGURE 3.26 Specifying the font family

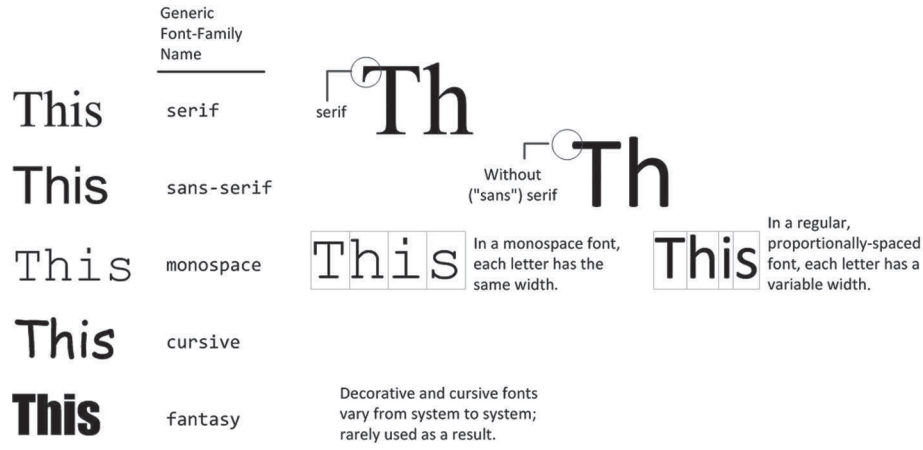


FIGURE 3.27 The different font families

One common approach is to make your font stack contain, in this order, the following: *ideal*, *alternative*, *common*, and then *generic*. Take for instance, the following font stack:

```
font-family { "Hoefler Text", Cambria, "Times New Roman", serif; }
```

You might love the appearance of Hoefler Text, which is installed on most Macs, so it is your *ideal* choice for your site; however, it is not installed on very many PCs or Android devices. Cambria is on most PC and Mac computers and is your *alternative* choice. Times New Roman is installed on almost all PCs and Macs so it is a safe *common* choice; but because you would prefer Cambria to be used instead of Times New Roman, you placed Cambria first. Finally, Android or Blackberry users might not have any of these fonts, so you finished the font stack with the *generic* serif since all your other choices are all serif fonts.

Websites such as **http://cssfontstack.com/** can provide you with information about how prevalent a given font is on PC and Windows computers, so you can see how likely it is that ideal font is even installed.

Another factor to think about when putting together a font stack is the **x-height** (i.e., the height of the lowercase letters, which is generally correlated to the width of the characters) of the different typefaces, as that will have the most impact on things such as characters per line and hence word flow.

HANDS-ON EXERCISES

LAB 3 EXERCISE
CSS Font Sizes

3.7.2 Font Sizes

Another potential problem with web fonts is font sizes. In a print-based program such as a word processor, specifying a font size is unproblematic. Making some text

NOTE

Over the past few years, the most recent browser versions have begun to support the `@font-face` selector in CSS. This selector allows you to use a font on your site even if it is not installed on the end user's computer. While `@font-face` has been part of CSS for quite some time, the main stumbling block has been licensing. Fonts are like software in that they are licensed and protected forms of intellectual property.

Due to the ongoing popularity of open source font sites such as Google Web Fonts (http://www.google.com/webfonts) and Font Squirrel (http://www.fontsquirrel.com/), `@font-face` seems to have gained a critical mass of widespread usage.

