TUTORIAL 7

Designing a Web Form

Creating a Survey Form

OBJECTIVES

Session 7.1
- Explore web forms
- Work with form servers
- Create forms and field sets
- Create labels and input boxes
- Explore form layout

Session 7.2
- Work with date and time fields
- Create a selection list
- Create option buttons
- Create check boxes and text area boxes

Session 7.3
- Create spinners and range sliders
- Use data lists
- Create form buttons
- Validate a form
- Apply validation styles

Case | *Red Ball Pizza*

Alice Nichols is a manager at *Red Ball Pizza*, a popular pizzeria in Ormond Beach, Florida. She wants to conduct an online survey of Red Ball customers using a web form that will be placed on the restaurant's website. She has asked you to help design a prototype for the survey form. The form should record customer information, as well as each customer's perception of his or her last experience at the restaurant. Alice wants the form to include different tools to ensure that each user enters valid data. Once a customer completes the form, the information will be sent to the Red Ball server for processing and analysis.

STARTING DATA FILES

tutorial
rb_survey_txt.html
rb_forms_txt.css
+ 9 files

review
rb_build_txt.html
rb_customer_txt.html
rb_validate_txt.css
+ 15 files

code1
code7-1_txt.html
+ 2 files

code2
code7-2_txt.html
+1 file

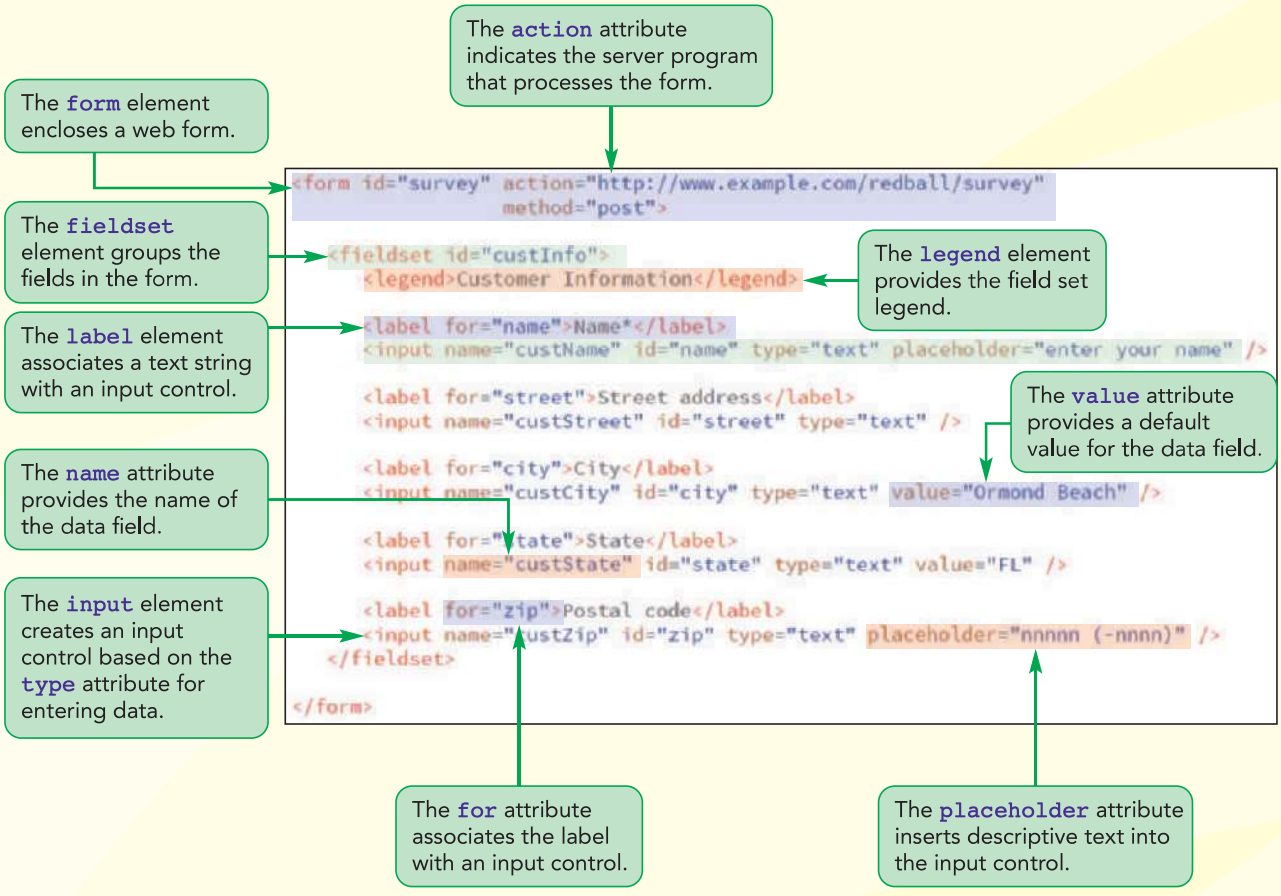
code3
code7-3_txt.html
code7-3_txt_valid.css
+ 2 files

code4
code7-4_txt.html
+ 1 file

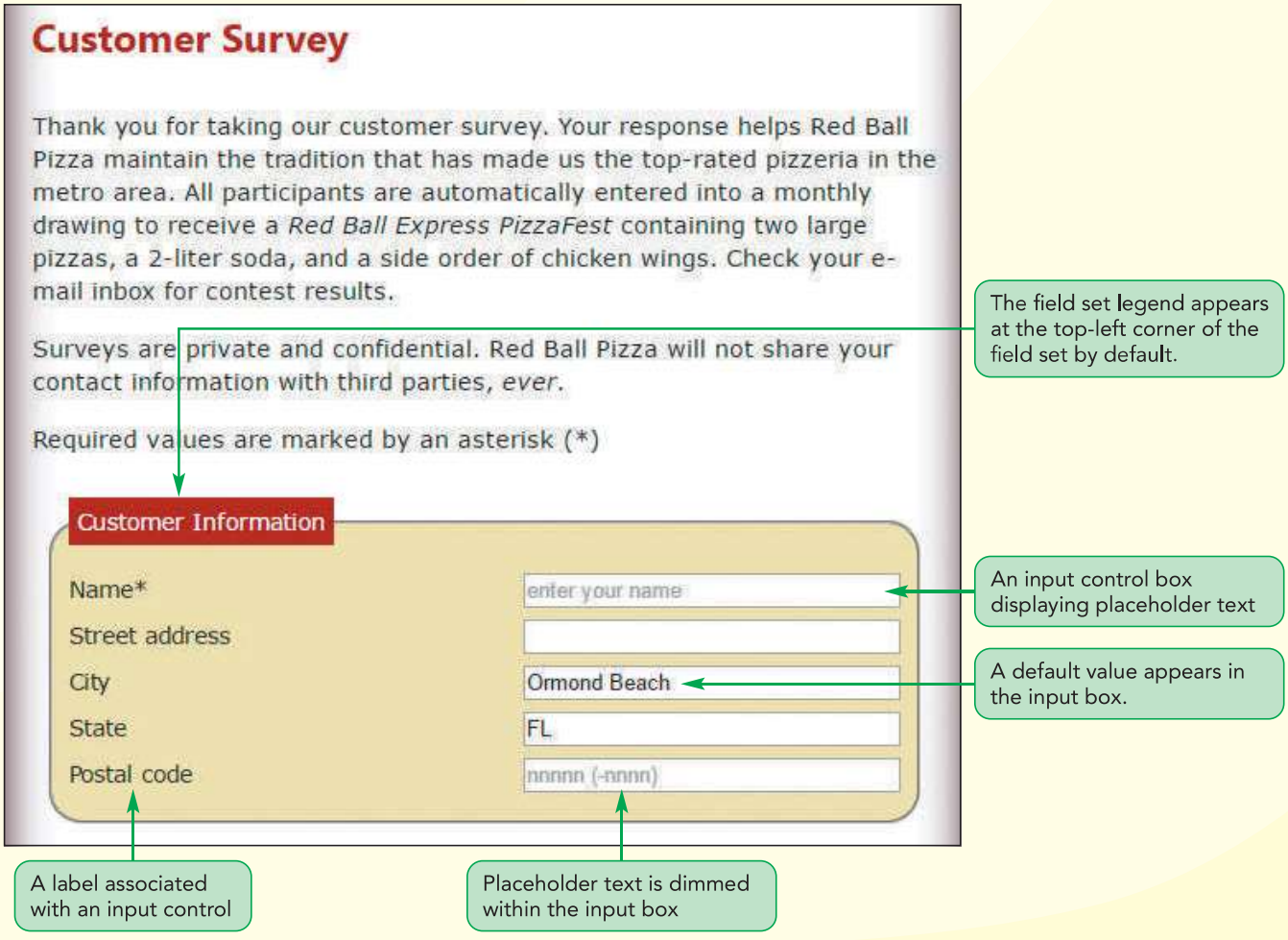
case1
cg_register_txt.html
cg_validate_txt.css
+ 9 files

case2
sb_payment_txt.html
sb_validate_txt.css
+ 16 files

Session 7.1 Visual Overview:



Structure of a Web Form



Introducing Web Forms

So far, the websites you have created have been passive: allowing the user to view information but not allowing the user to directly interact with the page's content, except via hyperlinks. Starting with this tutorial, you will begin working with more interactive websites that allow for user feedback. The most common way of accepting user input is through a **web form**, which allows users to enter data that can be saved and processed.

Parts of a Web Form

A web form contains **controls**, also known as **widgets**, which are the objects that allow the user to interact with the form. HTML supports several types of controls and widgets, including:

Controls

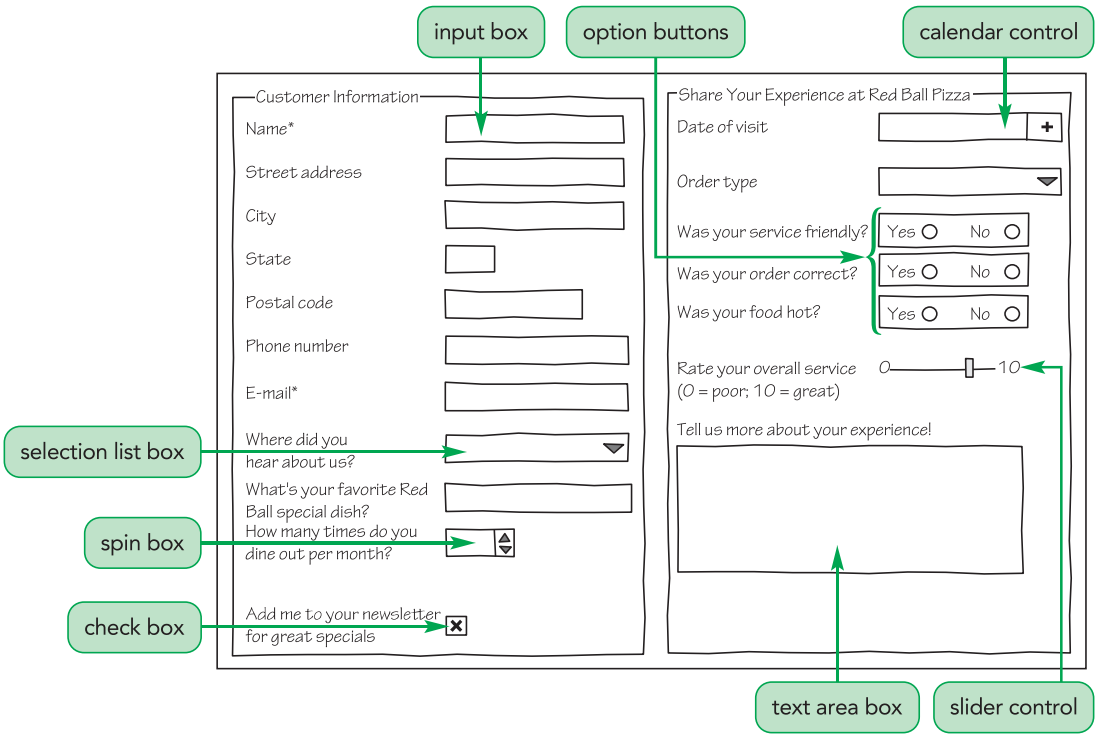
- **input boxes** for inserting text strings and numeric values
- **option buttons**, also called **radio buttons**, for selecting data values from a small predefined set of options
- **selection lists** for selecting data values from a more extensive list of options
- **check boxes** for selecting data values limited to two possibilities, such as "yes" or "no"
- **text area boxes** for entering text strings that may include several lines of content

Widgets

- **spin boxes** for entering integer values confined to a specified range
- **slider controls** for entering numeric values confined to a specified range
- **calendar controls** for selecting date and time values
- **color pickers** for choosing color values

Figure 7–1 shows Alice's sketch of the web form she wants you to create.

Figure 7–1 Proposed survey form



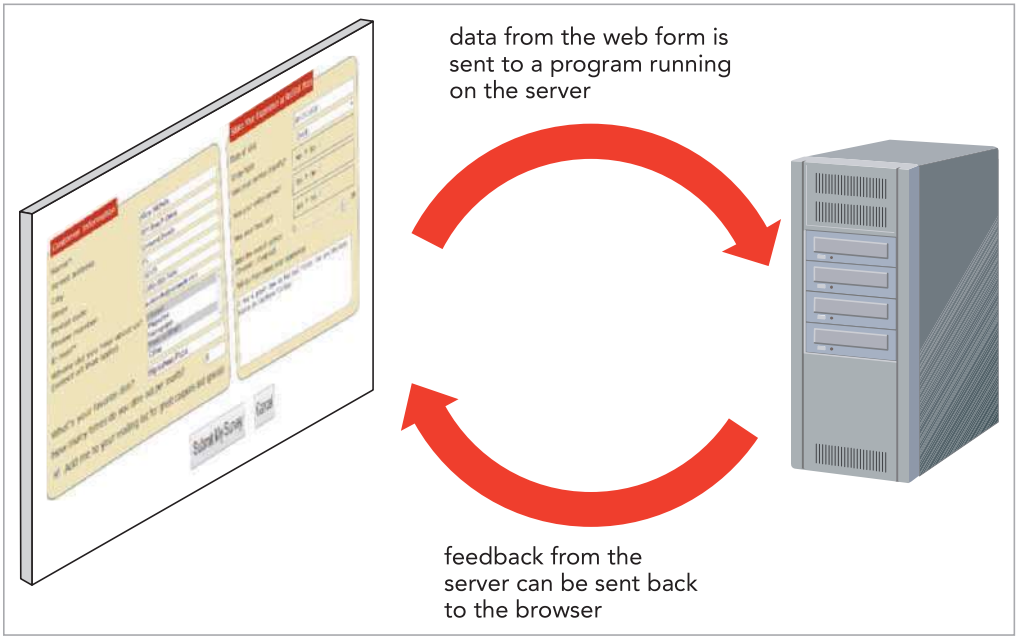
Alice's proposed form includes several of the controls discussed above, such as input boxes for entering the customer name, contact information, and email address; option buttons for storing the customer's service experience; and a selection list from which customers can choose how they heard about Red Ball Pizza from a long list of options.

Each data entry control is associated with a **data field** or **field** in which data values supplied by the user are stored. For example, the input box in which a customer enters his or her name is associated with the custName field, the calendar control in which the customer enters the date he or she visited Red Ball Pizza is associated with the visitDate field, and so forth.

Forms and Server-Based Programs

Once the field values have been entered by the user, they are processed by a program running on the user's computer or on a web server in a secure location. For example, a web form is used to collect sales data from the customer for an order and the server program processes that data and handles the billing and delivery of the sales items. See Figure 7–2.

Figure 7–2 Interaction between the web form and the server



Alice is already working with a programmer on a web server program that will store and interpret the survey results. You will not have access to that program, so Alice just wants you to concentrate on the design of the web form. Your colleagues will test your form to verify that the information is being collected and processed correctly by the web server.

INSIGHT

Restricting Access to Web Server Programs

Since the web form designer might not have permission to create or edit the programs running on the web servers, he or she will usually receive instructions about how to interact with the server programs. These instructions often include a list of fields that are required by the program and a description of the types of values expected in those fields.

There are several reasons to restrict direct access to these programs. The primary reason is that, when you run a server-based program, you are interacting directly with the server environment. Mindful of the security risks that computer hackers present and the drain on system resources caused by large numbers of programs running simultaneously, system administrators are understandably careful to maintain strict control over access to their servers and systems.

Server-based programs are written in a variety of languages. The earliest and most common of these programs is **Common Gateway Interface (CGI)**, which are scripts written in a language called **Perl**. Other popular languages widely used today for writing server-based programs include ASP, ColdFusion, C, Java, PHP, Python, and Ruby. You can check with your ISP or system administrator to find out what programs are available on your web server, and what rights and privileges you have in accessing them.

Starting a Web Form

TIP

HTML also supports the `name` attribute for uniquely identifying forms.

All web forms are marked using the following `form` element

```
<form id="text" attributes>
   content
</form>
```

where the `id` attribute identifies the form (which is important when more than one form is being used on the web page), *`attributes`* specify how the form should be processed by the browser, and *`content`* is the form's content. Forms typically contain many of the controls that were listed earlier, but they also can contain page elements such as tables, paragraphs, inline images, and headings. A `form` element can be placed anywhere within the body of the page.

REFERENCE

Inserting a Web Form

- To insert a web form, add

```
<form id="text" attributes>
   content
</form>
```

where *`text`* identifies the form, *`attributes`* control how the form is processed, and *`content`* is the content of the form.

Add a form to Alice's survey page now with the ID survey.

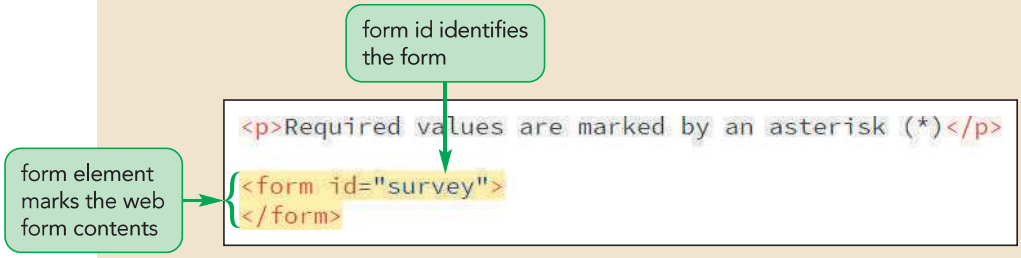
To insert a web form:

1. Use your editor to open the **rb_survey_txt.html** file from the html07 ▸ tutorial folder. Enter ***your name*** and ***the date*** in the comment section of the file and save it as **rb_survey.html**.
2. Scroll down and, directly after the third paragraph in the `section` element, insert the following `form` element:

```
<form id="survey">
</form>
```

Figure 7–3 shows the placement of the web form.

Figure 7–3 Inserting a web form



Next, you will include attributes that tell the browser how the form should interact with the web server.

