HTML5

TOPICS INCLUDE:

- Create semantic HTML documents
- Build interactive forms and validate data
- Embed native audio and video
- Apply advanced CSS techniques
- Incorporate eye-catching fonts
- Save user data in local storage

ADVANCED

HTML5:
Advanced
Student Manual

HTML5: Advanced

CEO, Axzo Press:	Ken Wasnock
Vice President, Content and Delivery:	Josh Pincus
Director of Publishing Systems Development:	Dan Quackenbush
Writer:	Brandon Heffernan
Copyeditor:	Catherine Oliver
Keytester:	Cliff Coryea

Student Manual
ISBN-10: 1-4260-2994-2
ISBN-13: 978-1-4260-2994-3

Printed in the United States of America
1 2 3 4 5 GL 06 05 04 03

Contents

Introduction

After reading this introduction, you will know how to:

A Use ILT Series manuals in general.

B Use prerequisites, a target student description, course objectives, and a skills inventory to properly set your expectations for the course.

C Re-key this course after class.

Topic A: About the manual

ILT Series philosophy

Our manuals facilitate your learning by providing structured interaction with the software itself. While we provide text to explain difficult concepts, the hands-on activities are the focus of our courses. By paying close attention as your instructor leads you through these activities, you will learn the skills and concepts effectively.

We believe strongly in the instructor-led class. During class, focus on your instructor. Our manuals are designed and written to facilitate your interaction with your instructor, and not to call attention to manuals themselves.

We believe in the basic approach of setting expectations, delivering instruction, and providing summary and review afterwards. For this reason, lessons begin with objectives and end with summaries. We also provide overall course objectives and a course summary to provide both an introduction to and closure on the entire course.

Manual components

The manuals contain these major components:

- Table of contents
- Introduction
- Units
- Course summary
- Glossary
- Index

Each element is described below.

Table of contents

The table of contents acts as a learning roadmap.

Introduction

The introduction contains information about our training philosophy and our manual components, features, and conventions. It contains target student, prerequisite, objective, and setup information for the specific course.

Units

Units are the largest structural component of the course content. A unit begins with a title page that lists objectives for each major subdivision, or topic, within the unit. Within each topic, conceptual and explanatory information alternates with hands-on activities. Units conclude with a summary comprising one paragraph for each topic, and an independent practice activity that gives you an opportunity to practice the skills you've learned.

The conceptual information takes the form of text paragraphs, exhibits, lists, and tables. The activities are structured in two columns, one telling you what to do, the other providing explanations, descriptions, and graphics.

Course summary

This section provides a text summary of the entire course. It is useful for providing closure at the end of the course. The course summary also indicates the next course in this series, if there is one, and lists additional resources you might find useful as you continue to learn about the software.

Glossary

The glossary provides definitions for all of the key terms used in this course.

Index

The index at the end of this manual makes it easy for you to find information about a particular software component, feature, or concept.

Manual conventions

We've tried to keep the number of elements and the types of formatting to a minimum in the manuals. This aids in clarity and makes the manuals more classically elegant looking. But there are some conventions and icons you should know about.

Item	Description
Italic text	In conceptual text, indicates a new term or feature.
Bold text	In unit summaries, indicates a key term or concept. In an independent practice activity, indicates an explicit item that you select, choose, or type.
`Code font`	Indicates code or syntax.

`Longer strings of ►`
`  code will look ►`
`  like this.`
	In the hands-on activities, any code that's too long to fit on a single line is divided into segments by one or more continuation characters (►). This code should be entered as a continuous string of text.

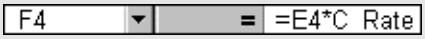
Select **bold item**	In the left column of hands-on activities, bold sans-serif text indicates an explicit item that you select, choose, or type.
Keycaps like ↵ ENTER	Indicate a key on the keyboard you must press.

Hands-on activities

The hands-on activities are the most important parts of our manuals. They are divided into two primary columns. The "Here's how" column gives short instructions to you about what to do. The "Here's why" column provides explanations, graphics, and clarifications. Here's a sample:

Do it!

A-1: Creating a commission formula

Here's how	Here's why
1 Open Sales	This is an oversimplified sales compensation worksheet. It shows sales totals, commissions, and incentives for five sales reps.
2 Observe the contents of cell F4	 The commission rate formulas use the name "C_Rate" instead of a value for the commission rate.

For these activities, we have provided a collection of data files designed to help you learn each skill in a real-world business context. As you work through the activities, you will modify and update these files. Of course, you might make a mistake and therefore want to re-key the activity starting from scratch. To make it easy to start over, you will rename each data file at the end of the first activity in which the file is modified. Our convention for renaming files is to add the word "My" to the beginning of the file name. In the above activity, for example, a file called "Sales" is being used for the first time. At the end of this activity, you would save the file as "My sales," thus leaving the "Sales" file unchanged. If you make a mistake, you can start over using the original "Sales" file.

In some activities, however, it might not be practical to rename the data file. If you want to retry one of these activities, ask your instructor for a fresh copy of the original data file.

Topic B: Setting your expectations

Properly setting your expectations is essential to your success. This topic will help you do that by providing:

- Prerequisites for this course
- A description of the target student
- A list of the objectives for the course
- A skills assessment for the course

Course prerequisites

Before taking this course, you should have knowledge of and experience with several HTML5 elements and attributes, as well as some experience with style sheets (CSS). Furthermore, this course assumes that you've completed the following course or have equivalent experience:

- *HTML5: Basic*

Target student

You will get the most out of this course if have some experience with HTML, and with HTML5 in particular. You should understand basic HTML structure and design techniques, and you should know how to use CSS to format HTML markup.

Course objectives

These overall course objectives will give you an idea about what to expect from the course. It is also possible that they will help you see that this course is not the right one for you. If you think you either lack the prerequisite knowledge or already know most of the subject matter to be covered, you should let your instructor know that you think you are misplaced in the class.

After completing this course, you will know how to:

- Plan a Web site project, control a browser's rendering mode, set character encoding and document language, comment your code effectively, ensure basic HTML5 support in old browsers, take advantage of semantic structural elements, and create page titles and descriptions that are optimized for search engine results pages (SERPs) and for other Web sites.
- Create interactive forms, set input field properties, define option groups, create data lists, enable spelling checks, create required fields, validate user data by using regular expressions, create and format fieldsets and legends, and identify best practices in form design.
- Embed native audio and video that works in all HTML5-supporting browsers, enable user controls, set audio and video properties, create a poster image, and embed videos hosted externally.
- Format elements based on their location in the document structure to avoid having to introduce new HTML, incorporate special fonts by using the Google Web Fonts API, apply shadow effects, and insert and format generated content.
- Create editable content, implement a script that saves user data in local storage, format an editable region, and rotate an element.

Skills inventory

Use the following form to gauge your skill level entering the class. For each skill listed, rate your familiarity from 1 to 5, with five being the most familiar. *This is not a test*. Rather, it is intended to provide you with an idea of where you're starting from at the beginning of class. If you're wholly unfamiliar with all the skills, you might not be ready for the class. If you think you already understand all of the skills, you might need to move on to the next course in the series. In either case, you should let your instructor know as soon as possible.

Skill	1	2	3	4	5
Planning a Web site project					
Creating a basic HTML5 document					
Adding comments to markup and styles					
Ensuring basic HTML5 support in outdated browsers					
Using document outlines for planning and development					
Creating a semantically meaningful document structure					
Optimizing page titles and descriptions for SERPs					
Creating a form and applying labels					
Creating email fields, checkboxes, and radio buttons					
Setting input field size and limiting data length					
Creating a password field					
Controlling autofocus and autocomplete					
Creating a text area field and placeholder text					
Creating a select list					
Defining option groups for a select list					
Creating a data list and checking spelling					
Creating required input fields					
Requiring numeric data with limits					
Validating input for a specific date format					
Creating fieldsets and legends					
Formatting fieldsets and legends					
Identifying best practices in form design					

Applying native audio and enabling user controls					
Setting audio attributes					
Applying native video and setting properties					
Controlling data preload and creating a poster image					
Embedding a video hosted on YouTube					
Applying the <iframe> element					
Applying styles based on parent-child structure					
Applying styles based on location and type					
Alternating table row colors					
Writing efficient style sheet code and limiting redundancies					
Applying Google Web Fonts to your site					
Applying shadow effects					
Generating content from a style sheet					
Applying images as CSS content					
Creating an editable content section					
Saving user data in local storage					
Rotating an element					
Using vendor prefixes in style rules					
Conducting basic site testing					

Topic C: Re-keying the course

If you have the proper hardware and software, you can re-key this course after class. This section explains what you'll need in order to do so, and how to do it.

Hardware requirements

Your personal computer should have:

- A keyboard and a mouse
- A 500 MHz processor (or faster)
- At least 256 MB RAM
- At least 1.5 GB of available hard disk space
- An SVGA monitor (1024×768 or higher resolution)
- Working sound output and speakers for the "Audio and Video" unit.

Software requirements

You will need the following software:

- Windows 7 or Windows XP, updated with the most recent service packs

  Note: This course was written for Windows 7. Throughout the course, the simple text editor Notepad is used for development. This program comes with Microsoft Windows systems, so no download or setup is required. For other systems, any simple text editor will work.
- The latest version of Google Chrome (version 20 or later)
- The latest version of Firefox (version 13 or later)

  Note: To successfully key this course and get the most out of its content, you need to use Chrome and Firefox as your browsers. Both are used throughout the course to observe differences in appearance and support and to learn how to manage those differences.

Network requirements

The following network components and connectivity are also required for re-keying this course:

- Internet access, for the following purposes:
  - Downloading the latest critical updates and service packs
  - Completing Activity B-3 in the unit titled "Audio and video"
  - Completing Activities B-1 and B-2 in the unit titled "CSS techniques"

Setup instructions to re-key the course

Before you re-key the course, you will need to perform the following steps.

1. Use Windows Update to install all available critical updates and service packs.
2. With flat-panel displays, we recommend using the panel's native resolution for best results. Color depth/quality should be set to High (24 bit) or higher.

   Please note that your display settings or resolution may differ from the author's, so your screens might not exactly match the screenshots in this manual.
3. Configure Windows Explorer to display file extensions as follows:
   a. Open Windows Explorer.
   b. Choose Organize, Folder and Search Options.
   c. On the View tab, clear "Hide extensions for known file types," and click OK.
4. If you have the data disc that came with this manual, locate the Student Data folder on it and copy it to the desktop of your computer.

   If you don't have the data disc, you can download the Student Data files for the course:
   a. Connect to http://downloads.logicaloperations.com.
   b. Enter the course title or search by part to locate this course.
   c. Click the course title to display a list of available downloads.
      Note: Data Files are located under the Instructor Edition of the course.
   d. Click the link(s) for downloading the Student Data files.
   e. Create a folder named Student Data on the desktop of your computer.
   f. Double-click the downloaded zip file(s) and drag the contents into the Student Data folder.

Unit 1

Authoring guidelines

Complete this unit, and you'll know how to:

A Plan a Web site effectively.

B Discuss HTML5 authoring options, control a browser's rendering mode, set character encoding and the document language, comment your code effectively, and ensure basic HTML5 support in old browsers.

C Create document outlines that make optimal use of the semantic structural elements of HTML5.

D Create page titles and descriptions that are optimized for search engine results pages (SERPs) and for other Web sites.

Topic A: Planning a Web site

Explanation

When you start a new Web site, it's important to first plan the site project carefully before you start the design and development work.

Basic elements of a site project plan

When you plan a Web site, you need to consider more than your visual design approach. Other factors, including usability, structure, navigation, and consistent rendering in multiple browsers, are equally important.

Another critical component of any Web development project is an analysis of the site's purpose and audience. Before you begin development, consider these factors:

- **Objective** — Set well-defined objectives for your Web site. An objective should be specific, measurable, and realistic.
- **Content** — The content should clearly convey the purpose of the site, and it should be relevant to the intended users. The language, tone, graphics, and level of detail should be based on an analysis of the site's objectives and target audience.
- **Audience** — Analyze the interests, age, experiences, background, and expectations of your audience. Many of your design and content choices will likely be based on this analysis.
- **Site structure** — Determine the site's content and navigation structure at the beginning of the design process. Making wholesale changes in these structures later on can have a ripple effect that's time-consuming to deal with.

Analyzing the site's purpose and audience

Spend some time defining the audience for the site and the goals you want to accomplish with the site. Ask yourself or your development team the following general questions as part of a comprehensive (and ongoing) analysis:

- What do we (or what does the client) want to achieve with the site?
- How will the site achieve the established goals?
- Whom are we trying to reach, and what are the audience demographics—the education level, age range, gender, or special interests of the target audience?
- What does the intended audience already know about the information that will be presented on the site?
- What values or experiences might members of this audience have in common?
- What are the client's goals?
- In what ways will the audience be transformed or edified by the information presented on the Web site?
- How will we keep visitors interested and engaged over the long term?
- What are the specific needs of the site's users, and how will we tailor the design and content to those needs?
- Which browsers are predominantly used by the target audience?
- What Internet access speed is typical, on average, for the target audience?

Design considerations

You can attract and retain users by designing pages that make it easy for them to find the information they're looking for. You can achieve this by creating an intuitive content and navigation structure, and by using graphics, color, and typography thoughtfully. Before you begin actual design and content work, think about how best to structure your content and how you should present your information.

When planning a site's structure and design, keep the following factors in mind:

- **Navigation** — The navigation scheme should reflect the site's structure, and it should be consistent on every page.
- **Fonts** — Choose fonts that provide optimal readability and that suit the target audience. For example, you might use standard fonts designed for the screen, such as Verdana or Georgia for a corporate site, but a more playful font, such as Comic Sans MS, for a site targeted to kids.
- **Page length** — Break information into manageable chunks. A page that contains an excessive amount of text that requires a lot of vertical scrolling can be tedious to many users, who tend to prefer to consume information in small pieces.
- **Color and contrast** — Choose your color scheme carefully. In addition to helping to establish the look and feel of your site, color can draw the user's eye to specific content and convey emphasis. The color of your text is especially important. It must be easy to read against its background color, with sufficient contrast to make the text clear and crisp.
- **Load time** — Nothing ruins an online experience more than pages that are slow to load. If you keep visitors waiting, they're likely to go elsewhere. Use media files and other potentially "heavy" components conservatively, and don't place too many "heavy" components on any one page.
- **Typography** — In addition to choosing fonts thoughtfully, adjust your font styles, such as line height and letter spacing, to improve readability. Be careful not to over-style text in articles or other body text.

Keep your most important information "above the fold"

Above the fold is a term and technique from the newspaper industry—information that is considered to be the most important is presented on the front page, above the fold, so that the big headline is quickly visible, even at a glance. You can apply the same concept to a Web page by placing the most important information near the top of the page so that a user does not have to scroll down to view that information.

Do it!

A-1: Discussing basic planning and design considerations

Questions and answers

1 What are some important factors to consider when planning a Web site?

2 What content factors are important to consider when planning a site?

3 What questions can you ask yourself or your project team to help ensure that content and design choices stay relevant to the site's overall objectives?

4 Why is it important to choose background colors and text colors that provide sufficient contrast?

5 Besides the importance of foreground and background contrast, name two important factors to consider when choosing colors.

6 Why is it important to break text into logical, manageable chunks?

7 Name two important factors to consider when formatting text.

Topic B: HTML5 authoring

Explanation

One of the benefits of HTML5 is that it simplifies some of the fundamental code requirements of Web authoring. For example, it's easier to remember and apply the right document type and content encoding, and to create valid markup, than it was when XHTML was the de facto markup standard on the Web. To get started with HTML5 development, you need a solid understanding of these foundational aspects, of what has changed and why, and of authoring options and techniques.

What's HTML5?

HTML5 is the latest iteration of HTML (Hypertext Markup Language), the markup standard for documents on the Web. The term "HTML5" has come to imply a lot more than just the elements and attributes included in the HTML5 specification, though. HTML5 has become a loose reference to many Web technologies working in concert for a faster, more functional platform for Web-based applications.

In a practical sense, the term "HTML5" has become similar to "dynamic HTML," or DHTML, a buzzword from the late '90s which referred to using HTML, CSS, and scripting to produce interactive functionality in Web sites. But strictly speaking, HTML5 is just the markup behind Web documents, and it includes many new elements and attributes that enable developers to create semantically meaningful documents and rich Web applications.

Browser developers have adopted HTML5 faster and more consistently than they did with past Web standards, in part because of the actions of Google and Apple, two influential companies that were early proponents of the concepts and technologies behind HTML5.

HTML5 applications

Many modern Web sites and applications that are considered HTML5 sites are a combination of HTML, CSS3, JavaScript, Scalable Vector Graphics (SVG), and other technologies, like geolocation and local storage. All these technologies work with HTML to produce the types of environments that many people have come to associate with HTML5.

These corresponding technologies are beyond the scope of this course for the most part. However, in the real world, the developers who work with HTML are typically the same people who apply CSS to format the HTML. Therefore, in the context of this training, it makes practical sense to include some commonly used CSS techniques that are relevant to HTML5 elements and attributes.

Flexible authoring requirements

Unlike XHTML, HTML5 is not strict in terms of syntax and authoring requirements. For example, you can write tags and attributes in uppercase or lowercase and omit quotation marks around attribute values. However, if you're already in the habit of following XHTML requirements, as many developers are, you don't have to change anything. It's still valid HTML5, whether or not you use lowercase, wrap attribute values in quotation marks, and self-close empty tags.

Some of the old XHTML guidelines remain useful, while others have become less important. For example, if you like to write uppercase tags and/or uppercase attributes, or omit quotation marks around attributes, none of today's browsers will penalize you for it.

However, it remains important to properly nest the elements in your document outline and to close your containing elements. There are also some important benefits of establishing and adhering to a set of authoring standards. These benefits include the following:

- **It will be easier to collaborate with other developers.** It's important that you create documents that can be easily read, understood, and maintained, both for your own benefit and for other developers who will need to read or modify the documents at some point. If multiple people use different authoring habits, making modifications can become difficult, time consuming, and error-prone.
- **Your CSS styles will work more reliably and consistently.** Such habits as properly nesting your elements and closing all container elements will help a lot when you apply CSS styles. When HTML gets messy and inconsistent, you can end up with styles that aren't working as you expect, and it can be difficult and time consuming to locate the offending code. So, you can save yourself time and hassle by at least properly nesting the elements in your document outline and closing all container tags.
- **You can be more confident that your content will be displayed consistently in a variety of browsers and devices.** Today's browsers do an excellent job of handling errors and making sense of imperfect markup. But some bad habits can lead to formatting problems and functionality issues when scripts interact with elements. Again, not all browsers will render pages consistently if duplicate closing tags are present, if containing elements are not closed, or if elements are improperly nested, so it makes a lot of practical sense to do at least those things.

So, if you're already in the habit of adhering to some of the authoring requirements of XHTML, there's no need to change anything. And if you're not in the habit, there's no need to start now—go ahead and take advantage of the looseness of HTML5. But even then, create a set of authoring standards for yourself and/or your organization and stay consistent with it. Down the road, you'll be glad you did.

The HTML5 doctype

The *doctype* (document type) declaration should be the first thing that appears in an HTML document; it tells the browser what type of document it's rendering. In previous versions of HTML (and XHTML), the doctype declaration was long and difficult to remember, resulting in many mistakes and therefore inaccurately identifying document types. The HTML5 doctype declaration is simplified and easy to remember. To define a document as HTML5, add the following code to the top of your page:

```
<!DOCTYPE html>
```

Writing DOCTYPE in all caps used to be a requirement, but it's not in HTML5, although many Web authors are still in the habit of doing so. Either way is fine. This declaration not only tells the browser that the document contains HTML5 markup, but also affects how the browser displays the page.

Standards mode and quirks mode

If your document begins with the HTML5 doctype declaration, modern browsers will render the document in standards mode (also called "no-quirks mode"). This means that the page will be displayed as it should be displayed, with proper implementations of CSS and the Document Object Model (DOM).

If you don't have any doctype declaration, modern browsers will render the page in quirks mode, so it might render certain aspects of a page the way older browsers used to. Typically, quirks mode is used only to ensure backward-compatibility in legacy sites that were designed when browsers implemented some things wrong. There's a third rendering mode, called "almost standards mode" or "limited quirks mode," which specifically accommodates pages that were designed when table formatting was not implemented as the CSS specification had intended.

To simplify all this, here's what you need to know:

- For all new development, start your documents with the HTML5 doctype declaration to trigger standards mode and take full advantage of modern browsers.
- For legacy documents that aren't displayed as originally designed when they're viewed in standards mode, consider omitting a doctype declaration to trigger quirks mode. Another option is to use the HTML 4.01 Loose doctype declaration or the XHTML 1.0 Transitional doctype, both of which trigger "limited quirks mode."

It's also important to note that using a doctype declaration to trigger a specific rendering mode does not guarantee that all your pages, old and new, will be displayed consistently in all browsers. Before you deploy a new site, you should always test it in all of the most popular Internet browsing clients in use today.

Character encoding: boring but important

A file's *character encoding* determines the alphanumeric characters that a browser can display. There are different character encodings out there for various languages, like Chinese, Japanese, Russian, and English. Some characters are common across many encodings, while others are unique to a particular language. The only one that really matters for our purposes is UTF-8 because it works for any language and includes all the dashes, quotation marks, and other characters that are commonly and uncommonly used.

If you've ever seen a Web page with strange characters in the text, like black diamonds, a string of question marks, or other gobbledygook, it's probably due to a lack of proper character encoding. Strange characters often appear when content is pasted in from applications like Microsoft Word.

Every Web document you create should include a character encoding statement. It's needed not only to prevent nonsense characters from being displayed but also to prevent a type of security vulnerability called cross-site scripting (XSS).

Servers are typically configured to automatically send the right encoding in the HTTP headers. If that doesn't happen—due to oversight, bad policy, or migration issues, or because you open a local copy of a page (which is not delivered from a server)—then you'll have a potential security hole if the encoding is not manually specified.

Another danger exists with forms. If browsers aren't told which encoding to use, either by the server or by the document itself, and users attempt to submit characters outside of the browser's default encoding, a variety of problems can occur, and they're all bad.

So, play it safe and always specify the UTF-8 character encoding in every document or template in your site. All you need to do is add the following code to the top of the `<head>` section of each document:

```
<meta charset="utf-8" />
```

Setting the document language

To specify a document's primary language, you use the `lang` attribute in the `<html>` element. In the following example, the `lang` attribute declares that the document's primary language is English.

```
<html lang="en">
```

Other commonly used language abbreviations include `fr` for French, `de` for German, and `es` for Spanish. Specifying the language of a page or section can help search engines determine relevance, and it can help alternative devices like screen readers to properly identify changes in a document's primary language.

For example, if you have a document whose primary language is English, and you use a French term within a paragraph, you would write:

```
<p>Thanks for your order. <i lang="fr">Bon appétit!</i></p>
```

In this example, the change in the document language is limited to the scope of the `<i>` element. A screen reader would then revert to `lang="en"` at the point where the `<i>` element is closed. The `<i>` element is chosen here to create italics without implying emphasis.

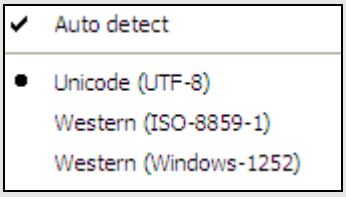
Do it!

B-1: Setting up a basic HTML5 document

The files for this activity are in Student Data folder **Unit 1\Topic B**.

Here's how	Here's why
1 In Chrome, open titles.html	(In the current topic folder.) You can right-click the file and choose Open with, Google Chrome.
2 Observe the simple layout	This document does not contain an HTML5 doctype declaration, so it's being rendered in quirks mode. This is not recommended; HTML5 documents should contain the HTML5 doctype.
3 In Notepad, open titles.html	(In the current topic folder, right-click the file and choose Open with, Notepad.) You can also use an equivalent simple text editor such as TextEdit on the Macintosh.
4 At the very top of the document, type: `<!DOCTYPE html>`	To identify the document as an HTML5 document and to trigger standards mode.
5 Save your changes and refresh the page in Chrome	Notice the subtle change in the font size in the table. In older browsers, CSS styles were not properly inherited in tables. The HTML5 doctype declaration tells the browser to render the page as it should be displayed.
6 In the upper-right corner of the browser, click the wrench icon	To open the browser's main menu.
Point to **Tools**	To open a submenu.
Point to **Encoding**	To display the Encoding options. There are many encoding options, and when a document or its issuing server does not specify an encoding, the browser is left to guess which one to use. Chrome guesses that the encoding is Western (ISO-8859-1).
7 Switch to the code	
At the top of the `<head>` section, type: `<meta charset="utf-8" />`	To specify the character encoding for the document. A closing slash for empty elements is optional in HTML5.
8 Save your changes and refresh the page in Chrome	

9 Open the Encoding menu again



(Click the wrench icon and then point to Tools, Encoding.) The browser now identifies the page as being UTF-8 encoded.

10 Add the following bold code to the `<html>` tag:

```
<html lang="en">
```

To define the document's primary language as English. Setting this attribute can help search engines determine relevance, and help devices like screen readers identify changes in the natural language.

11 Save your changes and refresh the page in Chrome

Setting the document language doesn't have a visible effect on the page, but it's an important fundamental step in the development of your site pages and templates.

12 True or false? For HTML5 code to be considered valid, it must be authored with the same coding requirements as XHTML.

13 Why is UTF-8 usually the best character encoding to use on the Web?

14 Why is it important to establish and adhere to a set of authoring standards?

15 Describe two reasons that it's important to include the HTML5 doctype declaration at the top of your new Web documents and templates.

16 Describe two reasons that it's important to specify the proper character encoding in all of your site pages and templates.

17 Describe two reasons that using the `lang` attribute is important.

Using comments

Explanation

It's important to add comments to your HTML markup and style sheets so that you and other developers can easily read, understand, and modify your documents and templates.

HTML comment tags

HTML (and XHTML) comment tags have a specific required syntax. They start with an exclamation point, followed by double dashes, and they end with double dashes and the greater-than sign, as follows:

```
<!-- THIS IS A COMMENT -->
```

Browsers do not display comments unless they are written incorrectly. They are not meant to be displayed, but they serve important purposes. Comments make your code easier to read, and they provide information about the various sections of a document or template. For example:

```
<!-- PRIMARY NAVIGATION SECTION -->
```

Your comment text doesn't have to be written in all caps as shown in these examples, but the capital letters can help make comments stand out from the surrounding markup. Anything that helps you to easily locate your comments will in turn help you to locate and identify various sections of your documents.

Additional use cases, guidelines, and benefits

It can be difficult to locate specific sections of code that you need to modify, especially in long documents that contain several `<div>` sections that look alike. Writing clear comments that identify or describe each section of a page or template will help you and others to read and understand your pages or templates when it's time to modify them. This information is especially important when you're working in development teams.

Another common use of comments is to hide certain blocks of code that you want to store but not display. For example, if you have a section of markup that you want to use intermittently on a page or template, for such things as periodic messaging, you can surround that section of HTML markup in a comment tag to temporarily disable it.

Comment guidelines

To write and use comments most effectively:

- Write your comments as you write code, rather than trying to add them later. You'll likely write comments that are most clear and effective when the purpose of each section or series of markup tags is fresh in your mind. If you put off documenting your code, you might never get around to it, or your documentation might not be as thorough or accurate as it could have been.
- Use comments to record the code author's name and the date of the last modifications. This information can be particularly useful when you're working in teams.
- When working in teams, develop commenting conventions for all developers to follow. This consistency will pay off down the road when updates need to be made.
- Update your comments when you update your code. Outdated comments aren't helpful and can actually cause confusion and incorrect conclusions to be drawn about the purpose of certain code blocks.

Commenting your style sheets

As with HTML, commenting your style sheet rules is an important habit to develop—comments make it easier for you and other developers to read and understand the code and to edit it quickly. Use CSS comments to document how particular style rules are acting on the document and to clarify rules that would not be easily understood at first read.

CSS comments have a different syntax than HTML comments. CSS comments consist of forward slashes and asterisks, with the comment text in between, as follows:

```
/* This is your comment text */
```

Anything inside these comment delimiters will have no effect on a page. Use them to provide author information, the last revision date, or information about a particular rule or a group of rules. For example:

```
/* Nudge the main navigation box down 4 pixels */
```

Like HTML comments, CSS comments can span multiple lines. For example:

```
/*
Author: John Clements
Last Revised on: 12/23/12

*/
```

All the same recommendations and guidelines for writing effective HTML comments apply to writing CSS comments.

Do it!