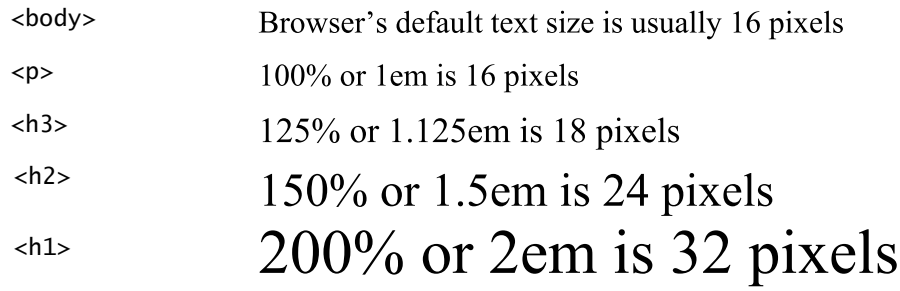
The following example illustrates how to use Droid Sans (a system font also used by Android devices) from Google Web Fonts using @font-face.

```
@font-face {
  font-family: "Droid Sans";
  font-style: normal;
  font-weight: 400;
  src: local("Droid Sans"), local("DroidSans"),
    url(http://themes.googleusercontent.com/static/fonts/droidsans/v3/
        s-BiyweUPV0v-yRb-cjciBsxEYwM7FgeyaSgU71cLG0.woff)
        format('woff');
}
/* now can use this font */
body { font-family: "Droid Sans", "Arial", sans-serif; }
```

Notice that the `src` property specifies the URL of the font file, which in this case is a `WOFF` (web open font format) file hosted on Google's servers. If you wanted, you could host the file on your own server and then you would use a normal relative URL.

12 pt will mean that the font's bounding box (which in turn is roughly the size of its characters) will be 1/6 of an inch tall when printed, while making it 72 pt will make it roughly one inch tall when printed. However, as we saw in Section 3.2.3, absolute units such as points and inches do not translate very well to pixel-based devices. Somewhat surprisingly, pixels are also a problematic unit. Newer mobile devices in recent years have been increasing pixel densities so that a given CSS pixel does not correlate to a single device pixel.

So while sizing with pixels provides precise control, if we wish to create web layouts that work well on different devices, we should learn to use relative units such as em units or percentages for our font sizes (and indeed for other sizes in CSS




```
/* using 16px scale */

body { font-size: 100%; }
p { font-size: 1em; }       /* 1.0  x 16 = 16 */
h3 { font-size: 1.125em; }  /* 1.25 x 16 = 18 */
h2 { font-size: 1.5em; }    /* 1.5 x 16  = 24 */
h1 { font-size: 2em; }      /* 2 x 16 = 32 */
```

```
<body>
  Browser's default text size is usually 16 pixels
  <p>100% or 1em is 16 pixels</p>
  <h3>125% or 1.125em is 18 pixels</h3>
  <h2>150% or 1.5em is 24 pixels</h2>
  <h1>200% or 2em is 32 pixels</h1>
</body>
```

FIGURE 3.28 Using percents and em units for font sizes

as well). One of the principles of the web is that the user should be able to change the size of the text if he or she so wishes to do so; using percentages or em units ensures that this user action will "work," and not break the page layout.

When used to specify a font size, both em units and percentages are relative to the parent's font size. This takes some getting used to. Figure 3.28 illustrates a common set of percentages and their em equivalents to scale elements relative to the default 16-px font size.

While this looks pretty easy to master, things unfortunately can quickly become quite complicated. Remember that percents and em units are relative to their parents. Figure 3.29 illustrates how in reality it can quickly become difficult to calculate actual sizes when there are nested elements. As you can see in the second screen capture in Figure 3.29, changing the `<article>` element's size changes the size of the `<p>` and `<h1>` elements within it, thereby falsifying their claims to be 16 and 32 px in size!

For this reason, CSS3 now supports a new relative measure, the **rem** (for root em unit). This unit is always relative to the size of the root element (i.e., the `<html>` element). However, since early versions of Internet Explorer (prior to IE9) do not support the rem units, you need to provide some type of fallback for those browsers, as shown in Figure 3.30.

