HP doesn’t format and present text directly. Usually, PHP programs create HTML pages as their output. Also, most PHP programs get their parameters from HTML forms. It’s important to ensure you can build HTML code by hand, because your code must generate HTML. This book—and PHP in general—rely on some advanced HTML skills that you may not have needed when you were building basic HTML documents. This appendix reviews the critical HTML commands including tables, forms, and cascading style sheets (CSS).

- Review HTML commands
- Use cascading style sheets to enhance your Web pages
- Build HTML forms
Building Basic HTML Pages

The basic unit of Web development is the HTML page. This text document contains special tags that describe the page’s data. Although you might already be familiar with HTML, it makes sense to review these skills; PHP programming is closely tied to HTML.

I strongly urge you to use a plain text editor as you begin. You can use Notepad or one of the many free editors available. There are some exceptional free editors available on the CD that accompanies this book. Word processors usually do not save files in plain text format (which PHP and HTML require) and many of the fancy Web editors (such as FrontPage or Dreamweaver) tend to write clunky code that gets in your way once you start to add programming functionality.

Creating the Hello, World! Page

HTML is mainly text. The Web author adds special markups to a text document to indicate the meaning of various elements. When a user requests a Web page, the text document is pulled from the Web server and the browser interprets the tags to determine how to display the document. Figure A.1 illustrates a very simple Web page.

If you look at the code you see that it’s pretty easy to understand, even if you aren’t terribly familiar with HTML code.

```html
<html>
<head>
<title>Hello, World</title>
</head>

<body>
<center>
<h1>Hello, World!</h1>
This is my first HTML page
</center>
</body>
</html>
```
Many words are encased in angle braces: `<>`. These words are called tags and they are interpreted as instructions. Most tags come in pairs. For example, the entire document begins with `<html>` and ends with `</html>`. The slash (`/`) indicates an ending tag.

Each HTML document contains a head area surrounded with a `<head></head>` pair. The header area contains information about the document in general. It almost always contains a title, which is often displayed in the Web browser’s title bar. However, there are no guarantees. HTML tags describe the meaning of an element, not necessarily how it is to be displayed. It’s up to each browser to determine how something will be displayed.

The bulk of an HTML document is contained in the body, indicated with the `<body></body>` tags.

You can use tags within the body to define various page characteristics. Usually, you can guess at the meanings of most of the tags. For example, the `<center></center>` pair causes all the text between the tags to be centered (if the browser can support this feature).

It’s vital to understand that HTML tags are not commands to the browser as much as suggestions. This is because there are so many different types of computers and Web browsers available. It’s possible that somebody might look at your Web page on a palm-sized computer or a cell phone. These devices will be unable to display information in the same way as full-size computers. The Web browser tries to follow your instructions, but ultimately, the way the page looks to the end user is not under your direct control.
The `<h1>` tags designate that the text contained between the tags is a level-one (highest priority) heading. HTML supports six levels of heading, from `<h1>` to `<h6>`. You can’t be exactly sure how these headings will appear in a user’s browser, but any text in an `<h1>` pair will be strongly emphasized, and each descending head level causes the text designated by that code to have less and less emphasis.

**Basic Tags**

HTML has a number of tags. Most are used to determine the meaning of a particular chunk of text. Table A.1 illustrates some of these tags.

Of course, there are many other HTML tags, but those featured in Table A.1 are the most common. Figure A.2 illustrates several of Table A.1’s tags.