Interacting with the Web Server

To specify where to send the form data and how to send it, include the following `action`, `method`, and `enctype` attributes

```
<form action="url" method="type" enctype="type">
   content
</form>
```

where the `action` attribute provides the location of the web server program that processes the form, the `method` attribute specifies how the browser should send form data to the server, and the `enctype` attribute specifies how the form data should be encoded as it is sent to the server.

The `method` attribute has two possible values: `get` and `post`. The default is the **get method**, which tells the browser to append the form data to the end of the URL specified in the `action` attribute. The **post method** sends the form data in its own separate data stream. Each method has its uses, but the post method is considered to be a more secure form of data transfer. Your website administrator can supply the necessary information about which of the two methods you should use when accessing the scripts running on the server.

The `enctype` attribute has three possible values summarized in Figure 7–4.

Figure 7–4 Values of the enctype attribute

Value	Description
application/x-www-form-urlencoded	The default format in which the data is encoded as a long text string with spaces replaced by the + character and special characters (including tabs and line breaks) replaced with their hexadecimal code values
multipart/form-data	The format used when uploading files in which no encoding of the data values occurs
text/plain	The format in which data is transferred as plain text with spaces replaced with the + character but no other encoding of the data values occurs

Alice tells you that your survey form will be processed by the CGI script using the `action` attribute accessing a server program located at the fictional URL address http://www.example.com/redball/survey with the post method. You do not have to specify a value for the `enctype` attribute because the default value of application /x-www-form-urlencoded is sufficient. Add this information to your web form.

To specify how the form interacts with the server:

1. Return to the opening `<form>` tag and add the following attributes:

```
action="http://www.example.com/redball/survey"
method="post"
```

Figure 7–5 highlights the newly added form attributes.

Figure 7–5 Associating the web form with an action and a method

```
<p>Required values are marked by an asterisk (*)</p>

<form id="survey" action="http://www.example.com/redball/survey" method="post">
</form>
```

location of server processing the form

method by which the form data is sent to the server

2. Save your changes to the file.

Because http://www.example.com/redball/survey does not correspond to a real CGI script running on the web and thus cannot process the survey form you will create in this tutorial, you will add a JavaScript program named rb_formsubmit.js to handle the form. The purpose of this JavaScript program is to intercept the content of the form before the browser attempts to contact the CGI script and report whether or not the data contained in the survey form has been correctly filled out. The JavaScript program has already been created, so you will create a link to the file using the following `script` element:

```
<script src="rb_formsubmit.js"></script>
```

A `script` element is an HTML element used to access and run JavaScript programs that will run within the user's browser. You will learn more about scripts and their applications in Tutorial 9, but for now, you add this code to the document head so that it can be applied throughout this tutorial.

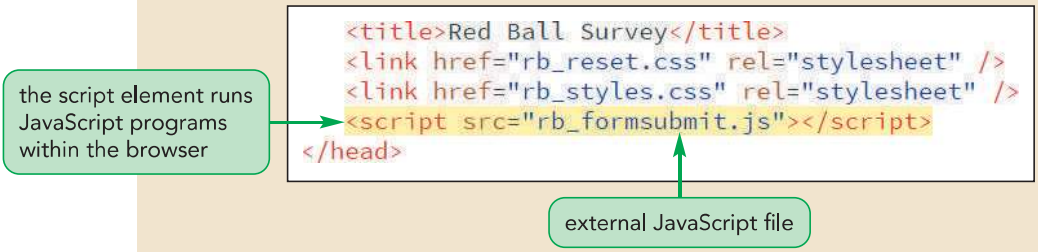
To insert a script:

1. Scroll up to the document head and insert the following code directly above the closing `</head>` tag:

```
<script src="rb_formsubmit.js"></script>
```

Figure 7–6 highlights the code for the `script` element.

Figure 7–6 Using a script to manage the form submission



2. Save your changes to the file.

Now that you have added the `form` element to the survey page, you can start populating the survey form with controls and other form features. You will start by adding field sets.

Creating a Field Set

Because a web form can have dozens of different fields, you can make your form easier to interpret and more accessible by grouping fields that share a common purpose into a **field set**. Field sets are created using the following `fieldset` element

```
<fieldset id="id">
   content
</fieldset>
```

where *id* identifies the field set and *content* is the form content within the field set. An id is not required, but it is useful in distinguishing one field set from another.

Marking a Field Set

Alice wants you to organize the form into two field sets: the custInfo field set will enclose the fields containing contact information for *Red Ball Pizza* customers and the expInfo field set will enclose the fields that record those customers' impressions of the restaurant.

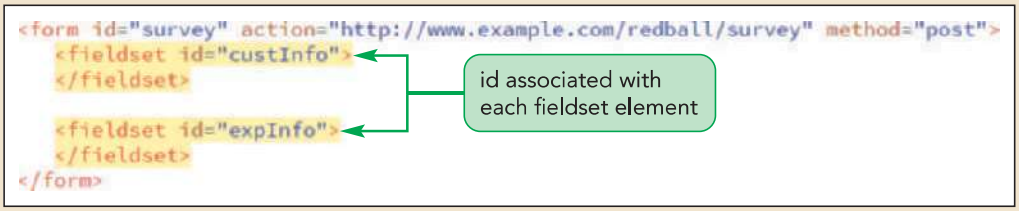
To add field sets to a form:

1. Scroll back to the web form and, within the `form` element, insert the following `fieldset` elements:

```
<fieldset id="custInfo">
</fieldset>

<fieldset id="expInfo">
</fieldset>
```

Figure 7–7 highlights the code for the two new field sets.

Figure 7–7 Inserting field sets



2. Save your changes to the file.

The default browser style is to place a border around the field set to set it off visually from other elements in the web form. Field sets act like block elements that expand to accommodate their content. Before viewing the two field sets in your browser, you will add a legend.

Adding a Field Set Legend

Every field set can contain a legend describing its content using the following `legend` element

```
<legend>text</legend>
```

where *text* is the text of the legend. The `legend` element contains only text and no nested elements. By default, legends are placed in the top-left corner of the field set box, though they can be moved to a different location using the CSS positioning styles.

REFERENCE

Creating a Field Set

- To create a field set, add

```
<fieldset id="id">
   content
</fieldset>
```

where *id* identifies the field set and *content* is the form content within the field set.

- To add a legend to a field set, place the following element within the `fieldset` element:

```
<legend>text</legend>
```

where *text* is the text of the legend.

Based on Alice's sketch from Figure 7–1, add the legend text "Customer Information" and "Share Your Experience at Red Ball Pizza" to the two field sets you just created.

To add legends to the field sets:

1. Within the custInfo field set, add the following `legend` element:

```
<legend>Customer Information</legend>
```

2. Within the expInfo field set, add the legend:

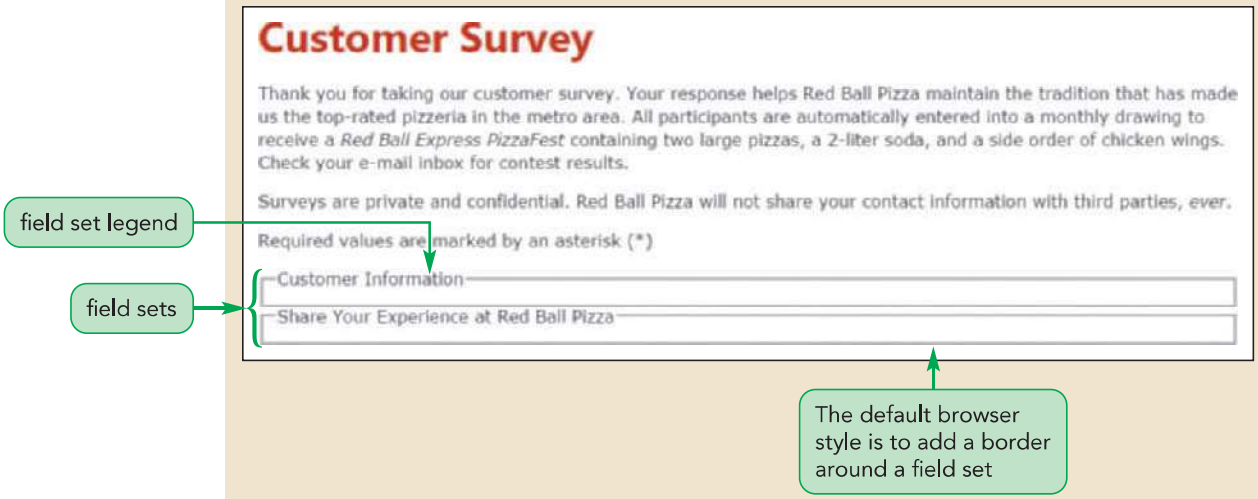
```
<legend>Share Your Experience at Red Ball Pizza</legend>
```

Figure 7–8 highlights the code for the two legends.

Figure 7–8 Adding legends to the field sets

3. Save your changes to the file and then open the **rb_survey.html** file in your browser. Figure 7–9 shows the appearance of the two field sets.

Figure 7–9 Legends displayed in the field set box



The field sets you added are currently empty, so they appear small and narrow on the survey page. Next, you will populate the field set with the controls that will be used to insert different field values.

Creating Input Boxes

Because most form controls are designed to receive user input, they are marked using the following `input` element

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

where the `name` attribute provides the name of the data field associated with the control, the `id` attribute identifies the control in which the user enters the field value, and the `type` attribute indicates the data type of the field. When the form is submitted to the server, the field name is paired with the field value; thus, you always need a `name` attribute if you are submitting the form to a server. The `id` attribute is required only when you need to reference the control, as would be the case when applying a CSS style to format the control's appearance.

Input Types

At the time of this writing, HTML supports twenty-two different values for the `type` attribute. Each input type is associated with a different form control, usually one that is tailored to make it easy for the user to enter data that matches the input type. For example, an input type of `password` is displayed as an input box that hides the input text for security purposes. Figure 7–10 describes the different `type` values for the `input` element and how their controls are typically displayed in most current browsers. If no `type` value is specified, the browser assumes a default value of `text` and adds a simple text input box to the web form.

Figure 7–10 Controls and the input type attribute

Type Value	Control Displayed by the Browser
button	A button that can be clicked to perform an action
checkbox	A check box for yes/no or true/false responses
color	A widget from which users can select a color
date	A widget from which users can select a calendar date
datetime-local	A widget from which users can select a calendar date and time
email	An input box used for email addresses
file	A widget from which users can select a local file
hidden	A control that is hidden from the user
image	An image that can be clicked to perform an action
month	A widget from which users can select a calendar month and year
number	A spin box from which users can select a numeric value
password	An input box in which the entry value is hidden by * symbols
radio	A radio or option button that can be clicked by the user
range	A slider from which users can select a numeric value within a defined range
reset	A button that can be clicked to reset the web form
search	A widget that can be used to search for a defined term
submit	A button that can be clicked to submit the form for processing
tel	An input box used for telephone numbers
text	An input box used for text entries (the default)
time	A widget from which users can select a time value
url	An input box used for entering URLs
week	A widget from which users can select a week value

REFERENCE

Creating an Input Control

- To create an input control for data entry, add the element

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

where `name` provides the name of the field associated with the control, `id` identifies the control in which the user enters the field value, and *type* indicates the type of control displayed by the browser.

The first `input` elements you will add to the survey form will be input boxes in which the customer enters his or her name, street address, city, state, postal code, phone number, and e-mail address. For the program running on the web server, these input boxes with be associated with data fields named custName, custStreet, custCity, custState, custZip, custPhone, and custEmail, respectively. You will identify the controls for these fields with the ids: name, street, city, state, zip, phone, and mail. Before each `input` element, you will insert a text string that describes the content of the input box.

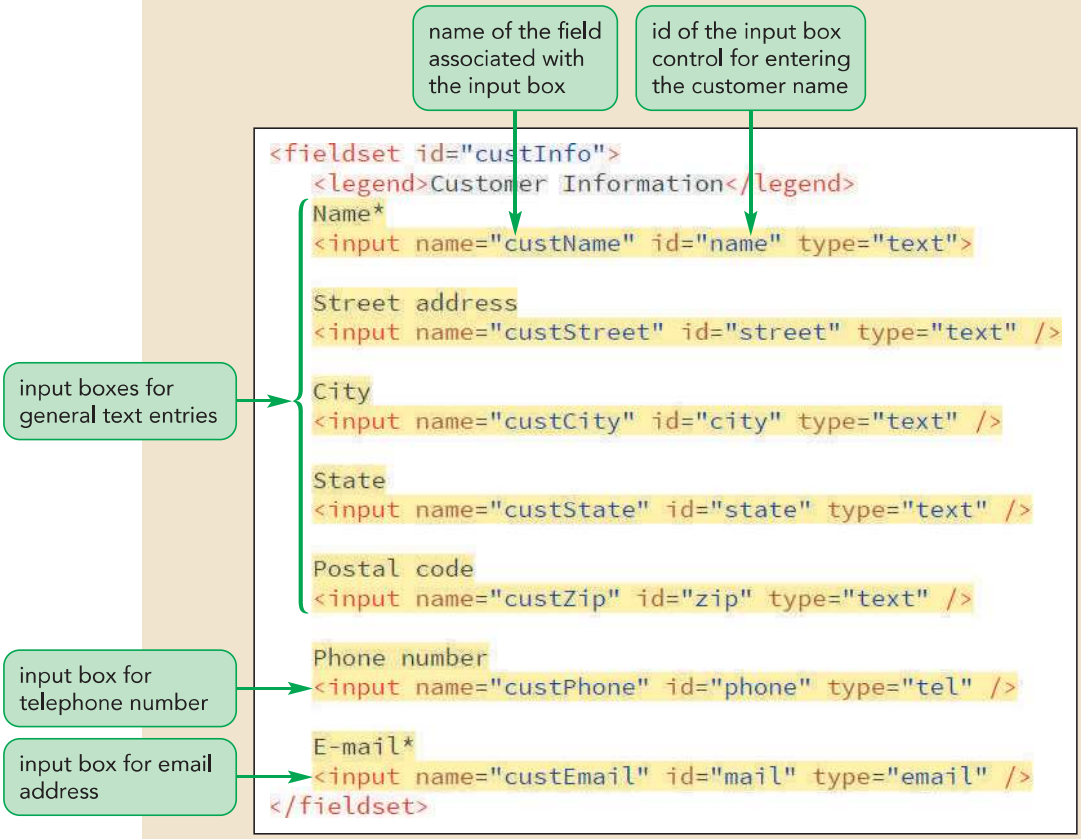
To add input elements:

1. Within the custInfo field set, add the following text strings and `input` elements:

```
Name*
<input name="custName" id="name" type="text" />

Street address
<input name="custStreet" id="street" type="text" />

City
<input name="custCity" id="city" type="text" />

State
<input name="custState" id="state" type="text" />

Postal code
<input name="custZip" id="zip" type="text" />

Phone number
<input name="custPhone" id="phone" type="tel" />

E-mail*
<input name="custEmail" id="mail" type="email" />
```

Figure 7–11 highlights the code for the newly inserted `input` elements.

Figure 7–11 Adding input elements to the form

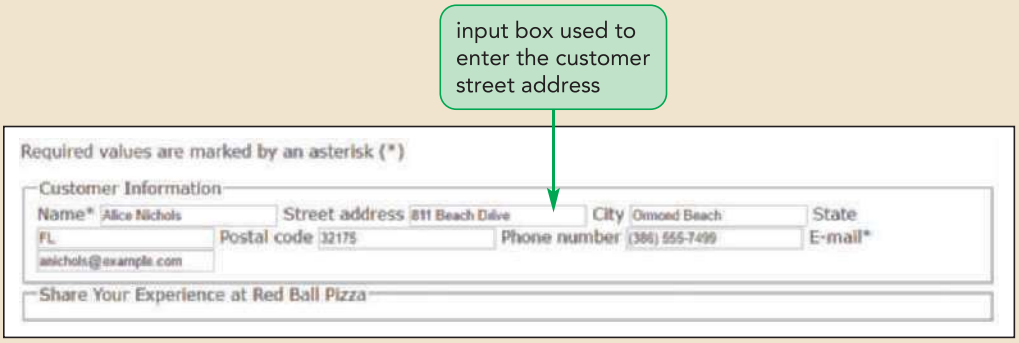


2. Save your changes to the file and then reload the rb_survey.html file in your browser.
3. Click the **Name*** input box on the form to make it active and type ***your name*** in the input box. Press the **Tab** key to move the insertion point to the next input box.
4. Complete the form by entering ***your contact information*** in the remainder of the form, pressing the **Tab** key to move from one input box to the next. Figure 7–12 shows the completed data entry for the form.

TIP

You can prevent users from entering data into a control by adding the attribute `disabled` to the element tag.

Figure 7–12 Displaying input boxes



By default, browsers display input boxes as inline elements with a default length of 20 characters. Later, you will explore how to format these controls to make them easier to read and work with.

INSIGHT

Navigating Forms with Access Keys

You activate controls like input boxes either by clicking them with your mouse or by tabbing from one control to another. As your forms get longer, you might want to give users the ability to jump to a particular input box. This can be done with an access key. An **access key** is a single key on the keyboard that you press in conjunction with another key, commonly the Alt key for Windows users or the control key for Mac users, to jump to a spot in the web page. You create an access key by adding the `accesskey` attribute to the HTML element that creates the control. For example, to create an access key for the custName input box, you would enter the following code:

```
<input name="custName" id="custName" accesskey="l" />
```

If a user types Alt+1 (or control+1 for Mac users), the insertion point automatically moves to the custName input box. Note that you must use letters that are not reserved by your browser. For example, Alt+f is used by many browsers to access the File menu and thus should not be used as an access key. Access keys also can be used with hypertext links and are particularly helpful to users with impaired motor skills who find it difficult to use a mouse or others who prefer not to use a mouse.

Note that you should test your access keys with different browsers since a keyboard shortcut on one browser might not work with another. Your form can be enhanced through the use of access keys but it should not rely on them.

Input Types and Virtual Keyboards

Most mobile and tablet devices do not have physical keyboards; instead, they use **virtual keyboards** that exist as software representations of the physical device. One way that web forms can be made responsive to the needs of mobile and touch devices is by displaying different virtual keyboards for each input type. With an input box for telephone numbers, it is more convenient to have digits (instead of alphabetic characters) prominently displayed on the keyboard. Figure 7–13 shows the virtual keyboards that will be displayed based on the value of the `type` attribute.

Figure 7–13 Virtual keyboards for different input types

type="text"

type="email"

type="number"

type="tel"

type="url"

TIP

Always include a `type` attribute in your input box so that a user's device can choose a keyboard best suited for the form control.

Note that for email addresses the `@` key is prominently displayed as well as a key that inserts the `.com` character string. Similarly, for url data, the virtual keyboard includes a key that inserts the `www.` character string. The choice and layout of the virtual keyboard is determined by the operating system of the device.