HANDS-ON EXERCISES

LAB 3 EXERCISE
CSS Paragraphs

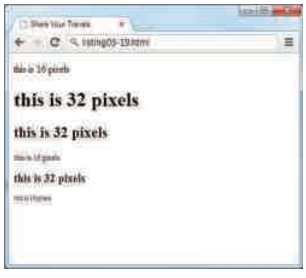
3.7.3 Paragraph Properties

Just as there are properties that affect the font in CSS, there are also a range of CSS properties that affect text independently of the font. Many of the most common text properties are shown in Table 3.10, and like the earlier font properties, many of these will be familiar to anyone who has used a word processor.

```
<body>
  <p>this is 16 pixels</p>
  <h1>this is 32 pixels</h1>
  <article>
     <h1>this is 32 pixels</h1>
     <p>this is 16 pixels</p>
     <div>
        <h1>this is 32 pixels</h1>
        <p>this is 16 pixels</p>
     </div>
  </article>
</body>
```


```
/* using 16px scale */

body { font-size: 100%; }
p    { font-size: 1em; }      /* 1 x 16 = 16px */
h1   { font-size: 2em; }      /* 2 x 16 = 32px */
```




```
/* using 16px scale */

body { font-size: 100%; }
p    { font-size: 1em; }
h1   { font-size: 2em; }

article { font-size: 75% }    /* h1 = 2 * 16 * 0.75 = 24px
                                 p = 1 * 16 * 0.75 = 12px */

div  { font-size: 75% }       /* h1 = 2 * 16 * 0.75 * 0.75 = 18px
                                 p = 1 * 16 * 0.75 * 0.75 = 9px */
```

FIGURE 3.29 Complications in calculating percents and em units


```
/* using 16px scale */

body { font-size: 100%; }
p {
      font-size: 16px;  /* for older browsers: won't scale properly though */
      font-size: 1rem;  /* for new browsers: scales and simple too */
}
h1   { font-size: 2em; }

article { font-size: 75% }  /* h1 = 2 * 16 * 0.75 = 24px
                               p = 1 * 16 = 16px */

div  { font-size: 75% }      /* h1 = 2 * 16 * 0.75 * 0.75 = 18px
                               p = 1 * 16 = 16px */
```

FIGURE 3.30 Using rem units

Property	Description
letter-spacing	Adjusts the space between letters. Can be the value `normal` or a length unit.
line-height	Specifies the space between baselines (equivalent to leading in a desktop publishing program). The default value is `normal`, but can be set to any length unit. Can also be set via the shorthand `font` property.
list-style-image	Specifies the URL of an image to use as the marker for unordered lists.
list-style-type	Selects the marker type to use for ordered and unordered lists. Often set to none to remove markers when the list is a navigational menu or a input form.
text-align	Aligns the text horizontally in a container element in a similar way as a word processor. Possible values are `left`, `right`, `center`, and justify.
text-decoration	Specifies whether the text will have lines below, through, or over it. Possible values are: none, `underline`, `overline`, `line-through`, and `blink`. Hyperlinks by default have this property set to `underline`.
text-direction	Specifies the direction of the text, left-to-right (`ltr`) or right-to-left (`rtl`).
text-indent	Indents the first line of a paragraph by a specific amount.
text-shadow	A new CSS3 property that can be used to add a drop shadow to a text. Not yet supported in IE9.
text-transform	Changes the capitalization of text. Possible values are none, `capitalize`, `lowercase`, and `uppercase`.
vertical-align	Aligns the text vertically in a container element. Most common values are: `top`, `bottom`, and `middle`.
word-spacing	Adjusts the space between words. Can be the value `normal` or a length unit.

TABLE 3.10 Text Properties

3.8 Chapter Summary

Cascading Style Sheets are a vital component of any modern website. This chapter provided a detailed overview of most of the major features of CSS. While we still

have yet to learn how to use CSS to create layout (which is relatively complicated and is the focus of Chapter 5), this chapter has covered most of the CSS that most web programmers will probably need to learn.