The source code for the `basic.html` document illustrates how the page was designed.

```html
<html>
<head>
<title>Basic HTML Tags</title>
</head>
<body>
<h1>Basic HTML Tags</h1>
<h1>This is an h1 header</h1>
<h2>This is an h2 header</h2>
<h3>This is an h3 header</h3>
<h4>This is an h4 header</h4>
<h5>This is an h5 header</h5>
<h6>This is an h6 header</h6>
<center>
This text is centered
</center>
<b>This is bold</b>
<br>
<i>This is italic</i>
<hr>
</body>
</html>
```
<table>
<thead>
<tr>
<th>Tag</th>
<th>Meaning</th>
<th>Discussion</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;b&gt;</code></td>
<td>Bold</td>
<td>Won’t work on all browsers</td>
</tr>
<tr>
<td><code>&lt;i&gt;</code></td>
<td>Italics</td>
<td>Won’t work on all browsers</td>
</tr>
<tr>
<td><code>&lt;h1&gt;</code></td>
<td>Level 1 header</td>
<td>Strongest headline emphasis</td>
</tr>
<tr>
<td><code>&lt;h6&gt;</code></td>
<td>Level 6 header</td>
<td>Weakest headline level (levels 2-5 also supported)</td>
</tr>
<tr>
<td><code>&lt;ul&gt;</code></td>
<td>Unordered list</td>
<td>Must contain list items (<code>&lt;li&gt;</code>)&lt;/li&gt;)</td>
</tr>
<tr>
<td><code>&lt;li&gt;</code></td>
<td></td>
<td>Used for bulleted lists</td>
</tr>
<tr>
<td><code>&lt;ol&gt;</code></td>
<td>Ordered list</td>
<td>Must contain list items (<code>&lt;li&gt;</code>)&lt;/li&gt;)</td>
</tr>
<tr>
<td><code>&lt;a href = &quot;anotherPage.html&quot;&gt; go to another page&lt;/a&gt;</code></td>
<td>Anchor (hyperlink)</td>
<td>Places a link on the page. Text between <code>&lt;a&gt;</code> and <code>&lt;/a&gt;</code> is visible on page as a link; when user clicks link, browser goes to the specified address</td>
</tr>
<tr>
<td><code>&lt;img src = &quot;imgName.gif&quot;&gt;</code></td>
<td>Image</td>
<td>Adds the specified image to the page; Images should be in GIF, JPG, or PNG formats</td>
</tr>
<tr>
<td><code>&lt;font color = &quot;red&quot; size = 5&gt; this text is red&lt;/font&gt;</code></td>
<td>Modify font</td>
<td>Will not work in all browsers; it's possible to modify font color, size, and face (typeface), although typeface will often not transfer to client machine</td>
</tr>
<tr>
<td><code>&lt;br&gt;</code></td>
<td>Break</td>
<td>Causes a carriage return in the output; does not have an ending tag</td>
</tr>
<tr>
<td><code>&lt;hr&gt;</code></td>
<td>Horizontal rule</td>
<td>Add a horizontal line to the page; does not have an ending tag</td>
</tr>
</tbody>
</table>

The `h1` through `h6` headers create headlines of varying size and emphasis. The `<b>` tag causes text to be bold, and `<i>` formats text in italics. Finally, the `<hr>` tag draws a horizontal line on the page.
More HTML Tags

The rest of the tags shown in Table A.1 are featured in Figure A.3.

The tags in more.html add lists, links, and images to a Web page. The code used to produce this page looks like this:
HTML supports two types of lists. The `<ol>` set creates *ordered* (numbered) lists. Each element in the list set (specified by an `<li>` pair) is automatically numbered. The `<ul>` tags produce unnumbered lists. Each `<li>` element is automatically given a bullet.

*Hyperlinks* are the elements that allow your user to move around on the Web by clicking specially designated text. The `<a>` tag designates a hyperlink. The `<a>` tag almost always includes an `href` attribute, which indicates an address. The
user is redirected to the indicated address when he clicks the link. The text (or other html) between the `<a>` and `</a>` tags is designated as the hyperlink. That text appears on the page as a link (usually blue and underlined). The more.html example has a link to one of my home pages (http://www.cs.iupui.edu). When the user clicks the Andy's Home Page link in the browser, she is transported to that page.

The other feature, the `<img>` tag, includes images into a Web page. Most browsers readily support .gif and .jpg files, and many now can support the newer .png format.

If you have an image in some other format or an image that needs to be modified before use, use free software such as irfanView or the Gimp (both included on the CD that accompanies this book).

### Tables

You might be working with large amounts of information that could benefit from table-style organization. HTML supports a set of tags for use in building tables. These tags are illustrated in Figure A.4.

![FIGURE A.4](image)

Tables can be basic or complex, with cells occupying multiple rows and columns.

The code for the simpler table looks like this:

```html
<table border = "1">
```
Tables are created with the `<table></table>` tags. Inside these tags, you create rows using the `<tr></tr>` (table row) tags. Each table row can contain table heading (`<th></th>`) or table data (`<td></td>`) elements.

The Web browser ignores spaces and indentation, but it's very smart to use whitespace in your HTML code, which makes it easier to read. Notice how I indented all elements inside each table row. This makes it much easier to see that all the information within the `<tr></tr>` set is part of one group.