Adding Field Labels

In the last set of steps, you entered a descriptive text string above each `input` element to indicate what content should be entered into the input box. However, nothing in the HTML code explicitly associates that descriptive text with the input box. To associate a text string with a control, you enclose the text string within the following `label` element

```
<label for="id">label text</label>
```

where *id* is the id of the control that you want associated with the label, and *label text* is the text of the label. For example, the following code associates the label text "Street address" with the input box for the custStreet control:

```
<label for="street">Street address</label>
<input name="custStreet" id="street" type="text" />
```

You also can make this association implicitly by nesting the control, such as an `input` element, within the `label` element as in the following code:

```
<label>
   Street address
   <input name="custStreet" id="street" />
</label>
```

Notice that you do not need to include a `for` attribute when you nest the control since the association is made implicit.

Which approach you use depends on how you want to lay out a form's content. When you use the `for` attribute, you can place the label text anywhere within the page and it will still be associated with the control. However, by nesting the control within the label, you can treat both the control and its label as a single object, which can make form layout easier because you can move both the label text and the control as a single unit around the page. Depending on the layout of your form, you might use both approaches.

REFERENCE

Creating a Field Label

- To explicitly associate a text label with a control, use the following `label` element and the `for` attribute

  ```
  <label for="id">label text</label>
  ```

  where *id* identifies the control associated with the label.
- To implicitly associate a text label with a control, nest the control within the `label` element as follows

  ```
  <label>
     label text
     control
  </label>
  ```

  where *control* is the HTML code for the form control.

Once you associate a label with a control, clicking the label activates the control. In the case of input boxes, clicking the label would automatically move the insertion point into the input box, making it ready for data entry. With date or color types, clicking the label will display the calendar or color picker widget.

Use the `label` element and `for` attribute now to associate the text strings you entered in the last set of steps with their corresponding input boxes.

The value of the `for` attribute should match the value of the id attribute for the control.

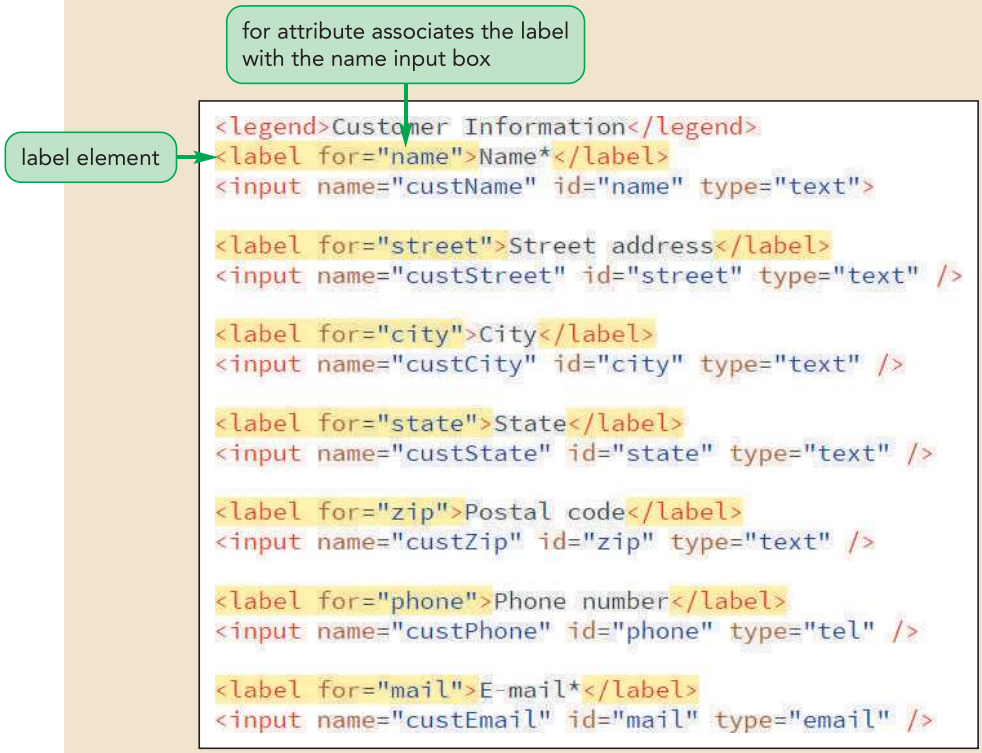
To insert form labels:

1. Return to the **rb_survey.html** file in your editor.
2. Go to the custInfo field set and enclose the text string *Name** within the following `label` element:

```
<label for="name">Name*</label>
```

3. Repeat Step 2 for the remaining descriptive text strings in the custInfo field set, using the `for` attribute to associate each text string with the id of the subsequent `input` element. Figure 7–14 highlights the newly added code in the web form.

Figure 7–14 Adding form labels



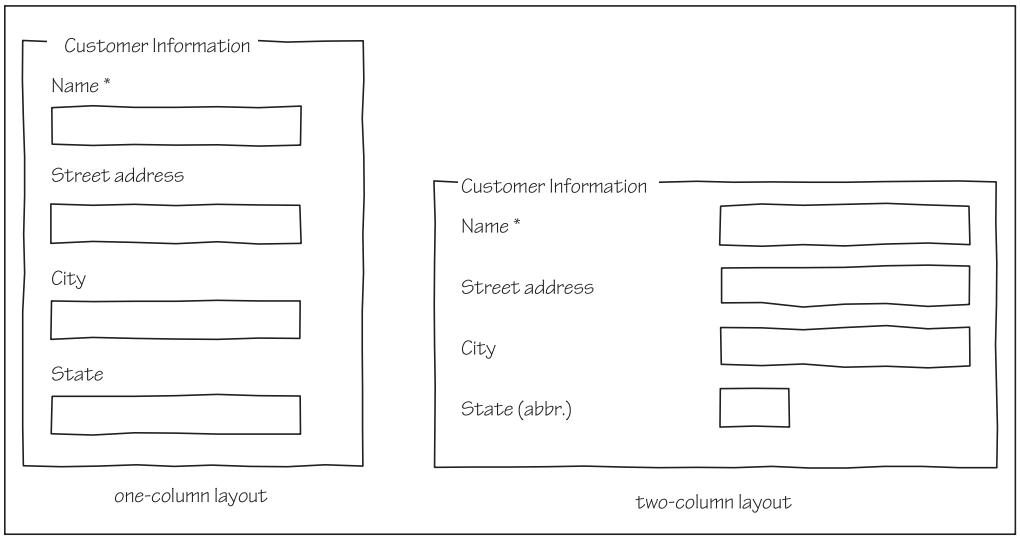
4. Save your changes to the file and then reload the rb_survey.html file in your browser.
5. Test the labels by clicking each label and verifying that the insertion point appears within the corresponding input box, making that control active on the form.

Alice stops by to see your progress on the survey form. In its current state, the form is difficult to read. She wants you to design a layout that will be easier to read and that will be responsive to both mobile and desktop devices.

Designing a Form Layout

To be effective, the layout of your form should aid the user in interpreting the form and navigating easily from one input control to the next. There are two general layouts: one in which the labels are placed directly above the input controls in a single column and the other in which the labels and controls are placed side-by-side in two columns. See Figure 7–15.

Figure 7–15 Form layouts



TIP

In a two-column layout, you can move the label text even closer to the input controls by right aligning the label text.

Usability studies have shown that a single column layout is more accessible because the labels are placed more closely to their input controls. However, for long forms involving many fields, a single column layout can be difficult to work with due to the extensive vertical space required.

Alice wants you to use a single column layout for mobile devices due to the limited horizontal space on those devices, but she wants a two-column layout for devices with larger screen widths. To accomplish this, you will use a flex layout that will allow the labels and controls to assume flexible widths based on the available screen width of the device being used.

First, you will nest each label and input box within a `div` element that will act as a flexbox container.

To create a flexbox for the label and input elements:

1. Return to the **rb_survey.html** file in your editor and scroll down to the custInfo field set.
2. Nest the label and input box for the custName field within the following `div` element, indenting the code to make it easier to read:

```
<div class="formRow">
   <label for="name">Name*</label>
   <input name="custName" id="name" type="text" />
</div>
```

3. Repeat Step 2 for the remaining label and input box pairs, nesting each pair within a `div` element belonging to the formRow class. Figure 7–16 highlights the new code in the file.

Figure 7–16 Nesting labels and input controls within div elements

```
<div class="formRow">
   <label for="name">Name*</label>
   <input name="custName" id="name" type="text" />
</div>

<div class="formRow">
   <label for="street">Street address</label>
   <input name="custStreet" id="street" type="text" />
</div>

<div class="formRow">
   <label for="city">City</label>
   <input name="custCity" id="city" type="text" />
</div>

<div class="formRow">
   <label for="state">State</label>
   <input name="custState" id="state" type="text" />
</div>

<div class="formRow">
   <label for="zip">Postal code</label>
   <input name="custZip" id="zip" type="text" />
</div>

<div class="formRow">
   <label for="phone">Phone number</label>
   <input name="custPhone" id="phone" type="tel" />
</div>

<div class="formRow">
   <label for="mail">E-mail*</label>
   <input name="custEmail" id="mail" type="email" />
</div>
```

Next, you will create a style rule that displays each `div` element of the formRow class as a flexbox and the objects that are direct children of those `div` elements as flex items.

To add styles for a flexible form layout:

1. Scroll to the top of the rb_survey.html file and then, within the document head and directly above the `script` element, add a link to the **rb_forms.css** style sheet.
2. Save your changes to the file and then use your editor to open the **rb_forms_txt.css** file from the html07 ▸ tutorial folder. Enter ***your name*** and ***the date*** in the comment section of the file and save it as **rb_forms.css**.

3. Within the Forms Layout Styles section, add the following style rule to display the formRow `div` element as a flexbox with row orientation and a 7-pixel top and bottom margin:

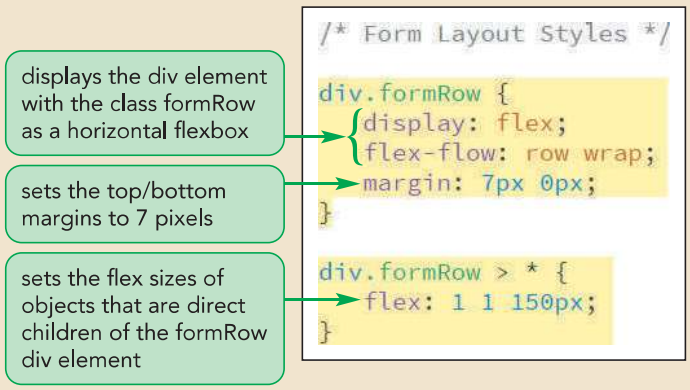
```
div.formRow {
   display: flex;
   flex-flow: row wrap;
   margin: 7px 0px;
}
```

4. Add the following style rules to set the growth, shrink, and basis values of the objects that are direct children of the formRow `div` element:

```
div.formRow > * {
   flex: 1 1 150px;
}
```

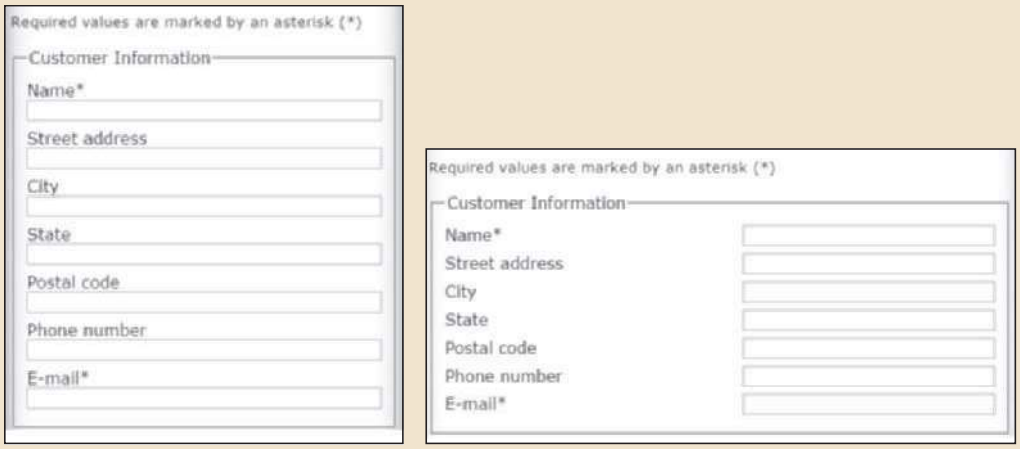
Figure 7–17 shows the new style rules in the style sheet.

Figure 7–17 Adding styles to create a flexible layout



5. Save your changes to the style sheet and then reload rb_survey.html in your browser.
6. Resize your screen width to verify that the form layout changes between one and two columns as the screen changes width. See Figure 7–18.

Figure 7–18 Flex layout of the labels and text input controls



narrow screen layout wide screen layout

Another way to set the width of an input box is by adding the following `size` attribute to the `input` element in the HTML file

```
size="chars"
```

where `chars` is the width of the input box in characters. For example, the following `input` element sets the width of the input box for the custState field to two characters:

```
<input id="state" size="2" type="text" />
```

Note that this is not an exact measure because the width of individual characters varies depending on the typeface and font style.

Alice suggests that you also use a flexible layout for the two field sets so that they are displayed side-by-side for wider screen devices and stacked for narrow screens. Create style rules now that will change the web form to a flexbox and the field sets as items within that flexbox. You will also add styles to change the appearance of the field set boxes themselves.

To create a flexible layout for the form:

1. Return to the **rb_forms.css** file in your editor.
2. At the top of the Form Layout Styles section, insert the following style rule to display the survey form as a flexbox:

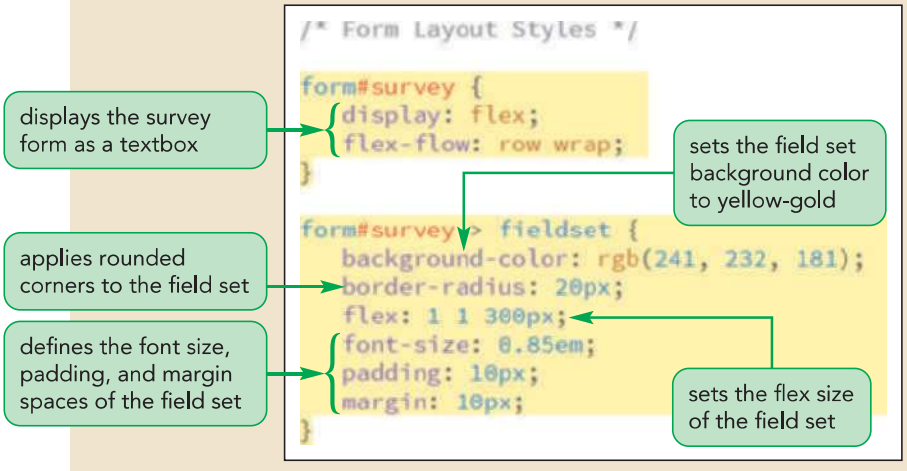
```
form#survey {
   display: flex;
   flex-flow: row wrap;
}
```

3. Add the following style rule to display the field sets within the survey form as flex items:

```
form#survey > fieldset {
   background-color: rgb(241, 232, 181);
   border-radius: 20px;
   flex: 1 1 300px;
   font-size: 0.85em;
   padding: 10px;
   margin: 10px;
}
```

Figure 7–19 shows the new style rules in the style sheet.

Figure 7–19 Creating a flexible layout for the form field sets



4. Save your changes to the style sheet and then reload rb_survey.html in your browser.
5. Resize your screen width and verify that the form layout changes from one column to two columns, with both field sets side by side as shown in Figure 7–20 when the screen width is larger, to one column, with the field sets stacked vertically when the screen width is reduced.

Figure 7–20 Flex layout of the field sets for a wide screen layout

Finally, Alice wants the field set legends to stand out from the border. She suggests you change the text and background color of the legends.

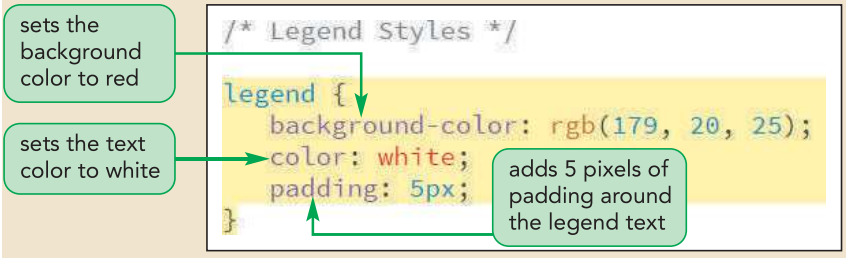
To set the style of the field set legend:

1. Return to the **rb_forms.css** file in your editor.
2. Go to the Legend Styles section and insert the following style rule:

```
legend {
   background-color: rgb(179, 20, 25);
   color: white;
   padding: 5px;
}
```

Figure 7–21 highlights the style rule for the field set legend.

Figure 7–21 Style rule for the field set legend



3. Save your changes to the style sheet and then reload rb_survey.html in your browser. Verify that the two field set legends appear in white font on a red background.

INSIGHT

Using the autocomplete Attribute

Many browsers include an autocomplete feature that automatically completes an input control based on previous user entries. For example, a user who routinely fills in his or her street address in a multitude of web forms can enable the browser to remember that information and to insert it automatically into any address field from a web from.

The autocomplete feature is a useful time-saver in most cases, but it also can be a security risk when using a computer located in a public place. After all, you may not want to have a private credit card number or password automatically filled in by a browser on a computer that other people will be using.

One way to prevent this problem is through the `autocomplete` attribute which enables or disables the browser's autocomplete function. For example, the following `input` element prevents the browser from automatically filling out the creditCard field.

```
<input name="creditCard" autocomplete="false" type="text" />
```

To enable the browser's autocomplete capability, set the value of the `autocomplete` attribute to `true`.

It is easier to use the form with the new layout. However, more than 90% of Red Ball Pizza customers come from Ormond Beach in Florida. Rather than forcing these customers to enter that data, it would be simpler to have those values entered for them. You can do that using default values.

Defining Default Values and Placeholders

To specify a default field value, you add the following `value` attribute to the HTML element for the form control

```
value="value"
```

where *value* is the value that will be entered by default into the control unless the user enters a different value. For example, the following `input` element sets the default value of the custCity field to Ormond Beach:

```
<input name="custCity" id="city" type="text" value="Ormond Beach" />
```

TIP

You can replace the default field value by entering a new value for the field.

Set the default values for the custCity and custState fields to *Ormond Beach* and *FL* respectively.

To define a default field value:

1. Return to the **rb_survey.html** file in your editor and scroll down to the custInfo field set.
2. Add the attribute **value="Ormond Beach"** to the `input` element for the custCity input control.
3. Add the attribute **value="FL"** to the `input` element for the custState input control.

Figure 7–22 highlights the attributes that add default values to the custCity and custState fields.

Figure 7–22 Defining the default field value

4. Save your changes to the file and then reload rb_survey.html in your browser. Verify that the City and State input boxes show the text strings *Ormond Beach* and *FL*, respectively.