3.8.1 Key Terms

absolute units
attribute selector
author-created style sheets
browser style sheets
cascade
class selector
collapsing margins
combinators
contextual selector
CSS
CSS3 modules
declaration
declaration block
descendant selector
element box
element selectors
em units
embedded style sheets
external style sheets
generic font
grouped selector
id selector
inheritance
inline styles
internal styles
location
percentages
presentation
property:value pair
pseudo-class selector
pseudo-element selector
relative units
rem
responsive design
selector
specificity
style rules
TRouBLe
universal element selector
user style sheets
vendor prefixes
web font stack
x-height

3.8.2 Review Questions

1. What are the main benefits of using CSS?
2. Compare the approach the W3C has used with CSS3 in comparison to CSS2.1.
3. What are the different parts of a CSS style rule?
4. What is the difference between a relative and an absolute measure unit in CSS? Why are relative units preferred over absolute units in CSS?
5. What is an element selector and a grouped element selector? Provide an example of each.
6. What are class selectors? What are id selectors? Briefly discuss why you would use one over the other.
7. What are contextual selectors? Identify the four different contextual selectors.
8. What are pseudo-class selectors? What are they commonly used for?
9. What does cascade in CSS refer to?
10. What are the three cascade principles used by browsers when style rules conflict? Briefly describe each.
11. Illustrate the CSS box model. Be sure to label each of the components of the box.
12. What is a web font stack? Why are they necessary?

3.8.3 Hands-On Practice

PROJECT 1: Share Your Travel Photos, Time for Some Style

DIFFICULTY LEVEL: Beginner

HANDS-ON EXERCISES
PROJECT 3.1

Overview

This project updates your existing project from Chapter 2 to add some visual stylistic improvements with CSS.

Instructions

1. Use your chapter02-project01.html file from the last chapter as a starting point but save it as chapter03-project01.html.
2. Create an external style sheet called reset.css that removes all the browser formatting from the main HTML elements and reference inside chapter03-project01.html as follows:

```
html, body, header, footer, hgroup, nav, article, section, figure,
figcaption, h1, h2, h3, ul, li, body, div, p, img
{
    margin: 0;
    padding: 0;
    font-size: 100%;
    vertical-align: baseline;
    border: 0;
}
```

3. Create another external style sheet named chapter03-project01.css and include it in your HTML file as well.
4. Add styles to chapter03-project01.css so that it looks similar to that shown in Figure 3.31. Do not modify the markup within the <body> element.

Be sure to group your style rules together in appropriate commented sections and to make your sizes scalable (i.e., don't use pixels for font sizes, padding, or margins).

Here's a hint for the header and footer.

```
header, footer {
  color: white;
  background-color: #3D6271;
  margin: 0em 4em 0.5em 4em;
}
```

Testing

1. Although an exact match is not required, see how closely you can make your page look like the one in Figure 3.31. Be sure to test in multiple browsers and at different browser widths.