B-2: Adding comments to markup and styles

Here's how	Here's why
1 Switch to the code	
Directly above the `<siteNav>` element, type:	To document the purpose of the code block that follows it.
`<!-- SITE NAVIGATION BAR -->`	
2 Save your changes and refresh the page in Chrome	No change is evident because comments are not displayed in a browser.
3 Directly above the `<thead>` element, type:	This page is a very simple example, but in larger, more complex pages, this kind of documentation is critical, especially when you're working in teams.
`<!-- TABLE HEADERS -->`	
4 Directly above the `<tfoot>` element, type:	Comments don't have to be written in all caps, but that convention can help make comments easy to find amidst the code.
`<!-- TABLE FOOTER -->`	
5 Above the `<tbody>` element, add a **TABLE BODY** comment	

6 From the styles folder, open styles.css

You'll add comments to the style sheet.

7 To the right of the body rule's closing brace, write:

To add a comment here.

```
/* Brown background color */
```

It's often helpful to include comments next to color codes so you and other developers don't have to guess which colors they represent.

8 For the h2 rule, add a comment that reads **Light brown**

As styles get more complex in fully developed sites, this type of documentation becomes critical for easy maintenance and for personnel transitions.

9 Save your changes

10 Why is it important to comment your markup and style sheet code?

11 Why is it important to write your comments as you code rather than some time later?

12 Why is it critical to update your comments when you update your code?

Ensuring HTML5 support in Internet Explorer

Explanation

Most Internet users today are browsing the Web with the latest version of Chrome, Firefox, Safari, or Internet Explorer. However, some organizations still use older operating systems which cannot run modern versions of Internet Explorer. To reach the widest audience possible, you need to ensure that the HTML5 methods you employ aren't preventing some users from accessing your content the way you intend.

The HTML5 enabling script

The current version of Microsoft's Internet Explorer browser does a good job of rendering HTML5 elements, but version 8 (and anything earlier) does not. However, achieving backward compatibility is fairly easy. All you need to do is include the "HTML5 shim," which is also called the "HTML5 shiv" or, most accurately, the "HTML5 enabling script."

Linking to the HTML5 enabling script

The HTML5 shim is a JavaScript file hosted on a public Google server and available for anyone to use in their site. This process involves a `<script>` element that points the browser to the JavaScript file. The `<script>` element is surrounded by a special conditional comment which is read only by Internet Explorer. Here's what the "shim" looks like:

```
<!--[if lt IE 9]>
<script
src="http://html5shim.googlecode.com/svn/trunk/html5.js">
</script>
<![endif]-->
```

This code means, "If the browser is Internet Explorer and earlier than version 9, then run this script." No other browsers will link to the script, so it's an efficient solution. Put simply, the script makes it possible for Internet Explorer 8 and earlier to parse and display HTML5 elements.

All you need to do is copy this conditional code block and paste it into the `<head>` section of all of your HTML5 documents and templates. It's critical that this code block is in the `<head>` section because Internet Explorer needs to run the script before it begins to parse the HTML5 document.

Thorough site testing remains important

Even with the HTML5 shim, it's still critical that you test your site in multiple browsers and platforms before you deploy it, especially in Safari, Chrome, Firefox, and Internet Explorer, on both the Windows and Apple platforms. In time, tools like the HTML5 shim probably won't be necessary, as the use of outdated browsers around the world will continue to decrease.

Do it!

B-3: Ensuring basic HTML5 support in outdated browsers

The files for this activity are in Student Data folder **Unit 1\Topic B**.

Here's how	Here's why
1 Open HTML5 shim.txt	From the current topic folder.
Press CTRL + A	To select all the code.
Press CTRL + C	To copy the code.
2 In your text editor, switch to titles.html	
3 Paste the code block in the `<head>` section, after the title	(Press Ctrl+V.) Paste the code after the title element and before the link to the style sheet.
4 Save your changes	
5 Reload the page in your browser	There is no visible change because you're using a modern browser, which ignores the link to the script because it's wrapped in a conditional comment that only Internet Explorer can read. For Internet Explorer 8 and earlier, the script makes it possible for the browser to parse and display HTML5 elements.
6 Why is it critical for the HTML5 shim to be in the `<head>` section?	
7 Close all open files	

Topic C: Content organization and structure

Explanation

Organizing your site content into an outline is an important part of site planning and the early stages of development. It allows you to view your content in a hierarchical form, which reveals your document structure. From there, it's easier to decide which elements to use for each component of your document structure. And with so many new elements available in HTML5, it makes practical sense to map out your content structure to take full advantage of HTML5.

The document outline

When you're starting a new Web site project, it can be helpful to use brainstorming sessions with a team, or to create storyboards and site mockups, to provide a framework for the scope, design, content, features, and functionality decisions that you'll need to make. It's also helpful to think of your site in terms of a traditional outline, like the simple example in Exhibit 1-1.

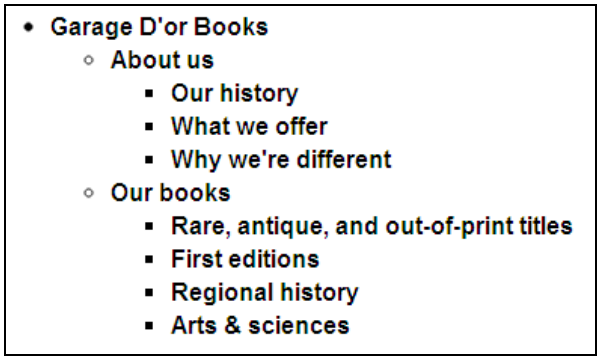


Exhibit 1-1: A simple example of a document outline

The document outline is part of the HTML5 specification

In addition to helping in the site planning and content organization phase, mapping your content into an outline is important because modern browsers use an outline model to parse HTML documents. Browsers use a document's sections, headings, and other components to build an outline to optimally interpret and display the content. This outline is not visible to the user but exists behind the scenes.

Why it's important to develop with a document outline in mind

Now more than ever, successful site development goes beyond what users see. You also need to think in terms of what other applications will see and how those applications will interact with your content, both now and in the future.

If your pages are poorly organized, with markup that's not well formed and that does not follow a logical outline, it might still *look* okay to a person. But other applications and devices probably will not be able to get the most out of your site. Other developers will also find it difficult to read, understand, and modify your site. For example, such a site will be only marginally useful to users with assistive devices like screen readers, and search engines might not be able to interpret your site's content.

On the other hand, if you do construct your pages as logical document outlines with semantic elements to help form a meaningful content flow, then:

- Your content will likely be optimally accessible to a wide range of devices.
- Search engines and other Web crawlers might be able to index your site more effectively.
- You can more easily syndicate your content to other sites.
- You and your development team will be able to read, understand, and update your site pages or templates more easily.

Do it!

C-1: Discussing document outlines

Questions and answers

1 What is a document outline?

2 Why is it important to build documents with the outline model in mind?

3 What are some benefits of creating Web documents with a logical structure and semantic HTML5 elements?

4 What site planning methods have you found to be effective, and which mockup or wireframe tools have been helpful to you?

Match your content with the right elements

Explanation

Semantically meaningful elements are those elements that are self-descriptive; they describe the content they contain. For example, a `<section>` element defines a section in a document, and the `<nav>` element defines a region that contains navigation links.

You need to decide which elements best describe your content. Sometimes the choice is obvious, and sometimes it's not. In some cases there's no one answer, and you could make an argument for one of several choices.

It often helps to ask questions about the purpose of a particular section of HTML markup. The following table lists some examples of the kinds of questions you can ask yourself or your development team as you work on creating HTML5 document outlines that are optimized for your content.

Question	If yes, then...
Does the content serve as an introduction for other content?	Use the `<header>` element and/or an appropriate heading tag, `<h1>` through `<h6>`.
Are there adjacent, related headings?	Consider wrapping them in an `<hgroup>` element to group them logically and share styles.
Does the content represent a distinct section of the document?	Use the `<section>` element.
Does the content contain site navigation links?	Use the `<nav>` element.
Does the content contain author information, dates, or legal or copyright information?	Use the `<footer>` element, which can be used for individual sections in addition to the footer for an entire document.
Will the content be used or distributed independently of the other content, like a blog entry or syndicated article?	Use the `<article>` element.
Is the content peripherally related to its adjacent content? Does it provide ancillary information for a main article or content region?	Use the `<aside>` element.
Does the content not fit any of the semantic HTML5 elements, and/or is it being added merely for layout or style purposes?	Use the `<div>` element.

Benefits of using the semantic elements of HTML5

There are many advantages to using HTML5's structural elements in your document outline. First, using them can make the development process easier and more consistent. With new elements that logically describe your content, you can avoid creating documents overflowing with `<div>` tags. Such documents can be difficult to read, understand, and modify. Other benefits include the following:

- Because you have more elements at your disposal, you can reduce the number of classes and IDs in your documents, and thereby simplify your documents and your style sheets.
- Your document structure will be easier to read and understand at a glance.
- Your documents will be easier to maintain and update.
- Code that's efficient and well formed can allow other applications to interact with your page content more effectively.

Create your own authoring standards for your content categories

You'll likely find it beneficial to create official element standards and guidelines for your organization so that everyone on your development team uses the same process and ultimately produces consistent markup that's efficient, semantically meaningful, and easy for everyone involved to understand and update.

Use site mockups

Many developers also find it helpful to create site mockups in design programs such as Adobe Fireworks or Illustrator, or one of several tools specific to creating site mockups and wireframes. After you have created a mockup, you can identify which HTML5 elements are best suited for each section and then create a basic structural skeleton for your site markup.

There are many ways you can do this; what's most important is that it's done. Plan your site, create a general outline, and use that outline as the framework for an efficient, semantically meaningful document structure.

Do it!

C-2: Structuring a document with semantic elements

The files for this activity are in Student Data folder **Unit 1\Topic C**.

Here's how	Here's why
1 In Chrome, open index.html	From the current topic folder.
2 In your text editor, open index.html	In the current topic folder, right-click the file and choose Open with, Notepad.
Scroll through and observe the HTML structure	You'll modify this markup so that it has a more semantically meaningful structure using HTML5 elements.
3 What are your observations about the overall document structure?	

4 Scroll down to the bottom of the code	There are so many `<div>` sections that it becomes difficult to identify which closing tag is associated with which opening tag. The result is a "div soup" that's difficult to read and modify, but this is typical of how many documents on the Web have been structured.
5 Locate the SITE NAVIGATION BAR comment	(Scroll up.) Below this comment, there's a div with the ID "siteNav," which contains the site's internal navigation links.
Which element would be best suited to replace `<div>` here?	
6 Locate the "content" div	
Which element would be best suited to replace `<div>` here?	
7 Locate the adjacent headings	
Which element could you use to group these related headings?	
8 Locate the "inStock" div	
Which element would be best suited to replace `<div>` here?	
9 Locate the two `<div>` containers that hold the two articles	
Which element would be best suited to replace the two `<div>` elements here?	
10 Locate the "pageFooter" div	
Which element would be best suited to replace `<div>` here?	
11 Scroll up to locate the "page" div	
What makes the `<div>` element a good choice in this case?	

12 Update the markup for the navigation bar	(Change `<div>` to `<nav>` for both the opening and closing tags.) Don't remove the siteNav ID; this is used to apply styles and add more structural context. (The `<nav>` element could be used elsewhere on the page for different link collections that aren't part of the main navigation scheme.)
13 Update the markup for the "content" section	(Use the `<section>` element.) Again, don't remove its ID. The closing tag for this is directly above the "pageFooter" element.
14 Group the two adjacent headings, `<h1>` and `<h2>`	Use the `<hgroup>` element.
15 Update the markup for the "inStock" element	Use the `<aside>` element.
16 Update the markup for the two articles Why is it possible to have separate `<h1>` elements within these sections?	Use the `<article>` element for both.
17 Update the markup for the "pageFooter" element Why is it NOT redundant to write `<footer id="pageFooter">` ?	Use the `<footer>` element. Because the `<footer>` element is not intended solely to define page footers; it can be used elsewhere in the document to define footers for individual sections. So, using an ID or class such as "pageFooter" or "sectionFooter" adds more structural and semantic value.
18 Take a look at your changes in the document structure	
19 How has the document structure been improved?	
20 Save your changes and reload the page in Chrome	The appearance of the page should not change. If there are layout oddities, they might be caused by missing closing tags. Switch to the code and check to make sure that all your opening and closing elements match.
21 Close all open files	

Topic D: Basic SEO

Explanation

Search engine optimization (SEO) is a broad topic that's beyond the scope of this course. However, there are some basic things you can do and some guidelines you can follow that can help you optimize your content snippets on *search engine results pages* (SERPS). This optimization is an important step in achieving high click-through rates.

Metadata content

In the HTML5 content model, content is organized into categories such as metadata, sectioning, heading, phrasing, and embedded content. Metadata content is responsible for the presentation or behavior of the document, or establishes relationships with other resources, via such elements as `<script>`, `<link>`, `<style>`, and `<meta>`. Metadata content belongs in the `<head>` section of your documents. Two metadata content elements that play an important role in SEO are page titles and the meta description element.

Page titles

Page titles might seem like the most basic of HTML elements, and perhaps they are, but their importance is often overlooked. Many Web authors just write their company name into every document title, while others make the mistake of omitting title text altogether. The page title is one of the most important pieces of your metadata content. By creating unique page titles for every document, you can establish relevance in search engine results and therefore bring more users to your site.

To create a page title, you use the `<title>` element inside the `<head>` section of a document. For example:

```
<title>Garage D'or Books: Our History</title>
```

Easy to remember, but the important aspect of page titles is how you write them.

Guidelines for optimizing your page titles

Titles are the first piece of information that users see on search engine results pages (SERPs), and they serve as the links to your pages or resources. So, it's critical that every one of your site pages contains a title and that each title is meaningful. Here are some important guidelines to help you write effective page titles:

- **Write titles that concisely describe the page content.** Usually, page titles are best when they describe the page content in as few words as possible. Search engines will display only a segment of a long title. You can always use the meta description element to provide important details.
- **Use title casing.** Studies have shown that title case results in better click-through rates than lowercase and even sentence case. Capitalize the first and last words in your titles, plus any nouns, pronouns, verbs, adverbs, and adjectives in between.
- **Check to make sure all the pages in your site have unique titles.** If several pages share the same title, it will be more difficult for search engines and other Web crawlers to distinguish each page.
- **Avoid writing page titles that are too generic.** Titles such as "Products" or "Services" are vague and don't give people or Web crawlers sufficient context.

- **Incorporate branding into your page titles.** Perhaps the easiest way to brand your page titles is to start each title with your company or organization's name, followed by a concise description of the page content. An example is "Outlander Spices: User Registration." This convention also provides a simple way to maintain consistency within the site while keeping each page title unique.
- **Don't insert a list of search keywords into your page titles.** Google and most other search engines don't use the `keywords` meta tag in their page rank algorithms, but that doesn't mean that page titles are a substitute location for such content. Search engines will likely penalize you in terms of page rank if you use page titles as a place to stuff keywords.

Page descriptions

Page descriptions (or "meta descriptions") work hand-in-hand with your page titles. Too often, Web authors copy the home page's default description and use it on every page in the site. This is a missed opportunity to establish search relevance for each individual page in your site and thereby achieve better click-through rates.

Page descriptions are not included in ranking algorithms, so they don't affect your page rank in Google and other search engines. However, they're critical because page descriptions are displayed as snippets on search engine results pages (SERPs), and therefore they can *influence* users when they see your description content.

Think of these snippets as opportunities to sell your content. People don't typically click the first item in a SERP, then the second, and so on; they scan their SERPs for the title and snippet that represent the best content and context match for their search criteria. For example, Exhibit 1-2 shows an actual Google search result. The search term that produced this result was **CEH Certification value**.

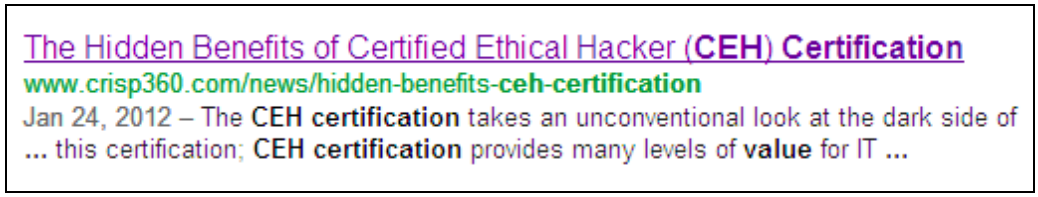


Exhibit 1-2: This Google search result shows how titles and descriptions are used

Here, you can see how important the document title and description are. The title serves as the main text about and link to the resource, and the description is pulled in as the first snippet under the title. Now imagine how much less useful this search engine result would be if the title were only the name of the organization.

As for the snippet that Google generates, notice that it uses an ellipsis to separate sources. The description for this page begins with this sentence: "The CEH certification takes an unconventional look at the dark side of computer network security." Google shows a part of that to provide relevance to the user, but then switches to another source—the document content itself—to provide further relevance to the search criteria. Google also makes the keywords that match the search criteria bold so the user can quickly scan the snippet and evaluate its relevance.

Page descriptions aren't used only by search engines; many popular sites and applications use them, too. Two examples among many are the professional networking site LinkedIn and the social network Facebook, both of which display page descriptions under titles when you share links with your network.

To create a page description, you use the `<meta>` tag along with two attributes, as follows:

```
<meta name="description"
 content="This is where your page description goes." />
```

The `name` attribute specifies the type of meta element (there are several others), and the `content` attribute contains the page description text.

Guidelines for optimizing your page descriptions

As with page titles, good page descriptions are an important part of search engine optimization (SEO) and can result in higher click-through rates. Here are some important guidelines to help you write effective page descriptions:

- **Try to keep your page descriptions to 155 characters or fewer.** It might not always be possible, but providing the detail you need in as few words as possible is an important goal. Search engines and other sites that display page descriptions will cut them off at around 155 characters or so, which might result in an awkward cut-off.
- **Use proper grammar and punctuation.** Page descriptions are highly visible, and they represent you and your organization, so it pays to look professional. That starts with the correct use of language and punctuation. Poorly written descriptions can look like spam and reflect poorly on your organization, leading readers to expect a similarly low level of quality at your site.
- **Have people first in mind, and Web crawlers second.** If you try to write descriptions that you think are optimized for search engines, you're less likely to write descriptions that are useful for people. For example, if your page descriptions are filled with keywords that do not have a natural grammatical flow, you can be penalized in terms of page rank. Describe your page content for human readers, and remember that page descriptions are not included in search engine algorithms.
- **Avoid repeating the same description on multiple pages.** Each page in your site should have a unique description. If you copy the same language into all your page descriptions, you're missing an opportunity to provide both users and machines with the details and context that can translate into increased traffic to your site.
- **Emphasize whatever distinguishes your organization or content.** Remember, SERP snippets are an opportunity to sell your content. It's important to emphasize the real benefits that your organization or content provides and whatever sets you apart from others. For instance, you might mention affiliations with well-known brands (to enhance your credibility), a special sale notice, free shipping, or some other value proposition.
- **When applicable, summarize the most important data.** In some cases, it makes more sense to provide a summary of the most important data in a page rather than to write a description in sentence or paragraph format. For example, a page description for a Ceylon cinnamon product page might look like something like this:

```
<meta name="description" content="Imported from: Sri
Lanka, Price: $23.99, Weight: 16 oz. (finely ground)" />
```

- **When applicable, include some kind of call to action.** There's nothing wrong with using some marketing copy in your page descriptions. You have a brief opportunity to persuade readers to click through to your page; a specific call-to-action statement can help make that happen. For example, write "Learn how you can…" or "Discover how we can save you 50%…." Use action verbs such as *read*, *improve*, *learn*, and *discover*—sometimes giving a little direction can make a big difference.
- **Make sure your descriptions are relevant to your page content.** One of the worst things you can do is write descriptions that are intended to entice people but have no real connection to the actual content on your pages. Such attempts at manipulation will have the opposite effect; if people find that your site contains content that is not as you advertised, you can significantly damage your site's reputation.
- **Avoid generic marketing clichés and unsubstantiated claims.** Words and phrases like "number one," "world class," and "greatest" are empty clichés that can't be quantified or substantiated. These words are a turn-off for most people because they suggest insincerity, exaggeration, and a lack of credibility, and show that very little thought and effort were put forth.

Do it!

D-1: Optimizing page titles and descriptions for SERPs

The files for this activity are in Student Data folder **Unit 1\Topic D**.

Here's how	Here's why
1 In Chrome, open index.html In your text editor, open index.html	From the current topic folder.
2 Locate the current page title	The current page title is "Home."
3 Why is this NOT an optimal title?	
4 What factors make a good page title?	

5 Why is it so important to write effective page titles?

6 What would make a better title for this page?

7 Change the page title to your better alternative

8 What factors make for a good, effective page description?

9 What page description would be useful for this page?

10 Why is it so important to write effective page descriptions?

11 Where in the HTML code must the page description be placed?

12 Type the following in the proper location in the document:

```
<meta name="description" content="" />
```

13 Between the quotation marks, type your description

14 How effective do you think your page description will be on search engine results pages and other sites that display descriptions when you share a link to the page?

15 Save your changes and reload the page in Chrome

There are no visible changes other than the page title in the title bar. Meta descriptions are not displayed in the browser.

Close all open files

Unit summary: Authoring guidelines

Topic A In this topic, you learned some basic guidelines you can follow to effectively **plan a Web site** development project.

Topic B In this topic, you learned about HTML5 applications, trends, and **authoring options**. You learned how to use the HTML5 doctype to control the browser's **rendering mode**, and you learned how to specify the **character encoding** of your documents. Finally, you learned how to set the document language, how to **comment** your code effectively, and how to ensure basic HTML5 support in outdated browsers.

Topic C In this topic, you learned why organizing your site content into an **outline** is an important part of site planning. You learned about the benefits of using **semantic elements** to mark up your content, and you applied these elements to a page structure. You also learned why it's important to establish **authoring standards** for you and/or your development team.

Topic D In this topic, you learned some fundamental and important **SEO concepts**. You learned that **page titles and descriptions** are two critical aspects of effective SEO because together they provide users with a snippet that serves as your opportunity to attract users to your content. You learned several guidelines for creating successful page titles and descriptions.

Review questions

1 Name some design factors that are important to analyze and plan before you get started on a Web development project.

2 What happens when you insert the HTML5 doctype declaration at the top of your document code?

   A Modern browsers will render the page using quirks mode.

   B The page will render correctly in outdated browsers.

   C The page will render exactly the same in old browsers and modern browsers.

   D Modern browsers will render the page using standards mode.

3 Which of the following is a valid HTML comment?

   A `<-- LAST MODIFIED ON 12/23/12 -->`

   B `/* Last modified on 12/23/12 */`

   C `<!- LAST MODIFIED ON 12/23/12 ->`

   D `<!-- Last modified on 12/23/12 -->`

4 Which of the following is a valid CSS comment?

A `// LAST MODIFIED ON 12/23/12 //`

B `/* Last modified on 12/23/12 */`

C `*/ LAST MODIFIED ON 12/23/12 /*`

D `<!-- Last modified on 12/23/12 -->`

5 True or false? If you use the HTML5 shim, you don't have to spend time testing your site on multiple browsers.

6 True or false? A document can contain multiple `<footer>` elements.

7 True or false? The `<nav>` element is intended only for a site's main navigation section.

8 True or false? Using semantic elements can help reduce the need for some classes and IDs, thereby simplifying your documents and reducing the amount of code required.

9 True or false? Metadata content belongs in the `<head>` section of your documents.

10 True or false? In page descriptions, it's not important to be grammatically correct.

11 True or false? In HTML5 documents, the `<div>` element doesn't have much use.

12 True or false? HTML5 does not require that attribute values be contained in quotation marks.

Unit 2

HTML5 forms

Complete this unit, and you'll know how to:

A Create forms, add a variety of input fields, create labels, set input field properties, enhance a form's usability, define option groups, create a data list, and enable spelling check.

B Create required fields, and validate user data by using the pattern attribute and regular expressions.

C Create and format fieldsets and legends.

D Identify best practices in form design.

Topic A: Interactive forms

Explanation

Forms allow users to interact with your Web site in a variety of ways. For example, you can ask for performance feedback, create a threaded discussion, or create a product order form. If you want to collect information from users, you need to create a form.

Introduction to forms

Forms consist of a variety of input fields designed to obtain certain types of information from users. For example, Exhibit 2-1 shows a simple registration form.

The data that users enter into these fields is typically submitted to and processed by a script on a Web server, which then sends the information to a database for storage. The script then returns information to the user by confirming the submission or displaying other relevant information about the completed transaction.

Exhibit 2-1: A simple form

The <form> element

To create a form, you start with the `<form>` element, which contains all the form input fields. The `<form>` element has one required attribute, `action`, which you use to specify the location of the script that processes the form data. For example:

```
<form action="scripts/form_processor.php">

...your form input fields here...

</form>
```

The data that users enter into the form's input fields is submitted to the processor you specify with the `action` attribute. So, the value of the `action` attribute is the path and file name for your specific form processor.

Form processing

Every input field must have a name, and the data that a user enters provides the value. Together, these pieces of information create *name/value pairs*. When a form is submitted, all the name/value pairs are sent to the form processor through one of two methods, *post* or *get*. You can use the `method` attribute to specify which of these methods to use. If no `method` attribute is present, the *get* method is assumed. This method sends the name/value pairs by appending them to a URL.

The *get* method is typically preferred when you want to retrieve (get) information. For example, search forms often use the *get* method. However, this method should not be used when sensitive data such as passwords or credit card numbers will be processed. In addition, some browsers limit the number of characters allowed in a URL.

The *post* method sends form data by using an HTTP post transaction. This method is more secure, and there's no limit to the number of name/value pairs you can create.

There are several server-side scripting options for processing Web forms; options include PHP, Active Server Pages (ASP), Java Server Pages (JSP), and many other languages.

The <input> element

After you have created a form container, you can insert input fields. You can use the `<input>` element to create a variety of input fields. This is an empty element, not a container, so it does not have a closing tag. If you choose to follow the authoring guidelines of XHTML, however, you need to add a terminating slash at the end of the tag.

The `type` attribute determines the data type to be used for a given `<input>` element. For example, to create a text input field for a user's first name, you could write:

```
<input type="text" name="FirstName" />
```

The `name` attribute establishes the name/value pair that will be sent to the processing script. You can use any name you want, but the name should reflect the data you're collecting. For example, logical variations of the previous example would include `name="first"` or `name="Fname"`.

Submit and reset buttons

Forms require a submit button so that user data can be sent to the form processor. To create a submit button, you use the `type="submit"` attribute in the `<input>` element. By default, most browsers display the text "Submit" on a submit button, but you can change this by using the `value` attribute. For example, to create a button that reads "Send" instead of the default button text, you would write:

```
<input type="submit" value="Send" />
```

If you want to give users a chance to start over and clear all data entered in your form fields, you can include a reset button by using the `type="reset"` attribute. Browsers display the text "Reset" by default, but again you can change that by using the `value` attribute. For example, to create a reset button that reads "Clear," you would write:

```
<input type="reset" value="Clear" />
```

The <label> element

The <label> element is an inline element that you use to create the text prompts for your form's input fields. You should label all input fields to tell users what kind of information they should enter into each field.

To associate a label with a particular input field, you can place the <input> element inside the <label> element, as follows:

```
<label>First Name:
<input type="text" name="FirstName" />
</label>
```

Or you can use the for attribute to explicitly associate a label with an input field by matching the label's value to the input's id value, as follows:

```
<label for="FirstName">First Name:</label>
<input type="text" id="FirstName" name="FirstName" />
```

Again, the value of the for attribute must match the ID of its associated input field, even if the values of input's name attribute and ID are the same, as shown in the previous example. The name attribute's value is used to create the name/value pair that's sent to the form processor, and it does not need to match the ID, although it can if you prefer to maintain that consistency.

If a user enters "John" in this input field and submits the form, the form processor will receive the name/value pair FirstName="John".

Labels improve form usability

When you use the <label> element to define a text prompt for each input field, either by placing the <input> element and its text prompt within a <label> element or by explicitly associating an input field and its label by using matching for and id attributes, you create a larger selection area for each input field. Users can then change the focus from one input field to another by clicking either the input field itself or its text label.

Do it!

A-1: Creating a form and applying labels

The files for this activity are in Student Data folder **Unit 2\Topic A**.

Here's how	Here's why
1 In Chrome, open membership.html	From the current topic folder, in the current unit folder.
Open membership.html in Notepad	(In the current topic folder, right-click the file and choose Open with, Notepad.)
2 Scroll down to locate the MEMBERSHIP FORM comment	
3 Under this comment, add the following code:	To start a form container and add one text input field.