In the `<table>` tag, you can use the `border` attribute to indicate border thickness.

Browsers are inconsistent in their default values. If you don’t specify the border width, some browsers show a border and some show no border at all. It’s best to specify a border width every time. If you don’t want a border, set the width to 0.
Sometimes you need table cells to take up more than one row or column. The code for the second table in `table.html` shows how to accomplish this:

```html
<table border = "4">
    <tr>
        <th></th>
        <th>Monday</th>
        <th>Tuesday</th>
        <th>Wednesday</th>
        <th>Thursday</th>
        <th>Friday</th>
    </tr>
    <tr>
        <th>Morning</th>
        <td>One</td>
        <td colspan = "2"><center>Two</center></td>
        <td>Three</td>
        <td rowspan = "2">Four</td>
    </tr>
    <tr>
        <th>Afternoon</th>
        <td>A</td>
        <td>B</td>
        <td>C</td>
        <td>D</td>
    </tr>
</table>
```

Notice that the cell containing the value *Two* has its `colspan` attribute set to 2. This tells the cell to take up two cell widths. Since this cell is twice as wide as normal, it is only necessary to define five `<td>` or `<th>` elements for this row instead of the six elements used for each row of the simpler table.

Look also at the cell containing the value *Four*. This cell takes up two rows. I used the `rowspan` attribute to set up this behavior. Notice that I needed fewer elements in the next row, because one of the columns is taken by this expanded element.
Enhancing Your Pages with CSS

Basic HTML is easy to write, but it creates dull pages. Modern browsers support cascading style sheets (CSS) elements, which specify how to display a particular tag. Entire books have been written about CSS, but the basic ideas are reasonably simple. You can define a style, which is a set of formatting rules, and attach it to various elements in your pages. An example clears things up.

Creating a Local Style

Figure A.5 illustrates a Web page with some features that are not available in normal HTML.

The h2 tag does not normally generate blue text, but I added a style to the text to make it blue. The code for the blue headline looks like this:

```html
<h2 style = "color:blue">
This H2 has a custom style turning it blue
</h2>
```

I added a style attribute to the <h2> tag. This style attribute has a number of options that can be set. The color option allows you to assign a color to a style. The object that uses that style appears in that color.

Many other style options are available. The larger paragraph in Figure A.5 uses a number of them. The code for that paragraph appears here:
This paragraph has a custom style. The custom style adds characteristics such as background color and border that aren't ordinarily available in HTML. Also, the font size can be specified in points. </p>

You can see that this paragraph tag has a more complex style attribute with a number of elements. Each element has a name and a value separated by a colon; semicolons separate the elements. A list of the most commonly used style elements is shown in Table A.2.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Possible values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color</td>
<td>Foreground color</td>
<td>Valid color names (blue), hex color values (0000FF)</td>
</tr>
<tr>
<td>Background color</td>
<td>Background color</td>
<td>Valid color names, hex color values</td>
</tr>
<tr>
<td>Font family</td>
<td>Font to show</td>
<td>Font name must be installed on client computer</td>
</tr>
<tr>
<td>Font size</td>
<td>Size of font</td>
<td>Can be described in pixels (px), points (pt), centimeters (cm), or inches (in)</td>
</tr>
<tr>
<td>Border width</td>
<td>Size of border</td>
<td>Usually measured in pixels (px), centimeters (cm), or inches (in)</td>
</tr>
<tr>
<td>Border style</td>
<td>How border will be drawn</td>
<td>Some choices are groove, double, ridge, solid, inset, outset</td>
</tr>
<tr>
<td>Border color</td>
<td>Color of border</td>
<td>Valid color names (blue), hex color values (0000FF)</td>
</tr>
</tbody>
</table>
Page-Level Styles

Although it is sometimes convenient to attach a style directly to an HTML element, sometimes you wish to modify a number of elements in a particular page. You can specify the default styles of several elements by adding a style element to your document. Figure A.6 illustrates how this page-level style works.

With page-level styles, you use a <style></style> segment in your document header to specify how each listed tag should be displayed. The code for the pageStyle.html page illustrates how a page-level style sheet can be created:

```html
<html>
<head>

<code>
</head>
</html>
```
The main body looks pretty much like normal HTML code (because it is). The interesting part of this page is the code between the <style> and </style> tags. This code describes how the various tags should be displayed.