   Trouble? If the form does not reload with the replacement text, close the file and reopen it in your browser, which clears all fields and opens a new copy of the form.

Placeholders are text strings that appear within a form control, providing a hint about the kind of data that should be entered into the field. However, unlike a default field value, a placeholder is not stored in the control as the field's value. Placeholders are defined using the following `placeholder` attribute

```
placeholder="text"
```

where *text* is the text of the placeholder. For example, the following `placeholder` attribute provides guidance about the format users should use when entering values for the custPhone field:

```
<input name="custPhone" id="phone" placeholder="(nnn) nnn-nnnn" />
```

When the browser displays the form, the text *(nnn) nnn-nnnn* appears grayed out in the input box indicating to the user that he or she should enter a phone number, including both the area code and the seven-digit number. The placeholder automatically disappears as soon as a user selects the control and begins to enter a value.

Alice asks you to add placeholders to the input boxes for the custName, custZip, and custPhone fields.

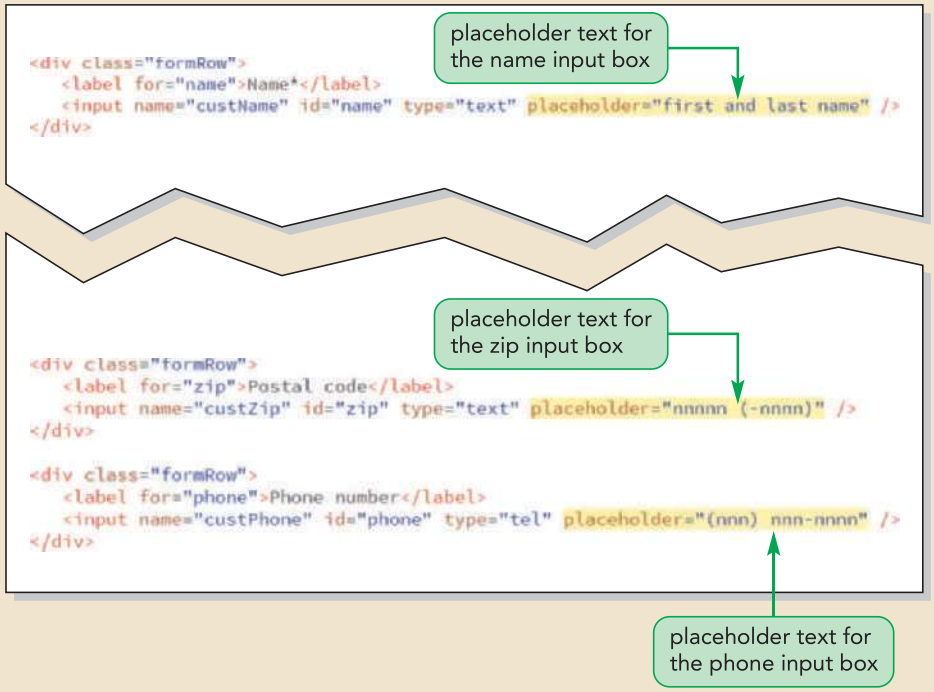
To define a placeholder:

1. Return to the **rb_survey.html** file in your editor.
2. Add the attribute **`placeholder="first and last name"`** to the `input` element for the custName field.
3. Add the attribute **`placeholder="nnnnn (-nnnn)"`** to the `input` element for the custZip field.

4. Add the attribute **placeholder="(nnn) nnn-nnnn"** to the `input` element for the custPhone field.

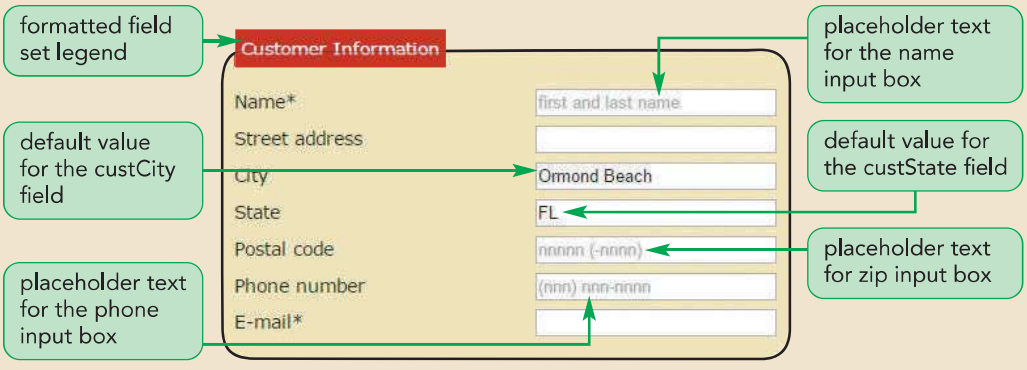
   Figure 7–23 highlights the `placeholder` attributes added to the form.

Figure 7–23 Defining placeholder text



5. Save your changes to the file and then reload rb_survey.html in your browser. As shown in Figure 7–24, placeholder text has been added to the Name, Postal code, and Phone input boxes.

Figure 7–24 Viewing default values and placeholder text



The style of the placeholder text is determined by the browser. There are no CSS styles to format the appearance of the placeholder but all major browsers include their own browser extensions for placeholders. Depending on the browser, the placeholder is treated either as a pseudo-class or a pseudo-element named either `input-placeholder` for the `webkit` and `ms` extensions or `placeholder` for the `moz` extension.

TIP

The moz extension for Firefox version 18 and earlier treats the placeholder as a pseudo-class rather than a pseudo-element.

The following code shows a cross-browser style sheet that changes the text color of the placeholder text for every input box to light red.

```
input::-webkit-input-placeholder {
   color: rgb(255, 151, 151);
}

input:-ms-input-placeholder {
   color: rgb(255, 151, 151);
}

input::-moz-placeholder {
   color: rgb(255, 151, 151);
}
```

Note that you cannot place different browser extensions within the same style rule because if style rule contains a selector that the browser doesn't recognize, the entire rule will be ignored.

PROSKILLS

Decision Making: Creating Cross-Browser Compatible Forms

Several form attributes, such as the `placeholder` attribute, might not be supported by older browsers. This poses a problem for designers who must decide whether or not to use such attributes. One school of thought holds that a web form should look and function the same across all browsers and browser versions. Thus, a feature like the `placeholder` attribute should not be used. If a placeholder is needed, it should be created using a JavaScript program that can be applied uniformly across browsers and browser versions. The opposing view holds that the best design is one that uses each browser to its utmost capabilities, and that the web will only improve in the long run if the most current features are employed because their use will encourage their more rapid adoption across the browser market.

To decide between these two approaches, you must evaluate whether the form feature you're adding is critical to understanding and using your web form. If it is, you need to include workarounds so that all users are supported regardless of their browser. On the other hand, if the feature enhances the user's experience but is not essential to working with the web form, it can be safely added without leaving older browsers behind.

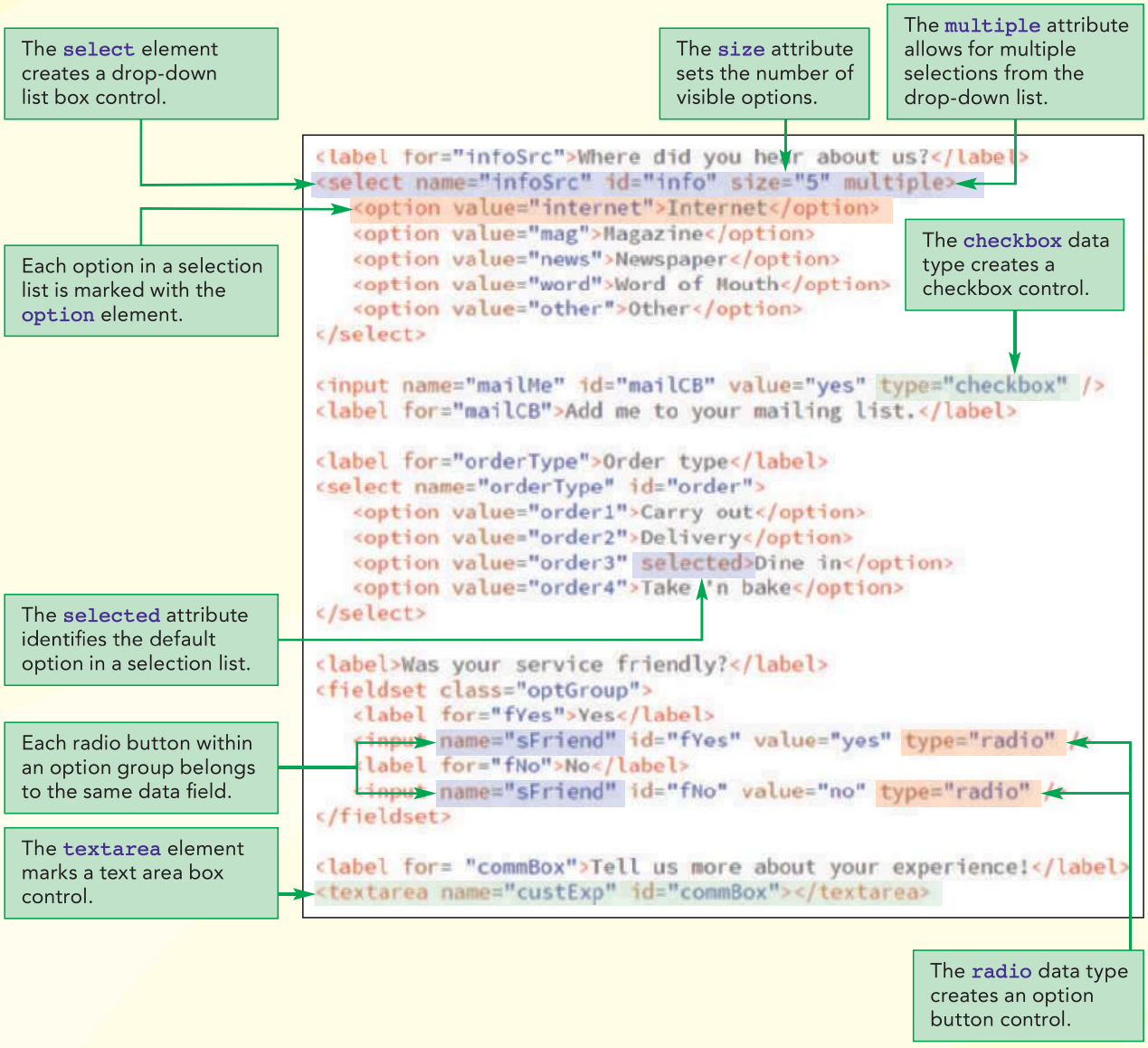
You have finished the initial stage of developing the survey form. Alice is pleased with the form's appearance and content. In the next session, you will extend the form by adding new fields and controls, including calendar widgets, selection lists, option buttons, and check boxes.

REVIEW

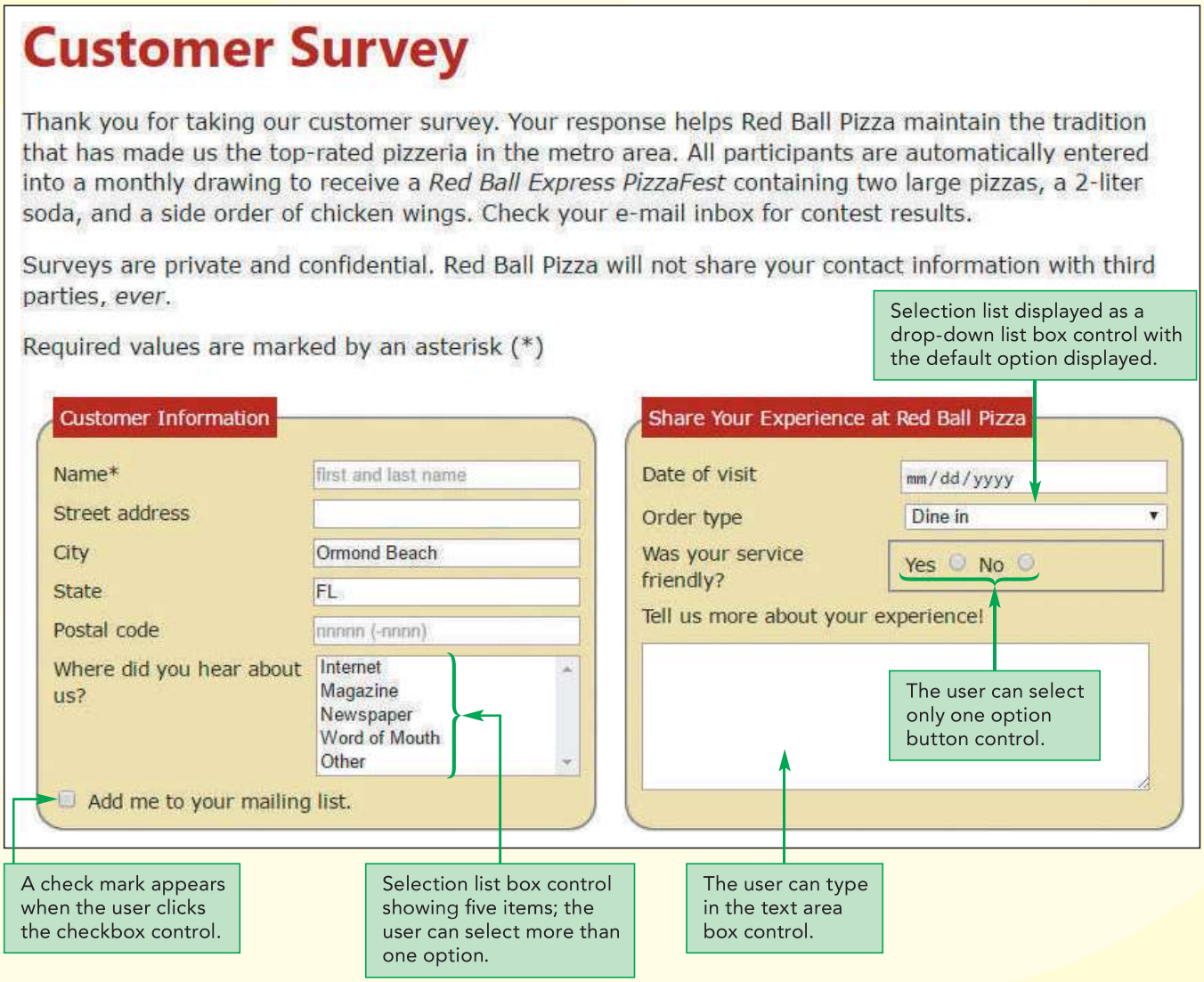
Session 7.1 Quick Check

1. What attribute do you add to the `<form>` element to access the CGI script at *www.example.com/cgi-bin/registration*?
   a. `action="www.example.com/cgi-bin/registration"`
   b. `method="www.example.com/cgi-bin/registration"`
   c. `src="www.example.com/cgi-bin/registration"`
   d. `url="www.example.com/cgi-bin/registration"`
2. What attribute identifies the form control?
   a. `name`
   b. `id`
   c. `control`
   d. `method`
3. What attribute do you use to indicate a password field?
   a. `password="true"`
   b. `method="password"`
   c. `id="password"`
   d. `type="password"`
4. What attribute do you use for an input box containing a telephone number?
   a. `type="telephone"`
   b. `type="text"`
   c. `type="tel"`
   d. `type="number"`
5. What attribute do you use for an input box containing a hypertext link?
   a. `type="url"`
   b. `type="link"`
   c. `rel="link"`
   d. `type="search"`
6. What attribute do you use to associate a label with the input control pWord?
   a. `type="pWord"`
   b. `for="pWord"`
   c. `url="pWord"`
   d. `src="pWord"`
7. What attribute do you use to set the default value of a field to "United States"?
   a. `default="United States"`
   b. `text="United States"`
   c. `placeholder="United States"`
   d. `value="United States"`
8. What code do you enter to specify the placeholder text "nnn-nn-nnnn"?
   a. `default="nnn-nn-nnnn"`
   b. `text="nnn-nn-nnnn"`
   c. `placeholder="nnn-nn-nnnn"`
   d. `value="nnn-nn-nnnn"`

Session 7.2 Visual Overview:



Web Form Widgets



Entering Date and Time Values

To ensure that users enter data in the correct format, you can use controls specifically designed for the field's data type. Consider, for example, the following code that creates an input box for a birthdate field:

```
<label for="bdate">Date of Birth</label>
<input name="bdate" id="bdate" />
```

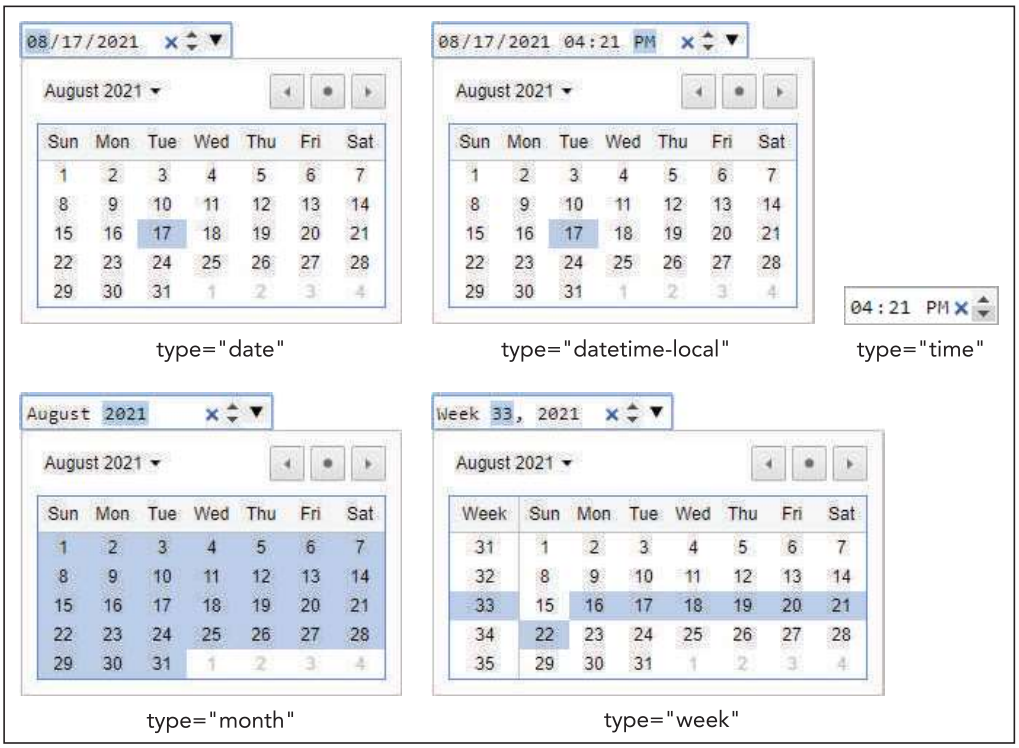
There is nothing to prevent users from entering the same date in a wide variety of formats such as September 14, 2021, 9/14/21, or 2021-09-14. The lack of uniformity in these date formats makes it difficult for a web server program to store and analyze the data.