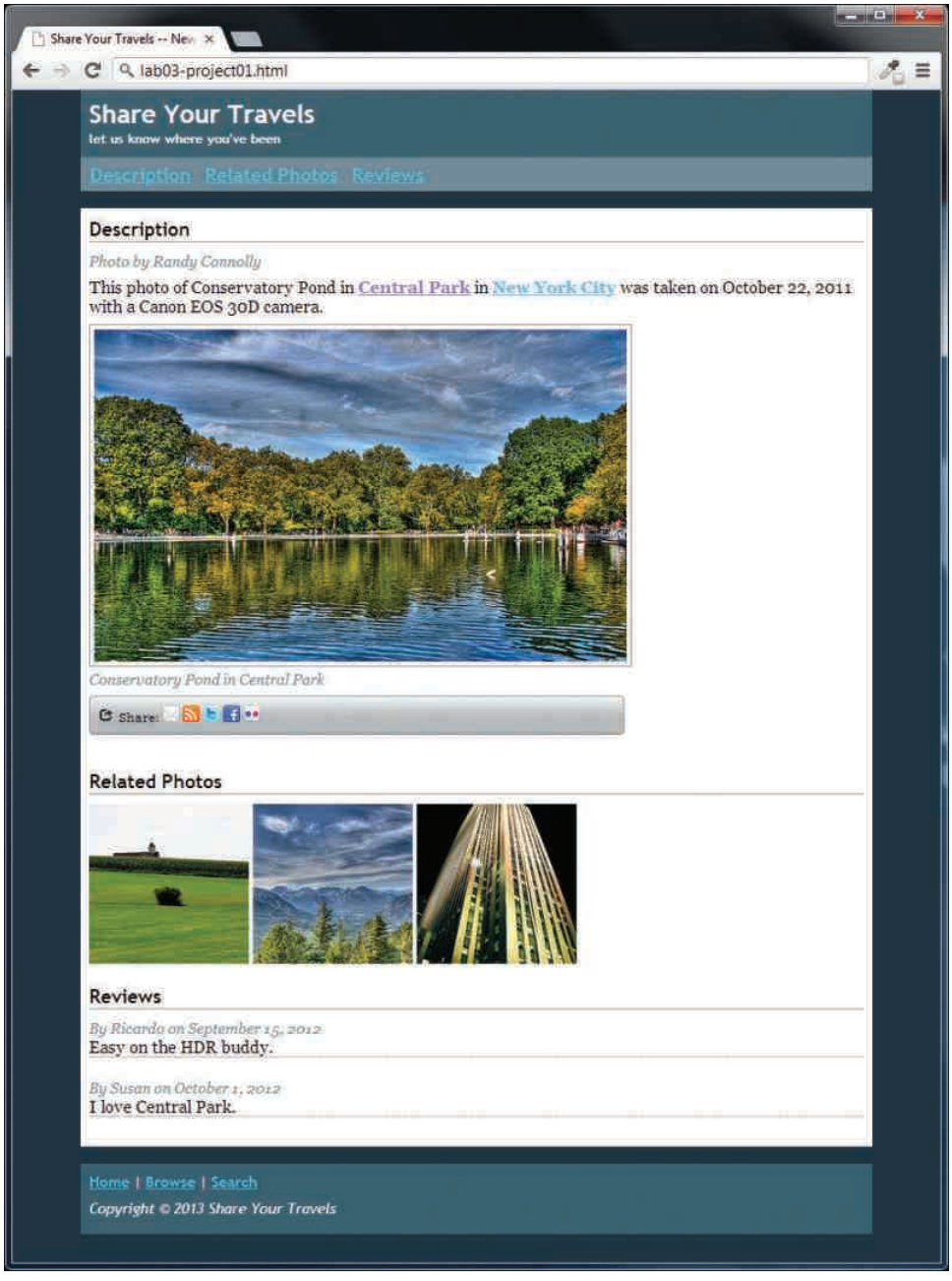


FIGURE 3.31 Completed Project 1

PROJECT 2: Book Rep Customer Relations Management

DIFFICULTY LEVEL: Intermediate

HANDS-ON EXERCISES
PROJECT 3.2

Overview

This project updates the CRM HTML page you started in Project 2.2 to add some visual style and make it look professional.

Instructions

1. Use your lab02-project02.html file from the last chapter as a starting point (and rename it) or take our chapter03-project01.html starting point file.
2. Import your existing reset.css from Project 1 to reset all default styles.
3. Create an external style sheet named chapter03-project02.css and import as well.
4. Add styles to chapter03-project02.css so that it looks similar to that shown in Figure 3.32. Do not modify the markup within the <body> element. This means defining styles for the header, footer, section, and other tags.

Hint: Notice the backgrounds for each of the section headers. Use attribute selectors for the mail and telephone link icons as shown below:

```
a[href^="mailto"] {
    background: url(images/email.png) no-repeat 0 3px;
    padding-left: 1em;
}
a[href^="tel"] {
    background: url(images/call.png) no-repeat 0 3px;
    padding-left: 1em;
}
```

Testing

1. Visually compare your output to that shown in Figure 3.32.

PROJECT 3: Art Store

DIFFICULTY LEVEL: Advanced

HANDS-ON EXERCISES
PROJECT 3.3

Overview

This project builds on the art store example from the previous chapter (Project 2.3), but purposefully leaves you having to dig a little deeper into CSS.

Instructions

1. Create a new file named chapter03-project03.html and remove all default styles via a reset.css stylesheet, as done for the previous two projects.
2. Modify chapter03-project03.html and an associated style sheet so that your output looks similar to that shown in Figure 3.31. Do not modify the markup within the <body> element.