Your opening tag should read <style type = "text/css"> to specify you’re using an ordinary style sheet. Inside the style element, list each tag you wish to define. After the tag name, encase the various stylistic elements in a pair of braces: {}.
The style elements are listed just like in the style attribute. Each element consists of a name/value pair. A colon separates the name and value, and a semicolon separates each pair.

Notice the second paragraph element, which looks like this:

```css
p.cursive {
    background-color: yellow;
    font-family: cursive
}
```

Like most HTML programming, the style element is not picky about where you have spaces or carriage returns. However, judicious use of these whitespace elements can make your code much easier to read and modify. I lined up each element so they are easy to read. I aligned the closing brace (}) directly under the HTML element's name to easily see how the various parts of code are related. You'll see the same kind of attention to indentation throughout your programming career.

By adding a period and another name (in this case, .cursive) to the HTML element's name, I created a second type of paragraph tag. You can create as many variations of a tag as you wish. This is especially handy if you want varying text styles. You might want one kind of paragraph for quotes, for example, and another kind for ordinary text. To use the special tag form, use the class attribute in the HTML:

```html
<p class = cursive>
This is a cursive paragraph
</p>
```

### External Style Sheets

Most Web browsers support a third kind of style sheet, called the *external style sheet*. Figure A.7 illustrates a page using an external style sheet.

The user cannot tell what type of style sheet was used without looking at the code. Although the external style example looks much like the page-level style sheet program, the underlying code is different. Here is the code for `externStyle.html`:

```html
<html>
<head>
<link rel="stylesheet"
    type="text/css"
    href = "externStyle.css">
</head>
```
The main code is identical to that in the pageLevel program, but the style sheet is not embedded directly into the document. Instead, the style is stored in another file called externStyle.css. The contents of this file are the same style rules in the earlier page:

```css
h1 {
    text-align:center;
    color:green;
    border-color:red;
}
```
border-style:double;
border-size: 3px
}

p {
  background-color: yellow;
  font-family: monospace
}

p.cursive {
  background-color: yellow;
  font-family: cursive
}

When you have the CSS rules stored in a separate file, you can use the link tag to import them. The advantage of this approach is being able to reuse one set of CSS rules for many pages.

**In the Real World**

External style sheets are very useful when working on a project that must be consistent across many pages. Most sites go through several iterations, and it could be a real pain to change the font color in 20 pages every time the client wants to try some new variation. If all your style rules are stored in one CSS document and all your pages refer to that document, you only have to change the style rules one time.

**Using Form Elements**

HTML pages often utilize form elements for user input. These elements include basic tools for user input and are useless in plain HTML. Although they are rather easy to put on a page, they don’t do much unless some kind of program is attached. Much of what you do as a PHP author involves getting information from Web-based forms, so it’s important to be familiar with the most common form elements.

**The Text-Based Elements**

Most of the form elements are about getting some sort of text information from the user to a program. The first set of such elements are those that simply allow the user to enter text. There are four such elements, illustrated in Figure A.8.
The code that generates **textForm.html** is reproduced here:

```html
<html>
<head>
<title>Text-Based Form Elements</title>
</head>
<body>
<h1>Text-Based Form Elements</h1>
<form>
text box:
<input type = "text"
     name = "txtInput"
     value = "your text here">
<br>

text area:
<textarea name = "txtBigInput"
         rows = 10
         cols = 40>
This is the stuff inside the textarea
</textarea>
</form>
</body>
</html>
```

**FIGURE A.8**

You can add text boxes, text areas, password boxes, and hidden fields to your Web pages.
**Creating a Text Box**

The most common input element of all is the humble text box. To make a plain-vanilla text box, I used the following code:

```html
<input type = "text"
    name = "txtInput"
    value = "your text here">
```

The element is a basic input element. By setting the type to text, I signify how the element is to be displayed on the screen—as something that the user can type text into. An input element using the text type is usually called a text box. Text boxes cannot include multiple lines of text, but you can specify the length of the text box with the size attribute. (If you set the size to 20, you are allowing for roughly 20 characters.)

It is important to add a name attribute to your text boxes (and indeed to all form elements) because you later write programs that try to retrieve information from the form. These programs use the various form element names to refer to what the user inserted.
Naming an input element is something of an art form. The name should be reasonably descriptive. (or albert is usually not good, because they don’t explain what kind of information is expected to be in the object). Object names should not have spaces.