TIP

If a browser does not support date and time controls, it will display an input box, leaving the user free to enter the date or time value in whatever format he or she wishes.

Starting with HTML 5, date and time fields could be indicated using one of the following `type` attributes: `date`, `time`, `datetime-local`, `month`, and `week`. Each of these `type` attribute values has a different control associated with it, enabling the user to select the date, time, month, or week value. The text into the input box is based on the user's selection in the control widget, ensuring the date or time text is entered in the same format for every user. Figure 7–25 shows examples of the widgets used by the Google Chrome browser.

Figure 7–25 Date and time controls



The expInfo field set will contain fields in which the customer can describe his or her experience at the pizzeria. Alice wants the field set to include a calendar control that users can use to enter the date of their visit to Red Ball Pizza.

To create a date field:

1. If you took a break after the previous session, make sure **rb_survey.html** is open in your editor.
2. Go to the expInfo field set and insert the following `label` and `input` elements:

```
<div class="formRow">
   <label for="visit">Date of visit</label>
   <input name="visitDate" id="visit" type="date" />
</div>
```

Figure 7–26 highlights the code for the `label` and `input` elements.

Figure 7–26 Creating a date field

```
<fieldset id="expInfo">
   <legend>Share Your Experiences at Red Ball Pizza</legend>

   <div class="formRow">
      <label for="visit">Date of visit</label>
      <input name="visitDate" id="visit" type="date" />
   </div>

</fieldset>
```

sets the data type of the visitDate field to "date"

3. Save your changes to the file and then reload rb_survey.html in your browser.
4. Click the Date of visit control and select a date to verify that the text of the date is entered into the input box.

Creating a Selection List

The next part of the survey form records how customers place their orders from Red Ball Pizza. A customer order can be placed in one of four ways: pickup, delivery, dine in, or, in the case of pizzas, uncooked pizzas that customers can take home and bake. Alice doesn't want customers to enter their order types into an input box because customers will enter this information in different ways, and the large variety of spellings and text will make it difficult to group and analyze the survey results. Instead, she wants each user to select the order type from a predetermined group of options. This can be accomplished using a selection list.

A selection list is a list box that presents users with a group of possible values for the data field and is created using the following `select` and `option` elements

```
<select name="name">
   <option value="value1">text1</option>
   <option value="value2">text2</option>
   ...
</select>
```

where *name* is the name of the data field, *value1*, *value2*, and so on are the possible field values, and *text1*, *text2*, and so on are the text of the entries in the selection list that users see on the web form. Note that the field value does not have to match the option text. In most cases, the option text will be expansive and descriptive, while the corresponding field value will be brief and succinct for use with the server program analyzing the form data.

The first option in the selection list is selected by default and thus contains the field's default value. To choose a different option as the default option, add the `selected` attribute to the `option` element as follows:

```
<option value="value" selected>text</option>
```

Note that XHTML documents require the attribute `selected="selected"` to be compliant with XHTML standards for attribute values.

REFERENCE

Creating a Selection List

- To create a selection list, add the elements

```
<select name="name">
   <option value="value1">text1</option>
   <option value="value2">text2</option>
   ...
</select>
```

where *name* is the name of the data field, *value1*, *value2*, and so on are the possible field values, and *text1*, *text2*, and so on are the text entries displayed in the selection list on the web form.

- To allow users to make multiple selections, add the attribute `multiple` to the `select` element.
- To set the number of options displayed at one time in the selection list, add the following attribute to the `select` element

```
size="value"
```

where *value* is the number of options displayed in the selection list at any one time.

- To specify the default value, add the `selected` attribute to the `option` element that you want to set as the default.

Add a selection list to the Red Ball Pizza survey form to record the type of order placed by the customer, storing the value in the orderType field. Identify the selection list control with ID order. Alice knows that most of the survey respondents dine in at the restaurant. Although she wants the options for the orderType field listed in alphabetical order, she would like the Dine in option selected by default.

To create a selection list:

1. Return to the **rb_survey.html** file in your editor.
2. Within the expInfo field set, add the following code to create the label and selection list:

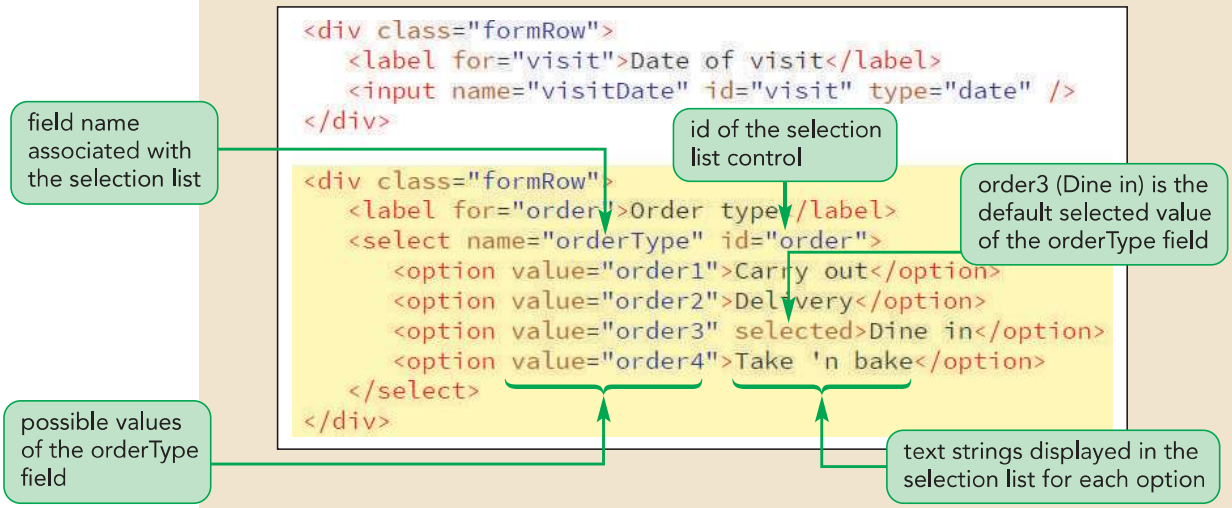
```
<div class="formRow">
   <label for="order">Order type</label>
   <select name="orderType" id="order">
      <option value="order1">Carry out</option>
      <option value="order2">Delivery</option>
      <option value="order3" selected>Dine in</option>
      <option value="order4">Take 'n bake</option>
   </select>
</div>
```

Figure 7–27 highlights the code for the selection list.

TIP

The default width of the selection box is equal to the width of the longest option text unless the width is set using a CSS style.

Figure 7–27 Creating a selection list for the orderType field



3. Save your changes to the file and then reload rb_survey.html in your browser.
4. Click the Order type selection list and verify that the list of order types appears in the drop-down list box and that Dine in is the default selected option. See Figure 7–28.

Figure 7–28 Viewing the selection list options

Working with select Attributes

By default, a selection list appears as a drop-down list box. To display a selection list as a scroll box with more than one option visible in the web form, add the following `size` attribute to the `select` element

```
<select size="value"> ... </select>
```

where *value* is the number of options that the selection list displays at one time. For example, a size value of 5 would display 5 items in the scroll box.

The default behavior of the selection list is to allow only one selection from the list of options. To allow more than one item to be selected, add the following `multiple` attribute to the `select` element:

```
<select multiple> ... </select>
```

TIP

To assist your users in completing your form, you can include instructions on your page detailing how to select multiple options from a selection list.

There are two ways for users to select multiple items from a selection list. For noncontiguous selections, users can press and hold the Ctrl key (or the command key on a Mac) while making the selections. For a contiguous selection, users can select the first item, press and hold the Shift key, and then select the last item in the range. This selects the two items, as well as all the items between them.

Alice has another selection list to add to the survey form, which will record how a customer heard about Red Ball Pizza. The survey presents the user with five options: Internet, Magazine, Newspaper, Word of Mouth, or Other. Alice wants the form to display all of the options, so you set the value of the `size` attribute to 5. She also wants customers to be able to select multiple options from the selection list.

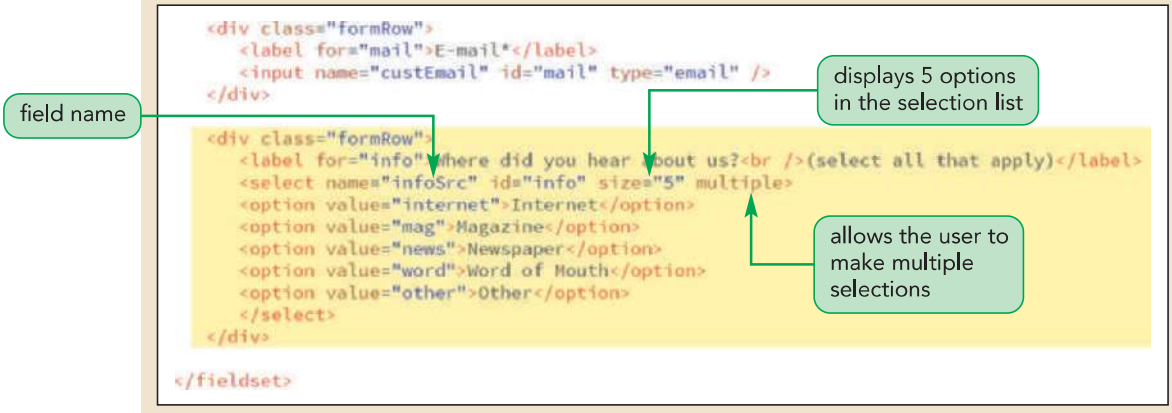
To apply the size and multiple attributes:

1. Return to the **rb_survey.html** file in your editor and go to the custInfo field set.
2. At the bottom of the field set, insert the following code:

```
<div class="formRow">
<label for="info">Where did you hear about us?
<br />(select all that apply)
</label>
<select name="infoSrc" id="info" size="5" multiple>
   <option value="internet">Internet</option>
   <option value="mag">Magazine</option>
   <option value="news">Newspaper</option>
   <option value="word">Word of Mouth</option>
   <option value="other">Other</option>
</select>
</div>
```

Figure 7–29 highlights the code for the selection list.

Figure 7–29 Inserting a selection list for the infoSrc field



3. Save your changes and then reload rb_survey.html in your browser. Figure 7–30 shows the infoSrc selection list box.

Figure 7–30 Viewing the selection list for the infoSrc field

4. Verify that you can now select multiple values for the infoSrc field by using the **Ctrl+click**, **Command+click**, **Shift+click** keyboard and mouse combinations.

If you use a multiple selection list in a form, be aware that the form sends a name/value pair to the server for each option the user selects from the list. Verify that your server-based program can handle a single field with multiple values before using a multiple selection list.

Grouping Selection Options

In long selection lists, it can be difficult for users to locate a particular option value. You can organize selection list options by placing them in option groups using the `optgroup` element

```
<select>
   <optgroup label="label1">
      <option>text1</option>
      <option>text2</option>
   </optgroup>
   <optgroup label="label2">
      <option>text3</option>
      <option>text4</option>
   </optgroup>
</select>
```

where *label1*, *label2*, and so forth are the labels for the different groups of options. The text of the label appears in the selection list above each group of items but it is not a selectable item from the list. Figure 7–31 shows an example of a selection list in which the options are divided into two groups.

Figure 7–31 **Grouping options in a selection list**

```
<label for="appetizers">Starter Menu</label>
<select name="meal">
   <optgroup label="Appetizers">
      <option value="sms">Spicy Mozzarella Sticks</option>
      <option value="pr">Pepperoni Rolls</option>
      <option value="tr">Toasted Ravioli</option>
   </optgroup>
   <optgroup label="Salads">
      <option value="sms">Pasta Salad</option>
      <option value="tbs">Tuscan Bread Salad</option>
      <option value="pr">Caesar Salad</option>
   </optgroup>
</select>
```

option group labels

The appearance of the option group label is determined by the browser. You can apply a style to an entire option group including its label, but there is no CSS style to change the appearance of the option group label alone.

INSIGHT

Hidden Fields

Some fields have predefined values that do not require user input and are often not displayed within the web form. You create a **hidden field** by setting the value of the `type` attribute to hidden as follows:

```
<input name="name" value="value" type="hidden" />
```

where *name* is the name of the data field and *value* is the value stored in the field. With a hidden field, both the field value and the input control are hidden from the user. Even though hidden fields are not displayed by browsers, the field values still can be read by examining the source code; for this reason, you should not put any sensitive information in a hidden field.

Creating Option Buttons

Option buttons, also called radio buttons, are like selection lists in that they limit fields to a set of possible values; but, unlike selection lists, the options appear as separate controls in the web form. Option buttons are created with a group of `input` elements with a `type` attribute value of "radio", sharing a common data field name as follows

```
<input name="name" value="value1" type="radio" />
<input name="name" value="value2" type="radio" />
<input name="name" value="value3" type="radio" />
…
```

TIP

To show that a group of radio buttons are associated with the same field, place the radio button controls within a field set.

where *name* is the name of the data field and *value1*, *value2*, *value3*, and so on are the field values associated with each option. While a user can select multiple items in a selection list, a user can only click or check one option in a group of radio buttons. Selecting one radio button automatically deselects the others and sets the value of the field to the value of the checked radio button.

For example, the following code creates a group of option buttons for the sFriend field, limiting the possible field values to "yes" or "no".

```
Was your service friendly?
<label for="fYes">Yes</label>
<input name="sFriend" value="yes" id="fYes" type="radio" />
<label for="fYes">No</label>
<input name="sFriend" value="no" id="fNo" type="radio" />
```

Note that the two radio button controls are given different ids and field values to distinguish them from each other, however they share the same field name, "sFriend".

By default, an option button is unselected; however, you can set an option button to be selected as the default by adding the following `checked` attribute to the `input` element:

```
<input name="name" type="radio" checked />
```

REFERENCE

Creating an Option List

- To create a group of option buttons associated with the same field, add the `input` elements

```
<input name="name" value="value1" type="radio" />
<input name="name" value="value2" type="radio" />
<input name="name" value="value3" type="radio" />
…
```

  where *name* is the name of the data field, and *value1*, *value2*, *value3*, and so on are the field values associated with each option.
- To specify the default option, add the `checked` attribute to the `input` element.

In the next part of the form, Alice wants to ask customers general questions about their experiences at the restaurant. She wants to know whether the service was friendly, whether orders were recorded correctly, and if the food was delivered hot. She suggests that you present these questions using radio buttons, placing each group of radio buttons within a `fieldset` element belonging to the optGroup class.

The value of the name attribute must be the same for all option buttons within a group.

To create a set of option buttons:

1. Return to the **rb_survey.html** file in your editor and go to the expInfo field set.
2. At the bottom of the expInfo field set, add the following code to create radio buttons for the sFriend field:

```
<div class="formRow">
   <label>Was your service friendly?</label>
   <fieldset class="optGroup">
      <label for="fYes">Yes</label>
      <input name="sFriend" id="fYes" value="yes"
type="radio" />
      <label for="fNo">No</label>
      <input name="sFriend" id="fNo" value="no"
type="radio" />
   </fieldset>
</div>
```

3. Add the following group of radio buttons for the oCorrect field:

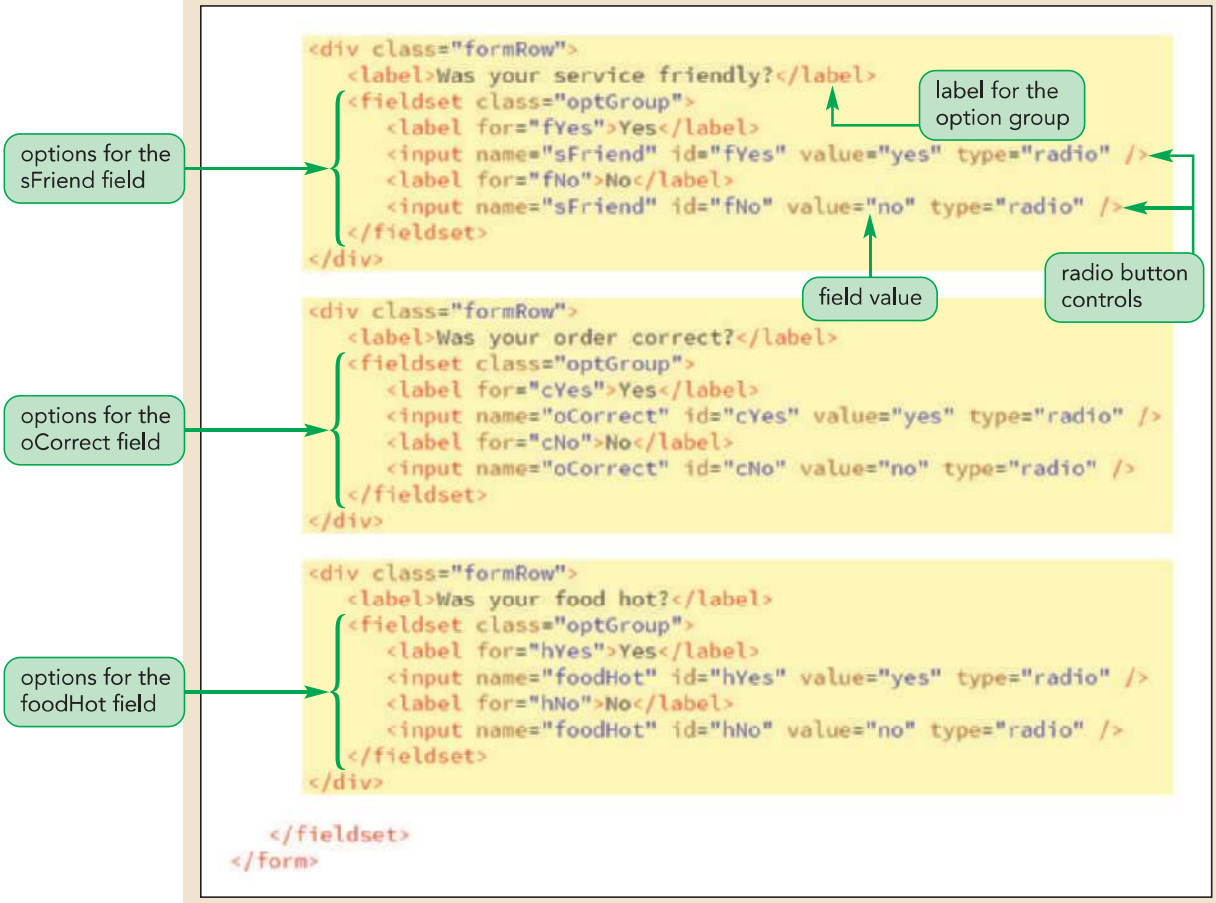
```
<div class="formRow">
   <label>Was your order correct?</label>
   <fieldset class="optGroup">
      <label for="cYes">Yes</label>
      <input name="oCorrect" id="cYes" value="yes"
type="radio" />
      <label for="cNo">No</label>
      <input name="oCorrect" id="cNo" value="no"
type="radio" />
   </fieldset>
</div>
```

4. Finally, add the following group of radio buttons for the foodHot field:

```
<div class="formRow">
   <label>Was your food hot?</label>
   <fieldset class="optGroup">
      <label for="hYes">Yes</label>
      <input name="foodHot" id="hYes" value="yes"
type="radio" />
      <label for="hNo">No</label>
      <input name="foodHot" id="hNo" value="no"
type="radio" />
   </fieldset>
</div>
```

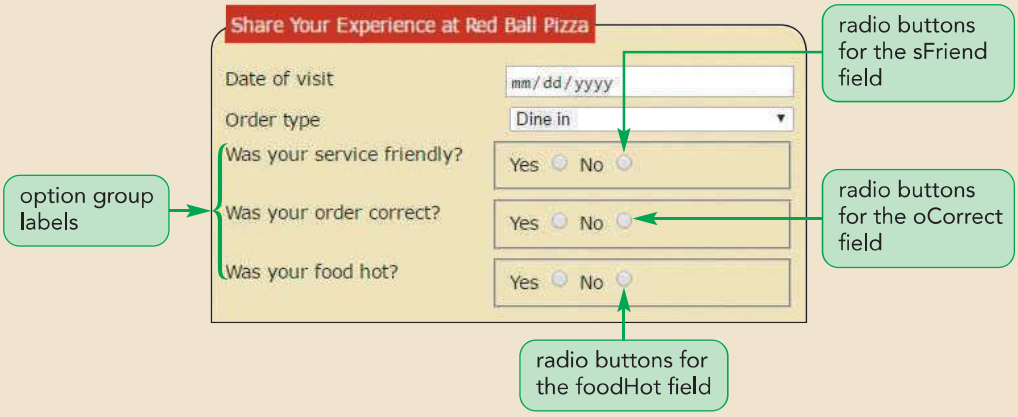
Figure 7–32 highlights the code for the set of option buttons.