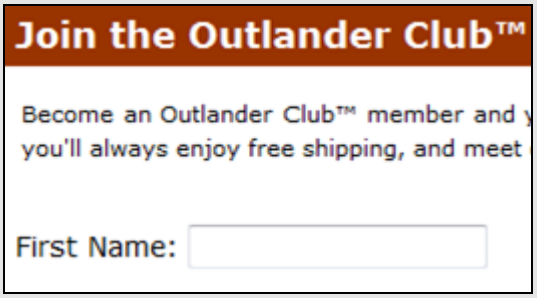
```
<form action="processor.php" method="get">

 <label for="first">First Name:</label>
 <input type="text" id="first" name="FirstName" />

</form>
```

Notice that the `id` and `name` values do not need to match, but the `for` attribute and the `id` value must match.

4 Save your changes and reload the page in Chrome

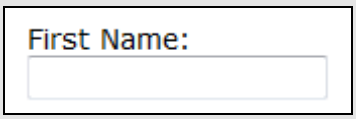


The label (First Name) and the input field are displayed on the same line because both are inline elements by default.

5 From the styles folder, open globalstyles.css	(In the current topic folder.) The file opens in Notepad. You'll make labels block elements.
Scroll to the bottom of the style sheet	
6 Under the FORM STYLES comment, add the following rule:	To make labels block elements and give them a 10-pixel top margin.

```
label { display: block; margin-top: 10px; }
```

7 Save the style sheet and reload the page in Chrome



The label now appears on its own line.

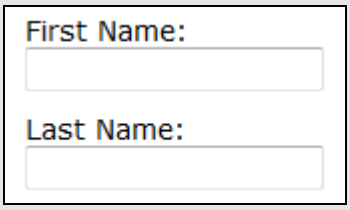
Close the style sheet

8 After the First Name input, add the following code:

```
<label for="last">Last Name:</label>
<input type="text" id="last" name="LastName" />
```

The value of the `for` attribute must match the input field's `id` value. The input field's `name` and `id` values can be the same, but this is not required.

9 Save your changes and reload the page in Chrome



The labels and input fields are arranged vertically because `<label>` elements are now block elements, and a new line is created before and after each block element.

10 After the Last Name input, add the following code:

```
<label for="zip">ZIP code:</label>
<input type="text" id="zip" name="ZIP" />
```

11 After the ZIP code input, add the following code:

```
<input type="submit" />
```

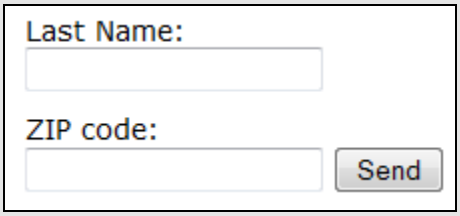
12 Save your changes and reload the page in Chrome

By default, the submit button reads "Submit." In Firefox, the default text is "Submit Query." You'll change this default text.

13 Add the following bold code to the submit input:

For submit and reset input fields, the `value` attribute creates the text label.

```
<input type="submit" value="Send" />
```

14 Verify the result	 (Save your changes and reload the page in Chrome.) The text on the button has changed.
15 Click anywhere on the text **First Name**	The insertion point appears in the First Name input field because the text label and the input field are associated by the `<label>` element. This association improves form usability by expanding the clickable region for an input field.
Click anywhere on the text **Last Name**	To move the insertion point to the Last Name input field. If the fields had no labels, clicking the text prompts would have no effect.
16 Type your first and last names in the appropriate fields	
Click **Send**	You receive a "This webpage is not found" error because the form is not linked to an actual processing script. Normally, with a fully functioning form, a new page should be displayed, verifying the form submission or taking the user to follow-up information.
17 Maximize the browser window	If necessary.
In the Address bar, observe the end of the URL	(If necessary, drag to the right in the Address bar to see the end of the URL.) This form's method is `get`, which appends the form's name/value pairs to the URL.
Click the browser's Back button	To go back to the form page.
18 In the navigation bar, click **Register**	To reload the current page and clear the form variables from the address.

Other input types

Explanation

There are many other input types in addition to the `text` and `submit` input types that you've already applied. For example, you can create checkboxes and radio buttons, or create input fields that collect e-mail addresses or passwords. The following table describes some other input fields you can create by using the `<input>` element.

Input type	Description
`<input type="password" />`	Creates a field in which users can enter passwords. Characters typed into a password field are displayed as asterisks or dots to ensure privacy.
`<input type="checkbox" />`	Creates a checkbox. If a user selects a checkbox, a checkmark appears in the box to indicate the selection. Clicking the box again clears the checkmark.
`<input type="radio" />`	Creates a radio button. Radio buttons are typically used in a group of two or more when only one option can be selected. For example, radio buttons are often used for answering Yes or No, which are mutually exclusive.
`<input type="email" />`	Creates a text input field specific to e-mail addresses.
`<input type="search" />`	Creates a text input field for collecting search terms.
`<input type="number" />`	Creates an input field for collecting numerical values.
`<input type="tel" />`	Creates an input field for collecting phone numbers.

Unsupported input types fail gracefully

The `email`, `search`, `number`, and `tel` input types are new in HTML5. (There are other new input types that are not supported at the time of this writing.) Fortunately, the `text` input type is the default input type, so any browser that encounters an HTML5 input type but doesn't support it will default to the standard `text` input type. Therefore, it's safe to use these input types to take advantage of the features they provide in modern browsers.

E-mail input fields

E-mail input fields look just like text input fields but are intended specifically to collect e-mail addresses. This input type, which is new in HTML5, helps to simplify the task of validating the user's input. It also improves the usability of your form when it's viewed on mobile devices, like the iPhone, that have touchscreens instead of physical keyboards. On these devices, when you select an e-mail input field, a digital keyboard will appear that's automatically optimized for entering an e-mail address.

For example, compare the two images in Exhibit 2-2. On the left is a site that does not use the `email` input type, while the example on the right does. Notice the difference in the keyboards that are displayed. The keyboard on the right is automatically optimized for entering an e-mail address; it shows the @ symbol and the period, and the alternate number keyboard changes to include only those characters that are allowed in an e-mail address.

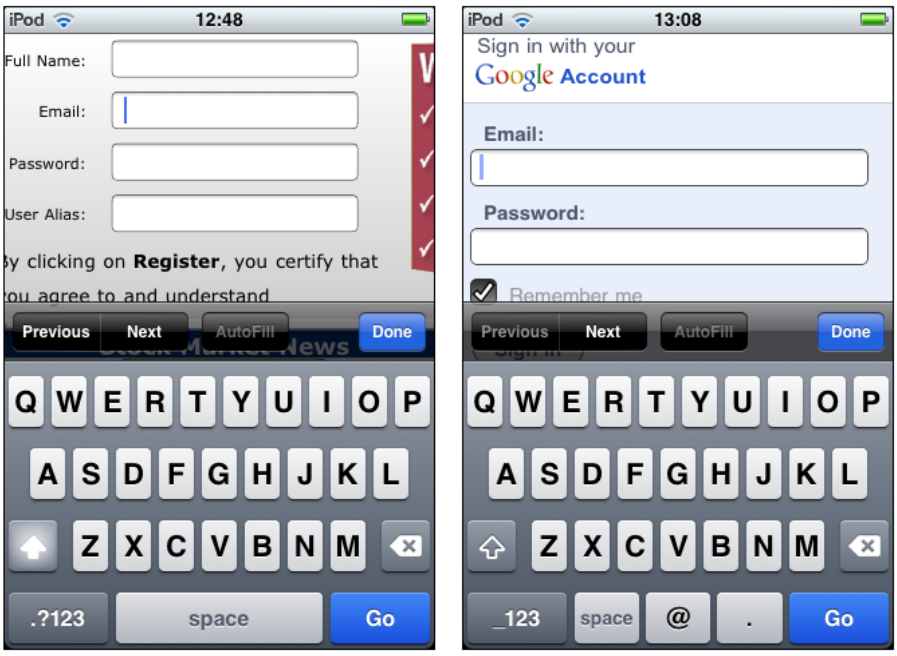


Exhibit 2-2: Using the email input type optimizes the keyboard on some devices

Telephone input fields

When you use the `tel` input type, users of iPhones, iPads, and many other mobile devices will automatically be presented with a number pad when they click the input field. Exhibit 2-3 shows a telephone input field that does not use the `tel` input type (left) and one that does (right). The mobile audience is important—anything you can do to make a user's experience easier, faster, and more intuitive, you probably *should* do.

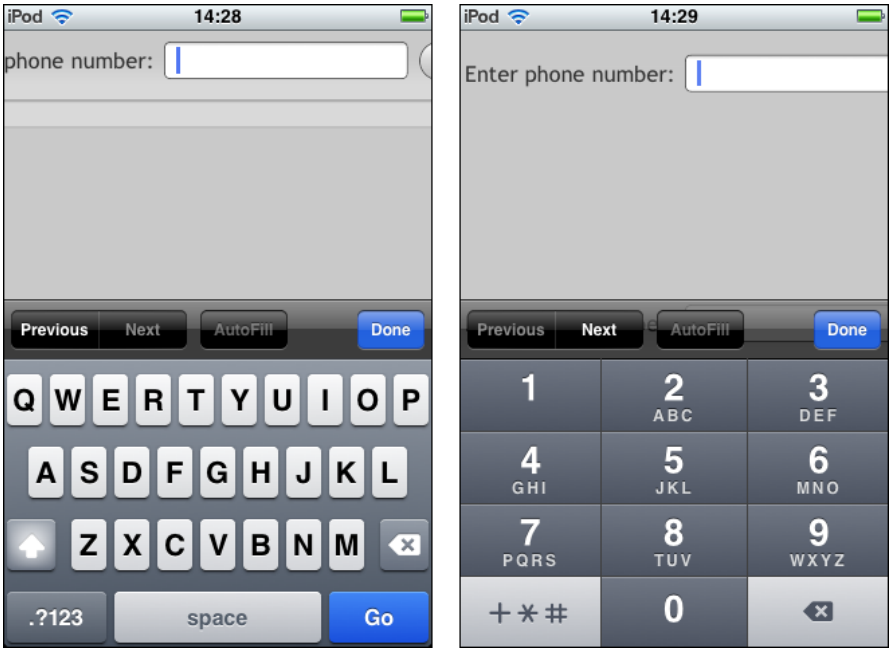


Exhibit 2-3: The tel input type optimizes touchscreens for entering phone numbers

Radio buttons

Radio buttons are meant to be used when a group of options are mutually exclusive; that is, when it's valid to select only one option in a group. For example, you can use radio buttons to ask for a user's gender or credit card type. To create a group of radio buttons, you give them the same name by using the `name` attribute. Then, use the `value` attribute to establish a unique value for each option. For example, to create the two radio buttons shown in Exhibit 2-4, asking users to indicate their gender, you would write:

```
<label>
 <input type="radio" name="gender" value="male" /> Male
</label>

<label>
 <input type="radio" name="gender" value="female" /> Female
</label>
```

In this example, each input element and its text prompt are contained in a `<label>` element, which is an alternative to using matching `for` and `id` attributes.

When this form is submitted, the name/value pair will be either `gender="male"` or `gender="female"`. If the user selects one option and then selects the other, the first option is deselected automatically.

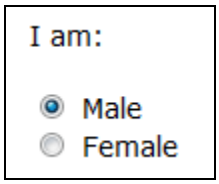


Exhibit 2-4: Radio buttons, with the Male option selected

Checkboxes

You can use checkboxes to prompt users to select an individual option, as shown in Exhibit 2-5, or you can create a group of related checkboxes so that users can select one or more options. A checkbox that is not checked is not submitted with other data in the form.

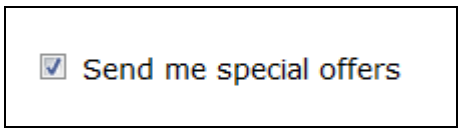


Exhibit 2-5: A checkbox input (selected)

Depending on the type of options you're providing, you can set a checkbox value to "yes." For example, the following code creates the example shown in Exhibit 2-5.

```
<label>
 <input type="checkbox" name="promos" value="yes" />
 Send me information on specials and promotions
</label>
```

If this box is checked, the form will submit `promos="yes"` to the processor.

To create a group of related checkboxes, give them the same name with different values. For example, the following code creates the checkbox options shown in Exhibit 2-6.

```
<label>
 <input type="checkbox" name="spice" value="cinnamon" />
 Cinnamon
</label>

<label>
 <input type="checkbox" name="spice" value="curry" />
 Curry
</label>

<label>
 <input type="checkbox" name="spice" value="anise" />
 Anise
</label>
```

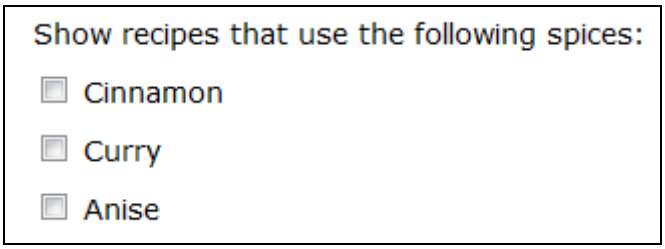


Exhibit 2-6: A group of related checkboxes

Creating a default selection

If you want a radio button or checkbox to be selected by default, you use the `checked` attribute. This is a *Boolean attribute*, which implies either a true or false condition. In HTML5, Boolean attributes are true if they're present and false if they're not present—actual "true" and "false" values are not valid. For example, to make a radio button or checkmark selected by default, you would write:

```
<label>Standard delivery
<input type="checkbox" name="standard" value="yes" checked>
</label>
```

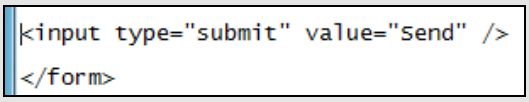
Or, if you prefer to use XHTML, which requires that all attributes have quoted values, you would repeat the attribute as the value, as follows:

```
<label>Standard delivery
<input type="checkbox" name="standard" value="yes"
checked="checked" />
</label>
```

If a checkbox is checked by default, a user can clear it by clicking it. If a reset button is present on the form, clicking it will return the checkbox to its default selected state.

Do it!

A-2: Creating e-mail fields, checkboxes, and radio buttons

Here's how	Here's why
1 In the HTML, click before the `<submit>` input tag, as shown	
	To place the insertion point here.
Press ENTER several times	To create some space for additional input fields before the Submit button.
2 In the space above the submit input, add the following code:	

```
<label for="email">E-mail:</label>
<input type="email" id="email" name="Email" />
```

3 On the next line, add the following code:	To define the text as a level-three heading. This text prompt is not a label for an input field.

```
<h3>Gender:</h3>
```

4 On the next line, add the following code:	To create two radio buttons to collect gender data.

```
<label>
 <input type="radio" name="gender" value="male" />
 Male
</label>
<label>
 <input type="radio" name="gender" value="female" />
 Female
</label>
```

	By nesting each input inside a `<label>` element, you associate the text label with its input field, so you don't need matching `for` and `id` attributes.
5 On the next line, add the following code:	To create a single checkbox.

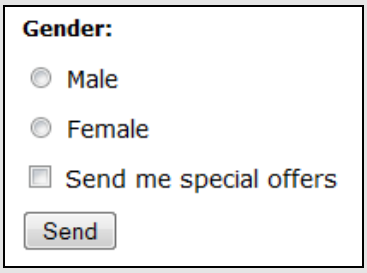
```
<label>
 <input type="checkbox" name="Offers" value="Yes" />
 Send me special offers
</label>
```

6 Wrap the submit input in a label, as follows:

```
<label>
<input type="submit" value="Send" />
</label>
```

Even though the value attribute provides the text label, using a label here maintains consistency and even spacing between the button and adjacent input elements.

7 Save your changes and reload the page in Chrome



8 Click either **Male** or **Female**

Select the other option — The previous selection is automatically deselected.

9 Click anywhere on the checkbox text — To select the checkbox. When you use labels, the text label for an input field becomes clickable, making the form easier to use.

Clear the checkbox — Click it again.

10 Switch to Notepad

Add the following bold code to the checkbox input:

```
<label>
 <input checked="checked" type="checkbox"
  name="Offers" value="Yes" /> Send me special offers
</label>
```

11 Save your changes and reload the page in Chrome — The checkbox is selected by default.

Clear the checkbox — Creating a default selection does not force the selection; a user can still deselect it.

12 When you're collecting phone numbers, what input type should you use and why?

Controlling the size of input fields

Explanation

You can use the `size` attribute to set the width of input fields. Values of the `size` attribute are measured not in pixels but in the number of text characters in a monospaced font, such as Courier. For example, to create an input field that's 30 monospaced characters long, you would specify `size="30"`. The default size of an input field is 20 characters.

Note that the `size` attribute, though measured by the number of characters you specify, controls only the appearance of the field onscreen; the `size` attribute doesn't actually limit the number of characters that users can enter in the field. To do that, you use the `maxlength` attribute.

Limiting the number of characters in an input field

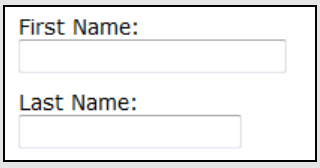
A text input field allows an unlimited number of characters, but that's often not the best way to collect data. You can help guarantee the integrity of the data you collect by setting limits on the number of characters allowed in a given text field. For example, if you have a U.S. ZIP code field, you might want to limit the number of characters to 5.

To set a limit on the number of characters allowed in a field, you use the `maxlength` attribute. For example, the following code limits the number of characters to 10:

```
<input type="text" name="UserName" maxlength="10" />
```

Do it!

A-3: Setting input field size and limiting data length

Here's how	Here's why
1 Switch to Notepad	
Add the following bold code to the First Name input:	To increase the size of the input field from the default size of 20 monospaced characters to 25.
`<input **size="25"** type="text" id="first" name="FirstName" />`	
2 Save your changes and reload the page in Chrome	
	The First Name field is now longer than the Last Name field.
3 Make the Last Name field the same size as the First Name field	
4 Set the E-mail input field's size to **35**	To accommodate longer e-mail addresses.
5 Set the ZIP code input field's size to **10**	
Save and verify your changes	Next, you'll control the number of characters allowed in certain fields.

6 In the ZIP code field, enter 15 or more numbers	There's no limit to the number of characters you can enter in a field, unless you manually limit the number of characters.
Double-click in the ZIP code field	To select the numbers.
Press DELETE	To delete the numbers.
7 Switch to Notepad	
Add the following bold code to the ZIP code input:	Standard ZIP codes are five digits.

```
<input maxlength="5" size="10" type="text" id="zip" ►
name="ZIP" />
```

8 Save your changes and reload the page in Chrome	
9 Try to enter more than five digits in the ZIP code field	The input field now accepts entries of only five or fewer characters.
10 Why is it often important to limit the number of characters allowed in a field?	
11 Close the Notepad file	

Password fields

Explanation

Password fields look the same as text input fields until you enter data into them. Instead of showing alphanumeric characters, a password field displays a series of asterisks or bullets, depending on the browser, to help protect sensitive information like passwords or PIN codes. To create a password field, you would write:

```
<input type="password" name="password" />
```

The value of the `name` attribute doesn't have to be "password" or an abbreviation such as "pwd" or "pass," but either one is a logical choice when you're collecting a password. If you're collecting some other kind of sensitive data, such as a PIN code, you might use "pin" as the value of the `name` attribute. Again, the `name` attribute and the value that a user enters create the name/value pair that's sent to the form processor.

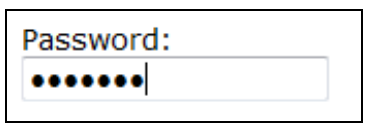


Exhibit 2-7: Password content is obscured for data entry privacy

When your form contains a password field or any other kind of sensitive information, such as credit card numbers, the form should use the `post` method, which sends form data by using an HTTP post transaction. This method is more secure, and there's no limit to the number of name/value pairs you can create.

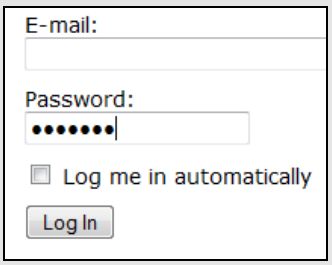
Do it!

A-4: Creating a password field

The files for this activity are in Student Data folder **Unit 2\Topic A**.

Here's how	Here's why
1 In Chrome, click **Log In**	(In the navigation bar.) To open the login.html page.
2 In Notepad, open login.html	You can right-click the file and choose Open with, Notepad.
3 Add the following bold code to the `<form>` element: `<form action="processor.php" **method="post"**>`	Under the LOGIN FORM comment. This form will submit passwords, so you'll use the `post` method, which submits the data by using an HTTP post transaction rather than by appending the data to the URL.

4 Under the E-mail input, enter the following code:
 	
`<label for="pass">Password</label>`
`<input type="password" id="pass" name="pwd" />`	

5 Save your changes and reload the page in Chrome	
Type data in the Password field	 The data is obscured for privacy.

Autofocus

Explanation

When a user opens a page with a form, the insertion point does not appear in any of the form input fields by default. To start entering data, the user has to click in a form field and begin typing. To improve the usability of a form, you can use the `autofocus` attribute to give focus to an input field as soon as the page loads so that the user can begin typing immediately.

The `autofocus` attribute is another Boolean attribute, implying either a true or false condition by its presence or absence. To give an input field automatic focus when the page loads, you would write:

```
<input type="text" name="username" autofocus>
```

Or, if you're using XHTML, you repeat the attribute as the value, as follows:

```
<input type="text" name="username" autofocus="autofocus" />
```

Setting autofocus on the first input field in a form can be especially helpful to users who prefer to navigate a form by pressing the Tab key. When the page loads, they can begin typing immediately, press Tab to move to the next input field, and ultimately navigate to the Submit button and press Enter to send their information without ever having to use the mouse or touchpad. Autofocus can be applied to only one form input per HTML document.

Autocomplete

Autocomplete is a common browser feature that helps users fill out form fields that they have completed in the past. For example, if you enter your e-mail address in an e-mail input field and then return to the form at a later date, you can simply click the field to display your previously entered e-mail address, and then click the address to populate the input field with that address.

Autocomplete is on by default in most browsers, but you can use the `autocomplete` attribute to control this feature. To disable autocomplete for an entire form, put the attribute in the `<form>` element and set its value to "off," as follows:

```
<form autocomplete="off" action="scripts/processor.php" />
```

You can also use the `autocomplete` attribute in individual input fields. This makes it possible to have autocomplete "on" for a form but "off" for one or more input fields, or vice versa.

In password input fields, autocomplete is not enabled by default. You should disable Autocomplete for any input field that collects other types of sensitive information, or information that's not likely to be re-used, such as a download key or coupon code.

Do it!

A-5: Controlling autofocus and autocomplete

Here's how	Here's why
1 Click **Log In**	(In the navigation bar.) To reload the login.html page.
Observe the form's input fields	The insertion point does not appear in any of the input fields by default.
2 Switch to Notepad	

3	Add the following bold code to the E-mail input: `<input **autofocus="autofocus"** size="35" type="email" id="email" name="Email" />`	To allow the user to begin typing as soon as the page loads.
4	Save your changes and reload the page in Chrome	The insertion point now automatically appears in the E-mail input field when the page loads.
5	Type your e-mail address	Or use a fake but valid e-mail address such as *someone@somewhere.com*.
6	Submit the form	(Click the Log In button.) A "File not found" page is displayed because the processor.php file is just an example and is not an actual file in the current directory.
	In the Address bar, observe the end of the URL	(If necessary, drag to the right in the Address bar to see the end of the URL.) This form's method is `post`, so the name/value pairs are not appended to the URL.
	Go back to the previous page	Click the browser's Back button (the left-pointing arrow).
7	Reload the page	
	Click in the E-mail field	The e-mail address you entered previously is displayed as an option. (If it doesn't appear automatically, type the first letter of the e-mail address.)
	Click the e-mail address	To populate the input field.
8	Switch to Notepad	
	Add the following bold code to the E-mail input: `<input **autocomplete="off"** autofocus="autofocus" size="35" type="email" id="email" name="Email" />`	
9	Save your changes and reload the page in Chrome	
	Click in the E-mail field	The e-mail address you entered previously is not displayed.
10	Close all open windows	

The <textarea> element

Explanation

When you want to collect comments from users or gather any other type of text information that may involve multiple lines of text, such as site feedback and suggestions, use a text area field. Text area fields are large to accommodate the amount of text that a user might enter. Exhibit 2-8 shows a text area field with a default size.

Exhibit 2-8: A text area field in its default size

To create a text area field, you use the `<textarea>` element. This is a unique container element because it doesn't actually contain anything within the HTML document. As always, you use the `name` attribute to establish the name/value pair—the text that a user enters into the field creates the value.

Text in a text area field appears in a monospaced font by default. To change it, you need to specifically set the font for the text area element.

Setting the size of a text area field

You can control the size of a text area field by using the `rows` and `cols` attributes, which stand for rows and columns. The `rows` attribute controls the height of a text area field, and the `cols` attribute controls the width. Both values must be greater than zero.

If you want a text area field to show initial text that the user can edit or delete, enter that text inside the `<textarea>` container. However, this isn't commonly done because some users might leave the default text there, resulting in unnecessary and duplicate data being sent to the form processor and the database.

Controlling the resizing of a text area field

In some browsers, including Firefox and Chrome, text area fields have a small resize handle in the lower-right corner by default, as shown in Exhibit 2-8. This handle (which some people call a "grabber") indicates that the text area field can be resized. This might suit you just fine—it often makes sense to allow users to view as much of their text entry as possible, especially if the information they're providing tends to be lengthy.

In some cases, however, allowing your text area fields to be resized can have an undesirable effect on your page layout, so you should get in the habit of checking. On each page that has a text area field, drag the field's resize handle horizontally and vertically to verify that content doesn't shift awkwardly or overlap adjacent areas.

To disable the resizing of text area fields, just open your style sheet and add the following rule:

```
textarea { resize: none; }
```

When this style is used, the resize handle disappears and the text area field can't be resized. The `resize` property accepts three other values in addition to `none`. The other values are `horizontal`, used to enable only horizontal resizing, `vertical`, for only vertical resizing, and `both`, which creates the original behavior.

Using placeholder text

With the `placeholder` attribute, you can create placeholder text in your form input fields. Placeholder text should not be used to replace labels, but rather should be used to give users a brief hint or guideline for the data they can enter in a particular field. For example, the placeholder shown in Exhibit 2-9 provides guidance for proper data entry.

Exhibit 2-9: Placeholder text

Placeholder text always appears inside an input field until a user either clicks the input field to select it or presses the Tab key. By default, placeholder text is gray. You can use CSS to customize the appearance of placeholder text.

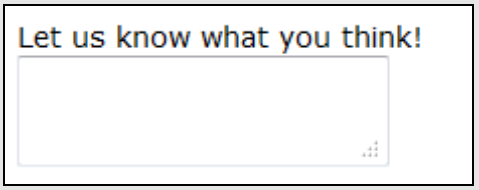
Do it!

A-6: Creating a text area field and placeholder text

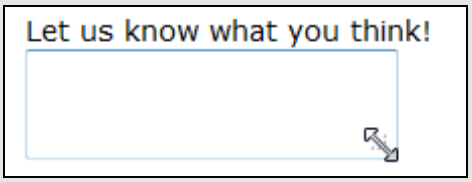
The files for this activity are in Student Data folder **Unit 2\Topic A**.

Here's how	Here's why
1 In Firefox, open feedback.html	(Right-click the file and choose Open with, Firefox.) If a message appears that asks if you want to make Firefox your default browser, clear the "Always perform this check when starting Firefox" option, and then click No.
2 In Notepad, open feedback.html	
3 Scroll down to the form	
4 Under the TEXTAREA comment, add the following code:	Be sure there's no space between the opening and closing tags, or the browser will interpret the space as default content.

```
<label for="feedback">Let us know what you think!</label>
<textarea id="feedback" name="comments"></textarea>
```

5 Save your changes and reload the page in Firefox	
	The text area field appears with a default size.
Observe the lower-right corner	The dotted triangle indicates that this is a draggable region. This indicator will look slightly different in other browsers.

6 Point to the lower-right corner, as shown



The pointer changes to a diagonal arrow, indicating that the text area field can be resized by dragging. This is the default behavior in some browsers.

Drag the text area field to the right

To increase its size. In some layouts, allowing users to resize input fields can affect the flow of adjacent content in undesirable ways.

7 From the styles folder, open globalstyles.css

In the current topic folder.

8 Under the FORM STYLES comment, add the following rule:

```
textarea { resize: none; }
```

Save the style sheet

Reload the page in Firefox

The text area field returns to its default size, and the lower-right corner no longer contains the resize indicator.

9 Switch to the HTML file

(Feedback.html.) You'll modify the text area field.

Add the following bold code to the `<textarea>` tag:

To increase the size of the text area field.

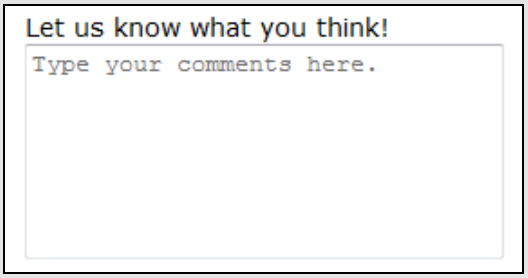
```
<textarea rows="6" cols="30" id="feedback" name="comments">
```

10 Save your changes and reload the page in Firefox

The text area field is now larger than the default size.