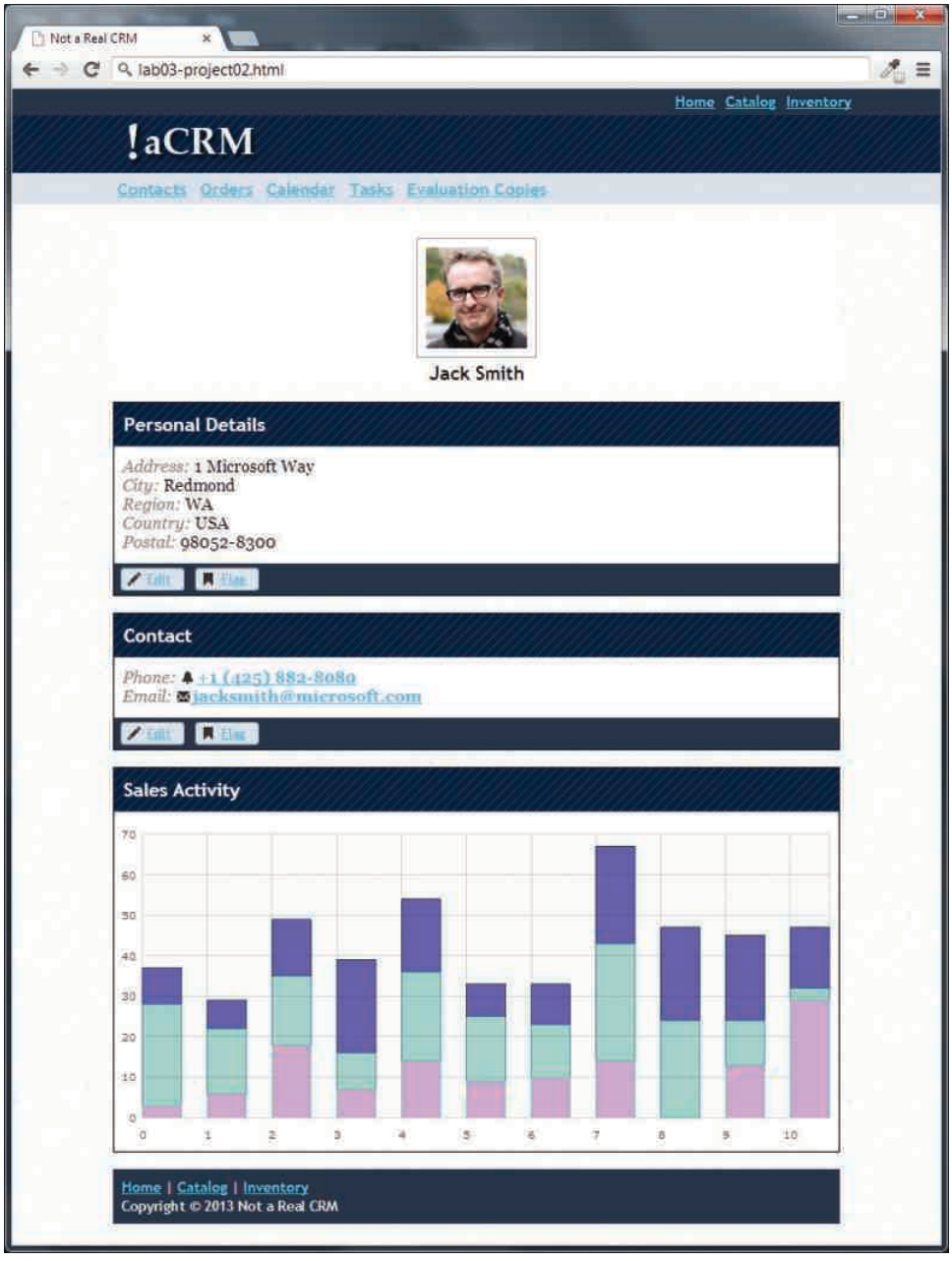


FIGURE 3.32 Completed Project 2

3. You will have to use a CSS3 feature that will require some research on your own. The `background-size` property can be used to force a background image to resize to the browser window.
4. Notice that two of the blocks in Figure 3.33 are partially transparent. Remember that CSS3 allows you to specify the alpha transparency of any color.
5. Finally, the header uses the font `Six Caps`, which will have to be supplemented with other options in the font stack in the event that font is not present on the client's computer.