The value attribute is used to set a default value for the text area. This value appears in the text area when the user first sees your form. It’s a good idea to put default values in forms when you can, because this gives you a chance to show the user what kind of information you’re expecting.

Creating a Text Area

Text boxes are very handy, but sometimes you want to let the user type in more than one line’s worth of information. For example, you might want to have a feedback page where the user can type in some comments. For this, you usually want to use an object called the text area. The code looks like this:

```
<textarea name = "txtBigInput"
    rows = 10
    cols = 40>
This is the stuff inside the textarea
</textarea>
```

The text area is created using a pair of `<textarea></textarea>` tags. The text area has a name attribute, as well as attributes for determining the size of the text box in rows and columns. Text areas should also be named using the name attribute, but the textarea object does not have a value attribute. Instead, anything between the `<textarea>` and `</textarea>` tags is considered the contents of the text area object.

Don’t forget to close the text area with a `</textarea>` tag. If you don’t, everything in the page after the `<textarea>` tag appears inside the text area—if the page renders at all!

Building a Password Field

Password fields are almost identical to text boxes. The code for creating a password is very much like the text field:

```
<input type = "password"
    name = "secret"
    value = "you can't read this">
```
The value typed into a password field is shown as asterisks onscreen. Presumably this keeps the KGB from peering over the shoulders of your users while they type passwords into your pages.

The password field offers virtually no real security. Information sent to the server via a password field is transmitted entirely in the clear, so it is only nominally secret.

Making a Hidden Field
Believe it or not, the text box has an even more secretive cousin than the password field. The hidden field is much like the text box in code, but it doesn’t appear on the page at all. Here’s how the code looks:

```html
<input type = "hidden"
    name = "mystery"
    value = "secret formula">
```

Its uses might not be obvious now, but it does come in handy when you want your page to communicate with a server-side program but don’t need the user to know all the details.

Creating the Selection Elements
It’s very easy to add text elements to your Web pages, but requiring users to enter text can interrupt the program flow. Whenever possible, experienced programmers like to give the user choices that do not involve typing. HTML forms have a number of simple elements for allowing the user to choose from a list of options.

Making the user’s life easy is a good reason to use some of these other input features, but there’s another reason. You never know what a user will enter into a text box. It can be difficult to write code that anticipates all the possible wrong things a user can enter. If you use the various selection elements described here, you predetermine all possible values your program will need to deal with (at least in most circumstances).

Figure A.9 shows a number of these selection-style elements on a Web page.
Creating Checkboxes

The first type of input to consider is the checkbox. Checkboxes usually look like, well, boxes that can be checked. Usually there is some kind of text near the checkbox. The box can be selected or deselected. Here’s the code used to create the checkboxes in the selectForm.html page:

```html
<input type = "checkbox" name = "chkBurger">cholesto-burger
<input type = "checkbox" name = "chkFries">fries
<input type = "checkbox" name = "chkDrink">drink
```

A checkbox is simply an input element of type checkbox. You can specify a checkbox’s value attribute, but it isn’t always necessary. Note that the caption next to the checkbox is plain HTML text. Each checkbox is a completely independent entity. Even though several checkboxes appear together in the HTML document, the value of one checkbox has no bearing on the value of any other.

Checkboxes are appropriate when any combination of the various elements is appropriate. For example, the user might want the burger, fries, and a drink. The user might want none of these things, or any combination. Checkboxes are not as appropriate when the options are mutually exclusive. For example, if asking what size a drink should be, only one size should be allowed per drink. That kind of situation is a perfect place to use another feature called radio buttons.
Selecting with Radio Buttons

You can use radio buttons (sometimes called option buttons) to let the user choose an item from several options. Radio buttons get their name from the radios on cars that had several buttons sticking out (at least they did when I was a kid). To select a station, you pressed the corresponding button in, which caused all the other buttons to pop out. HTML radio buttons have similar behavior. Radio buttons are grouped so that when you select one button, all the others in the group are automatically deselected.