11 Add the following bold code to the `<textarea>` tag:

```
<textarea placeholder="Type your comments here." rows="6"
cols="30" id="feedback" name="comments">
```

12	Save your changes and reload the page in Firefox	 By default, text in a text area field appears in a monospaced font. To change it, you need to set the font for the `<textarea>` element explicitly.
13	Click in the text area field	The placeholder text disappears.
	Type some text	
	Click **Clear**	To clear the form. The placeholder text is displayed again.
14	Close Notepad	

Select lists

Explanation

Select lists (also called select boxes or select menus) provide users with drop-down lists of predefined options. These lists allow you to offer multiple options without occupying any space on the screen. The options in the list are hidden, as shown in Exhibit 2-10, until a user clicks the drop-down arrow to display them, as shown in Exhibit 2-11.Select lists look slightly different depending on the browser and platform, but they function the same way in all browsers.

Exhibit 2-10: A select list in its default state

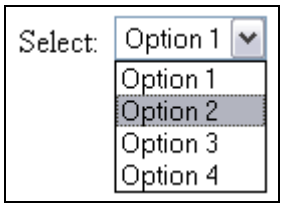


Exhibit 2-11: The same select list when activated

The <select> and <option> elements

To create a select list, you use the `<select>` element with two or more `<option>` elements. Each `<option>` element defines an option in the list. You use the `name` attribute in the `<select>` element, and each `<option>` element uses the `value` attribute to match the value of its text label. For example:

```
<select name="creditCard">
 <option value="mc">MasterCard</option>
 <option value="visa">Visa</option>
 <option value="amex">American Express</option>
</select>
```

When a user makes a selection, the name/value pair is generated by the combination of the select list's name and the value of the selected option. The width of a select list is determined by the size of the widest option in the list, plus the width of the drop-down arrow.

Setting a default option

By default, the first option in a select list is displayed. If the first option is the most frequently selected option based on your audience, it makes sense to leave it as the first option. Or you can use the `selected` attribute to define the default option. This is another Boolean attribute; to make an option pre-selected when the page loads, you would write:

```
<option value="US" selected>United States</option>
```

Or, if you're using XHTML, you repeat the attribute as the value, as follows:

```
<option value="US" selected="selected">United States</option>
```

Creating an empty "null" option

Sometimes a default selection isn't the best choice. For example, if you're collecting credit card information, making one credit card type pre-selected might make some users overlook the selection; the card type and number might then be mismatched. When a default selection can lead to user error, or in any other situation where a default option doesn't seem like the best choice for the user interface, you can create a *null option*.

To create a null option, specify "null" (or any other value that indicates nothing) as the first option value. Then, either add a prompt, such as "Select…," instead of an actual value, or leave the option container blank. For example:

```
<option value="null">select...</option>
```

This code would create the result shown in Exhibit 2-12.

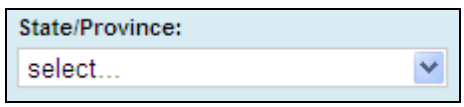


Exhibit 2-12: A select list with a "null" option used as a prompt

If the null option is not the first option in the select list, it will not be visible by default unless you use the `selected` attribute to make it the default. You would use a simple script to prevent a null option from being submitted mistakenly.

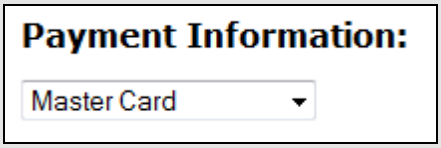
Do it!

A-7: Creating a select list

The files for this activity are in Student Data folder **Unit 2\Topic A**.

Here's how	Here's why
1 In Firefox, open order.html	(From the current topic folder.) You'll add a select list to this form.
2 In Notepad, open order.html Scroll to the bottom of the file	

3 Under the CREDIT CARD INFO comment, type:
 	
`<select name="CardType">`
`<option value="MC">MasterCard</option>`
`<option value="Visa">Visa</option>`
`<option value="Amex">American Express</option>`
`</select>`
	To create a select list with three options.

4 Save and verify your changes	 The first option in the list is displayed by default.
Click the drop-down arrow	To open the list. Notice that the width of the select list is determined by the longest option.
Select **American Express**	The selected option appears in the field.
Click **Clear form**	To reset the form. The first selection is displayed again.
5 Add the following option as the first option in the list: `<option value="null">Select</option>`	
	When a default selection can lead to user error, it often makes sense to create either an empty default option or a null option that serves as a selection prompt.
6 Save your changes and reload the page in Firefox	
Click **Clear form**	"Select" now appears by default. (You would use a simple script to prevent a null option from being submitted mistakenly.)

Option groups

Explanation

You can define option groups if the options in your select list are best presented in separate named groups. For example, if your select list contains a list of available coffees, you might want to group them by country of origin, as shown in Exhibit 2-13. This is another way you can improve the usability of your Web forms.

The <optgroup> element

To create an option group, you use the `<optgroup>` element. This element is valid only inside a `<select>` element. You place related `<option>` elements inside an `<optgroup>` container, and use the required `label` attribute to create the text label that organizes the options in the list. For example, the following code creates the select list shown in Exhibit 2-13.

```
<select name="coffee">
 <option value="null"> </option>
 <optgroup label="Africa/Arabia">
  <option value="Kenya">Kenya (bold)</option>
  <option value="Ethiopia">Ethiopia (bold)</option>
  <option value="Sanani">Arabian Sanani</option>
 </optgroup>

 <optgroup label="Asia/Pacific">
  <option value="Sumatra">Sumatra (bold)</option>
  <option value="Java">Mocha Java</option>
 </optgroup>
</select>
```

Exhibit 2-13: Option groups in a select list

Option group labels

The `label` attribute is valid only within the `<optgroup>` element. It provides the text label for a group of related options in a select list. It's not the same as the `<label>` element.

By default, most browsers make option group labels bold and italic, with their related options indented beneath them, as shown in Exhibit 2-13. Labels cannot be selected.

Do it!

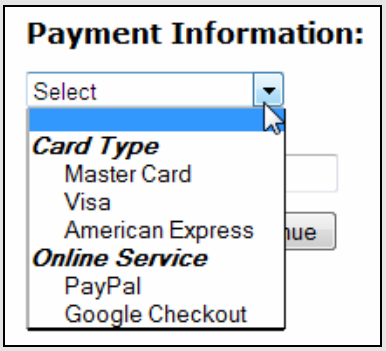
A-8: Defining option groups for a select list

Here's how	Here's why
1 After the null option, add the following bold code:	To define an option group for the three card types. Be sure to close the option group before the closing select tag, as shown.

```
<select name="CardType">
 <option value="null">Select</option>
 <optgroup label="Card Type">
 <option value="MC">MasterCard</option>
 <option value="Visa">Visa</option>
 <option value="Amex">American Express</option>
 </optgroup>
</select>
```

2 After the first option group, add the following bold code:	To define a second option group, one that contains payment alternatives.

```
<select name="CardType">
 <option value="null">Select</option>
 <optgroup label="Card Type">
 <option value="MC">MasterCard</option>
 <option value="Visa">Visa</option>
 <option value="Amex">American Express</option>
 </optgroup>
 <optgroup label="Online Service">
 <option value="PP">PayPal</option>
 <option value="GC">Google Checkout</option>
 </optgroup>
</select>
```

3 Save your changes and reload the page in Firefox	
Open the select list	 The options in the list are grouped under different headings.
Observe the label styles	Most browsers make option group labels bold and italic by default. Google Chrome applies only bold formatting.
4 Try to select an option group label	Option group labels are not selectable options.

Data lists

Explanation

You can use a data list with a text input field to combine the benefits of the select list with the freedom of the standard input field. In other words, you can make it easy for users to select predefined items, as in a select list, but without limiting users to only those options set forth in the list.

The <datalist> element

The `<datalist>` element is a container that defines a list of options for a text input field. When a user begins typing into an input field that has a data list, a drop-down list appears with the predefined options. This drop-down list also has basic autocomplete functionality, wherein the options displayed respond to the user's entry in real time. As soon as the user enters a character that's not part of any predefined entry, the drop-down list closes and the user can finish typing the data.

The `<datalist>` element requires an `id` attribute with a value that exactly matches the value of the `list` attribute in the associated `<input>` element. Then, to define each option in the data list, you use the `<option>` element, just as you would for a traditional select list. For example, the following code produces a text input field with a data list for entering commonly entered country names.

```
<label for="country">Country:</label>
<input type="text" list="country-list" name="country"
      id="country" />
<datalist id="country-list">
      <option value="United States" />
      <option value="Canada" />
      <option value="United Kingdom" />
      <option value="Australia" />
</datalist>
```

Here, the `<datalist>` element's ID, `country-list`, binds directly to the input field via the `list` attribute's matching value. When the text input field receives focus, nothing happens. But when the user begins typing in the field, the options that appear in the drop-down list update in real time, as shown in Exhibit 2-14.

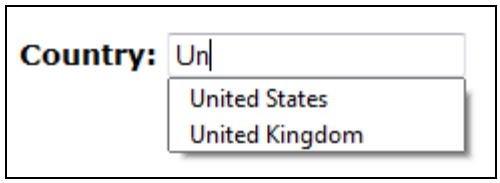


Exhibit 2-14: The data list displaying the only two entries that contain "un"

Also, notice that text values are not required for the individual options, as they are with options in a `<select>` element. In the code example above, the `<option>` elements are self-terminated for XHTML compliance, but you could also simply omit the terminating slash or closing tag.

Using a data list helps you to collect valid data without having to create a giant select list that includes infrequently selected options. It also provides the user with two input methods: the mouse for clicking a predefined option, and the keyboard for entering data manually.

Checking spelling

By default, most browsers provide spelling check functionality in text area fields. This functionality is controlled by the `spellcheck` attribute. The value of this attribute must be either `true` or `false`. If it's set to true, the element's content will be checked. If the attribute is set to false, the element's content will not be checked.

This functionality can also be useful in other fields, such as data lists, where data entry accuracy is important and the field is not collecting names or other data for which spelling is not applicable.

If you have several input fields for which you want to check spelling, you don't have to add a `spellcheck` attribute to each field. Instead, you can apply the `spellcheck` attribute to a parent element, such as a `<div>` container. All elements inside it will inherit its `spellcheck` value.

Do it!

A-9: Creating a data list and checking spelling

The files for this activity are in Student Data folder **Unit 2\Topic A**.

Here's how	Here's why
1 Under the COUNTRY comment, add the following bold code:	(Scroll up.) You will match the value of the list attribute to the data list's ID to bind the two elements.
`<input **list="country-list"** size="25" type="text" id="country" name="country" />`	
2 On the next line, add the following elements:	

`<datalist id="country-list">`
`  <option value="United States" />`
`  <option value="Canada" />`
`  <option value="United Kingdom" />`
`  <option value="Australia" />`
`</datalist>`	

3 Save your changes and reload the page	The Country field looks like an ordinary text input field.
Click inside the Country field	It still looks like an ordinary text input field.
4 Type **u**	Three of four options in the data list appear in a drop-down list because they contain the letter "u." ("Canada" does not.)
Type **n**	To form "un"; now two options are displayed—the only two that contain contiguous "u" and "n" characters.

5 Point to United States	
	The option is highlighted and the cursor changes to an arrow.
Click **United States**	To select the option. Using a data list can be helpful in certain situations. You can collect data without asking users to scroll through a large select list, and it gives users two input methods: the mouse for clicking an option, and the keyboard for typing data.
6 In the navigation bar, click **Feedback**	To open the feedback page you worked on earlier.
7 Click in the text area field	
Type a random misspelled word	A red squiggly line appears under the word; this line is a standard user interface convention for indicating a misspelling or unidentified word. Modern browsers enable this functionality by default, but only for text area fields.
8 In Firefox, go back to the order.html page	(Click the "Go back one page" button.) You'll continue to change order.html.
In the Country field, type **Portgal**	To misspell "Portugal."
Click outside the field	The spellcheck feature is not enabled in input fields by default.
9 In Notepad, switch to order.html	
Add the following bold code to the Country input:	
Save your changes and reload the page in Firefox	

```
<input spellcheck="true" list="country-list" size="25"
type="text" id="country" name="country" />
```

10 Click **Clear form**	If necessary, to clear the previous entry in the Country field.
In the Country field, type **Portgal**	
Click outside the field	Now, the browser flags the word as a misspelling or unknown word.
11 Right-click the misspelled word	A menu opens. Just as in word processing programs, you can select from a list of guesses, or add the word to your dictionary so that the browser doesn't flag it in the future.
Select **Portugal**	To apply the correct spelling of the country.
12 Close all open files	
13 If you have several input fields for which you want to check spelling, do you need to add a `spellcheck` attribute to each individual input?	

Topic B: Data validation and error feedback

Explanation

Before browsers supported HTML5, you had to use JavaScript to validate user data on the client side. Now, browsers provide basic, built-in form validation, so you don't have to add scripts to check for basic things such as whether a required field was completed.

Native input field validation

In the past, validating user input involved a lot of scripting. Now, with HTML5's new form elements and attributes, browsers can implement this key functionality natively. This means far less code and more consistent behaviors that are relatively fast and easy to deploy.

For example, you can provide basic e-mail validation simply by using the `email` input type—no additional code is required. If a user enters information that is not in a valid e-mail address format, an error message is displayed. Exhibit 2-15 shows a default e-mail error message in Google's Chrome browser.

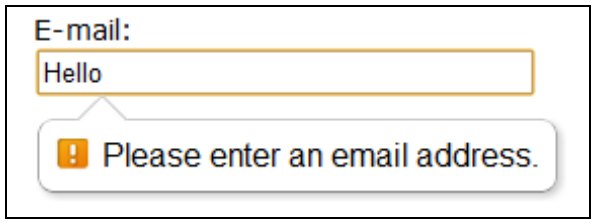


Exhibit 2-15: A data validation message (viewed in Chrome)

This kind of basic validation helps to guarantee the integrity of the data you collect. In browsers that don't support the `email` input type, the `text` input type is the default, so the `email` input type will be ignored and the data entered will render just as it always has. So, there's no disadvantage to using the `email` input type.

Required fields

You can use the `required` attribute to make an input field required—the form cannot be submitted until that field contains data. The `required` attribute is another Boolean attribute, implying either a true or false condition. To make an input field required, you would write:

```
<input type="text" name="username" required>
```

Or, if you're authoring in XHTML, you repeat the attribute as the value, as follows:

```
<input type="text" name="username" required="required" />
```

One of the great benefits of using the `required` attribute is that it's faster and more efficient both for users and for your server load. In many older implementations of this type of functionality, users are not given feedback until after they submit the form. The form entries are validated on the server side, and a revised page is returned with error feedback. By using the `required` attribute, you can provide this important error messaging without leaving the current page. This technique improves the usability of the form and reduces the processing required on the server end.

Disabling built-in validation

When you use the new input types of HTML5 or use the `required` attribute, built-in validation becomes automatic. If you want to take advantage of the new input types but you prefer to use custom script-based validation, you can add the `novalidate` attribute to the `<form>` element. This attribute disables automatic validation so that you can take advantage of the benefits of the new HTML5 input types while sticking with your own script-based data validation solutions.

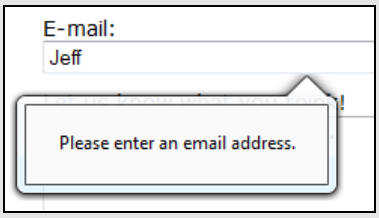
The `novalidate` attribute can stand on its own, or if you're authoring with XHTML, you repeat the attribute name as its value, as follows:

```
<form novalidate="novalidate" action="processor.php"
method="post">
```

Do it!

B-1: Creating required input fields

The files for this activity are in Student Data folder **Unit 2\Topic B**.

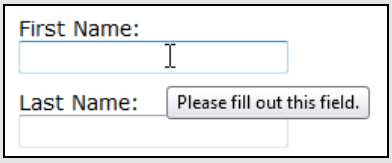
Here's how	Here's why
1 In Firefox, open feedback.html	(In the current topic folder.) Right-click the file and choose Open with, Firefox.
2 In the E-mail input field, type your name	(Not your e-mail address.) You'll observe a validation error message.
Press ENTER	 To try to submit the form. Firefox validates e-mail input types and displays a default error message if the entry is not in a valid e-mail address format.
Click in the E-mail input field	The message disappears and Firefox highlights the field in red to indicate the error. These default styles will be slightly different in other browsers.
3 In Notepad, open feedback.html	(From the current topic folder.) You'll make the First Name, Last Name, and E-mail fields required.
Add the following bold code to the First Name input: `<input required="required" size="25" type="text" id="first" name="FirstName" />`	
4 Add the same attribute to the Last Name and E-mail input fields	To make them required fields.

5 Save your changes and reload the page in Firefox

Click **Clear**

If necessary, to clear the input fields.

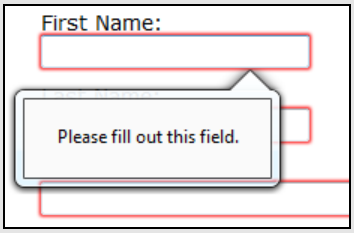
6 Point to the First Name field



In Firefox, a default tooltip appears when you point to a required field. (At the time of this writing, not all browsers provide these default tooltips.)

7 Click in the First Name field

Press ↵ ENTER



To try to submit the form without entering any information. Firefox displays an error message indicating that the First Name field must be filled out before the form can be submitted. Also, all empty required fields are highlighted in red to indicate their error status.

8 In Chrome, open feedback.html

You'll observe the default validation message in Google's Chrome browser.

9 Click **Send**

To try to submit the form. Even though three fields are required, Chrome highlights only the first field and displays an error message. Browsers have different details, but the core functionality is consistent.

10 In Notepad, close feedback.html

Keep Chrome open.

11 Name one advantage of using this data validation technique.

12 Why might you want to disable browser-based validation by using the `novalidate` attribute?

Basic pattern validation

Explanation

Using HTML5 input types and the `required` attribute isn't the only way you can take advantage of native validation in today's browsers. You can also use the `pattern` attribute to verify that user data matches a specific format or data type. For example, you can require that data entered into a specific field is numeric, or you can require that a date be entered in a specific format. You do this by using regular expressions.

Regular expressions

Regular expressions are used in scripting languages such as Perl and JavaScript. They provide a way to match strings of text, such as words, numbers, special characters, or patterns of characters. In the context of Web forms, regular expressions are useful for validating user-entered data against specific criteria written using regular expression syntax. Regular expressions are also called *regex* or *regexp*.

Some form input types use default regular expression matches. For example, the `email` input type automatically verifies that its data contains text followed by the @ symbol. You can take this further by requiring additional e-mail address components; you'd do this by using the `pattern` attribute to match a regular expression.

The basics of regex syntax

Expressions are used to match specific patterns. For starters, brackets `[ ]` are used to indicate a range of characters, and braces `{ }` are used to set limits, such as minimum and/or maximum numbers allowed. The following table provides some basic examples.

Expression	Description
`[0-9]`	This expression matches any digit between 0 and 9.
`[1-9]`	This expression matches any digit between 1 and 9.
`[a-z]`	This expression matches any lowercase character between a and z.
`[A-Z]`	This expression matches any uppercase character between A and Z.
`[a-Z]`	This expression matches any character from lowercase a to uppercase Z.
`{5}`	This indicates that the pattern *preceding* it must match exactly five times. For example, the expression `[0-9]{5}` translates to "any numbers between 0 and 9, but there must be exactly five digits."
`{5,}`	By adding a comma, you indicate that the pattern preceding it must match five or more times. For example, the expression `[0-9]{5,}` translates to "any numbers between 0 and 9, but there must be at least five digits."
`{5,10}`	By adding a number to the right of the comma, you set a maximum limit. For example, the expression `[0-9]{5,10}` translates to "any numbers between 0 and 9, but there must be at least 5 digits and no more than 10."

Ensuring that user data is numeric

One simple way that you can use regular expressions to validate your form data is to ensure that data entered in a numeric field is actually numeric. Or you might want to validate that the data is numeric and set a minimum number of digits. This capability can be useful for such things as postal code fields or standardized number schemes like ISBNs.

Validating for specific data types like this can help to guarantee data integrity. You don't want a database full of random data; if you're asking users to enter numeric data, such as a ZIP code, you should use some form of data type validation.

Using the previous table as a guide, you can guarantee that the information entered into a field is numeric, and that at least 5 numbers but no more than 10 numbers are entered, by using the following expression in a form input field:

```
pattern="[0-9]{5,10}"
```

Titles improve usability with regular expressions

When you use the `pattern` attribute to limit user data, you should also use the `title` attribute to provide users with a brief description of the required pattern. When you point to an input field that has a title, your message is displayed in the form of a tooltip, as shown in Exhibit 2-16.

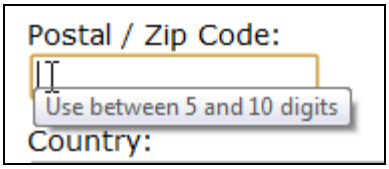


Exhibit 2-16: An input title appears as a tooltip

In Firefox, this default title is added automatically whenever the `pattern` attribute is used in a form input field. But not all browsers apply default titles, so it's important to specify your own. To do so, just add the `title` attribute to any input field that has a `pattern` attribute. The value of the `title` attribute will be displayed as a tooltip and will be used by assistive technologies like screen readers.

Do it!

B-2: Requiring numeric data with limits

The files for this activity are in Student Data folder **Unit 2\Topic B**.

Here's how	Here's why
1 In Chrome, open order.html In Notepad, open order.html	From the current topic folder.
2 Make Postal / Zip code a required field	Scroll down to locate the POSTAL / ZIP comment, and add the `required` attribute to the PostalCode input.
3 Add the following to the PostalCode input: `pattern="[0-9]{5,10}"`	To limit the postal code field to accept only numbers: a minimum of 5 and maximum of 10.

4 Add the following to the PostalCode input:

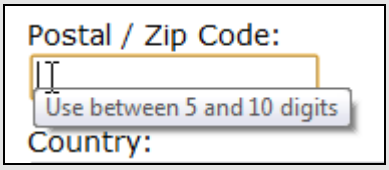
```
title="Use between 5 and 10 digits"
```

5 Save your changes and reload the page in Chrome

6 Enter your first name and last name in the appropriate fields

These are required fields.

7 Point to the Postal / Zip Code field



8 In the Postal / Zip Code field, type **1234** and press ↵ ENTER

The error message combines Chrome's default error message with your title content. Firefox does this also, although the formatting is different.

9 Type a fifth number

The error message disappears, indicating that the data entered meets the input requirements.

Press ↵ ENTER

To try to submit the form. Instead of the form data being submitted, the next empty required field is highlighted.

10 In the Postal / Zip Code field, add more numbers to total 11 or more

Press ↵ ENTER

The pattern limits this field to between 5 and 10 digits, so the error message is displayed again. This sort of pattern limitation can help to guarantee the integrity of the data you collect.

Validating for a specific date format

Explanation

To keep the data you collect consistent, you might want to validate user data for a specific format. You can use the `pattern` attribute and regular expressions to validate more than just minimum or maximum values. For example, you can also ensure that users submit data that follows a specific format, such as a date format or a URL format.

Date formats

On Web forms, you'll typically see date fields that require a certain format, such as MM/DD/YYYY, an example of which would be 03/21/2013, or MM/YY, an example of which would be 09/13. There are many ways that you can collect date information; one way is to use a series of Select menus that gather month, day, and year information separately. Or, you can use an ordinary text input field and use a regular expression to validate the data.

The following table lists two regular-expression characters that you can use to validate date information. All other characters except `[\^$.|?*+(){}` are treated as literal instances of themselves. For example, `/` (a forward slash) means that the forward slash itself is part of the expression.

Expression	Description
`( )`	Parentheses form a logical group within a string.
`|`	The pipe character is used to indicate alternates. For example, `a|b` is the same as saying "a or b."

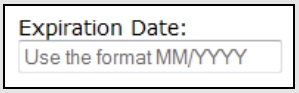
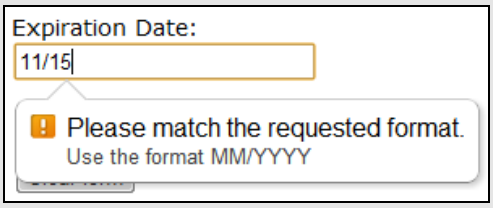
The importance of placeholders and titles

Using placeholders is important when you're validating data for a specific format. You want to clearly inform users of what you expect and make it easy for them to understand the feedback if their entry returns an error. So whenever you are asking for a specific format, use the `placeholder` attribute to concisely indicate the format. In addition, use the `title` attribute to specify the same information or provide more detailed instructions.

Do it!

B-3: Validating input for a specific date format

Here's how	Here's why
1 Switch to Notepad	
2 Scroll down to the input under the EXPIRATION comment	You'll validate data to make sure it matches a specific date format.
3 Create a placeholder that reads **Use the format MM/YYYY**	(Add a `placeholder` attribute with a value that matches the date format.) To request dates with a two-digit month and four-digit year. The input already has a title with this message.

	Save your changes and verify the result	 This text serves as a cue for users to enter their card's expiration date in this format.
4	Make Expiration Date a required input field	Add the `required` attribute.
5	Add the following code to the input: `pattern="(0[1-9]|1[012])/(20)[1-9][0-9]"`	This expression will validate that the first digit (the month) begins with either 0, any number between 1 and 9, or 10, or 11, or 12, followed by a forward slash, followed by 20, followed by any digit from 1 to 9 (which will exclude past dates), followed by any digit between 0 and 9.
6	Save your changes and reload the page in Chrome	
7	Fill out the First Name and Last Name fields	These are required fields.
8	Fill out the Postal / Zip Code and Password fields	These are also required.
9	In the Expiration Date field, type **11/15** and press ↵ ENTER	 The default error message precedes the message specified in the title.
	Change the year to **2015** and press ↵ ENTER	The date is accepted and submitted. The "page not found" error is expected; the form is not actually linked to a processing script.
	Go back to the form page	Click the Back button.
10	Would the year 2009 be a valid entry in this Expiration Date field?	
11	What is the last valid year allowed by this date rule?	
12	Close Notepad and both browsers	

Topic C: Form structure and styles

Explanation

In addition to using labels to improve the usability of your forms, you can also use fieldsets and legends. A *fieldset* is just what it sounds like: an element you use to group your form input fields into logical sections, which can make it easier for users to understand and interact with your forms. Legends provide labels for fieldset groups, helping to make your input prompts more intuitive.

Grouping inputs with the <fieldset> element

To logically group your form input fields, you use the `<fieldset>` element. For example, Exhibit 2-17 shows input controls grouped logically into two fieldsets. Grouping form controls into related sections not only makes a form more usable and intuitive, but also provides you with more design options. By default, browsers draw a box around input fields in a fieldset. This is a standard user interface convention that's often seen in software dialog boxes to make them easy to understand and use. You can customize the appearance of fieldset borders by using CSS.

Exhibit 2-17: Two fieldsets, grouping related input fields

The <legend> element

The text prompts at the top of the fieldsets shown in Exhibit 2-17—"Personal Information" and "Send me information on"—are examples of fieldset *legends*. Instead of using standard text elements, you can use the `<legend>` element inside a fieldset to logically structure the fieldset and provide a caption for a group of related fields. For example, to create the first fieldset shown in Exhibit 2-17, you would write:

```
<fieldset>
 <legend>Personal Information</legend>
 <label for="UserName">User Name:</label>
 <input type="text" name="UserName" id="UserName" />
 <label for="email">E-mail:</label>
 <input type="email" name="email" id="email" />
</fieldset>
```

To define a group of related input fields, place them inside a fieldset container, and make the legend the first element contained in the fieldset.

Dividing up your form input fields into separate fieldsets is particularly important when you're building large forms with many input fields, or when a single prompt applies to several input controls, as shown in the second fieldset in Exhibit 2-17.

Do it!