Figure 7–32 Creating option groups for the sFriend, oCorrect, and foodHot fields



5. Save your changes to the file and then reload rb_survey.html in your browser. Figure 7–33 shows the appearance of the three groups of radio buttons in the survey form.

Figure 7–33 Option buttons for the serve field



6. Click the radio buttons with each option group and verify that if you select one radio button, the other button in that group is automatically deselected.

Creating Check Boxes

Check boxes are designed for fields that record the presence or absence of an object or event. The check box control is created using the following `input` element with the `type` attribute set to "checkbox"

```
<input name="name" value="value" type="checkbox" />
```

TIP

The default field value for a check box control is "On".

where the `value` attribute contains the value of the field when the check box is checked, and the `type` attribute indicates that the input box is a check box. By default, the check box is not checked, however you can make a check box selected automatically by adding the `checked` attribute to the `input` element.

For example, the following code creates a check box for the orderDone field, recording whether an order has been completed:

```
<label for="orderCB">Order Completed</label>
<input name="orderDone" id="orderCB" value="yes" type="checkbox" />
```

If the check box is selected by the customer, the browser will send a name/value pair of orderDone/yes to the script running on the web server when the form is submitted. A name/value pair is sent to the server only when the check box is checked by the user. If the control is not checked, then no name/value pair is sent when the form is submitted.

REFERENCE

Creating a Check Box

- To create a check box, add the element

  ```
  <input name="name" value="value" type="checkbox" />
  ```

  where `type` is the type of input control, *`name`* is the name of the data field, and *`value`* is the data field value if the check box is selected.
- To specify that a check box is selected by default, add the `checked` attribute to the `input` element.

Alice wants her survey form to include a check box that customers can select if they wish to be added to the pizzeria's email list for specials and promotions. Add a check box for the mailMe field to the custInfo field set now.

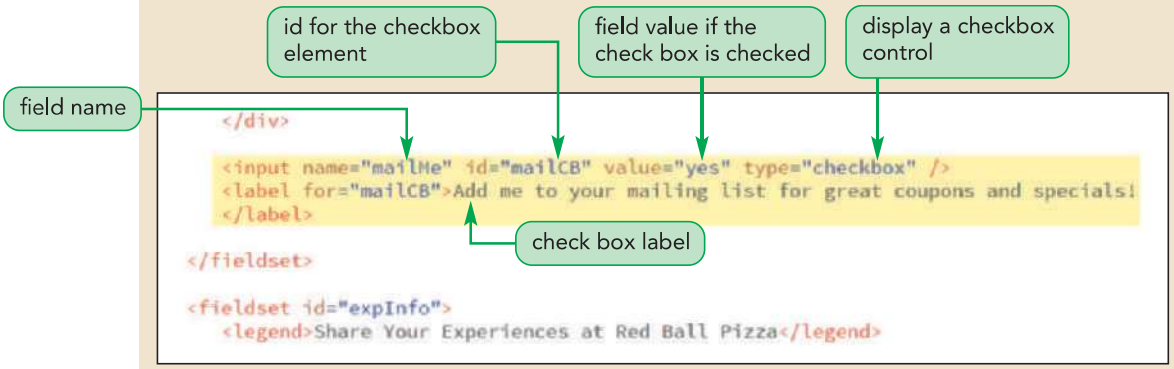
To add a check box control:

1. Return to the **rb_survey.html** file in your editor and go to the end of the custInfo field set.
2. Add the following code to create a check box followed by the label for the `checkbox` control:

   ```
   <input name="mailMe" id="mailCB" value="yes" type="checkbox" />
   <label for="mailCB">Add me to your mailing list for great
   coupons and specials!</label>
   ```

   Figure 7–34 highlights the code for the check box and label.

Figure 7–34 Creating a checkbox control



3. Save your changes and then reload rb_survey.html in your browser. Figure 7–35 shows the placement of the check box within the form.

Figure 7–35 Check box for the mailMe field

INSIGHT

Tab Indexing and Autofocus

Typically, users navigate through a form using the Tab key, which moves the insertion point from one field to another in the order that the form controls are entered into the HTML file.

You can specify an alternate order by adding the `tabindex` attribute to any control in your form. When each control is assigned a tab index number, the insertion point moves through the fields from the lowest index number to the highest. For example, to assign the tab index number 1 to the custName field from the survey form, you add the following `tabindex` attribute to the control:

```
<input name="custName" tabindex="1" />
```

This code places the insertion point in the custName field when the form is first opened. (Fields with 0 or negative tab indexes are omitted from the tab order entirely.)

Another way to place the insertion point in a field when the form is initially opened is to use the following `autofocus` attribute:

```
<input name="custName" autofocus />
```

Older browsers that do not support tab indexing or the `autofocus` attribute simply ignore them and open a file without giving the focus to any form control. When a user tabs through the form in those older browsers, the tab order will reflect the order of the elements in the HTML file.

Creating a Text Area Box

Input boxes are limited to a single line of text and thus are not appropriate for extended text strings that might cover several lines of content. For that type of data entry, you create a text area box using the following `textarea` element

```
<textarea name="name">
   text
</textarea>
```

where *text* is the default value of the data field. You do not have to specify a default value; you can leave the text box empty or you can use the `placeholder` attribute introduced in the last session to provide a hint to users about what to enter into the text box.

The default browser style is to create a text area box that is about 20 characters wide and two or three lines high. You can increase the size of the box using CSS styles. HTML also supports the following `rows` and `cols` attributes to set the text area size

```
<textarea rows="value" cols="value"> ... </textarea>
```

where the `rows` attribute specifies the number of lines in the text area box and the `cols` attribute specifies the number of characters per line. While the `rows` and `cols` attributes represent the older standard, you may still encounter their use in older websites.

TIP

When you enter more text than can fit into a text area box, the browser automatically adds vertical scroll bars to the box.

Content in a text area box automatically wraps to a new line as needed. You can determine whether those line returns are included as part of the field value by adding the following `wrap` attribute:

```
<textarea wrap="type"> ... </textarea>
```

where *type* is either `hard` or `soft`. In a hard wrap, line returns are included with the data field value, while in a soft wrap, line returns are not included. The default value of the `wrap` attribute is `soft`.

REFERENCE

Creating a Text Area Box

- To create a text area box for multiple lines of text, use

```
<textarea name="name">
   text
</textarea>
```

where *name* is the name of the field associated with the text area box and *text* is the default text that appears in the box.

- To specify the dimensions of the box, use a CSS style or apply the following attributes

```
rows="value" cols="value"
```

where the `rows` attribute specifies the number of lines in the text area box and the `cols` attribute specifies the number of characters per line.

- To specify how the field value should handle wrapped text, use the attribute

```
wrap="type"
```

where *type* is either `hard` (to include the locations of the line wraps) or `soft` (to ignore line wrap locations).

Alice wants to include a text area box where customers can enter extended commentary about the pizzeria, storing their comments in the custExp field. You will set the dimensions of the text area box using CSS.

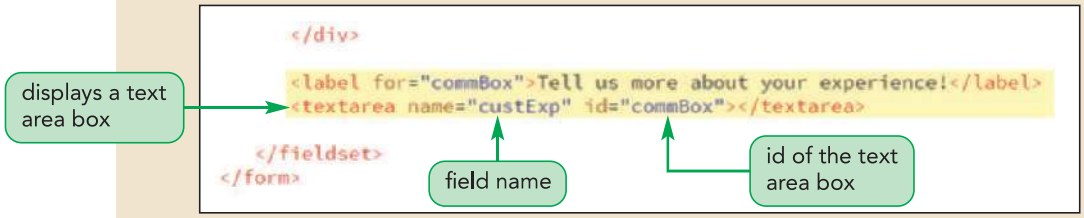
To add a text area box:

1. Return to the **rb_survey.html** file in your editor and go to the end of the expInfo field set.
2. Add the following code to create a text area box at the bottom of the field set:

```
<label for="commBox">Tell us more about your
   experience!</label>
<textarea name="custExp" id="commBox"></textarea>
```

Figure 7–36 highlights the code for the text area box and label.

Figure 7–36 Creating a text area box

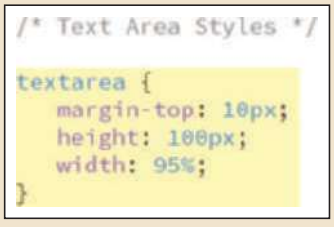


3. Save your changes and then return to the **rb_forms.css** file in your editor.
4. Go to the Text Area Styles section and insert the following style rule to set the size and top margin of the text area box.

```
textarea {
   margin-top: 10px;
   height: 100px;
   width: 95%;
}
```

Figure 7–37 shows the style rule for the text area box.

Figure 7–37 Styles for the text area box



5. Save your changes and then reload rb_survey.html in your browser. Figure 7–38 shows the appearance of the text area box.

Figure 7–38 **Text area box in the web form**

6. Test the text area box by clicking it and then typing a sample comment inside of the box.

PROSKILLS

Written Communication: Creating Effective Forms

Web forms are one of the main ways of getting feedback from your users, so it is important for the forms to be easily accessible. A well-designed form often can be the difference between a new customer and a disgruntled user who leaves your site to go elsewhere. Here are some tips to remember when designing a form:

- Keep your forms short and to the point.
- Mark fields that are required but also limit their number. Don't overwhelm your users with requests for information that is not really essential.
- Use the `autofocus` attribute to place users automatically into the first field of your form, rather than forcing them to click that field.
- Many users will navigate through your form using the Tab key. Make sure that your tab order is logical and easy for users to follow.
- Provide detailed instructions about what users are expected to do. Don't assume that your form is self-explanatory.
- If you ask for personal data and financial information, provide clear assurances that the data will be secure. If possible, provide a link to a web page describing your security practices.
- If you need to collect a lot of information, break the form into manageable sections spread out over several pages. Allow users to easily move backward and forward through the form without losing data. Provide information to users indicating where they are as they progress through your pages.
- Clearly indicate what users will receive once a form is submitted, and provide feedback on the website and through email that tells them when their data has been successfully submitted.

Finally, every form should undergo usability testing before it is made available to the general public. Weed out any mistakes and difficulties before your users see the form.

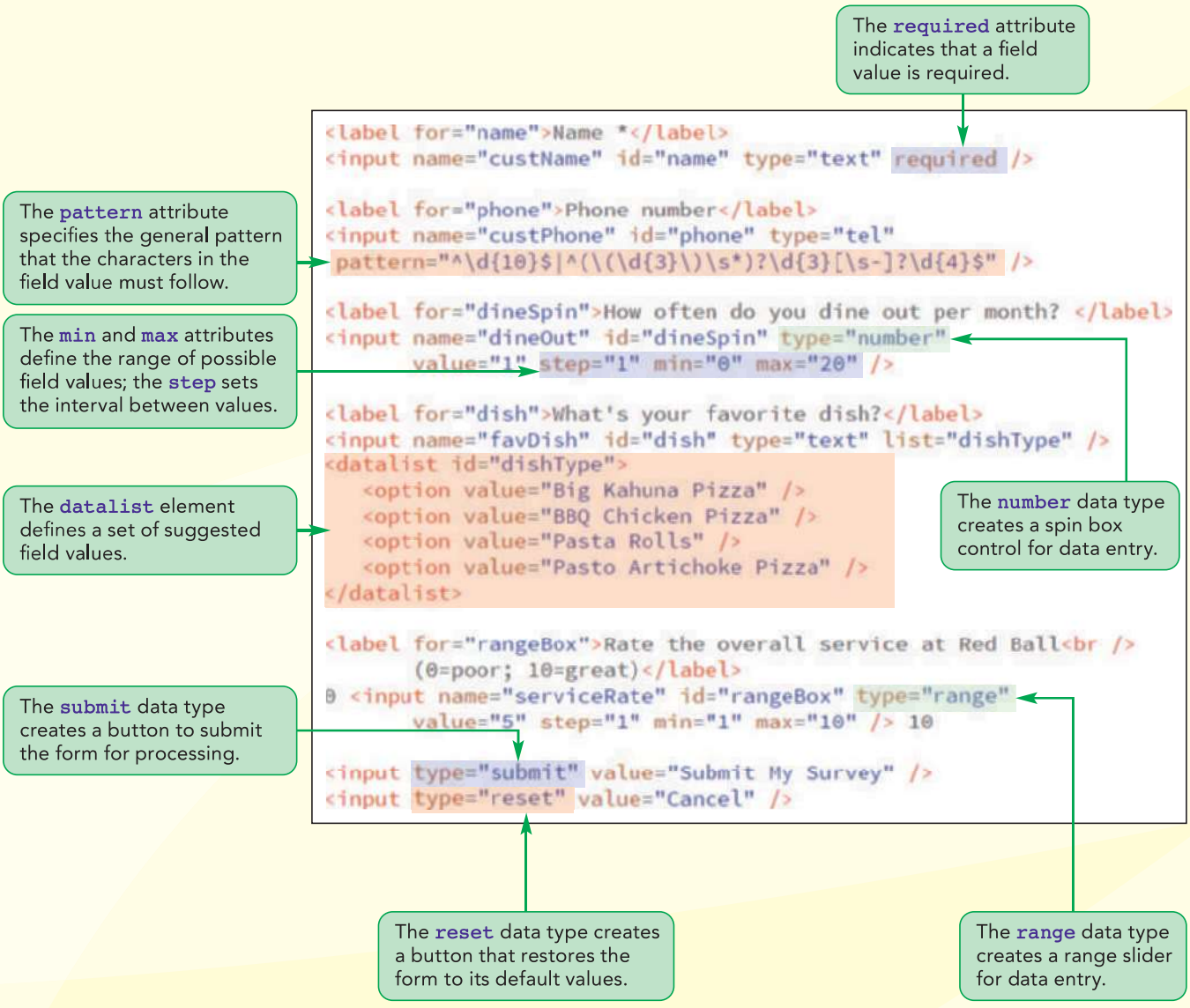
You have greatly extended the scope of the survey form through the use of a calendar control, selection lists, option button groups, a check box, and a text area box. In the next session, you will continue to work on the survey form by exploring how to design a form that verifies the user enters valid data before it is submitted to the web server for processing.

REVIEW

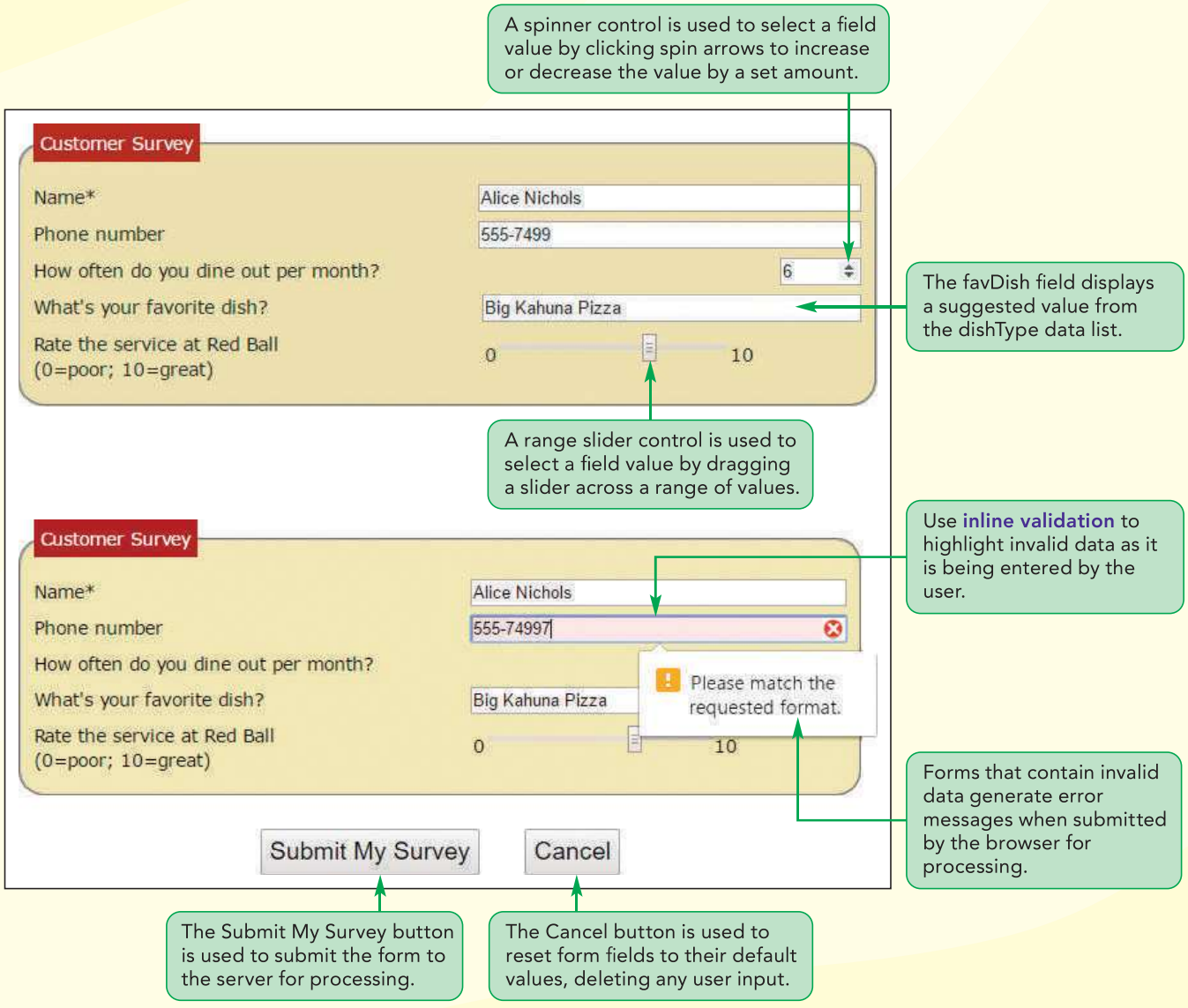
Session 7.2 Quick Check

1. What code do you enter to create a date control for the expireDate field?
   a. `<input name="expireDate" date />`
   b. `<input name="expireDate" type="date" />`
   c. `<input name="expireDate" src="date" />`
   d. `<input name="expireDate" method="date" />`
2. What code do you enter to create a date/time control for the orderDelivery field?
   a. `<input name="orderDelivery" date-time />`
   b. `<input name="orderDelivery" type="datetime-local" />`
   c. `<input name="orderDelivery" type="date/time" />`
   d. `<input name="orderDelivery" type="date-time" />`
3. What HTML elements are used to create a selection list?
   a. `<select><li /><li />…</select>`
   b. `<select><item /><item />…</select>`
   c. `<list><item /><item /> … </list>`
   d. `<select><option /><option /> … </select>`
4. What HTML elements are used to group option buttons?
   a. `<group><option /><option /> … </group>`
   b. `<optgroup><item /><item /> … </optgroup>`
   c. `<optgroup><option /><option /> … </optgroup>`
   d. `<group><item /><item /> … </group>`
5. What HTML element do you use to create an option button control?
   a. `<input type="option" />`
   b. `<option />`
   c. `<input type="radio" />`
   d. `<radio type="option" />`
6. What HTML element do you use to create a checkbox control?
   a. `<input type="check" />`
   b. `<input type="checkbox" />`
   c. `<check />`
   d. `<option type="check" />`
7. What HTML element do you use to create a text box control?
   a. `<text> … </text>`
   b. `<input type="text" />`
   c. `<textarea> … </textarea>`
   d. `<input type="textarea" />`

Session 7.3 Visual Overview:



Data Validation



Entering Numeric Data

In the last session, you worked with several form controls that restricted field values to a set of possible values, ensuring that the user submits valid data to the server for processing. HTML also supports restrictions on numeric values by specifying that the values must fall within a defined range.