Look at the code for the radio buttons and spot how the radio elements are grouped:

```html
<input type = "radio"
    name = "size"
    value = "small">small
<input type = "radio"
    name = "size"
    value = "medium">medium
<input type = "radio"
    name = "size"
    value = "large">large
```

The interesting thing about radio buttons is the way they are named. There are three radio buttons in this example, but they all have the same name. This trick groups the radio buttons so they act as expected. As soon as the user selects one item in a radio group, all other radio elements on the page with the same name are automatically deselected. Each of the radio objects has a distinct value. Your programs determine the value of whichever radio button in the group was selected.

Building Drop-Down List Boxes

Another common trick is to use some kind of drop-down list. These allow the user to choose from a list of options, but the various options only appear when the user is choosing from the list. This is especially useful when screen real estate is an issue or you want to keep the interface clean.

Drop-down lists are made with two different elements. The main object is select, which contains a series of option objects. (This is analogous to the way li objects appear inside a ul or ol object.) The code for building a drop-down list box makes it clear:

```html
<select name = "selColor">
    <option value = "red">red</option>
</select>
```
The select object has a name attribute. Each option has its own value attribute. Your program uses the value attribute of whichever element is returned. An option button’s value property doesn’t display anywhere. Place the text you want to have visible on the page between the <option> and </option> tags.

Creating a Multi-select List Box

One more selection element can be useful in certain situations. This isn’t really a new object at all, but a variation of the drop-down list. Here’s code for the last element in selectForm.html:

```html
<select name = "lstColor"
    size = 7
    multiple>
    <option value = "red">red</option>
    <option value = "orange">orange</option>
    <option value = "yellow">yellow</option>
    <option value = "green">green</option>
    <option value = "blue">blue</option>
    <option value = "indigo">indigo</option>
    <option value = "violet">violet</option>
</select>
```

The code looks identical to the previous (drop-down) list except for a few differences in the select tag itself. By setting the size attribute to a value of 7, I indicated that seven lines of the list should be shown at any time. This is useful when you want the user to see all (or many) of the choices all the time. The other interesting thing about this type of list box is it can allow for multiple selections if you include the multiple attribute. A multi-selection list box lets the user choose more than one element using standard multiple selection rules (for example, Shift+Click to select contiguous options or Ctrl+Click to add or remove a particular element from range).
Adding Buttons to Your Programs

The last major form element is the button. Buttons are important because the user is accustomed to clicking them to make things happen. Your programs should take advantage of this conditioning. Figure A.10 shows a page containing three distinct buttons.

![Figure A.10](image)

Although these buttons all look very similar to the user, they have distinctively different behaviors.

All three button types are variants of the basic \texttt{input} tag you’ve used so much in this appendix. The code for the \texttt{buttonForm.html} page illustrates this clearly:

```html
<html>
<head>
<title>Button Demo</title>
</head>
<body>
<h1>Button Demo</h1>
<form>
<textarea rows = 5>
Change the text here to see what happens when you click on the reset button.
</textarea>
<br><br>
</form>
</body>
</html>
```
The three different types of buttons look the same but behave differently.

- **Generic button.** When you set an input element’s type attribute to button, you create a generic button. These buttons are frequently used in client-side programming. To make something happen when the user clicks such a button, embed code in your Web page using a language such as JavaScript or VBScript. (Of course, there are exceptional books in the *Absolute Beginners*’ series describing exactly how to do this.) Server-side programming (which is this book’s focus) rarely involves the ordinary button object.

- **Reset button.** The reset button lets the user reset the page to its default condition. This is a handy feature, because it lets the user back up if the page gets messed up. It isn’t necessary to write any code for the reset button, because the browser automatically handles the resetting behavior.

- **Submit button.** This style is by far the most important kind of button for server-side programming. The submit button provides the link between Web pages and your programs. Most interactions in server-side programming involve sending an HTML page with a form to the user. The user presses the submit button when the user has finished making selections and typing values into the various form elements. This essentially bundles all the data in the form elements and sends them to a program.
Summary

You’ve already come a very long way. You’ve learned or reviewed all the main HTML objects. You’ve investigated CSS and how to use them to modify an HTML attribute. You experimented with the main form elements and learned how to add various kinds of text boxes and selection devices to your Web pages.

CHALLENGES

1. Create a Web-based version of your resume incorporating headings, lists, and varying text styles.

2. Modify one of your existing pages so that it incorporates CSS styles.

3. Build an HTML form that includes at least one instance of each of the following elements:
   - text box
   - list box
   - radio button group
   - check box
   - submit button