C-1: Creating fieldsets and legends

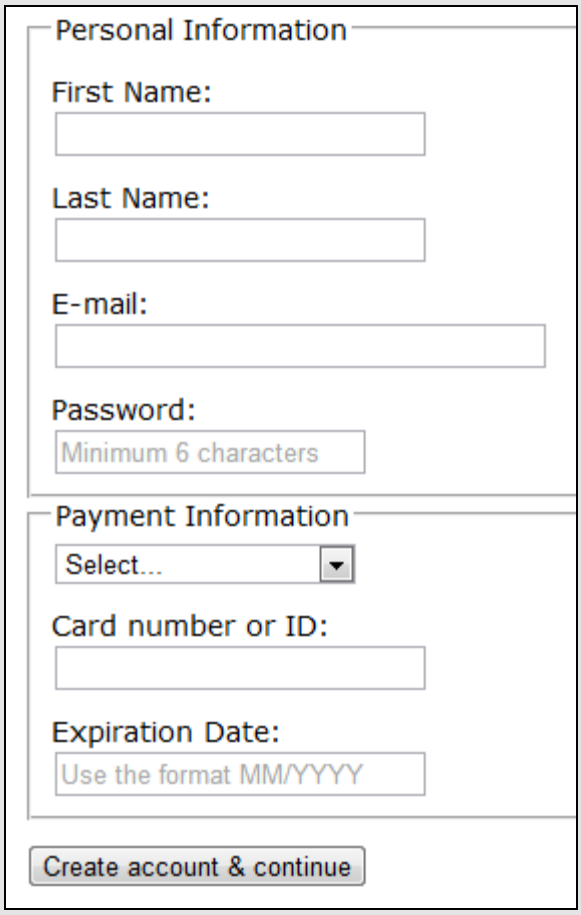
The files for this activity are in Student Data folder **Unit 2\Topic C**.

Here's how	Here's why
1 In Chrome, open order.html	(From the current topic folder.) You'll create two fieldsets on this page to logically organize the input fields.
Observe the form controls	There are some fields pertaining to user information and other fields pertaining to credit card information.
2 How would you separate these form fields into logical groups (fieldsets)?	
3 What do you think the legend text for these two fieldsets should be?	
4 Open order.html in Notepad	

5 Under the USER INFO comment, enter the following code:
 	
`<fieldset>`
`<legend>Personal Information</legend>`
	To start a fieldset and create a legend (text prompt) for it.

6 Scroll down to locate the PAYMENT INFO comment	
Directly above the PAYMENT INFO comment, close the fieldset	In the specified location, type `</fieldset>`.
7 Save and verify your changes	The input fields are grouped under the heading of "Personal Information." By default, browsers draw a thin border around a fieldset. You can customize this and other form styles with CSS.
8 Create a second fieldset for the payment fields, but do not include the button inputs	Close the fieldset element directly above the SUBMIT AND CLEAR comment.
Change the "Payment Information" heading to a legend	Change the `<h2>` tags to `<legend>` tags.

9 Save and verify your changes



The personal information and payment information fields are grouped separately, indicated by a default gray border around each group. You can control the appearance of these default borders by using CSS.

Customizing fieldset and legend styles

Explanation

As with most HTML elements, the default styles often won't be precisely what you're looking for. You might want different spacing, colors, fonts, or border styles, for example. You can customize the appearance of any element by using CSS. When it comes to fieldsets, commonly applied customizations include border styles and bold legend text. Find style combinations that enhance the usability of your forms, and be careful not to over-style.

Controlling fieldset width and spacing

Like other block elements, a fieldset will fill the entire available width of its parent element unless you specify a width for it. The width and height of a fieldset are indicated by the box drawn around it.

To control the width of a fieldset, you use the `width` property. For example, to set the width of all fieldset elements to 400 pixels, you would simply add the following rule to your style sheet:

```
fieldset { width: 400px; }
```

If you want to set the width for an individual fieldset, or for some but not all fieldsets in your site, you can use the `class` attribute in the fieldsets and then apply the style to that class name.

Other styles often applied to fieldsets include padding and margins. You can use the `padding` property to increase the amount of space between the fieldset borders and input fields, and you can use the `margin` property to create space between fieldset borders and adjacent elements.

Formatting legend text

By default, legends appear as normal text. But you'll probably want your legends to stand out like headings do, to provide context and an effective visual cue. For example, to make the text of all your fieldset legends 150% larger than the current or inherited font size and make them bold, you would write:

```
legend { font-size: 150%; font-weight:bold; }
```

Customizing fieldset borders

With CSS, you can add borders around any HTML element. Fieldsets automatically create a simple border, but you can change their default appearance in a variety of ways. Border styles consist of three properties, as described in the following table.

Property	Description
`border-color`	Sets the border color. You can use any valid color value.
`border-style`	Sets one of eight available styles: solid, double, dotted, dashed, ridge, groove, outset, and inset.
`border-width`	Sets the width of the border. You can use a pixel value or the keywords `thin`, `medium`, or `thick`.

You can combine all three of these properties into one single property by using the `border` shorthand property. For example, to create a solid gray border that's 4 pixels wide, you could write:

```
fieldset { border: 4px solid gray; }
```

The individual property names are implicit in their values, so it doesn't matter which order you specify the values in. In other words, you would get the same results as the preceding code if you wrote `border: solid gray 4px`.

Rounded corners

You can use the `border-radius` property to create rounded fieldset borders. This is a shorthand property that applies the value to all four sides of an element. For example, to soften the corner of your fieldset borders, you could write:

```
fieldset { border: 4px solid gray;
           border-radius: 4px; }
```

In addition to creating rounded border corners, you can use the `border-radius` property with a background color and no border styles to create the effect of a rounded background.

Be careful when styling form input fields

Form inputs and buttons are critical user interface components, so be careful not to over-style them. It's important that users can immediately recognize them for what they are. If the styles you apply make an input look less like an input, or a button look less like a button, you should dial it back.

Do it!

C-2: Formatting fieldsets and legends

The files for this activity are in Student Data folder **Unit 2\Topic C**.

Here's how	Here's why
1 In Notepad, open globalstyles.css	(From the styles folder, in the current topic folder.) You'll customize the fieldset and legend styles.
Under the FORM STYLES comment, add the following rule:	(At the bottom of the style sheet.) To make the legend text bold and 130% larger than its current size.
`legend { font-weight: bold; font-size: 130%; }`	
Save and verify your changes	The legend text is bold and larger.
2 Add the following rule:	To control the width of fieldset elements.
`fieldset { width: 400px; }`	
Save and verify your changes	Both fieldsets now have a fixed width of 400 pixels.

3 Add the following bold code to the fieldset rule:

To apply a custom border style to the fieldsets and add 20 pixels of space above the fieldsets.

```
fieldset { border: 2px solid #930; margin-top: 20px;
           width: 400px; }
```

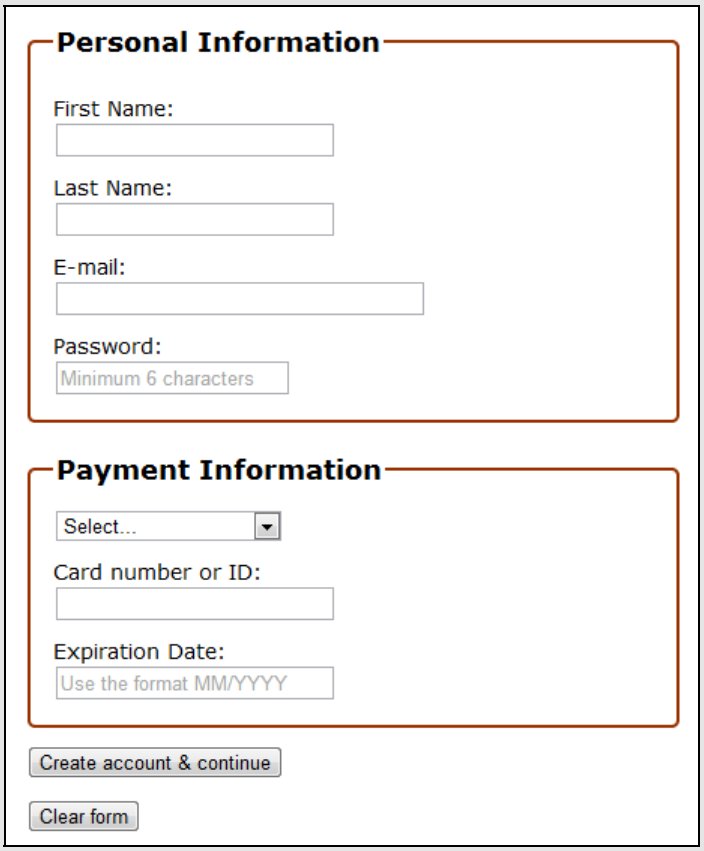
Save and verify your changes

4 Add the following bold code to the fieldset rule:

To add some space inside the fieldset and round the fieldset corners.

```
fieldset { border: 2px solid #930; margin-top: 20px;
           width: 400px;
           padding: 15px; border-radius: 5px; }
```

Save and verify your changes



These are just examples of the kind of style enhancements you'll probably need to make to get your forms looking optimal for your site.

5 Close all open files

Topic D: Form design considerations

Explanation

Forms are the bridge between an organization and its customers. Forms enable critical transactions and allow users to interact with a site in a variety of ways. Forms are a company's online cash register and its enrollment agent. They are arguably the most important pages on any given Web site, and yet too often, they are not well thought out or customer focused. Here, you'll learn some basic principles for designing effective forms.

Plan your form before you start building

Nobody enjoys filling out forms. Customers don't fill out forms to do you or your company a favor. They arrive at a form page with a goal in mind. They want something at the end of the transaction, so make it easy to get—whether it's a product, service, membership, or some form of participation. The last thing you want to do is to get in your customers' way when they're trying to complete a transaction.

Before you start building a form, think it through. A good plan will usually result in a good form. The process often starts with the database. It's important to structure your data simply and with your customers in mind so that when it's time to create the form, you don't force users to conform to unnecessary and rigid data requirements. For example, don't place limits on user names or require certain entries that aren't actually needed to complete the transaction. Instead of thinking of it as a form, think of it as a *transaction interface;* that's what it really is, and thinking of it as such might help you to raise the bar and focus on what's most important.

Don't ask for more than is necessary

If users wonder why they're being asked a particular question or why non-critical information is required, you have probably not adequately thought through your form's design. A surprisingly common mistake is to require membership or a subscription before or along with a purchase transaction. If you build your form this way, you'll probably lose a significant number of transactions because some people won't want to establish such a relationship; they want only what they came for. The transaction is the most important thing; you can always suggest or encourage membership after the transaction is complete.

When applicable, allow repeated user names

If registration is required or optional on your site, you probably want to do everything you can to facilitate loyalty. One simple thing you can do is avoid making user names a unique field in your database. This way, if Bob Smith wants to subscribe to your service as "Bob," you can let him be Bob and not force him to be Bob414 or some other user name he's likely to forget or not feel particularly tied to.

This concept applies to sites in which the displayed user name is private; this guideline does not apply to open forum sites or discussion boards where multiple people are collaborating and you need to prevent duplicate user names.

Questions to help you plan, design, and execute

When you're designing a form transaction interface and/or its corresponding database, consider the following questions to establish a foundation for an efficient and successful interface:

- **What is the users' objective in this transaction?** Align the interface with your customers' objective(s). If their objective is to buy a shirt, then make it as fast and easy as possible for them to get it.
- **What information is absolutely required to complete the transaction?** Omit anything that is not directly required for it. You can always gather more information when the transaction is complete.
- **What information can you ask for later, after the initial transaction is complete?** It's best to ask for information when the purpose of requesting that information is contextually obvious. In the rare case that it's not, briefly but clearly explain why you're asking for the information.
- **What is the most logical sequence for the input fields?** Order your input fields sensibly, and if you have to use several input fields, use fieldsets to group related fields into manageable chunks.
- **How can you optimize the usability of the interface?** To ensure a better user experience, you can use several techniques, including using placeholder text, using explicit labels to make the text prompts clickable, and taking advantage of some of the new input types in HTML5.
- **What reasonable data limits can you set to ensure data integrity but not inundate customers with rules?** This goes back to the database issue. Don't force customers to adhere to data limitations if those limitations detract from the simplicity and ease with which transactions can be completed.
- **How can you make the process of validating customer data as fast and efficient as possible?** Consider employing real-time validation. This way, users will see if they've made an error as soon as they click away from an input field. This is better than having them complete the form, submit it, scan for error messages, and then re-submit the form—often multiple times. This scenario can lead some frustrated users to abandon the registration process. Real-time validation also allows you to provide positive feedback as each entry is validated. This feedback gives users confidence as they get closer to finishing the transaction.

Do it!

D-1: Identifying best practices in form design

Questions and answers

1 What are some annoyances or hindrances that you have encountered on Web forms?

2 What techniques would you use to resolve these problems?

3 How can using the new input types of HTML5 help to make a user's experience with your form more successful?

4 Why is it critical to consider the user's objective when you're designing a form?

5 Why is it important not to ask for more information than is necessary at a given point in time?

6 Why is it important to logically sequence and group your input fields?

Unit summary: HTML5 forms

Topic A

In this topic, you learned how to create forms, add a variety of **input fields**, create labels to improve form usability, set input properties, control autocomplete and autofocus, create placeholders, define option groups, create a data list, and enable spelling checking in form input fields.

Topic B

In this topic, you learned about **browser-based validation**. You learned how to create required fields, and how to disable browser-based validation in case you need to use your own script-based data validation solutions. You also learned how to use regular expressions with the `pattern` attribute to validate user data entries.

Topic C

In this topic, you learned how to create **fieldsets** to group your form input fields into logical sections, making it easier for users to understand and interact with your forms. You also learned how to create **legends** and how to use CSS to customize the appearance of fieldsets and legends.

Topic D

In this topic, you learned some **best practices** in form design that will help you build effective transaction interfaces.

Review questions

1 True or false? Form inputs and controls must be placed inside a `<form>` container.

2 The value of a label's `for` attribute must match which of the following?

   A The value of the related input field's `name` attribute

   B The value of the related input field's `id` attribute

   C The value of the related input field's `type` attribute

   D The value of the related input field's `value` attribute

3 True or false? Another way to associate a label with an input field is to place the input inside a `<label>` element; no `for` attribute is required.

4 True or false? When your forms use labels, users can select an input field by clicking the text label; they don't have to click the input field itself.

5 Advantages of using the `email` input type to collect e-mail addresses include: [Choose all that apply.]

   A It simplifies the task of validating the user's input.

   B The touchscreen keyboards of mobile devices like the iPhone will be automatically optimized for entering an e-mail address.

   C Browsers that don't support it will default to the `text` input type, so there's no risk in using it.

   D E-mail input fields provide default placeholder text.

6 True or false? The `autofocus` attribute can be used as many times as necessary in a form.

7 True or false? It's possible to have autocomplete "on" for a form but "off" for one or more input fields within the form.

8 Which of the following expressions would set an input field to accept only numbers between 0 and 9, with a minimum of two digits and a maximum of five digits?

A `pattern="[0-9](2,5)"`

B `pattern="[0-9]{2,5}"`

C `pattern="[0-9{2,5}]"`

D `pattern="[0-9]{2|5} "`

Independent practice activity

In this activity, you will add elements to a form, set input field properties, create a custom validation rule, group input fields, and apply styles.

The files for this activity are in Student Data folder **Unit 2\Unit summary**.

1 Open register.html in both Notepad and Chrome. This page has an empty form container. You'll add input fields and properties.

2 In the empty form container, insert a text input field with the text label **User Name:** and make it a required field.

3 Add an e-mail input field with a label that reads **E-mail Address:** and make it a required field.

4 Add a password input field with a label that reads **Password:** and make it a required field.

5 Create a submit button that reads **Submit info**. Insert a line break before the submit button.

6 Set the User Name field to receive focus when the page loads.

7 Save your changes and compare your results to Exhibit 2-18. Make changes if necessary.

8 Wrap the three input fields in a fieldset with the legend **Account Info**.

9 For the password input field, add a placeholder that reads **Minimum 5 characters**.

10 In the page's embedded style sheet, create a rule that applies a **10-pixel border radius** and **10 pixels of padding** to the fieldset element. (**Note:** The page already contains a style that makes label elements block-level elements.)

11 For the legend element, create a rule that makes the font size **150%** larger than its current size.

12 In the existing rule for the label element, add new styles that give each label a **10-pixel top margin** and makes all label text bold. (*Hint:* Use the `margin-top` and `font-weight` properties.)

13 Save your changes and compare your results to Exhibit 2-19. Make changes if necessary.

14 Verify that you cannot submit the form without entering data into all three fields.

15 Try to reproduce the style changes shown in Exhibit 2-20.

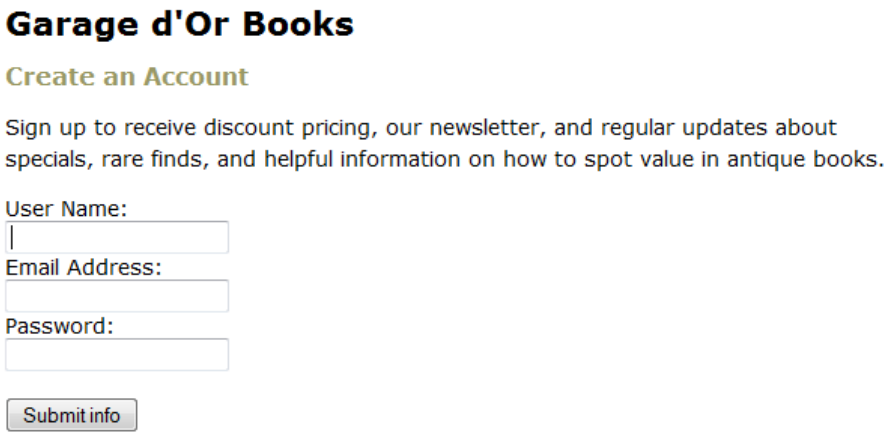


Exhibit 2-18: The form after Step 7

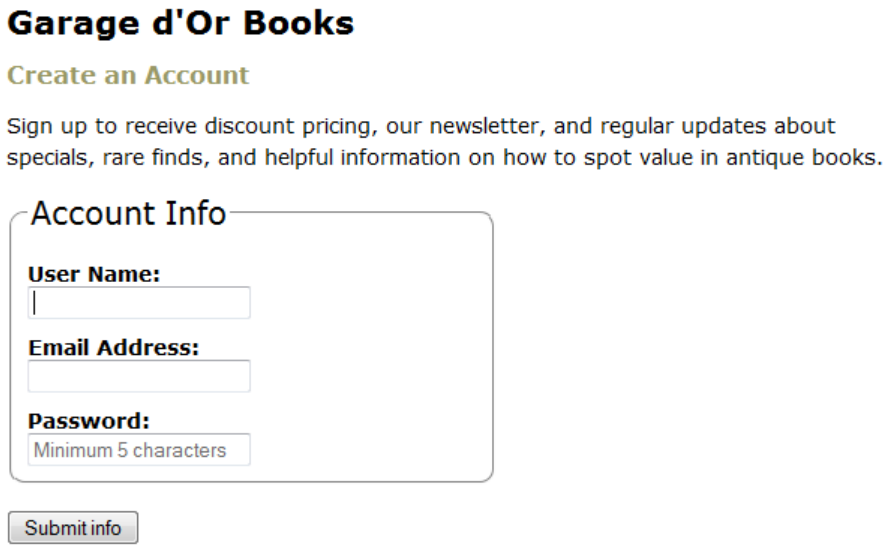


Exhibit 2-19: Styles for the Account Info fieldset

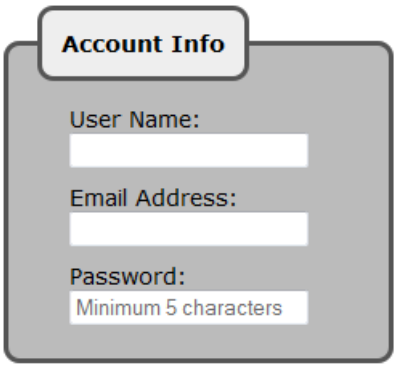


Exhibit 2-20: Alternate styles for the Account info fieldset

Unit 3

Audio and video

Complete this unit, and you'll know how to:

A Embed native audio that works in all HTML5-supporting browsers, enable user controls, and set audio properties.

B Embed native video that works in all HTML5 browsers, set a poster image, and embed videos hosted externally.

Topic A: Embedding native audio

Explanation

HTML5 has changed the landscape for embedded media by allowing developers and users to leave third-party browser plug-ins behind in favor of a simpler, easier, and more streamlined solution: audio and video playback capability built directly into the browser. Support for native audio and video is widespread; all you need is a few simple techniques and a couple format variations for your media files.

The <audio> and <source> elements

To embed native audio, all you need is the new and aptly named `<audio>` element combined with the `<source>` element. The `<audio>` element creates the audio container and activates the browser's native audio player. You use the `<source>` element to specify the URL or path to your audio file.

At its most basic, here's how the code looks:

```
<audio controls>
 <source type="audio/mp4" src="media/myAudio.m4a" />
 <source type="audio/ogg" src="media/myAudio.oga" />
</audio>
```

Audio controls

You need to use the `controls` attribute to make your audio a displayed component on the page. If you leave the `controls` out of your `<audio>` element, users won't see anything on the page associated with the audio file, and they won't have the ability to play or stop the audio. So if you're embedding HTML5 audio, the `controls` attribute is a must.

The `controls` attribute is another Boolean attribute, implying either a true or false condition. The attribute is in a true state if it's present, and a false state if it's not present (actual "true" and "false" values are not valid). In the previous example, the attribute does not have a corresponding value; this is valid HTML5 for Boolean attributes. However, if you want to write with XHTML authoring standards, you should include a value that matches the attribute name, as follows:

```
<audio controls="controls">
```

With controls enabled, users can control volume, pause and resume playback, and drag the playhead to a specific part of the audio clip. Native controls look slightly different depending on the browser, but they can be customized with JavaScript and CSS.

Exhibit 3-1: Firefox's native audio player

Specifying the audio content type

In each `<source>` element, the `type` attribute tells the browser what the media type is. It's important that you *always* use the `type` attribute because without it, browsers have to download the file just to determine whether it's supported, and that is a waste of bandwidth. It's also critical that your `type` values don't contain misspellings or inaccuracies; if they do, browsers might not be able to play the audio.

The src attribute

Also in each `<source>` element, the `src` attribute is required to provide the URL or path to the audio file. To allow as wide an audience as possible to access your audio, you'll need to include multiple versions of your audio files because HTML5 browsers support some audio formats but not others.

Audio formats

Before you get started with embedding audio and applying user controls, it will help to identify the various audio formats and the browsers that support them. Browser support for the following audio file types is likely to evolve over time, so it's important to stay informed of changes in the browser support landscape and thoroughly test your audio in multiple browsers and platforms. The following table is not a complete list of audio formats on the Web, but includes the formats most commonly used for HTML5 audio.

File type/ Extension	Description
OGG/.oga	The "Ogg Vorbis" audio format (.ogg or .oga) is an open-source audio format supported by FireFox, Chrome, and Opera. However, Safari and Internet Explorer do not currently support it, and users of those browsers represent a significant portion of your Internet audience.
MP3/.mp3	The MP3 audio format is widely supported; however, depending on what you do with your .mp3 audio, it's important to note that royalty licensing might come into play.
AAC/.m4a	AAC stands for Advanced Audio Coding. This audio format (.m4a) offers better audio quality than does MP3. AAC is also referred to as "MP4 audio" because the audio data is embedded in an MP4 container, hence the .m4a file extension. AAC does not have the restrictive licensing hindrances that come with the MP3 format. AAC is supported by Internet Explorer, Chrome, Safari, and Apple's iOS devices, like the iPhone and iPad. FireFox and Opera do not currently support it, but that might change in time.
WAV/.wav	The "wave" (.wav) audio format does not compress the audio data, so the quality is good but file sizes tend to be large. As a result, this file type is usually not the best option.

Audio format summary

Different browsers support different formats for HTML5 audio, just as in the old days of Web development, when there was always a new workaround required to get browsers to play nicely with your content and design. Fortunately, the bottom line is pretty simple:

- Use the AAC (.m4a) format to reach users with Internet Explorer, Chrome, Safari, and Apple's iOS devices, like the iPhone and iPad. If you're developing only for iOS devices, you need to deliver only AAC.
- Use the Ogg (.ogg or .oga) format to reach FireFox and Opera users.
- Know your audience and provide content that works for that audience. To reach a broad Internet audience, you need to minimally provide content that can be viewed by the most popular browsers: Safari, Chrome, Firefox, Internet Explorer, and the cross-platform versions of those browsers.

There are several free audio/video conversion tools that you can download and use to convert your audio files to the specific formats you need.

Optimizing the sequence of your audio types

The purpose of listing variants of your media files by using multiple `<source>` elements is to allow your content to reach as wide an Internet audience as possible. A browser that doesn't support the first media type will proceed to the next one, then the next one, and so on until it encounters a media type that it can play.

Some versions of iOS (the iPhone and iPad platform) will not play anything if the first media source that iOS encounters is unsupported. This will probably become less of a concern over time as older versions of the platform are fully phased out, but to be safe, you might consider first listing a media type that's supported by iOS devices. After that, you can determine your own criteria for the best order of media types. So, using the source list order shown here can help ensure that all HTML5 browsers can play your audio:

```
<audio controls>
 <source type="audio/mp4" src="media/myAudio.m4a" />
 <source type="audio/ogg" src="media/myAudio.oga" />
</audio>
```

Using this example, browsers and devices that support the ACC/MP4 format will play the first audio source, and the browsers that do not support that format (FireFox and Opera at the time of this writing) will skip over the first source and play the second source, the Ogg file.

Fallback text

In addition to providing at least two variants of your audio, you should always include fallback text for users who do not have a browser that supports the `<audio>` element. It's easy to do. At the end of your source list, include a paragraph with a brief message stating that the user's browser does not support native audio. You might even want to provide a link so users can download the audio or provide a link suggesting that they upgrade their browser.

```
<audio>
  <source type="audio/mp4" src="media/myAudio.m4a" />
  <source type="audio/ogg" src="media/myAaudio.oga" />
  <p>Your browser does not support native audio. Consider
     upgrading to an HTML5 browser. </p>
</audio>
```

Browsers that don't support the `<audio>` element will display this text instead. This fallback text should always be the *last* item in your `<audio>` container.

Do it!

A-1: Applying native audio and enabling user controls

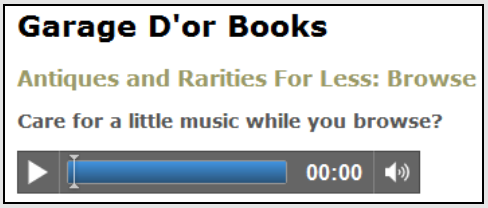
The files for this activity are in Student Data folder **Unit 3\Topic A**.

Here's how	Here's why
1 In Chrome, open titles.html	(From the current topic folder.) You'll add audio to this basic page.
2 In Firefox, open titles.html	You'll observe HTML5 audio in both browsers.
3 In Notepad, open titles.html	

4 Under the AUDIO comment, type the following:
 	
`<audio>`
`  <source type="audio/mp4" src="media/piano.m4a" />`
`</audio>`
 	
Save your changes	

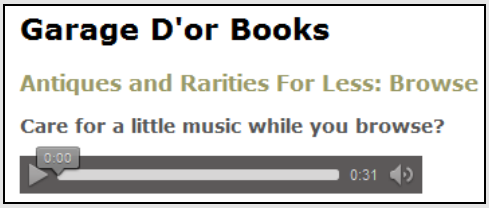
5 Reload the page in Chrome	There's no visible change on the page because the native player isn't set to show its controls.

6 Add the following bold code:
 	
`<audio **controls**>`
`  <source type="audio/mp4" src="media/piano.m4a" />`
`</audio>`
	To enable the audio controls.
	
	In HTML5, Boolean attributes don't require a value. If you want to use XHTML, specify "controls" as the value of the attribute.

7 Save your changes and reload the page in Chrome	 With the controls enabled, the user can see the native audio player and control the audio.
8 Reload the page in Firefox	Firefox can't load the audio because it does not currently support the MP4 format in the `<audio>` element. (This will likely change at some point in the future.)

9 Add the following second source to the audio:
 	
`<audio controls>`
`  <source type="audio/mp4" src="media/piano.m4a" />`
`  **<source type="audio/ogg" src="media/piano.oga" />**`
`</audio>`	

10 Save your changes and reload the page in Firefox



The Firefox player looks slightly different from the Chrome player. It also shows the total length of the audio, in seconds.

In either browser, play the audio

(Click the triangle.) The audio plays and the native player shows the duration of the song in seconds.

11 Add the following fallback text to the audio code:

```
<audio controls>
 <source type="audio/mp4" src="media/piano.m4a" />
 <source type="audio/ogg" src="media/piano.oga" />
 <p>Your browser does not support native audio. Consider
   upgrading to an HTML5 browser.</p>
</audio>
```

12 Why is it important to include fallback text?

13 Why is it important that the MP4 audio is first in the source list?

Audio attributes

Explanation

There are several other attributes you can use to customize the controls and audio experience: `autoplay`, `loop`, and `preload`. The `autoplay` and `loop` attributes are Boolean; their presence sets a "true" state, and their absence creates a false state.