Creating a Spinner Control

One way of restricting numeric values is through a **spinner control**, which displays an up or down arrow to increase or decrease the field value by a set amount. To create a spinner control, apply the following `input` element using the `number` data type:

```
<input name="name" id="id" type="number"
   value="value" step="value" min="value" max="value" />
```

where the `value` attribute provides the default field value, the `step` attribute indicates the amount by which the field value changes when a user clicks the spin arrow, the `min` attribute defines the minimum possible value, and the `max` attribute defines the maximum possible value of the field. For example, the following `input` element creates a spinner control with the ID attSpin for the attendance field with the spinner value ranging from 10 to 50 in steps of 5 units with a default value of 20:

```
<input name="attendance" id="attSpin" type="number"
   value="20" step="5" min="10" max="50" />
```

Add a new field to the survey form named dineOut that queries customers about how often they dine out, setting its default value to 1 and allowing the field value to range from 0 up to 20 in steps of 1 unit.

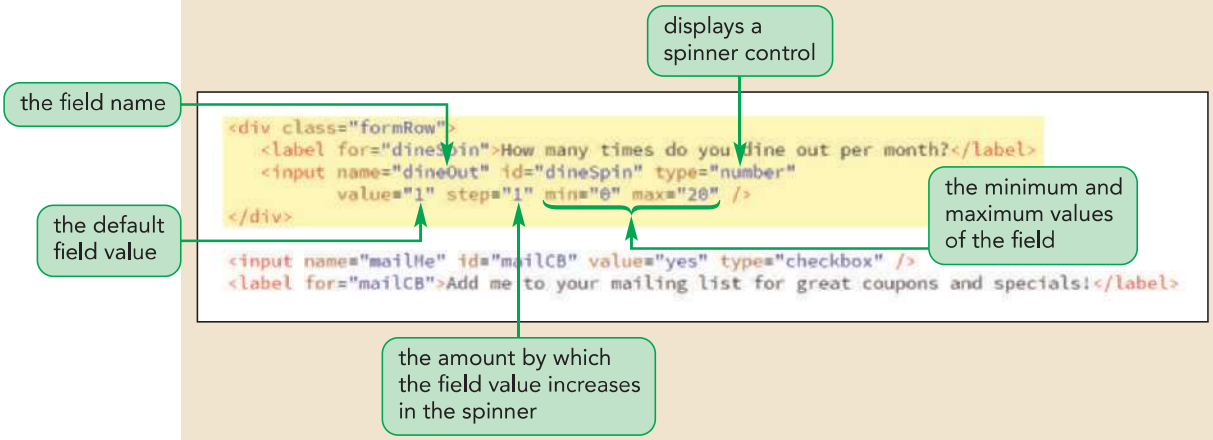
To add a spinner control:

1. If you took a break after the previous session, make sure **rb_survey.html** is open in your editor and scroll down to the custInfo field set.
2. Above the check box for the mailMe field, insert the following code to create a spinner control for the dineOut field:

```
<div class="formRow">
   <label for="dineSpin">How many times do you dine out per
month?</label>
   <input name="dineOut" id="dineSpin" type="number"
          value="1" step="1" min="0" max="20" />
</div>
```

Figure 7–39 highlights the code for the spinner control and label.

Figure 7–39 Creating a spinner control for the dineOut field



3. Save your changes to the file.

   A spinner control does not need to be as wide as the input boxes used for text entries. So, next, you will create a style rule that sets the width of the spinner control.

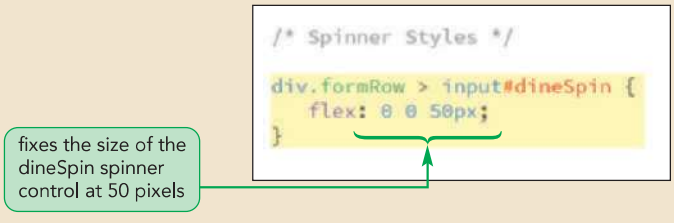
4. Return to the **rb_forms.css** file in your editor and scroll down to the Spinner Styles section.

5. Enter the following style rule for `input` elements with the number data type:

```
div.formRow > input#dineSpin {
   flex: 0 0 50px;
}
```

   Figure 7–40 highlights the new style rule in the style sheet.

Figure 7–40 Styles for the dineSpin spinner control



6. Save your changes to the style sheet and then reload rb_survey.html in your browser. Figure 7–41 shows the layout and appearance of the spinner control.

Figure 7–41 **Spinner in the web form**

Trouble? At the time of this writing, some browsers (such as IE and Edge) do not support the number data type and they also ignore the `step`, `min`, and `max` attributes. In those browsers, the spinner control is displayed as a text input box.

7. Click the input box for the spinner control and verify that you use the arrow buttons to increase and decrease the field value within the range 0 to 20.

Creating a Range Slider

Another way to limit a numeric field to a range of possible values is through a **slider control**, which the user can use to drag a marker horizontally across the possible field values. Slider controls are created by applying the range data type in the following `input` element

```
<input name="name" id="id" type="range"
   value="value" step="value" min="value" max="value" />
```

where the `value`, `step`, `min`, and `max` attributes have the same meanings as they did for the spinner control. Many browsers do not include a scale on the range slider widget, so it is a good idea to include the lower and upper values of the range before and after the slider control. For example, the following code creates a range slider for the attendance field with values range from 10 to 50 in steps of 5 and a default value of 20.

```
10
<input name="attendance" id="attSlider" type="range"
   value="20" step="5" min="10" max="50" />
50
```

REFERENCE

Creating Spinner Controls and Range Sliders

- To create a spinner control for numeric data, enter the `input` element with a type value of "number"

```
<input name="name" id="id" type="number"
   value="value" step="value" min="value" max="value" />
```

where the `value` attribute provides the default field value, the `step` attribute indicates the amount by which the field value changes when a user clicks the spin arrow, the `min` attribute defines the minimum value, and the `max` attribute defines the maximum value of the field.

- To create a range slider control for numeric data, use the following `input` element with a type value of "range":

```
<input name="name" id="id" type="range"
   value="value" step="value" min="value" max="value" />
```

Add a range slider to the survey form now, which customers can use to rate their experience at Red Ball Pizza from 0 (poor) up to 10 (great).

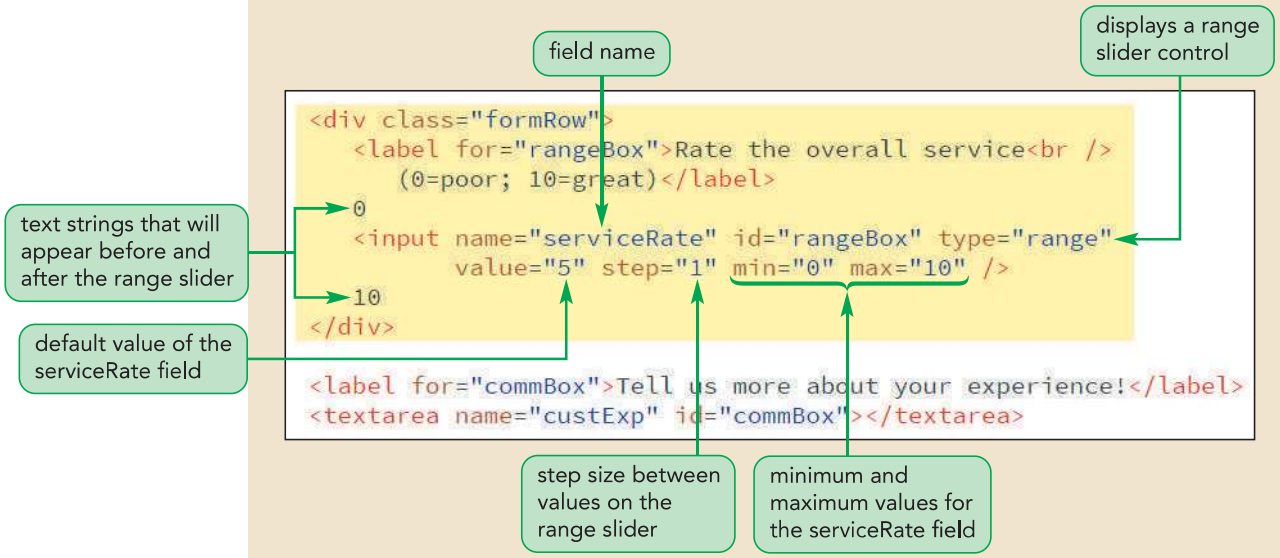
To add a range slider control:

1. Return to the **rb_survey.html** file in your editor and scroll down to the expInfo field set.
2. Directly above the text area control, add the following code to create a range slider control for the serviceRate field:

```
<div class="formRow">
   <label for="rangeBox">Rate the overall service<br />
          (0=poor; 10=great)</label>
   0
   <input name="serviceRate" id="rangeBox" type="range"
          value="5" step="1" min="0" max="10" />
   10
</div>
```

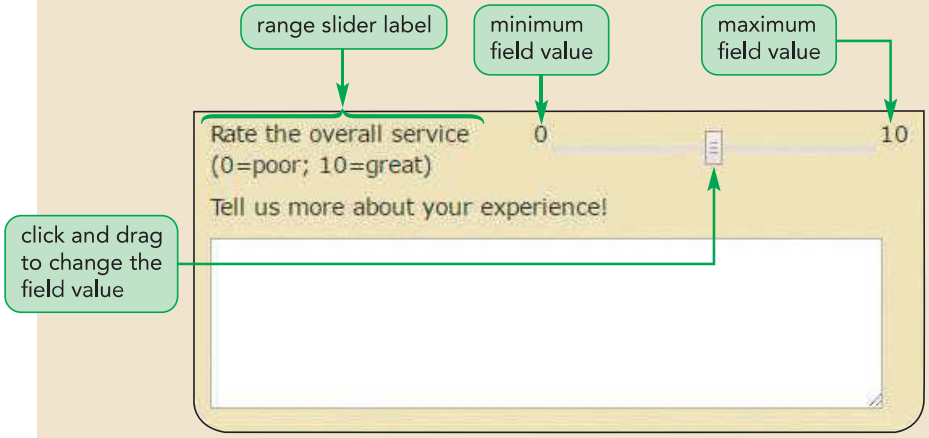
Figure 7–42 highlights the code for the range slider control and label.

Figure 7–42 Creating a range slider control for the serviceRate field



3. Save your changes to the file and then reload rb_survey.html in your browser. Figure 7–43 shows the appearance of the range slider control in the Google Chrome browser.

Figure 7–43 Viewing the range slider control



Trouble? Other browsers will display different styles for the range slider widget. For example, Microsoft Edge and Internet Explorer will display a colored bar with a pop-up window showing the current value of the serviceRate field. If your browser does not support the range slider widget, it will display a text input box.

4. Verify that you can drag the marker on the range slider to the left and right.

INSIGHT

Styles for Widgets

The appearance of a form widget is largely determined by the browser and there are no CSS styles to alter it. However, most browsers do provide style extensions that allow you to modify their widgets. One useful browser extension is the following `appearance` extension that defines the widget associated with the form control

```
-moz-appearance: type;
-webkit-appearance: type;
```

where *type* is the type of widget including `none` (for no widget), `button`, `checkbox`, `listbox`, `radio`, `range`, `spinner`, `textfield`, and many other types depending on the browser. For example, to display a selection list as an input box, you would apply the following style rule:

```
select {
   -moz-appearance: textfield;
   -webkit-appearance: textfield;
}
```

The selection list options will still appear but as pop-ups for the input box. You should use these browser extensions with care because they are not part of the CSS standard, and thus respond unpredictably.

The next data field that Alice wants added to the survey form is a text box, which customers can use to indicate their favorite Red Ball Pizza dish. There are a lot of possible answers and Alice doesn't want to limit the options to a selection list, but she does want to provide suggestions to customers as they type their entries. You can add these suggestions with a data list.

Suggesting Options with Data Lists

A **data list** is a list of possible data values that a form field can have. When applied to an input box, the data values appear as a pop-up list of suggested values. Data lists are defined using the following `datalist` element

```
<datalist id="id">
   <option value="value">
   <option value="value">
…
</datalist>
```

where the value assigned to the different `option` elements provides the suggested entry in the list for its associated `option` element. To apply a data list, add the following `list` attribute to the `input` element

```
<input list="id" />
```

where *id* references the id of the `datalist` element. For example, to create an input box for the favDish field that offers a few suggested items, you could enter the following code:

```
<input name="favDish" type="text" list="dishes" />
<datalist id="dishes">
   <option value="Antipasto Pizza">
   <option value="Big Kahuna Pizza">
   <option value="BBQ Chicken Pizza">
</datalist>
```

TIP

When applied to the range type, a data list appears as tick marks in the range slider widget.

The options in the dishes data list are just suggestions. The customer is not obligated to accept any options and can type a dish of his or her own choosing.

REFERENCE

Creating and Applying a Data List

- To create a data list of possible values, enter

```
<datalist id="id">
   <option value="value">
   <option value="value">
…
</datalist>
```

where each `value` attribute provides the text of a possible value in the data list.

- To reference the data list from an input control, add the `list` attribute

```
<input name="name" list="id" />
```

where *id* references the ID of the data list structure.

Add an input box for the favDish field to the survey form now and augment it with a data list of suggested Red Ball Pizza dishes.

To apply a data list to an input control:

1. Return to the **rb_survey.html** file in your editor and go to the custInfo field set.
2. Directly above the `div` element that encloses the spinner control for the dineOut field, enter the following code to create the input box for the favDish field along with the field's data list of suggested values.

```
<div class="formRow">
   <label for="dish">What's your favorite dish?</label>
   <input name="favDish" id="dish" type="text" list="dishType"
/>
   <datalist id="dishType">
      <option value="Anitpasto Pizza">
      <option value="Big Kahuna Pizza">
      <option value="BBQ Chicken Pizza">
      <option value="Mediterranean Herb Pizza">
      <option value="Pasta Rolls">
      <option value="Pasto Artichoke Pizza">
   </datalist>
</div>
```

Figure 7–44 highlights the code for the input box and data list.

Figure 7–44 Applying a data list to the favDish field

```
<div class="formRow">
   <label for="dish">What's your favorite dish?</label>
   <input name="favDish" id="dish" type="text" list="dishType" />
   <datalist id="dishType">
      <option value="Anitpasto Pizza">
      <option value="Big Kahuna Pizza">
      <option value="BBQ Chicken Pizza">
      <option value="Mediterranean Herb Pizza">
      <option value="Pasta Rolls">
      <option value="Pasto Artichoke Pizza">
   </datalist>
</div>

<div class="formRow">
   <label for="dineSpin">How many times do you dine out per month?</label>
   <input name="dineOut" id="dineSpin" type="number"
          value="1" step="1" min="0" max="20" />
</div>
```

data list containing suggested values

links the favDish field to the dishType data list

3. Save your changes to the file and reload rb_survey.html in your browser.
4. Click the input box for the favDish field and type the letter **p**. Note that the browser displays the list of dishes that start with the letter "p". See Figure 7–45.

Figure 7–45 Viewing suggested data values

Trouble? Currently the Firefox and Chrome browsers will display any data list entry that contains the letter "p" as opposed to only those data list values starting with the letter "p".

Now that you have entered most of the survey form fields, you will examine how to submit the form for processing. To do that, you will create a form button.

Working with Form Buttons

So far, all of your form controls have been used to enter field values. Another type of control is one that performs an action. This is usually done with **form buttons**, which can perform the following actions:

- Run a command from a program linked to the web form.
- Submit the form to a program running on the web server.
- Reset the form fields to their default values.

The first type of button you will examine is the command button.

Creating a Command Button

A **command button** is a button that runs a program, which affects the content of the page or the actions of the browser. Command buttons are created using the following `input` element with the `type` attribute set to `button`

```
<input value="text" onclick="script" type="button" />
```

where *text* is the text that appears on the button and *script* is the name of the program or the program code that is run when the button is clicked by the user. For example, the following `input` element creates a command button containing the text "Run Program", which runs the setup() program when the button is clicked:

```
<input value="Run Program" onclick="setup()" type="button" />
```

There is no need to use command buttons in the Red Ball Pizza survey form.