Testing

1. First, try resizing your browser to ensure the image resizes dynamically to fill the space, and the floating objects position themselves correctly.
2. Try out different browsers or platforms to see if it really works on all types of devices, including your mobile phone.

Hint: This is tricky if you have not yet set up a web server. You may have to return to finish this particular testing step until after you have access to a web server as described in Chapter 8.

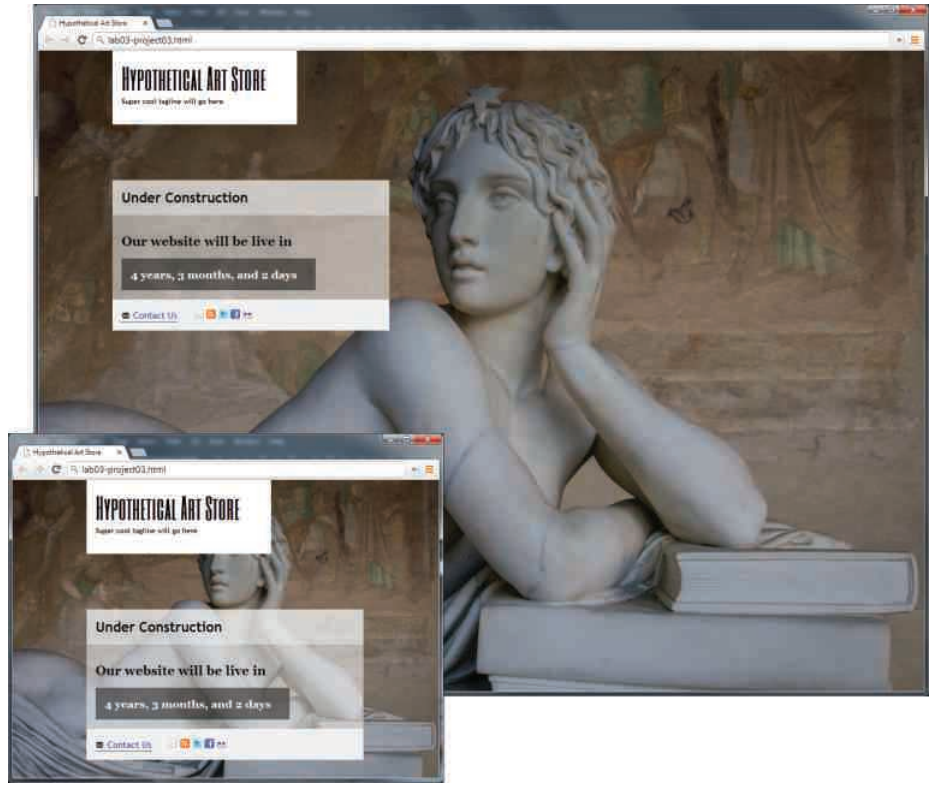


FIGURE 3.33 Completed Project 3

3.8.4 References

1. J. Teague, CSS3: Visual Quickstart Guide, Peachpit, 2012.
2. D. Cederholm and E. Marcotte, Handcrafted CSS, New Riders, 2009.
3. E. A. Meyer, CSS Web Site Design, Peachpit, 2003.
4. W3C, Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. [Online]. http://www.w3.org/TR/CSS2/.
5. T. Olsson and P. O'Brien, CSS Reference. [Online]. http://reference.sitepoint.com/css.
6. V. Friedman, "CSS Specificity: Things You Should Know." [Online]. http://coding.smashingmagazine.com/2007/07/27/css-specificity-things-you-should-know/.