Autoplay

When the `autoplay` attribute is present, the media file plays as soon as it loads. It's usually not a good idea to set your media files to play automatically—most people simply don't like it. However, there are occasions when it makes sense. If the page that contains the audio is accessible only via links that clearly indicate what to expect, then using autoplay makes sense because the user will have already requested the audio.

Looping audio

When the `loop` attribute is present, the media file will automatically start again from the beginning after it reaches the end and it will loop continuously. Generally, there aren't many good use cases for looping an audio track, but it could be useful if your intent is to provide background music or create an atmosphere with sound effects.

Preloading audio

The `preload` attribute is intended to give browsers a "hint" as to what you, the author, think will create the optimal user experience for a particular media file. The attribute accepts three values: `none`, `metadata`, and `auto`. If you specify `none`, the media file will not preload when the page loads. If you specify `metadata`, only the file's metadata will preload. If you specify `auto`, you leave it up to the browser to decide what to do.

Do it!

A-2: Setting audio attributes

Here's how	Here's why
1 Add the following bold code:	To set the audio to play continuously when the page loads.

```
<audio autoplay loop controls>
 <source type="audio/mp4" src="media/piano.m4a" />
 <source type="audio/ogg" src="media/piano.oga" />
 <p>Your browser does not support native audio. Consider
   upgrading to an HTML5 browser.</p>
</audio>
```

2 Save your changes and reload the page in Chrome	The audio plays as soon as the page loads, and it loops continuously when it reaches the end.
Stop the audio	
3 Reset the audio so it does not play automatically and does not loop	
Save your changes and verify the result in Chrome	
4 Why is it often *not* a good idea to set an audio file to autoplay?	
5 Describe a situation in which it would be reasonable to set an audio file to autoplay.	
6 Describe a reasonable use case for looping an audio track.	
7 If applicable, describe a use case for autoplay in the context of your own site or your organization's site.	
8 Close titles.html	In your text editor.

Topic B: Embedding native video

Explanation

Embedding native video is similar to embedding native audio. HTML5 browsers have their own built-in video players, precluding the need for plug-ins like Flash and Silverlight. Native video uses fewer processor resources, which is especially important on mobile devices which run on batteries. As with HTML5 audio, creating at least two variants of your video is important. All HTML5 browsers support the markup around native video, but not all of them support every video file format. Another benefit of native video is that you can modify its functionality and appearance with JavaScript and CSS.

The <video> and <source> elements

To embed native video, you use the `<video>` element along with multiple `<source>` elements. The `<video>` element creates the video container and activates the browser's native video player. The `<source>` element specifies the URL or path to your video.

For example:

```
<video controls>
 <source type="video/mp4" src="media/myVideo.mp4" />
 <source type="video/ogg" src="media/myVideo.ogv" />
</video>
```

Video formats

Before you get started with embedding video, it will help to identify the various video formats and the browsers that support them. Media files typically have three components: separate audio and video data compressed in specific formats called *codecs*, and the container format that binds them together. For example, "Ogg" is a media container format, "Theora" is the name of the Ogg video codec, and "Vorbis" is the name of the Ogg audio codec.

Browser support for the following video file types will likely evolve over time, so it's important to stay informed of changes in the browser landscape and thoroughly test your media in multiple browsers and platforms. There are many video file formats, some old and some fairly new. The following table lists three video formats that are commonly used with the `<video>` element.

File type/ Extension	Description
Ogg/.ogv	The "OGG Theora" video format (.ogg or .ogv) is an open-source video format supported by Firefox, Chrome, and Opera. However, Safari and Internet Explorer do not currently support it—and users of those browsers represent a significant portion of your Internet audience.
MP4/.mp4	The MP4 video format consists of the MPEG-4 container format, which holds video data in the H.264 codec and holds audio data in AAC format. The MP4 format is supported by Internet Explorer, Chrome, Safari, and iOS devices (mobile Safari). Support for the H.264 codec is likely to expand over time, due in part to its high quality and efficiency on mobile devices.
WebM/.webm	The WebM container format is royalty-free and high quality. It contains the VP8 video codec and the Ogg Vorbis audio codec. It's supported by Firefox, Chrome, and Opera, but at the time of this writing, Safari and Internet Explorer do not support it.

MP4/H.264 is dominant on popular mobile devices

Because the H.264 codec is high quality, runs efficiently on mobile devices, and plays a dominant role on Apple devices, support for it will probably expand, perhaps at the expense of some other perfectly viable open-source formats, such as Ogg and Webm.

Also, in early versions of iOS, mobile Safari will not play anything if it can't play the first media source it encounters. This will become less of a concern over time as older versions of the platform are fully phased out, but to be safe, make the MP4/H.264 format the first in your video source list.

Video format summary

To get your HTML5 video to play in all modern browsers:

- Start your video source list with the MP4 format. This code will reach users with Internet Explorer, Chrome, Safari, and Apple's iOS devices, like the iPhone and iPad (mobile Safari). If you're developing content that will be delivered only to iOS devices, you need to use only the MP4 format.
- Use the OGG (.ogg or .ogv) or Webm (.webm.) formats, both of which are open source, to reach Firefox and Opera users. At the time of this writing, Firefox and Opera do not support the H.264 codec, but that will probably change.
- Know your audience and test your pages to verify that you're providing content that works for that audience. To reach a broad Internet audience, you need to minimally provide content that can be viewed by the most popular browsers: Safari, Chrome, Firefox, Internet Explorer, Opera, and the cross-platform versions of those browsers.

There are several free video conversion tools that you can download and use to convert your video files to the specific formats you need.

A sample source order

After listing MP4 first, determine your own criteria for the best order of video types. For example, your source list might look something like this:

```
<video controls>
 <source type="video/mp4" src="media/myVideo.m4a" />
 <source type="video/webm" src="media/myVideo.webm" />
 <source type="video/ogg" src="media/myVideo.oga" />
 <p>Sorry, your browser is...on the old side. You can
 <a href="/media.myVideo.mp4">download the video here</a>.</p>
</video>
```

Webm provides better video quality than Ogg does, so depending on the nature of your video, you might want to list MP4 first, then Webm, and then Ogg. If the quality difference between your Webm version and your Ogg version is imperceptible, you might consider just using the Ogg format to reach Firefox and Opera users.

Another option is to use client-side or server-side detection to determine which format is supported by the visiting browser and then serve the right format. Applying this technique is beyond the scope of this course.

Specifying a single media source

If all major browsers one day support the same media formats, then you won't have to use separate `<source>` elements. You'll be able to put the `src` attribute directly in the `<video>` element, as follows:

```
<video src="media/myVideo.mp4" controls>
<p>Sorry, your browser is...on the old side. You can
<a href="/media.myVideo.mp4">download the video here</a>.</p>
</video>
```

Providing a Flash fallback

In place of or in addition to your plain-text fallback, you can paste in any legacy Flash-embed code that you might have, so that users with older browsers can still access a version of your video rather than seeing plain text and a link.

Video attributes

The same attributes that apply to native audio, such as the `controls`, `loop`, `autoplay`, `preload`, and `src` attributes, apply to native video. There are additional attributes specific to video, including `height`, `width`, and `poster`.

The src attribute must point to a video file

When you embed video, the `src` attribute must point to an actual video file. If you specify a path that does not end with an actual video file, such as a link to a YouTube video that ends with an alphanumeric ID assigned by YouTube, then your video won't work.

Video controls

As with native audio, you need to use the `controls` attribute to allow users to see the browser's native video controls. If you don't include a `controls` attribute in your `<video>` element, users will see that there's something there, but they won't be able to play the video. So if you're embedding HTML5 video, the `controls` attribute is a must. As with other Boolean attributes in HTML5, it's in a true state if it's present and a false state if it's not present.

With controls enabled, users can control volume, pause and resume playback, and drag the playhead to a specific part of the video clip. Controls look slightly different depending on the browser, but they can be customized with JavaScript and CSS.

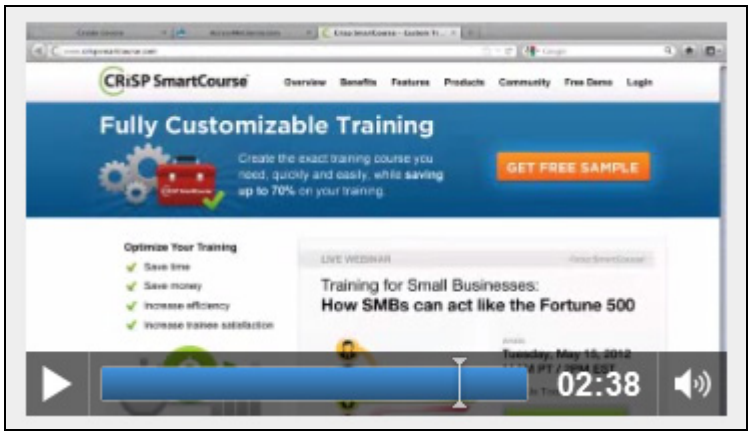


Exhibit 3-2: Chrome's native video player, toward the end of a video clip

Specifying the video content type

As with audio, each `<source>` element for your video should have a `type` attribute to tell the browser what format each video source is encoded with. Without the `type` attribute in each `<source>` element, browsers have to download a portion of the file to determine whether it's supported, and that is a waste of bandwidth. Be sure your `type` values don't contain typos or other errors; if they do, browsers might not be able to play the file. The `type` attribute does not apply to the `<video>` element.

Height and width

You specify the dimensions of your video by using the `height` and `width` attributes. These attributes should be familiar to you; they're the same ones you use to set image dimensions. For example:

```
<video controls height="360" width="640">
 <source type="video/mp4" src="media/myVideo.m4a" />
 <source type="video/webm" src="media/myVideo.webm" />
 <p>Sorry, your browser is...on the old side. You can
 <a href="/media.myVideo.mp4">download the video here</a>.</p>
</video>
```

The `height` attribute sets the height of the video element, and the `width` attribute sets the width of the video element—no surprises there. Values are based on pixels. If you set one value but not the other, the browser uses the specified value to calculate the height or width required to preserve the video's original aspect ratio.

Even if you specify height and width values that are not consistent with the video's aspect ratio, the video will not be crunched down or distorted larger, as an image file would be. Instead, HTML5 browsers always maintain the file's aspect ratio.

A note about iOS devices (the mobile Safari browser)

When videos are played on iOS devices like the iPhone and iPad, automatic playback is disabled—even if you use the `autoplay` attribute. Also, on the iPhone, videos always play full screen.

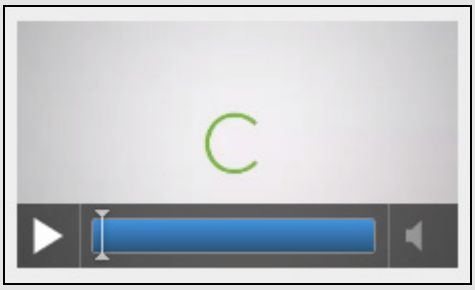
Do it!

B-1: Applying native video and setting properties

The files for this activity are in Student Data folder **Unit 3\Topic B**.

Here's how	Here's why
1 In Firefox and Chrome, open index.html	From the current topic folder.
In your text editor, open index.html	
2 Under the AD BAR comment, type the following:	(Type it exactly as shown.) To create a new section for advertising that contains a video.

`<section id="adRail">`
`<h1>Advertisements</h1>`
`<video controls>`
`<source type="video/ogg" src="media/smartcourse.ogv" />`
`</video>`
`</section>`	

	It's important to use the elements and `id` name shown because there are styles in the style sheet that are attached to these elements.
3 Save your changes and reload the page in Chrome	No height or width settings are specified yet, so the browser displays the video at its actual size.
4 Add the following bold code:	To set the width of the video.
`<video controls` **`width="220"`**`>`	
Save your changes and reload the page in Chrome	
	With the width specified, the browser uses the video's actual height/width ratio to establish the height needed to preserve the video's original aspect ratio.

5 Reload the page in Firefox

Firefox displays the same video in its native player. The controls look different, and by default Firefox displays the duration of the video in minutes and seconds.

6 Play the video for five seconds and then pause it

In either browser.

7 Add the following bold code to the video container

To include an MP4 version of the video as the first file in the source list.

```
<section id="adRail">
 <h1>Advertisements</h1>
 <video controls width="220">
  <source type="video/mp4" src="media/smartcourse.mp4 />
  <source type="video/ogg" src="media/smartcourse.ogv />
 </video>
</section>
```

8 Save your changes

9 In the current topic folder, open the media folder

There's only one version of the video, in .ogv format. You'll observe what happens when a browser tries to load a non-existent or unsupported video.

Close the media folder

10 If there were an MP4 version of the video in the media folder, what would be achieved by including this additional source?

11 Why should you put an MP4 version first in the source list?

12 Reload the page in Firefox	There is no visible change.
Play the video	Even though the MP4 is first in the list and Firefox currently doesn't support that format in the `<video>` element, Firefox recognizes it as a non-supported format and moves to the next source in the list, which it plays without delay.
Pause the video after a few seconds	
13 Reload the page in Chrome	
Play the video	If there were an MP4 version of the video in the media folder, Chrome would play it because it appears first in the list and Chrome supports that format. But because there's no file by that name, Chrome skips that source and moves to the Ogg video.
Pause the video after a few seconds	
14 Create a text fallback that includes a link to download the video	After the last `<source>` element in the `<audio>` container, insert a paragraph with a message stating that the user's browser does not support HTML5 video, and provide a link to the video file.
15 What other fallback options might be valuable when you're using HTML5 video?	

Preload options

Explanation

You can control how your videos are preloaded, which is important for a variety of contexts. For example, if you're displaying a video advertisement, you probably don't want to load all the video data when most users are not likely to play it. That would be a waste of bandwidth.

It would make more sense to wait until the user requests the video and then download the video data. However, if the video is a core content component of your page, then it's typically best to preload the video data so that the user doesn't have to wait for it and therefore has an optimal experience.

The preload attribute

The `preload` attribute is intended to give browsers a "hint" as to what you, the site author, think will create the optimal user experience for a particular media file. The attribute accepts three values: `none`, `metadata`, and `auto`.

The `preload="none"` value suggests to the browser that users might or might not play the media file; in other words, the file is not necessarily relevant to the page content, and therefore the browser should not preload any of its data to avoid wasting bandwidth. The previously mentioned advertising-video scenario is a use case for this option, as long as an image poster is used or some other form of advertising copy is visible.

The `preload="metadata"` value suggests to the browser that users might or might not play the media file, but downloading just the file's metadata (such as the duration of the file and the first frame of the video) would be useful.

The `preload="auto"` value lets the browser decide whether to preload some or all of the media file. Because this code creates a browser-specific action, results will vary. And a single browser type might take different actions depending on certain variables, such as network availability and connection speed or device settings that might prevent the preloading of media.

Similarly, if the `preload` attribute is not present, it's up to the browser to determine what to do. The HTML5 specification suggests that `metadata` is the optimal default behavior when the attribute is omitted because it minimizes bandwidth usage while providing users with some information about the file.

Autoplaying video

As with audio, when the `autoplay` attribute is present, the video file begins playing as soon as the page loads. It's often not a good idea to set your media files to play automatically, but there are situations when doing so makes sense. For example, if you provide a link to a video, you could set the video on the destination page to autoplay because the user will have specifically requested it. The popular site YouTube works this way: when you click a link for a particular video, that action serves as the request to play the video.

Creating a poster image

Depending on the preload settings, browsers might display the first frame of your video to provide some indication of the video content. However, the first frame of your video might not be the best choice for a starting image. In addition, your video's first frame might be empty or it might contain only a color or something else that isn't a good representation of the video content.

With the `poster` attribute, you can specify the path to an image file to serve as the initial image that people see before they choose to play the video. This poster image, which is often called a *thumbnail image*, will be displayed if no video data is available or while the video data loads. This attribute allows you to choose the best representative image for your video, and this image plays a critical role in getting people to watch your video.

To specify a poster image, add the `poster` attribute to the `<video>` element and specify a valid path to the file. For example:

```
<video poster="images/myPoster.jpg" controls>
```

Server configuration for various media

You or your server administrator might need to ensure that the servers responsible for delivering your content are configured to deliver all the different media types you intend to use. The *Internet media type* (also called the *content-type* or *MIME type*) defines the formats in use in your site.

In many cases, this server configuration involves modifying the .htaccess file (.htaccess stands for *hypertext access*). This is a configuration file that's used on many Web servers. All you need to do is open the .htaccess file in a simple text editor and add entries for each media type you intend to serve. The following is a sample list of media types and the syntax used to configure the .htaccess file:

```
AddType audio/mp4 .m4a
AddType video/mp4 .mp4
AddType audio/ogg .ogg .oga
AddType video/ogg .ogg .ogv
AddType audio/webm .webm
AddType video/webm .webm
```

In this example, the Ogg format is included along with its more specific audio (.oga) and video (.ogv) types because the .ogg file extension is also a valid type for both.

Do it!

B-2: Controlling data preload and creating a poster image

Here's how	Here's why
1 Add the following bold code to the `<video>` element: `<video **preload="none"** controls width="220">`	To disable the preloading of any data associated with the video file.
2 Save your changes and reload the page in Chrome	All that appears now are the browser's native controls. The video still occupies the same space, but there's nothing to indicate its content.
Reload the page in Firefox	 In Firefox, the video player appears but no information associated with the video is displayed. Firefox shows a gray background to indicate the dimensions of the video.
3 Describe a situation in which you might want to use `preload="none"`.	
4 Play the video	The video data loads and the video begins playing normally.
Pause the video	
Reload the page	To return the video player to a zero-data state.

5 Change the `preload` value as follows:

To preload only the video file's metadata—for example, its duration and the first frame.

```
<video preload="metadata" controls width="220">
```

Save your changes and reload the page in Firefox

Again, Firefox shows the video's first frame and duration, the same result you get by omitting the `preload` attribute. This occurs because loading metadata is the default setting used by most browsers. This might change over time, so it's best to declare your preference by using the `preload` attribute.

6 Change the `preload` value as follows:

To let the browser determine what data should be preloaded.

```
<video preload="auto" controls width="220">
```

Save your changes and reload the page

Both Chrome and Firefox display the file's metadata when `auto` is used, so there's no visible change. This result might vary depending on the circumstances, such as device settings and network detection.

7 Is the first frame of this video effective in terms of providing a thumbnail that gives users an indication of the video's content?

8 Add the following bold code:

```
<video poster="media/smartcourse.gif" preload="auto"
controls width="220">
```

Save your changes and reload the page in Chrome

Now the video's initial state shows an image captured from the video, showing the logo and URL for the site. This makes for a much more effective and useful video thumbnail.

9 Why is it important to specify a poster image for your videos?

10 Which preload option would make the most sense in the context of how you use video in your own sites or your organization's sites?

11 Describe a situation in which it makes sense to set a video to autoplay.

Embedding YouTube video

Explanation

There's more than one way to embed video in your site. You've learned how to embed native video by using the `<video>` element; next, you'll learn how to embed videos by using popular video hosting sites like YouTube.

Using resources like YouTube to host videos is a popular choice because it's free, it doesn't use up any of your organization's server bandwidth, and the service is as reliable as it gets in terms of delivery speed and minimal downtime. Using free video hosting sites also opens up your videos to a wide Internet audience via search, random browsing, and social network sharing.

The <iframe> element

Inline frames allow you to insert other documents and resources directly into a page and control its location on the page just like any other page element. Inline frames are used in a variety of ways, including displaying advertising content from external sources and displaying hosted video.

To create an inline frame, you use the `<iframe>` element and specify its height, width, and the path to the document or resource. For example:

```
<iframe src="URL-to-document" height="200"
width="400"></iframe>
```

Inline frames require a closing tag even though the element itself does not contain other elements or content. In addition to the inline frame code, you need the secondary document or resource to display inside the inline frame. This nested document will appear in the "window" that is created by the `<iframe>` element.

The seamless attribute

By default, most browsers draw a border around an inline frame. You can remove this border in two ways, but sometimes using both is the best solution. First, you can use the `seamless` attribute.

The `seamless` attribute is a Boolean attribute that's intended to make an inline frame appear as part of the containing page rather than as a separate content source within the page, to "seamlessly" blend in with the document. In other words, it prevents borders and scrollbars from appearing around the inline frame.

Using CSS to disable default iframe borders

At the time of this writing, the `seamless` attribute is not well supported. However, you can achieve the same effect by using CSS. For example, to turn off the default border around inline frames, you can add this rule in your style sheet:

```
iframe { border: none; }
```

If this style is added to your site-wide style sheet, it will turn off the default borders for all `<iframe>` elements in your site. If you intend to use the `seamless` attribute for forward-compatibility, you can specifically target only those inline frames that contain a `seamless` attribute. You do this by using an attribute selector. For example:

```
iframe[seamless] { border: none; }
```

Attribute selectors use brackets after the element name to select only those elements that contain the attribute indicated within the brackets. This rule selects only those `<iframe>` elements that contain the `seamless` attribute, thereby reinforcing the intent of the `seamless` attribute and ensuring that all browsers will display inline frames without any default border.

Do it!

B-3: Embedding a video hosted on YouTube

The files for this activity are in Student Data folder **Unit 3\Topic B**.

Here's how	Here's why
1 Inside the adRail section, below the closing video tag, type: `<iframe src="" width="220" height="125"></iframe>`	You'll paste the URL for the video in Step 3.
2 In the current topic folder, open YouTube link.txt Select and copy the entire URL	This is a link to a YouTube-hosted copy of the same video.
3 Paste the URL between the `src` quotation marks Save your changes and reload the page in Chrome	 The hosted video is displayed in the `<iframe>` element. The YouTube service also overlays your text title and a Play button.
4 Observe the default inset border	Most browsers draw some kind of border around an inline frame; these default border styles might change over time. You can turn off default borders by using two methods.
5 Open the styles folder Open styles.css	In the current topic folder. You'll create a rule that turns off the default border drawn around inline frames.
6 Create the following rule: `iframe { border: none; }` Save the style sheet and reload the page in Chrome Reload the page in Firefox	To disable borders for any inline frame. The default border disappears. To verify that you get the same result in Firefox.

7 Observe the video's initial screen	When you embed a hosted video with an inline frame, you can't directly specify a poster image. When you upload a video to YouTube, the service creates three thumbnail images that you can choose to establish the initial state of your video. These generated options might not be what you would have created, but at least you have options.
Play the video for a few seconds and then pause it	YouTube displays its own controls and options, which include the duration of the video, the elapsed time of the video, an option to watch it on YouTube, and an option to watch in full-screen mode. YouTube also allows you to click the video area to play and pause playback. YouTube will also provide the `<iframe>` code you need to share and embed your YouTube-hosted videos.
8 What are some advantages of hosting your own videos by using the `<video>` element?	
9 What are some advantages of using a hosting service like YouTube for your videos?	
10 Close all open files and browser sessions	

Unit summary: Audio and video

Topic A

In this topic, you learned how to **embed native audio** that works in all HTML5-supporting browsers. You learned about the different **audio formats** and which browsers support each content type. You also learned how to sequence an **audio source list** to ensure that all HTML5 browsers and iOS devices can access your audio content.

Topic B

In this topic, you learned how to **embed native video** that works in all HTML5-supporting browsers. You learned about various **video formats** and which browsers support each format. You also learned important details about several video-related attributes, and you learned how to set a **poster image** for a video. Finally, you learned how to use **inline frames** to embed externally hosted content such as YouTube videos.

Review questions

1 True or false? Native audio and video require less processing power than plug-in–based methods.

2 True or false? When a browser encounters an `<audio>` or `<video>` container, it goes through the source list from top to bottom until it finds a media type it can play.

3 What does specifying `preload="auto"` in a `<video>` element tell a browser?

   A It suggests that the browser should not preload any data until the user plays the video.

   B It suggests that the browser should load only the file's metadata.

   C It suggests that the browser should use the preload setting of its parent element.

   D It suggests that the browser should determine whether or not to preload some or all of the video data.

4 True or false? If you omit the `controls` attribute from the `<audio>` element, users won't see anything on the page associated with the audio file, and they won't have the ability to play or stop the audio.

5 True or false? You can use the `src` attribute directly in the `<video>` element to specify a single video source.

6 True or false? You should always use the `type` attribute when you specify a video source, even when you specify a single source.

7 True or false? If you set a video's width but not its height, the browser will compress or stretch the video, which often produces undesirable results.

8 True or false? The `<iframe>` element does not require a closing tag.

Independent practice activity

In this activity, you'll embed a video and set video properties.

The files for this activity are in Student Data folder **Unit 3\Unit summary**.

1 Open index.html in Chrome and your text editor.

2 Scroll to the bottom of the code to locate the AD BAR comment.

3 Under the comment, above the footer section, create a section by using a `<section>` element that has the ID **adBar**. **Note:** It's important that you use this exact ID name because the style sheet contains a style rule for this ID.

4 Insert an `<h1>` heading as the first element in your new section. Add **Advertisements** as the heading text.

5 Under the Advertisements heading, embed a video using a single video source, with the path **media/smartcourse.ogv**. (*Hint:* You can use a `src` attribute directly in the `<video>` element when you need to specify only a single version of a video.)

6 Enable the video controls and set the width to **350**. Let the browser decide which video data should be preloaded. Also, add fallback text to the video for non-supporting browsers.

7 Use the image file smartcourse.gif as a poster image for the video. (*Hint:* The file is also in the media folder.)

8 Save your changes and reload the page in both Chrome and Firefox. Compare your results to Exhibit 3-3.

Exhibit 3-3: The video file after Step 8

9 Close all open files.

Unit 4

CSS techniques

Complete this unit, and you'll know how to:

A Format elements based on their location in the document structure.

B Incorporate special fonts using the Google Web Fonts API, and apply shadow effects.

C Insert and format generated content.

Topic A: Styles that improve efficiency

Explanation

You should be somewhat familiar with CSS, or Cascading Style Sheets, the standard style language of the Web. HTML and CSS are inseparable; you need to learn some CSS to be an effective Web developer. Here, you'll learn some practical CSS techniques that promote efficiency by reducing the need to add new elements and attributes to your existing HTML structure.

Pseudo class selectors

You can use pseudo class selectors to format elements based on their location in the document structure. They are called *pseudo classes* because they aren't true classes. Rather than acting on elements based on their names or their content, pseudo classes classify elements based on characteristics that are not physically part of the document structure.

You should be familiar with pseudo class selectors such as `:hover` and `:visited`, which you use to control link styles—these classifications aren't physically present in the document but they represent a state or characteristic of an element. There are several other pseudo classes you can use, including:

```
:first-child
:last-child
:only-child
:first-of-type
:last-of-type
:only-of-type
```

The key benefit of using these selectors is that they enable you to format content without having to introduce any new elements or attributes into your HTML document structure. These selectors are also supported in all the major browsers: Chrome, Firefox, Safari, Opera, and Internet Explorer.

Pseudo class selector syntax

The syntax for a pseudo class selector is as follows:

```
element:pseudo-class { property:value; }
```

No spaces are allowed between the element you want to select, the colon, and the pseudo class. However, if you want to specify a parent to give the rule greater context, you list it first, followed by a space and then the rest of the rule, as follows:

```
parent element:pseudo-class { property:value; }
```

The first-child, last-child, and only-child pseudo classes

With the `first-child` pseudo class, you can select an element only when it's the first child of its parent element. For example, to make only the first item in your lists bold, you could write:

```
li:first-child { font-weight:bold; }
```

This would apply to both ordered and unordered lists (provided they do not contain any other elements), and the result would look like this:

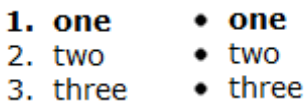


Exhibit 4-1: Selecting list items only if it's the first child of its parent

Similarly, you can use the `last-child` pseudo class to select an element only if it's the last child of its parent element, and you can use `only-child` to select an element when it's the only child element of its parent.

Adding context

As with all style rules, you can also declare a parent element to ensure that formatting applies only in a specific context. In the previous `first-child` example, the first list item in a list of any type would be selected, whether it's an unordered list or an ordered list. The following example makes the last item in ordered lists bold (only ordered lists, not unordered lists).

```
ol li:last-child { font-weight:bold; }
```

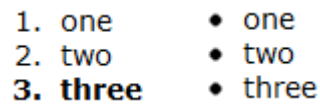


Exhibit 4-2: Selecting an ordered list item when it's the last child of its parent

So, to make a rule contextual, start the rule with the element name that provides the context, followed by a space, and then followed by the rest of your style rule.

Do it!

A-1: Applying styles based on parent-child structure

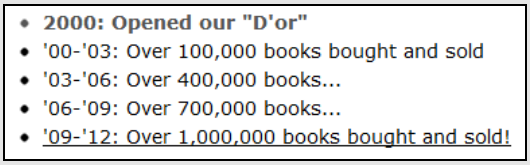
The files for this activity are in Student Data folder **Unit 4\Topic A**.

Here's how	Here's why
1 In Chrome, open about-us.html	From the current topic folder in the current unit folder.
2 From the styles folder, open styles.css	The styles folder is in the current topic folder, in the current unit folder.
3 Scroll down to the bottom of the style sheet	
Insert a CSS comment that reads **PSEUDO CLASSES**	CSS comments begin with `/*` and end with `*/`.
4 Under the comment, insert the following rule:	To make only the first item in the list bold and dark gray.
`li:first-child { font-weight:bold; color:#555; }`	
5 Save the style sheet and reload the page in Chrome	Only the first item in the list is bold.
6 In the nav bar, click **Home**	To open the home page. The new style has also affected the list on this page, which was not intended.
Click **About Us**	To return to the about-us.html page.
7 Open about-us.html in your text editor	
Locate the "about" section	Each page contains a section that's named for its page. This provides the opportunity to create contextual styles.
Minimize the about-us.html file	Don't close it.
8 In the style sheet, add the following bold code:	To make only the first list item that's in the "about" section dark gray.
`#about li:first-child { font-weight:bold; color:#555; }`	
Save the style sheet and reload the page in Chrome	The list on the About Us page is still formatted as intended.
9 Click **Home**	The first item in this list is no longer bold because the rule now selects only the first list item that's in the "about" section, which this page does not have.
Go back to the About Us page	

10 Create a new rule that underlines only the last list item on this page

Use the `:last-child` pseudo class and the `text-decoration: underline` property and value.

Compare your results to the example shown



11 How would you replicate these styles without pseudo classes?

Selecting by location and element type

Explanation

In addition to selecting and formatting elements based on their parent-child structure, you can be more specific by selecting only the first, last, or only element of a particular type within a parent element.

The first-of-type, last-of-type, and only-of-type pseudo classes

You can use the `first-of-type`, `last-of-type`, and `only-of-type` pseudo classes when you want to select only specific element types within a parent/child structure. Consider the following basic structure, for example:

```
<section>
 <h1>This is a heading</h1>
 <p>This is a paragraph</p>
 <p>This is another paragraph</p>
</section>
```

If you want to make only the first paragraph bold, using `p:first-child` would have no effect because there's no paragraph in this structure that is the first child of its parent—the heading is the first child in this structure.

However, you could write:

```
p:first-of-type { font-weight:bold; }
```

This would work because the first paragraph is the first of its type within its parent element.

Parent/child pseudo classes work well in tables and lists

Pseudo classes that select items based on parent/child structures tend to work well in lists and tables because the structure of lists and tables tends to be predictable and consistent.

Do it!

A-2: Applying styles based on location and type

Here's how	Here's why
1 Review the About Us content structure	The page contains a heading and subheading, followed by a paragraph, a list, another heading, two paragraphs, and an aside.
2 In the style sheet, after the existing pseudo classes, type: `#about p:first-child { background-color:#eee; border-radius:5px; }`	You want to format only the first paragraph on this page without adding any HTML.
3 Save your changes and reload the page in Chrome	There is no change because there is no paragraph in this document that is the first child of its parent.
4 Switch to the about-us.html file	
Review the HTML structure	The first paragraph's parent is the "about" section element. The first child in that element is the `<hgroup>` that contains both headings.
5 Change `first-child` to `first-of-type`, as follows: `#about p:`**`first-of-type`**` { background-color:#eee; border-radius:5px; }`	
6 Save your changes and reload the page in Chrome	Now the style works because the first paragraph is the first of its type in its parent element.
7 Create a new selector for only the last paragraph in the "about" section	
Follow your selector with this declaration: `{ border-bottom: 1px dotted #555; padding:20px; }`	To give the last paragraph in the "about" section a bottom border and bottom padding.
8 Save your changes and reload the page in Chrome	A small dotted border is drawn along the full length of the last paragraph, and some extra space is added between the content and the border.

The nth-child and nth-of-type pseudo classes

Explanation

You can also use the `nth-child` pseudo class to format elements based on their position in their parent element's structure. In particular, you can select the *n*th child element of its parent element. Consider the following list, for example:

```
<ul>
 <li>List item 1</li>
 <li>List item 2</li>
 <li>List item 3</li>
</ul>
```

Here, the parent of each list item is the `<ul>` element. To format only the third list item, you could add this rule to your style sheet:

```
li:nth-child(3) { background-color: silver; }
```

This rule says, "Select any list item that is the third child element of its parent element, and give it a silver background." The syntax for this pseudo class follows:

```
element:nth-child(n) { property:value; }
```

The argument (*n*) determines which element or elements will be selected for formatting. The easiest arguments to use are:

- A keyword, `even` or `odd`, to select even-numbered child elements or odd-numbered child elements
- A positive integer (e.g., 1, 2, 3, etc.) to select a specific child element in a sequence of elements, from top to bottom

As with other styles, you can establish a specific context by including a parent element. In the previous example, the third list item in a list of any type would be selected, whether it's an unordered list or an ordered list. To specify that you want to select the third item in only unordered lists, you would start the rule with `ul` followed by a space:

```
ul li:nth-child(3) { background-color: silver; }
```

This rule says, "Select any unordered list item that is the third child element of its parent, and give it a silver background."

The nth-of-type pseudo class

Similar to the `nth-child` pseudo class, you can use `nth-of-type` to format elements based on their position in their parent element's structure. However, whereas `nth-child` selects an element in a series of any child elements within a parent, `nth-of-type` selects only those elements of the type you specify. For example:

```
<section>
 <p>This is the first paragraph in this section.</p>
 <p>This is the second paragraph in this section.</p>
 <p>This is the third paragraph in this section.</p>
</section>
```

If you want just the second paragraph to have a silver background, you can write:

```
p:nth-child(2) { background-color: silver; }
```

However, if you change the markup to add a heading, like this:

```
<section>
 <h1>This is a heading</h1>
 <p>This is the first paragraph in this section.</p>
 <p>This is the second paragraph in this section.</p>
 <p>This is the third paragraph in this section.</p>
</section>
```

…then the second paragraph is no longer selected—the first one is, because the first paragraph is now the second child element of its parent. However, if you were to write:

```
p:nth-of-type(2) { background-color: silver; }
```

…then your intended style would still work; the second paragraph would still be selected.

So, when you use these pseudo class selectors, it's important to consider how your page content will change over time. In particular, be careful when you use the `nth-child` pseudo class. If you introduce new elements to the parent element structure, you're likely to get undesired results or no results at all.

Alternating table row colors

Alternating table row colors is a popular application of the `nth-child` and `nth-of-type` pseudo classes. This design technique makes reading table data much easier; the eye can more easily track each individual row of data. In the past, this technique required the addition of class names in each table row; this added a lot of extra HTML, especially in long tables. Using these pseudo classes makes the technique much faster and easier to apply—and without requiring any new markup.

Controlling table cell spacing and padding

By default, browsers apply default padding and spacing between table cells. A cell's padding is the space between its content area and its borders, while cell spacing is the space between adjacent cells. For years, developers used the `cellspacing` and `cellpadding` attributes in table tags to control or disable the default spacing and padding in table cells. These attributes are deprecated in HTML5, meaning that they are omitted from the specification in favor of CSS, and any further use is discouraged.

Now, the `padding` property controls cell padding, and the `border-spacing` property controls the space between adjacent cells.

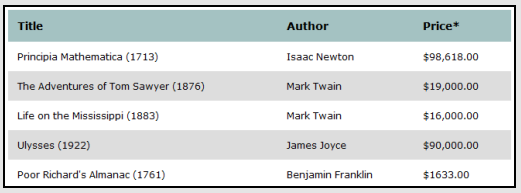
- The `padding` property must be applied to the `<td>` and/or `<th>` elements.
- The `border-spacing` property must be applied to the `<table>` element.

For example, to give all table data and header cells 6 pixels of padding, and to remove the border spacing that browsers apply by default, you would create two rules:

```
td, th { padding: 6px; }
table { border-spacing: 0; }
```

Do it!

A-3: Alternating table row colors

Here's how	Here's why
1 Click **Browse Titles**	(In Chrome.) To open the titles.html page.
2 Observe the gaps between each column	By default, browsers apply default spacing between adjacent cells. You'll start by disabling this default spacing.
3 Switch to the style sheet	
4 Scroll down to locate the TABLE STYLES comment	
5 Add the following style to the existing table rule: `border-spacing: 0;`	The border-spacing property has to be set on the table element.
6 Save your changes and reload the page in Chrome	The default spacing is reset to zero.
7 Under the PSEUDO CLASSES comment, add the following rule: `tr:nth-child(even) { background-color: #ddd; } /* Gray */`	To select only the even-numbered table rows and give them a gray background.
8 Save your changes and reload the page in Chrome	 The even-numbered rows now have a gray background, which helps make the data easier to read across each row.
9 Create a rule that makes only the Price data (the third column) bold	Start with the `<td>` element, and use the nth-child pseudo class.
Insert a CSS comment that reads **Price Column**	(Add the column after the style declaration.) Adding comments to your style rules will help you and others to make changes later on.

Title	Author	Price*
Principia Mathematica (1713)	Isaac Newton	$98,618.00
The Adventures of Tom Sawyer (1876)	Mark Twain	$19,000.00
Life on the Mississippi (1883)	Mark Twain	$16,000.00
Ulysses (1922)	James Joyce	$90,000.00
Poor Richard's Almanac (1761)	Benjamin Franklin	$1633.00

10 Compare your results to those shown here

Author	Price*
Isaac Newton	$98,618.00
Mark Twain	$19,000.00
Mark Twain	$16,000.00
James Joyce	$90,000.00
Benjamin Franklin	$1633.00

The Price data column is now bold.

11 Observe the column widths

Right now, there is no width setting applied to any of the columns, so they are sized based on the amount of content they contain.

12 Create a rule that gives only the Title column a width of **300px**

Insert a CSS comment that reads **Title Column**

Save your changes and reload the page in Chrome

The Title column is shortened slightly, so the other columns shift a bit to the left.

13 Will the alternating-row-colors style work if new rows are added?

14 What are the limitations of the other two `nth-child` rules?

15 Change the alternating-row-color rule to use `nth-of-type`

Replace `nth-child(even)` with `nth-of-type(even)`.

Save your changes and reload the page in Chrome

There's no change because both pseudo classes work in the context of this structure.

16 Will the other styles work if you change them to `nth-of-type`?

17 Are the following two selectors equivalent?

```
ol li:nth-child(1)
ol li:first-child
```

18 Close all open files

Guidelines for reducing style redundancies

Explanation

Because CSS is an integral part of HTML5 authoring, it's important to cover some practical style authoring guidelines. In addition to using advanced selectors that promote efficiency by precluding the need to introduce new document markup, you should try to keep your style sheets as efficient as possible. One way to do that is to consolidate styles to avoid redundancies.

Consolidate styles when possible

When you review your style sheet, you might find that you have several redundant styles. Redundancies can be difficult to avoid as you're developing, especially if you're working in a team, but once you're at the finish line, take some time to look over your style sheet. Look for opportunities to add additional helpful comments and to remove redundancies by consolidating styles.

As a simple example, say you have the following three styles:

```
h1 { color: #333; }
h2 { color: #333; }
h3 { color: #333; }
```

Here, three different heading elements share the same background color. You could achieve the same result with one rule by grouping the three selectors into a single selector and separating each selector with a comma, as follows:

```
h1, h2, h3 { color: #333; }
```

You should do this even when different rules have their own styles. As long as multiple rules are sharing a style, you can group the elements together and then handle individual styles in their own rules. For example, consider the following styles, in which three elements share the same text color but have different padding values:

```
h1 { color: #333; padding: 8px; }
h2 { color: #333; padding: 5px; }
h3 { color: #333; padding: 3px; }
```

It would be more efficient to write these styles as follows:

```
h1, h2, h3 { color: #333; }
h1 { padding: 8px; }
h2 { padding: 5px; }
h3 { padding: 3px; }
```

Here, four rules are more efficient than the preceding three rules because there are no redundancies—no repeating styles.

This might not seem like it would cut file size by much, but if you have multiple large style sheets, this technique can have a significant impact on their size and readability. Smaller file size isn't the main benefit of this type of consolidation, though. The real benefit is that it makes your style sheets easier to work with and to update.

Using the previous example, if you were to decide at a later date that you wanted to change the background color of your headings, you'd need to make only one change instead of three changes. Again, this might not seem significant in this small example, but in a larger context, with several instances of this kind of style duplication, the file size and effort efficiencies can be substantial.

Do it!

A-4: Exploring style sheet efficiencies

Questions and answers

1 What would be a more efficient way to write the following styles?

```
em { color: #333; font-style: italic; }
strong { color: #333; font-style: italic; }
cite { color: #333; font-style: italic; }
```

2 What would be a more efficient way to write the following styles?

```
p { color: #000; padding: 6px; background-color: #eee; }
h1 { color: #000; padding: 10px; }
div { color: #000; background-color: #eee; padding: 6px; }
```

3 What would be a more efficient way to write the following styles?

```
section { background-color: #eee; margin: 10px; }
section#intro { background-color: #eee; margin: 10px; }
section#articles { background-color: #ccc; margin: 10px; }
```

Topic B: Fonts in the cloud

Explanation

In the past, one of the most challenging aspects of Web design was figuring out how to take full advantage of the power of typography. Today, you can access free font services "in the cloud" to incorporate distinctive fonts into your site and rest assured that your text will look the same for everyone.

Typography on the Web

The fonts that are installed on computers depend on the platform, version, and software programs that have been installed. In the early years, practically every page on the Web used the same default Times New Roman. The art of typography was minimized on the Web because browsers could display a designer's font choices only if those fonts were installed on the user's computer.

Developers resorted to a variety of techniques, including using images for headings just to add unique typographical touches. All those images ate up precious bandwidth, the text wasn't searchable, and updating the text became a time-consuming and tedious task. None of that is necessary anymore.

Today you can incorporate distinctive fonts into your site by using open-source font services online. One of the most popular is Google's *Web Fonts API service*, located at http://www.google.com/webfonts. You choose the fonts you want to use in your site, and the service walks you through the steps to incorporate them into your site's pages or templates.

The font choices include fonts optimized for reading, handwriting fonts, exotic fonts for special purposes, and more. The service uses Google's servers, so you know you don't have to be concerned about downtime, reliability, or speed. To link your documents to the Web Fonts API, you use the `<link>` element.

The <link> element

With the `<link>` element, you can link your pages to other resources, such as style sheets and scripts. The `<link>` element has two required attributes, `href` and `rel`. The `href` attribute specifies the URL for the resource, and the `rel` attribute specifies the relationship of the referenced file to the main document. You can use the optional `type` attribute to specify the MIME type of the referenced file.

For example, to link to Google's Web Fonts API and use the font named "Acme," you would write:

```
<link rel="stylesheet" type="text/css"
href="http://fonts.googleapis.com/css?family=Acme" />
```

This code must be placed inside the head section of each document that you want to have access to the font.

Also, notice that in this example, the element is terminated with a slash at the end of the tag. The `<link>` element is not a container and therefore does not have a corresponding closing tag. HTML5 does not require a terminating slash for empty elements, but you can include one if you prefer to follow the guidelines of XHTML.

Multiple style-sheet sources

A page can contain multiple `<link>` elements, linking the page to several style sheets and other resources. For example, you might have separate style sheets to optimize your content for mobile devices and for optimizing the print output of your pages.

Setting the font

When you have established the link to the Web Fonts API, you can then use CSS to declare the fonts you specify in your `<link>` element. To set the font, you use the `font-family` property. To ensure that your site visitors will view your text in a font that is at least similar to your first choice, specify at least two or three fonts, with each one separated by a comma, as follows:

```
body { font-family: Acme, Verdana, sans-serif; }
```

Even when you're using the Web Fonts API, it's still a good idea to declare a generic family name at the end, just in case a user's browser can't load any of the specific fonts. Also, if the font you choose consists of multiple words, it must be enclosed in quotes.

Choose your fonts carefully

The font choices you make when you're developing a site are critical in many ways. First, you need to make sure that your text is readable, consistent across different platforms, and appropriate for the overall look and feel of the site.

Determine which fonts are best suited to your site. Make this determination for body text, where readability and consistency are the key factors, as well as for headings and other areas where you have opportunities to use unique design touches.

For example, Verdana is one of the most popular and widely used fonts on the Web today because it was designed specifically for reading on screen. Exotic fonts can work fine for headings and logos, but don't get fancy with body text where people need to read long lines.

Do it!

B-1: Applying Google Web Fonts to your site

The files for this activity are in Student Data folder **Unit 4\Topic B**.

Here's how	Here's why
1 In Chrome, open about-us.html	(From the current topic folder in the current unit folder.) You will format the text "Garage D'or Books" with a special font.
Open about-us.html in your text editor	
2 In the head section, after the title, add the following code:	
`<link rel="stylesheet" type="text/css" href="http://fonts.googleapis.com/css?family=Bilbo+Swash+Caps" />`	
3 Save your changes	Leave the file open.
4 From the styles folder, open styles.css	The styles folder is in the current topic folder, in the current unit folder.
5 Locate the current h1 rule	You will use the Bilbo Swash Caps font for the "Garage D'or Books" text, which is a level-one heading. However, the `<h1>` element is used in other areas because in HTML5, each section can have its own independent outline. We want to use this special font only for the company logo.
6 Switch to about-us.html	
Locate the `<hgroup>` element	The company name heading is the only `<h1>` element in an `<hgroup>` element, so you can use it as the context for a new rule.
7 Switch to the style sheet	
Add the following rule after the existing h1 rule:	To format `<h1>` elements only if they are inside an `<hgroup>` element.
`hgroup h1 { font-family: "Bilbo Swash Caps",cursive; font-size: 50px; }`	
Save your changes and reload the page in Chrome	 The browser displays the special font even though it's not installed on your computer.

8 Observe the space above and below the text

When you specify a large font size like this, you get large top and bottom margins. You can reduce them with CSS.

Add the following bold code to the rule:

```
hgroup h1 { font-family: "Bilbo Swash Caps",cursive;
            font-size: 50px;
            margin-top:10px;
            margin-bottom:10px; }
```

9 Save your changes and reload the page in Chrome

The margins are reduced.

10 Click **Home**

(In the navigation bar.) To open the home page. The font is not applied to the other pages because each page has to include the link to the Google font service.

11 In about-us.html, copy the entire `<link>` element

The one that links to the Google Web Fonts API.

12 In your text editor, open index.html

Paste the code into the top of the head section

Save and close the file

13 Paste the code into the titles.html document, and save and close it

14 In Chrome, click the three site links

To view the results on all three pages. The special font now appears on all pages. Next, you'll apply text effects with CSS.

Shadow effects

Explanation

When you use distinctive fonts for design purposes, you can customize them using various CSS properties. For example, you can apply a shadow effect, which is a popular design technique that can add appeal and draw the user's eye to particular items. To create text shadows, you use the `text-shadow` property.

The text-shadow property

With the `text-shadow` property, you specify four values separated by spaces. As shown in Exhibit 4-3, the first value sets the shadow's horizontal offset—a positive value offsets the shadow to the right of the element, and a negative value offsets the shadow to the left.

The second value specifies the vertical offset—a positive value offsets the shadow from the bottom of the element, and a negative value offsets the shadow from the top. The third value specifies the distance of the shadow blur, and the fourth value specifies the shadow color.

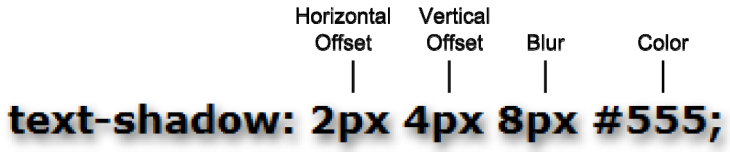


Exhibit 4-3: A text shadow effect and the code used to create it

The `box-shadow` property works just like the `text-shadow` property, but it applies to block elements such as `<section>`, `<div>`, and `<nav>`. You can use the `box-shadow` property to quickly create effects such as drop shadows and blurs.

Do it!

B-2: Applying shadow effects

Here's how	Here's why
1 Switch to styles.css	
2 Add the following style to the hgroup h1 rule: `text-shadow: 1px 1px 3px #333;`	To apply a gray shadow with 1px of horizontal offset, 1px of vertical offset, and 3px of blur.
3 Save your changes and reload the page in Chrome	 The shadow effect creates a sense of depth.
4 Add the following style to the hgroup h1 rule: `color: #a4c2c2; /* pale blue */`	To make the text color a pale blue.
5 Save your changes and reload the page in Chrome	 With a text color, you can sometimes more easily see the shadow effect.
6 Locate the `#page` style	This style rule formats the page wrapper div, the outermost container.
7 Add the following style to the `#page` rule: `box-shadow: 5px 5px 15px #444;`	
8 Save your changes and reload the page in Chrome	The page container now appears to be raised off the background. These are just examples of the kinds of effects you can create.
9 Close all open files	

Topic C: Generated content

Explanation

In addition to using CSS to format HTML content, you can also use it to add content for user interface enhancements and other stylistic purposes. *Generated content*, which is also called *CSS content* or *style content*, allows you to clarify and augment your documents with small pieces of content or other forms of indicators or notations. To create this content, you use pseudo elements.

Pseudo elements

Pseudo elements are similar to pseudo classes. Pseudo classes classify elements based on characteristics that are not physically part of the document tree, such as the "hover" or "visited" state of a link. *Pseudo elements* are so named because they are displayed just like element content, but they are not actually elements within the document structure.

List bullets are like a form of generated content except that browsers display them automatically. The bullets in an unordered list, for example, don't physically appear in the document markup, but they are a display characteristic of the `<li>` element.

Because pseudo elements enable you to apply a variety of interface enhancements without adding any non-essential markup to your documents, they help to keep your documents lean and free of style-oriented markup, thus preserving your document semantics.

Sample use cases

Sample applications of pseudo elements include displaying a link's `href` value as text, which is particularly useful in print style sheets, and displaying words, special characters, or images before or after certain elements for interface design purposes.

For example, say that you use `<strong>` elements throughout your site to mark up important text statements, such as "This offer expires on 12/31/13," and you later decide that you want the text "NOTE:" to precede all such notices. Instead of going into each document that contains such a notice and manually adding this content, you can select the elements and apply the content from the style sheet.

Depending on the number of entries, this solution could save a lot of time. Plus, if you ever want to remove the text, it's a matter of removing one rule in the style sheet rather than multiple instances of actual document content.

The :before and :after pseudo elements

To create style content, you need to specify what content you want to include and where it should be placed relative to actual document content. You use the `:before` and `:after` pseudo elements to set the location for generated content. For example, if you decide you want to add an asterisk before the content in your `<aside>` elements, you could write:

```
aside:before { content: "*"; }
```

Pseudo elements must be written at the end of a selector, after the selector subject and any classes or IDs that might provide context. Also, a style rule can have only one pseudo element.

The content property

As shown in the previous example, the `content` property declares the content that you want to appear before or after an actual element's content. If you specify string content (text content), it must be placed in quotation marks.

You can apply styles to generated content, just as you can with actual content. For example, you can control the color, font, borders, and position of generated content.

Don't use generated content for text or other content that belongs in the document tree. Generated content is not searchable and should be used only to inform users or otherwise provide stylistic enhancements. Browsers that support HTML5 support generated content, but older browsers have varying and inconsistent support.

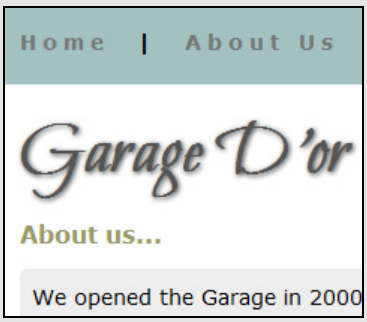
Do it!

C-1: Generating content from a style sheet

The files for this activity are in Student Data folder **Unit 4\Topic C**.

Here's how	Here's why
1 In Chrome, open about-us.html	From the current topic folder in the current unit folder.
2 Locate the level-two headings	The "About us" and "Why we're different" headings are both `<h2>` elements. You'll add an ellipsis to the end of both of these by using pseudo elements.
3 From the styles folder, open styles.css Scroll down to the PSEUDO ELEMENTS comment	The styles folder is in the current topic folder, in the current unit folder.
4 Under the comment, add the following rule: `#about h2:after { content: "..."; }`	To add an ellipsis to the end of the `<h2>` headings that are inside the "about" section.
5 Will this rule have an effect on any `<h2>` elements outside this page?	

6 Save your changes and reload the page in Chrome



The ellipsis is applied to both headings, but only the headings on this page.

7 Click **Home**

(In the navigation bar.) To open index.html. The style is not applied to the `<h2>` heading on this page.

8 In the style sheet, add the following new rule:

Under the PSEUDO ELEMENTS comment.

```
#main ul li { list-style-type: none; }
```

To remove the default bullets that browsers apply to unordered list items. This rule applies only to unordered lists in the "main" section, which is only on the home page.

9 At the end of the style declaration, add the following comment:

```
/* Replace default bullets with double colons */
```

10 Save your changes and reload the page in Chrome

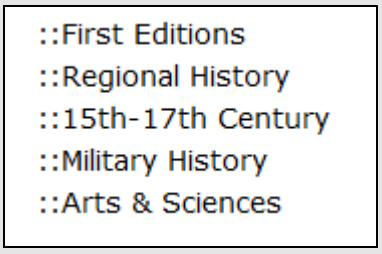
The default bullets are now gone. You'll create a custom bullet style for this section.

11 On the next line in the style sheet, add the following rule:

To create a custom bullet that consists of two adjacent colons.

```
#main ul li:before { content:"::"; }
```

12 Save your changes and reload the page in Chrome



The double colons appear before every list item in this section. Next, you'll add styles to these custom list bullets.

13 Add the following two styles shown in bold:

To style the CSS content.

```
#main ul li:before { content:"::";
      font-weight:bold; color: #996; }
```

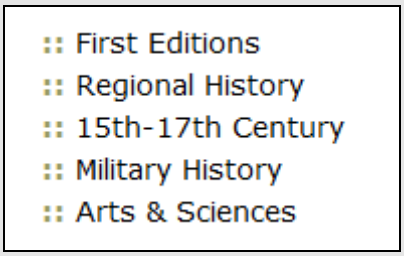
14 Save your changes and reload the page in Chrome

Generated content can be styled like ordinary content. Next, you'll add a space after the pseudo element.

15 Add a space after the pseudo element content, as shown:

```
#main ul li:before { content:":: ";
      font-weight:bold; color: #996; }
```

16 Save your changes and reload the page in Chrome



The list now has custom bullet icons created from plain text in the style sheet.

Images as style content

Explanation

In addition to using the `content` property with string content, you can use this property to specify the location of an image file and to display it before or after a selected element. This technique can be especially useful in data tables, where graphic icons can be employed to help users read and understand various data types or options.

Applying an image as generated content

To generate image content from a style sheet, you use the `content` property along with the following syntax:

```
selector:pseudo-element { content: url(path-to-image); }
```

The `content` property is followed by `url`, followed immediately by parentheses that contain the URL for the image file. The path cannot be in quotes.

When you specify a path to an image file or other resource from within a style sheet, your URL path will probably have to go up one level, depending on your directory structure. For example, if your style sheets are stored in a styles folder, a path to an image file will have to first go up one level to get outside the styles folder, and then provide the path to the image file. To do this, use two periods and a slash:

```
{ content: url(../images/myImage.png); }
```

Attribute selectors

Another way you can select elements for styling is to use attribute selectors. *Attribute selectors* select elements if they contain a specified attribute or attribute=value combination. Attribute selectors use the following syntax:

```
[attribute] { property: value; }
```

This syntax will apply the style to any element that contains the specified attribute. Or you could write:

```
element[attribute] { property: value; }
```

This code selects any element that contains the specified element. Finally, you can select only those elements that have specific attribute values:

```
element[attribute="value"] { property: value; }
```

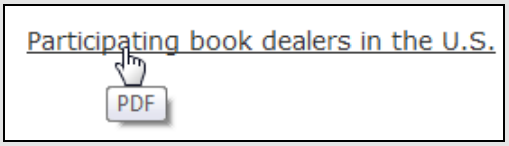
The attribute name is always placed within brackets, as shown. You can use any attribute as a selector. For example, to make italic only those elements containing the attribute and value `lang="fr"`, you could write:

```
[lang="fr"] { font-style: italic; }
```

There are a variety of use cases for attribute values, depending on your document structure. The value they represent is that like pseudo classes, they allow you to select and format specific elements without having to introduce new HTML markup.

Do it!

C-2: Applying images as CSS content

Here's how	Here's why
1 Click **Browse Titles**	To open titles.html
In your text editor, open titles.html	You'll create a link and then use generated content to append an image.
2 Scroll to the bottom	
Directly under the LINK TO PDF comment, type:	To create a link with a title indicating that the link opens a PDF file.
`<p><a href="#" title="PDF">Participating book dealers in the U.S.</a></p>`	
	The # value is just a placeholder because we're not linking to an actual resource.
3 Save your changes and reload the page	
Point to the link	
	Link titles can improve usability by providing more information about a link when you point to them. Next, you'll append an image.

4 Add the following rule to the end of the style sheet:

To select links that have a PDF title and append a PDF icon to them.

```
a[title="PDF"]:before { content:url(../images/pdf.png); }
```

5 Save your changes and reload the page

A PDF icon appears before the link. No other links on the page are affected.

6 Add 5px of right padding to the PDF icon

(Use the `padding-right` property.) To create some space between the icon and the link text.

7 Save your changes and reload the page

There's some extra space between the icon and the link text.

8 How can using CSS image content be an efficient design option?

9 How might you use generated content in your own site(s)?

10 Close all open files

Unit summary: CSS techniques

Topic A

In this topic, you learned how to use **pseudo classes** to format elements based on their location in the document structure. You learned that this technique can improve **document efficiency** by reducing the need to use new HTML to achieve design objectives. You also learned how to create **alternating table row colors** without having to change your HTML markup, and you learned some basic guidelines for creating efficient style sheets.

Topic B

In this topic, you learned how to **incorporate special fonts** into your site by using the Google Web Fonts API. You learned how to link to the service and how to apply the fonts with CSS. Finally, you learned how to apply **text shadow** effects.

Topic C

In this topic, you learned about **generated content**. You learned how to use **pseudo elements** and the **content property** to add content enhancements that are not part of the document tree. You also learned that you can format generated content the same as actual document content. Finally, you learned how to use **attribute selectors** to select and format specific elements without having to introduce new HTML markup.

Review questions

1 True or false? In a simple list containing 10 items, the selectors `li:last-child` and `li:nth-child(10)` will select the same list item.

2 True or false? Pseudo elements and pseudo classes are not part of the document tree.

3 True or false? The `<link>` element can be placed anywhere in the document structure except in the `<body>` element.

4 True or false? A document can contain multiple `<link>` elements.

5 True or false? When you use font services like Google's Web Fonts API, you can only display the fonts as they are; you can't apply CSS formatting to them.

6 Which two components are required to generate content from a style sheet? [Choose two.]

   A Any pseudo class

   B The `:before` or `:after` pseudo element

   C The `content` property

   D An attribute selector

7 Which of the following are benefits of using generated content? [Choose all that apply.]

A Generated content enables you to apply a variety of interface enhancements without adding any non-essential markup to your documents.

B Generated content allows you to quickly and easily make interface-related content updates quickly and without having to modify any HTML.

C Generated content is indexed by search engines.

D Generated content is supported in all browsers, old and new.

Independent practice activity

In this activity, you'll create alternating table row colors, apply a shadow effect, and change the placement of generated content.

The files for this activity are in Student Data folder **Unit 4\Unit summary**.

1 In Chrome, open titles.html.

2 In your text editor, open styles.css.

3 Using a color of your choice, create a rule that alternates the table row background color and the default white background color.

4 At the bottom of the style sheet, under the NAVBAR SHADOW comment, use the `box-shadow` property on the `#siteNav` element to reproduce the effect shown in Exhibit 4-4. (*Hint:* The first value is the horizontal offset, the second value is the vertical offset, the third value is the amount of blur, and the fourth value specifies the color of the blur.)

Exhibit 4-4: The <nav> section with a drop shadow effect

5 Set the PDF icon to appear on the right side of the link, and add some space between it and the link text, as shown in Exhibit 4-5.

Exhibit 4-5: The PDF icon after Step 5

6 Close all open files.

Unit 5

Local storage and site testing

Complete this unit, and you'll know how to:

A Create editable content, implement a script that saves user data in local storage, and rotate an element.

B Perform basic site testing before you publish a site.

Topic A: Editable content and local storage

Explanation

Beyond elements, attributes, and CSS techniques, HTML5 enables a variety of Web applications and practical functionality. Typically this functionality involves JavaScript, which is beyond the scope of this course. In this topic, you will enable editable content and apply an existing local storage script to create a persistent "sticky note."

Making an element's content editable

You can use the `contenteditable` attribute in an element to make its content editable. If you set the attribute to "true," that element's content—and any child content it contains—will be editable by the user. A "false" value indicates that the element is not editable. This is the same as omitting the attribute, but you could use it to prevent a child element from being editable when that element would otherwise inherit a "true" value from its parent element. For example:

```
<section contenteditable="true">
 <p> This paragraph is editable. </p>
 <ul>
  <li>And so is this list item...</li>
  <li>And this one...</li>
  <li>Etc.</li>
 </ul>
</section>
```

There are many practical use cases for editable content, and they typically require JavaScript to be truly functional. For example, for sites with collaborative content, you could make articles editable directly in the browser, or you could allow user comments to be editable. Or you might offer users the ability to create a simple to-do list or to record reminders and notes. There's practically no limit to the number and variety of applications you can create using HTML5, CSS, and JavaScript.

The spellcheck attribute

When you make an element's content editable, browsers check spelling in that region by default. Any words not recognized by the browser's built-in dictionary will appear with a red squiggly underline, which you're probably used to seeing in word processing programs and some form input fields.

This functionality is controlled by the `spellcheck` attribute, which accepts either a `true` or `false` value. A `true` value activates spelling checking, while `false` de-activates that feature when it would otherwise be enabled.

If you enable spelling check on a parent element, all of its child elements will inherit that `spellcheck` value unless you specify otherwise.

The outline property

Browsers have different default styles when it comes to things like user forms and editable content regions. For example, in Chrome, when you click an element that's editable, a yellow outline appears around the perimeter of the editable region, but there's no default outline in other browsers. To make these styles consistent, you can use the `outline` property in your style sheet.

The `outline` property works similarly to the `border` property in that it's a shorthand property that you can use to specify all the outline attributes: color, style, and width. For example, in the following rule, a class named "highlight" applies a 1px solid red outline:

```
.highlight { outline: 1px solid red; }
```

While the syntax is the same as for the `border` shorthand property, its effect and its purpose are not the same. Whereas borders are added to an element's height and width, outlines are not. An outline is placed "on top" of the outer edge of an element and therefore does not have an effect on element spacing or layout.

To ensure that an element does not show a default outline when it comes into focus, you can disable the outline by using the keyword value `none`. This technique also entails using the `:focus` pseudo class in your selector.

The :focus pseudo class

You're already familiar with pseudo classes like `:hover` and `:visited`, which you use to style link states. You can use the `:focus` pseudo class to style an element when it comes into focus because a user has clicked it or pressed the Tab key to navigate to it. This pseudo class is typically used in forms to highlight a selected input field, but you can also use it to format editable regions. For example:

```
div.editable:focus { outline: none; }
```

This code would disable default outlines for any `<div class="editable">` element so that editable regions look the same in all browsers.

Do it!

A-1: Creating an editable content section

The files for this activity are in Student Data folder **Unit 5\Topic A**.

Here's how	Here's why
1 In Chrome, open titles.html	(From the current topic folder in the current unit folder.) The page contains a square yellow section that you'll modify into a "sticky note."
2 In your text editor, open titles.html	
Read the code under the STICKY NOTE comment	The yellow, shadowed box is the "stickyNote" section, which contains a "Note" div, which in turn contains a paragraph. The "stickyNote" section provides the appearance, and the "Note" div will provide the functionality.
3 In Chrome, click the text **Add your comments here.**	As with all normal text, you can select it but you can't place the insertion point in it or modify it in any way.

4 Add the following bold code to the "Note" div: `<div id="Note" **contenteditable="true"**>`	
5 Save your changes and reload the page in Chrome	
Click the text in the sticky note	Chrome displays a yellow outline around editable regions by default. This outline shows the perimeter of the editable region. It does not contain the "Reset Note" link because we don't want that to be editable.
6 In Firefox, open titles.html	
Click the text in the sticky note	Firefox and other browsers do not display an outline by default.
Minimize Firefox	
7 In titles.html, locate the embedded style sheet	The styles that control the sticky note are embedded in a `<style>` container on this page because it's the only page that has a sticky note.
Under the embedded styles, add the following rule: `#Note:focus { outline:none; }`	To turn off the outline when the editable region is in focus.
8 Save your changes and reload the page in Chrome	
Click the text in the sticky note	Chrome no longer shows an outline around the perimeter of the editable region.
9 Select the default text	
Type your name	The content of the element changes. Words that are not recognized by the browser's dictionary are displayed with a squiggly red underline, similar to that used in word processing programs. You can turn off this default feature by adding `spellcheck="false"` in the element you have set as editable.
Add more text so that it spans more than one line	The text wraps within the confines of the editable region's width.
10 Reload the page in Chrome	Your changes disappear and the default text is displayed again. Next, you'll add a script and some other components that will make user content persistent.

Local storage

Explanation

Local storage is a powerful feature that's well supported in current browsers. *Local storage* allows user data to be stored directly in the browser. This data can then be delivered back to the user on later visits. This is useful in a variety of contexts, such as for storing user names and other form data, or for user notes or similar content.

Comparisons to cookies

You've probably heard of browser "cookies"; they've been around since the 1990s. Cookies are small text files that are sent in HTTP request headers and stored as individual files on a user's computer. They have been used for storing user preferences, login information, and other browser history and session data. Local storage works differently and is intended to provide functionality for which cookies are not considered an optimal solution.

How local storage works

Local storage, which is also called *Web storage*, uses key/value pairs that allow user data to persist in a user's browser. When the browser is closed, the data is stored indefinitely or until it's manually cleared. The data does not expire like cookies typically do, the data does not have to be sent to a server, and you can store a lot more data than you can with cookies—about 5MB. Local storage is also based on user action, unlike cookies, which are set automatically behind the scenes for a variety of purposes.

You can use JavaScript to set, get, and clear the data in local storage. For example, you can store user form entries as they are being typed so that the data can be retrieved in case the browser crashes or is inadvertently closed before the form data is submitted.

Some browsers, including Internet Explorer, do not support local storage for HTML files opened locally.

The <script> element

The `<script>` element is a container for embedding JavaScript in a document. Before HTML5, you had to specify `type="text/javascript"` to indicate what type of script the element contains. HTML5 browsers use that setting by default, though, so the `type` attribute is now considered optional in the `<script>` element. Including it does no harm.

A simple `<script>` container would look something like this:

```
<script>

  (Your JavaScript code goes here)

</script>
```

Depending on the purpose of the script, you can insert it in a document's head section or its body section. Most embedded scripts belong in the head section. You can include any number of scripts in the head or body of a document.

Scripts are stored as separate files and are linked to documents via the `<link>` element, just as external style sheets are linked.

The onclick and onkeyup event handlers

Event handlers are components of JavaScript that execute functions based on user events, such as the clicking of a button or link. Event handlers are not added to the main script within a `<script>` container or an external script file; they are included in the HTML elements to which the event pertains. For example, to make a link execute a JavaScript function, you could include the `onclick` event handler in the link, as shown below:

```
<a href="" id="clear" onclick="clearStorage();">Clear</a>
```

The `onclick` event handler does what you probably expect: it executes a specified function when the user clicks an object. The `onKeyUp` event handler executes a specified function when a keyboard key is released.

Hide storage-related features from browsers that don't support local storage

To prevent older browsers from displaying non-functioning page elements, you can include a script that tests whether the browser supports local storage, and then include the element in the document only if the script returns a "true" value.

Do it!

A-2: Saving user data in local storage

The files for this activity are in Student Data folder **Unit 5\Topic A**.

Here's how	Here's why
1 Locate the STICKY NOTE STORAGE comment	(In the head section.) You'll insert a script here that will store user content in local storage.
2 Under this comment, type:	To create a script container.

`<script type="text/javascript">`
 	
`</script>`	

	You could also create a link to the script, but because this is the only page that will use the script, embedding it in the document makes sense.
3 In your text editor, open sticky note script.txt	From the current topic folder, in the current unit folder.
4 Read the script	The script starts with a function that saves the content in the "Note" element as "UserNote." The next function retrieves "UserNote" from local storage. The "else" statement displays a default message if "UserNote" is not in local storage. Finally, the last function will be tied to the "Reset Note" link to clear "UserNote" from local storage. This is just one of many ways you can implement this type of functionality.

5 Press CTRL + A

To select the entire script.

Press CTRL + C

To copy the script.

Close the script file

6 Paste the script inside the script container you created

Verify that the entire script is between the `<script>` and `</script>` tags.

7 Select the paragraph element in the "Note" div, as shown

```
<!-- STICKY NOTE -->

<section id="stickyNote">

<div id="Note" contenteditable="true">
<p>Add your comments here.</p>
</div>
```

Press DELETE

To remove this prompt. In this implementation, the script creates the default text, so this prompt is no longer necessary.

8 In the same location, type:

Be sure to type this code inside the "Note" div.

```
<script>getUserNote();</script>
```

This script retrieves "UserNote" if it exists.

9 Save your changes

Next, you'll specify the event that executes the "SaveNote" function in the main script.

10 In the "Note" element, add the following code shown in bold:

```
<div id="Note" contenteditable="true"
onkeyup="saveNote(this.id);">
```

This event handler executes the "saveNote" function when the user releases a key over the "Note" element. So, "UserNote" will be saved when the last character is entered.

11 Save your changes

In Chrome, reload the page

12 In the sticky note, select the default text

Type your name, today's date, and a note of your choice

13 Reload the page in Chrome

Your note is saved on later visits—the content is saved in the browser's local storage and will remain there until you clear it or change the note. You might also notice that in Chrome, once the content is saved, any unrecognized words are no longer flagged.

14 Click **Reset Note**

The note does not reset because you have not yet specified the event handler to execute the "clearStorage" function in the script.

15 In the "resetNote" link, add the following code shown in bold:

When the user clicks this link, the "clearStorage" function executes. This function removes the data from local storage.

```
<a id="resetNote" href="" onclick="clearStorage();">
```

16 Save your changes and reload the page in Chrome

Enter some random notes

Reload the page

The note is saved in local storage and displayed whenever this browser opens the page.

17 Click **Reset Note**

The link now clears local storage and displays the default text again.

18 Test the sticky note again

19 What other features might be useful for this sticky note?

20 How might you implement local storage in your own site?

21 How would you serve browsers that don't support local storage?

Transforming an element with CSS

Explanation

You can use the `transform` property to manipulate the appearance of an element in several ways. For example, you can use the property to rotate an element by a specific number of degrees, or you can skew an element (change its shape) or change its scale (size).

The transform property

The `transform` property of CSS3 is relatively new. For its value, it accepts a transform function and related coordinates. There are several transform functions you can use, including `rotate()`, `skew()`, and `scale()`. For example, Exhibit 5-1 shows the sticky note slightly rotated to create the illusion that it was pressed onto the screen.

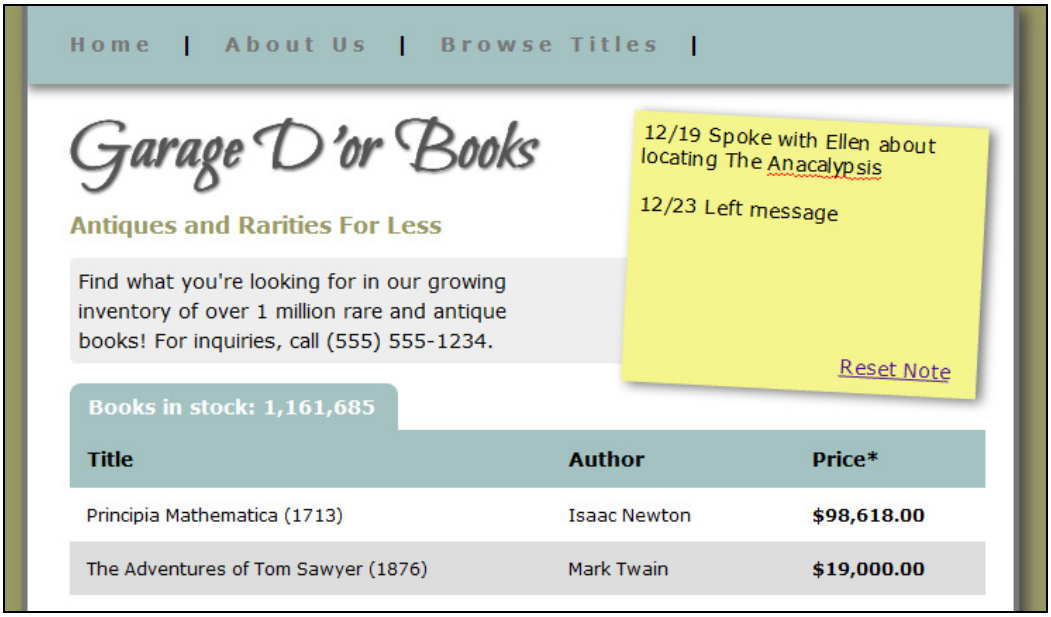


Exhibit 5-1: The sticky note, rotated slightly to complete the illusion

The rotate( ) function

The `rotate()` function sets an element's rotation according to the angle parameter inside the parentheses, in degrees (deg). For example, to rotate a `<div id="box">` element by 15 degrees (clockwise), you would write:

```
#box { transform:rotate(15deg); }
```

At the time of this writing, transforming an element by using the `transform` property and related functions is not widely supported. However, you can still transform your elements and get your transformations to work in all the major browsers by using proprietary style prefixes.

Vendor-prefixed styles

Developers and the companies who make browsers move a lot faster than the specification approval process does. For example, even though HTML5 is supported in all of today's browsers, and has been supported to varying degrees for some time, the official HTML5 specification isn't scheduled to be finalized until 2014. The same situation is true for CSS. Today, several CSS3 styles are supported only with vendor prefixes.

Browser makers have created proprietary prefixes to indicate that certain properties are "in progress." The idea is that eventually, when different browser implementations solidify into a consistent syntax, the prefixes will be omitted. The vendor prefixes for the different browser rendering engines are described in the following table.

Prefix	Description
`-webkit-`	"Webkit" is the name of the rendering engine used in the Chrome and Safari browsers and in their mobile variants, iOS Safari and Android.
`-moz-`	"Mozilla" is the name of the rendering engine used in the Firefox browser.
`-ms-`	"Ms" is an abbreviation for Microsoft and is the vendor prefix for the Internet Explorer browser.
`-o-`	"O" is short for Opera and is the prefix for the Opera browser.

So, for example, if you wanted to rotate that "box" element mentioned before by 10 degrees and you wanted it to work in all the main browsers, you would write:

```
#box { -webkit-transform:rotate(10deg);
       -moz-transform:rotate(10deg);
       -ms-transform:rotate(10deg);
       -o-transform:rotate(10deg);
       transform:rotate(10deg); }
```

In this example, every browser maker is supporting the same syntax (aside from the prefix), so it stands to reason that this will be the official syntax and will be unanimously supported without any prefix. However, the early implementations of many other CSS properties vary widely, and it's unclear which syntax will ultimately be made official.

As shown in the previous example, when you use vendor prefixes, it's a good idea to also include the non-prefixed version of it so that your styles are forward-compatible.

Do it!

A-3: Rotating an element

The files for this activity are in Student Data folder **Unit 5\Topic A**.

Here's how	Here's why
1 At the end of the `#stickyNote` rule, add the following style: `-webkit-transform:rotate(3deg);`	To rotate the sticky note by 3 degrees clockwise when viewed in Safari and Chrome.
2 Save your changes and reload the page in Chrome	The sticky note is rotated slightly, as shown in Exhibit 5-1.
3 Test the sticky note again	The style doesn't affect its functionality; it only alters the appearance of the text slightly, giving it a less formal appearance.
4 Open titles.html in Firefox	The sticky note works but is not rotated. At the time of this writing, you need to use the "moz" prefix to make rotation work in Firefox.
Add the following style under the webkit style: `-moz-transform:rotate(3deg);`	
5 Save your changes and reload the page in Firefox	The rotation style now works, and the sticky note looks the same as it does in Chrome.
6 Include the non-prefixed version at the end of the rule	
7 How might you implement local storage in your own site?	
8 Close all open files and browser sessions	

Topic B: Site testing basics

Explanation

It's critical that you test each site you create before you publish it. Not only do you want to verify that your site is optimally usable and accessible, but you should also seek to verify that your end result meets the goals that you set during the planning phase of the site project.

Testing and feedback

If your site does not immediately communicate what it offers to users, or if it's difficult to read, use, or navigate, it's likely that visitors will leave your site and not return. By testing your site for content, design, and usability factors, you can determine how successfully your site meets its objectives, determine how easy it is for users to find what they're looking for, and verify that the site's features work as intended in multiple browsers. A successful site test should give you confidence that your site will achieve its objectives. Testing your site also helps you find and fix any problems before the site is deployed.

Site testing typically involves multiple objective volunteers who are asked to navigate the site and to try to use its features in several browsers.

Formal site testing

In a formal site testing environment, which is typical for large sites and complex Web applications, a facilitator typically leads test participants through key areas of the site while observing the speed and ease with which the users independently complete specific objectives. User reactions can be monitored and recorded, and post-test interviews can provide further information about each user's experience with the site. Data is then collected and analyzed, and updates and improvements can be made before the site is published.

Informal site testing

Depending on the project, an informal testing environment can be just as effective. You might only need several people in a conference room or even individual participants working at their desks or at home, following general site testing instructions and recording their experiences with the site's features and components.

Provide a clear set of goals at the outset and then stand back

Whether the testing is formal or informal, you should provide participants with a clear set of goals and a consistent method for recording their experiences—even a simple notepad can do the trick. The testing participants should represent actual site visitors. The instructions you provide should set expectations and suggest what users should observe and record, but without providing specific direction about where to click, where to find content, or how to use certain features. Members of the development team should avoid providing any further direction and should let the testing participants attempt to navigate the site and solve problems independently.

Progressive enhancements

Even after you publish a Web site, the testing, feedback, and improvement process should continue. It's important that you continually monitor how your site meets the needs of its audience. After you have established a solid foundation that has been thoroughly tested, you should establish that foundation as a benchmark for later testing.

Hold onto the test results data and feedback and use that information to compare against future iterations of the site.

With your benchmark in place, you can apply the principle of *progressive enhancement* to gradually add new features and functionality to your site when multiple browsers support that functionality. You can conduct tests on each new component to verify that each new enhancement works as intended in a variety of browsers and is accessible to your full audience.

Do it!

B-1: Discussing site testing and feedback

Questions and answers

1 Why is testing critical before you publish a site?

2 What information should you provide to site testers?

3 What information should you not give during the testing process?

4 What can you do after a site is tested and published?

Unit summary: Local storage and site testing

Topic A

In this topic, you learned how to create an **editable content** section and apply a JavaScript that stores user data in **local storage**. You learned the basics of how local storage works, some possible use cases, and how local storage differs from cookies. You also learned how to control **outline** styles and use vendor-prefixed styles, and you learned how to **rotate** an element to achieve a design effect.

Topic B

In this topic, you learned some basic **site testing** and feedback principles that you can use to ensure that your site works optimally for your intended audience.

Review questions

1 True or false? To make practical use of editable content, you need to tie it to JavaScript to enable additional functionality.

2 True or false? Applying a thick outline style to an element can affect the element's size and location on the page.

3 True or false? You can use the `:focus` pseudo class to select and format an element when the user hovers the mouse pointer over it.

4 True or false? Local storage works the same as cookies.

5 True or false? You can include any number of scripts in the head or body of a document.

6 What is an event handler?

   A A script component that is included in the body of the script, inside the `<script>` container or inside the external JavaScript file

   B A script component that executes functions based on user events such as the clicking of a button or link

   C A script component that loops through an array

   D A script component that de-activates other script components

7 True or false? Event handlers are included in the HTML elements to which the events pertain.

Course summary

This summary contains information to help you bring the course to a successful conclusion. Using this information, you will be able to:

A Use the summary text to reinforce what you've learned in class.

B Determine the next courses in this series (if any), as well as any other resources that might help you continue to learn about HTML5.

Topic A: Course summary

Use the following summary text to reinforce what you've learned in class.

Unit summaries

Unit 1

In this unit, you learned how to **plan** a Web site project and you learned about HTML5 applications, trends, and authoring options. You learned how to control the browser's rendering mode, specify the **character encoding** of your documents, set the document language, **comment** your code effectively, and ensure basic HTML5 support in old versions of Internet Explorer. You also learned how to apply **semantic elements** to a page structure, and you learned some basic SEO concepts. Finally, you learned several guidelines for writing effective **page titles** and **descriptions**.

Unit 2

In this unit, you learned how to create **forms**, add a variety of **input fields**, create labels, set input properties, control autocomplete and autofocus, create **placeholders**, define option groups, create a data list, and activate the spelling check feature in form input fields. You also learned how to create **required fields**, use regular expressions to **validate** user data entries, use **fieldsets** and **legends** to group your form input fields into logical sections, and customize the appearance of fieldsets and legends. Finally, you learned some best practices in **form design** to build more effective transaction interfaces.

Unit 3

In this unit, you learned how to embed **native audio**, and you learned about different **audio formats**, including which ones are supported by various browsers. You also learned how to embed **native video** and apply various **video formats**. You then learned about several video-related attributes, and you learned how to set a **poster image** for a video. Finally, you learned how to use **inline frames** to embed externally hosted content, such as YouTube videos.

Unit 4

In this unit, you learned how to use **CSS pseudo classes** to format elements based on their location in the document structure. You also learned how to create alternating table row colors without having to change any HTML markup, and you learned some basic guidelines for creating **efficient style sheets**. You learned how to incorporate special fonts into your site by using the Google Web Fonts API, and you learned how to apply text shadow effects. Then you learned about generated content, and you learned how to use **pseudo elements** and the **content property** to add content enhancements that are not part of the document tree. You also learned how to use **attribute selectors** to select and format specific elements without having to introduce new HTML markup.

Unit 5

In this unit, you learned how to create an **editable content section** and apply a JavaScript that stores user data in **local storage**. You learned the basics of how local storage works and how local storage differs from cookies. You also learned how to control **outline styles** and use **vendor-prefixed styles**, and you learned how to **rotate an element** to achieve a design effect. Finally, you learned some basic **site testing** principles that you can use to ensure that your site works optimally for your intended audience.

Topic B: Continued learning after class

It is impossible to learn any subject completely in a single day. To get the most out of this class, you should begin applying your new skills and knowledge as soon as possible. We also offer resources for continued learning.

Next courses in this series

This is the last course in this series.

Other resources

For more information on HTML5, go to **www.Crisp360.com**. Crisp360 is an online community where you can expand your knowledge base, connect with other professionals, and purchase individual training solutions.

Glossary

API
Application Programming Interface, a method through which software components communicate with each other.

Attribute
Information that defines a specific characteristic of an HTML element.

Attribute selectors
CSS selectors used to select elements if they contain a specified attribute or attribute=value combination.

Boolean attribute
An attribute that implies either a true or false condition. In HTML5, Boolean attributes are true if they're present and false if they're not present.

Character encoding
A method of determining the alphanumeric characters that a browser can display.

Contextual selectors
Style selectors that target only those elements that are used in a particular context in the document.

DOCTYPE declaration
A statement that defines the variant of HTML used in a document.

Embedded style sheet
A style container that is defined by the <style> element, is inserted in a page's <head> section, and influences only that individual page.

Event handlers
Components of JavaScript that execute functions based on user events, such as the clicking of a button or link

Fieldset
An element you can use to group your form input fields into logical sections.

Generated content
Content added to a page from a style sheet for user interface enhancements and other stylistic purposes. Also called *CSS content* or *style content*.

Inline frame
An element that allows you to insert other documents and resources directly into a page and to control the item's location on the page just as you would with any other page element.

Local storage
A technology that allows user data to be stored directly in the browser. This data can then be delivered back to the user on later visits. Also called *Web storage*.

Margin
The space around an element's box.

Outline
A CSS style that is drawn on top of an element's box and does not add to the element's height or width.

Padding
The space between an element's content and its border (the boundary of its box).

Poster image
An image that serves as the initial image that people see before they choose to play a video. Also called a *thumbnail*.

Properties
CSS keywords that take specific formatting actions on an element.

Pseudo classes
CSS selectors used to classify elements based on characteristics that are not physically part of the document structure.

Pseudo elements
Elements that are displayed just like actual document content, but they are not actually elements within the document structure.

Regular expressions
Expressions used in scripting languages such as Perl and JavaScript. They provide a way to match strings of text, such as words, numbers, special characters, or patterns of characters. Also called *Regex* and *Regexp*.

Search engine optimization (SEO)
Techniques you can employ to improve your site's ranking and content display in the search results of search engines such as Google and Bing.

SERPs
Search engine results pages.

Semantic elements
Elements that are self-descriptive; they describe the content they contain.

Index

A

B

C

D

E

F

AX1426029942
ISBN-10 1-4260-2994-2
ISBN-13 978-1-4260-2994-3

TO LEARN MORE ABOUT THE ILT SERIES, VISIT US AT WWW.LOGICALOPERATIONS.COM