Creating Submit and Reset Buttons

The two other kinds of form buttons are submit and reset buttons. A **submit button** submits the form to the server for processing when clicked. A **reset button** resets the form, changing all fields to their original default values and deleting any field values that the user might have entered. Submit and reset buttons are created using the following `input` elements with the `type` attribute set to "submit" and "reset" respectively

```
<input value="text" type="submit" />
<input value="text" type="reset" />
```

where once again *text* is the text string that appears on the button.

REFERENCE

Creating Form Buttons

- To create a form button to run a command, use

  ```
  <input value="text" onclick="program" type="button" />
  ```

  where *text* is the text that appears on the button and *program* is the program that is run in response to the user clicking the button.
- To create a form button to submit the form and its fields and values to a script, use

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

- To create a form button to reset the form to its default values and appearance, use

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

Alice wants the survey form to include both a submit button and a reset button. The submit button, which she wants labeled "Submit My Survey", will send the form data to the server for processing when clicked. The reset button, which she wants labeled

"Cancel", will erase the user's input and reset the fields to their default values. Add these two buttons at the bottom of the form within a div element with the ID buttons.

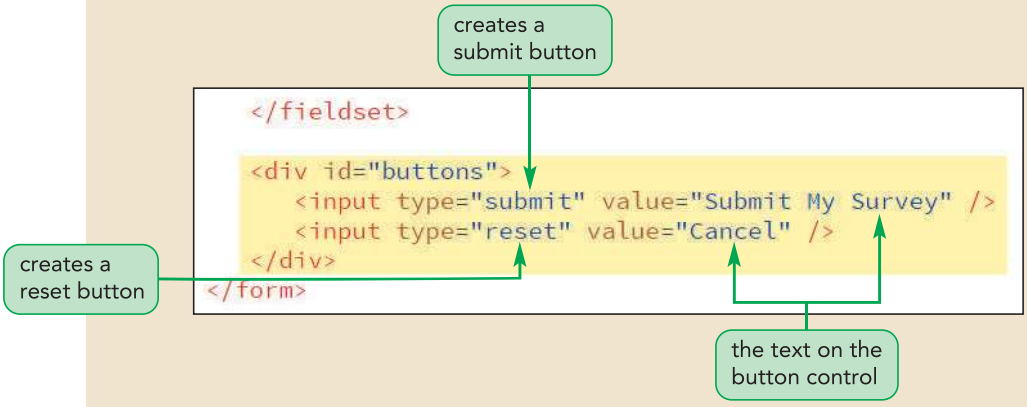
To create submit and reset buttons:

1. Return to the **rb_survey.html** file in your editor and scroll down to the closing `</form>` tag.
2. Directly above the closing `</form>` tag, insert the following code:

```
<div id="buttons">
   <input type="submit" value="Submit My Survey" />
   <input type="reset" value="Cancel" />
</div>
```

Figure 7–46 highlights the code to create the submit and reset buttons.

Figure 7–46 Creating submit and reset buttons



3. Save your changes to the file.

Next, you will format the appearance of the `div` element and the two buttons it contains.

4. Return to the **rb_forms.css** file in your editor and go to the Form Button Styles section.
5. Add the following style rule to set the width of the `div` element to 100% and to horizontally center its content.

```
div#buttons {
   text-align: center;
   width: 100%;
}
```

6. Add the following style rule to set the font size, padding, and margins for all submit and reset buttons in the page.

```
input[type='submit'], input[type='reset'] {
   font-size: 1.2em;
   padding: 5px;
   margin: 15px;
}
```

Figure 7–47 shows the style rules for the form buttons on the page.

Figure 7–47 Styles for the form buttons

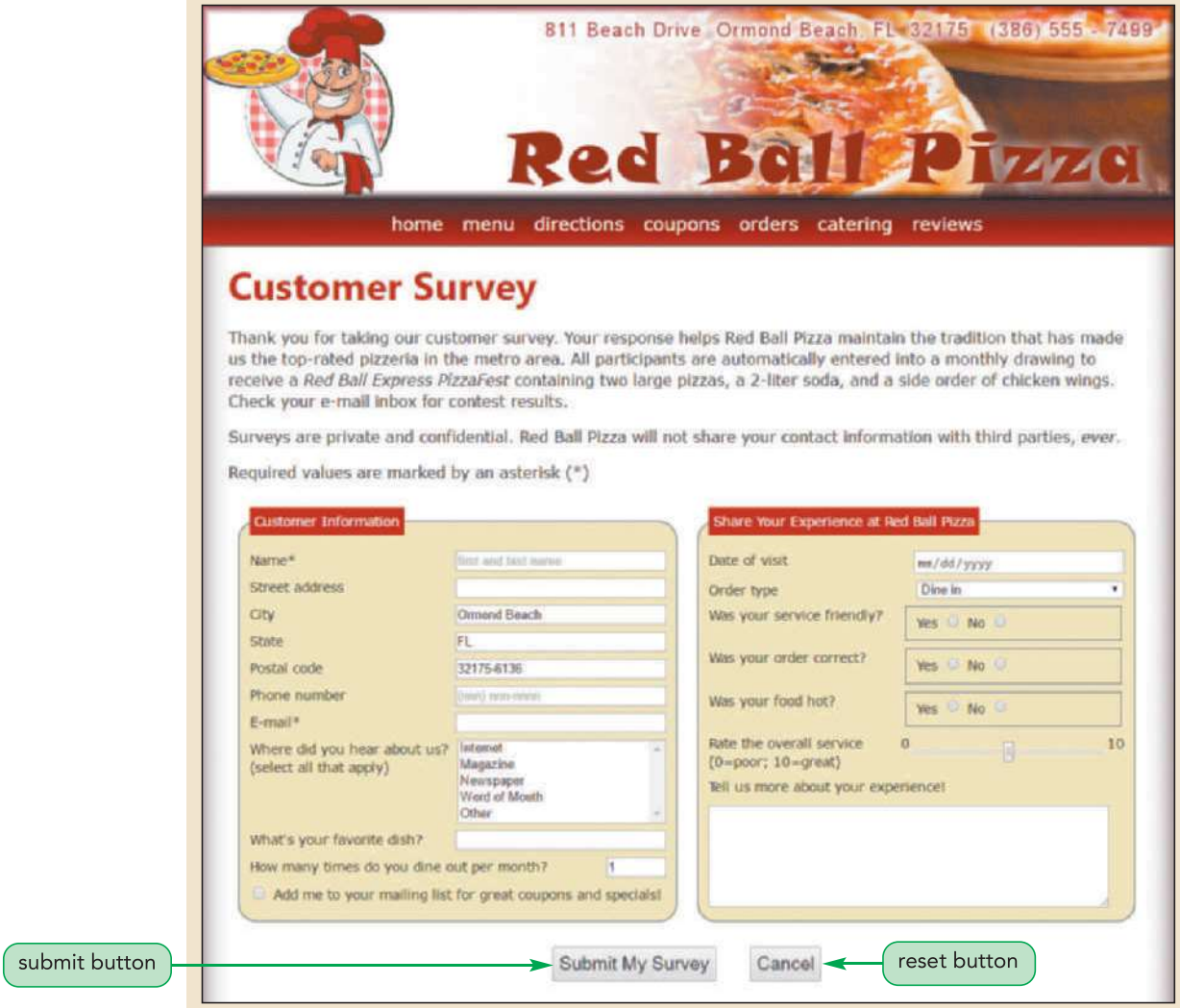
sets the width of the div element to 100% and centers its contents

sets the font size, padding, and margins of the submit and reset buttons

```
/* Form Button Styles */

div#buttons {
   text-align: center;
   width: 100%;
}

input[type='submit'], input[type='reset'] {
   font-size: 1.2em;
   padding: 5px;
   margin: 15px;
}
```

7. Save your changes to the file and then reload rb_survey.html in your browser. Figure 7–48 shows the layout and content of the completed web form.

Figure 7–48 Completed design and layout of the survey form



8. Enter some sample data into the form and then click the **Cancel** button to test the actions of your reset button. Verify that the form is reset to its initial state and the data fields return to their default values. You will test the actions of the submit button shortly.

Designing a Custom Button

The appearance of a command, submit, and reset button is determined by the browser. While you can modify some basic properties such as the button border, font, or background color, you can't add clipart graphics or other features. For more control over a button's appearance use the following `button` element

```
<button type="text">
   content
</button>
```

where the `type` attribute specifies the button type (`submit`, `reset`, or `button`—for creating a command button) and *`content`* are HTML elements placed within the button, including formatted text, inline images, and other design elements supported by HTML. For example, the following code demonstrates how an inline image and text marked as a paragraph can be nested within a submit button.

```
<button type="submit">
   <img src="orderImg.png" alt="" />
   <p>Place your order now</p>
</button>
```

You do not need a custom button in the survey form.

Validating a Web Form

The most important part of form design is ensuring that users enter reasonable values in the correct format. Part of this is accomplished through the use of form controls, such as option buttons and selection lists, which limit the user to a set of pre-approved values. However, there are other data fields that do not easily fit into those types of input controls. For example, how can you ensure that the user has entered a valid credit card number or an email address in the proper format?

TIP

You can turn off client-validation by adding the attribute `novalidate` to the `form` element.

The process of ensuring that the user has supplied valid data is called **validation** and can take place on the web server where it is known as **server-side validation** or within the user's own browser where it is referred to as **client-side validation**. Whenever possible, you should supplement server-side validation with client-side validation to reduce the server's workload. In a payment form, you should verify that the customer correctly completed all of the fields *before* submitting the data to the server so that the server does not have to deal with an improperly completed form.

Identifying Required Values

The first validation test you should perform is to verify that data has been supplied for all required data fields. To identify those fields that are required (as opposed to those that are optional), add the `required` attribute to the control. For example, the following code specifies that the custName field is required and cannot be left blank by the user:

```
<input name="custName" required />
```

In the same way, the `required` attribute can be added to the `select` element or the `textarea` box to make those data fields required. However, for a `select` element to be required, the first option in the selection list must have empty value or no text content.

If a required field is left blank in an input or `textarea` box, the browser will not submit the form but will return an error message instead, indicating that the required data field has not been filled out. If no option is chosen from a required selection list, the browser will once again return an error message.

For the Red Ball Pizza survey form, Alice wants every customer to enter a name and an email address, so she asks you to make the custName and custEmail fields required.

To create submit and reset buttons:

1. Return to the **rb_survey.html** file in your editor.
2. Add the attribute **`required`** to the `input` element for both the custName and custEmail fields.

   Figure 7–49 highlights the newly added required attribute.

Figure 7–49 Making custName and custEmail required fields

marks custName as a required field

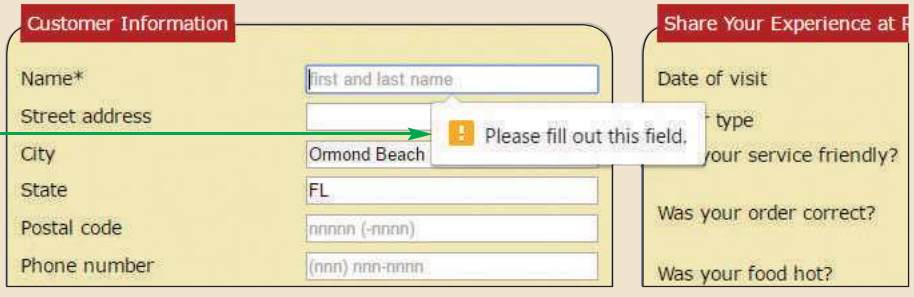
marks custEmail as a required field

```
<div class="formRow">
   <label for="name">Name*</label>
   <input name="custName" id="name" type="text" placeholder="first and last name" required />
</div>

<div class="formRow">
   <label for="mail">E-mail*</label>
   <input name="custEmail" id="mail" type="email" required />
</div>
```

3. Save your changes to the file and then reload rb_survey.html in your browser.
4. Test your form by clicking the **Submit My Survey** button without entering any values into the form itself. As shown in Figure 7–50, the browser fails to submit the form and instead displays a bubble containing the message that the custName field needs to be filled out.

Figure 7–50 Validation error message in Google Chrome

the form fails the validation test when no customer name is provided



Trouble? Figure 7–50 shows the error message rendered using the Google Chrome browser. The exact text and format of the validation bubble will vary from one browser to the next.

5. Enter ***your name*** into the custName field and then resubmit the form (without entering an email address). Verify that the custName field now passes validation but a bubble with a validation error message appears next to the blank email box.
6. Enter ***your email address*** into the custEmail field and then resubmit the form. Verify that the browser displays an alert message indicating that no invalid data have been detected.
7. Click the **OK** button to dismiss the dialog box.

The dialog box you encountered in Step 6 is not part of HTML or your browser. It was generated from the rb_formsubmit.js script file that you linked the web page to in Session 1. The confirmation dialog box only appears when no validation errors have been detected in the submitted form. Also, note that all of your data values have been preserved in the survey form. This is also a feature of the script file to avoid re-typing field values as you continue to test the web form. If you want to clear the form to see the default values, reopen the file in your browser.

Validating Based on Data Type

A form will fail the validation test if the data values entered into a field do not match the field type. For example, a data field with the `number` type will be rejected if non-numeric data is entered. Similarly, fields marked using the `email` and `url` types will be rejected if a user provides an invalid email address or text that does not match the format of a URL.

You have already specified data types for the survey form fields. Verify that the form will not accept invalid data for the custEmail field.

To verify the form does not accept invalid data:

1. Click the input box for the custEmail field in the survey form and type the text **Alice Nichols** (or any text string that does not represent an email address).
2. Click the **Submit My Survey** button to submit the form.

   As shown in Figure 7–51, the browser rejects the form based on the invalid data entered for the custEmail field.

Figure 7–51 Rejecting an invalid email address

3. Change the field value to **alice.nichols@example.com** and resubmit the form. Verify that the form now passes the validation test.
4. Click the **OK** button to close the JavaScript dialog box.

Accepting the email address does not mean that the email address is real; it only means that the text field value follows the proper general pattern for email addresses, which is a string of characters with no blank spaces followed by the `@` symbol and then followed by another string of nonblank characters. For validation tests that involve more complicated text patterns, you can do a pattern test.

Testing for a Valid Pattern

To test whether a field value follows a valid pattern of characters, you can test the character string against a regular expression. A **regular expression** or **regex** is a concise description of a character pattern. It is beyond the scope of this tutorial to discuss the syntax of regular expressions, but to validate a text value against a regular expression, add the following `pattern` attribute to the `input` element

```
pattern="regex"
```

where *regex* is the regular expression pattern. For example, the following code tests the value of the custZip field against the regular expression pattern `^\d{5}$`

```
<input name="custZip" pattern="^\d{5}$" />
```

where the regular expression `^\d{5}$` represents any string of 5 numeric characters. Thus, the value 85017 would match this regular expression, but values like 850177 or X8514 would not. Regular expressions are based on a rich language and can be written to match credit card numbers, phone numbers, email addresses, and so forth.

REFERENCE

Validating Field Values

- To indicate that a field is required, add the `required` attribute to the form control.
- To validate an email address, set the data type to `email`.
- To validate a web address, set the data type to `url`.
- To validate that a text input box follows a character pattern, add the attribute

  `pattern="regex"`

  where *regex* is a regular expression that defines the character pattern.

Alice has obtained regular expressions for phone numbers and 5- or 9-digit postal codes. Add the pattern attribute now to the custZip and custPhone fields to validate those field values. Note that some regular expressions are long and complicated, and you must type them exactly as written. If you make a mistake, you can copy the text of the regular expressions from the rb_regex.txt file in the tutorial.07/tutorial folder.

To test a field value against a regular expression:

1. Return to the **rb_survey.html** file in your editor and scroll down to the `input` element for the custZip field.
2. Add the following attribute to create a regular expression that matches 5- and 9-digit zip codes to the `input` element:

   ```
   pattern="^\d{5}(-\d{4})?$"
   ```

3. Go to the input element for the custPhone field and add the following attribute to create a regular expression that matches phone numbers with or without an area code:

   ```
   pattern="^\d{10}$|^(\(\d{3}\)\s*)?\d{3}[\s-]?\d{4}$"
   ```

   Figure 7–52 highlights the `pattern` attribute for both the custZip and custPhone fields.

Figure 7–52 Pattern matching with regular expressions

```
<div class="formRow">
   <label for="zip">Postal code</label>
   <input name="custZip" id="zip" type="text" placeholder="nnnnn (-nnnn)"
          pattern="^\d{5}(-\d{4})?$" />
</div>

<div class="formRow">
   <label for="phone">Phone number</label>
   <input name="custPhone" id="phone" type="tel" placeholder="(nnn) nnn-nnnn"
          pattern="^\d{10}$|^(\(\d{3}\)\s*)?\d{3}[\s-]?\d{4}$" />
</div>
```

regular expression pattern that matches 5- or 9-digit postal codes

regular expression pattern that matches phone numbers with or without area codes

4. Save your changes to the file and then reload rb_survey.html in your browser.
5. Enter ***your email address*** in the input box for the custEmail field so that the two required fields have a value.
6. Type **321** in the input box for the postal code and then submit the form. As shown in Figure 7–53, the browser rejects the field value because it does not match the pattern of either a 5-or 9-digit postal code.

Figure 7–53 **Rejecting an invalid postal code**

7. Change the postal code value to **32175** and resubmit the form. Verify that the form now passes the validation test.
8. Test the custPhone field by entering **5-7499** in the input box for the customer phone number and then submitting the form. Verify that the browser rejects the data as invalid.
9. Change the phone number to **555-7499** and resubmit the form, verifying that it now passes the validation test.

Defining the Length of the Field Value

Because older browsers might not support the `pattern` attribute, you can do a simple test based on character length using the following `maxlength` attribute

```
maxlength="value"
```

where *value* is the maximum number of characters in the field value. For example, the following `input` element limits the number of characters in the custZip field to 5, which means that field values with more than 5 characters will not be validated.

```
<input name="custZip" maxlength="5" />
```

Note that the `maxlength` attribute does not distinguish between characters and digits. A user could enter the text string *abcde* as easily as *32175* and have the field values pass validation.

INSIGHT

WebKit Styles for Validation Messages

Like widgets, the appearance of the bubble containing the validation message is determined by the browser. There is no standard CSS style to format the error message but there are browser extensions that give you more control over the error message style. For Google Chrome, the validation message is organized into the following pseudo-elements selectors:

- `::-webkit-validation-bubble:` Selecting the entire bubble containing the validation message
- `::-webkit-validation-bubble-arrow:` Selecting the pointing arrow above the validation bubble
- `::-webkit-validation-bubble-message:` Selecting the validation message within the bubble
- `::-webkit-validation-bubble-arrow-clipper:` Selecting the bubble behind the top arrow

To modify the appearance of the validation message, you can apply the following style rule, which displays the message in a gray font on an ivory background.

```
::-webkit-validation-bubble-message {
    color: gray;
    background: ivory;
}
```

Other browsers support their own collection of extensions to modify the appearance of the validation bubble. Because these are not part of the CSS standards, there is no common syntax yet for modifying the validation message. You can learn more about these extensions by viewing the documentation on the browser manufacturer's website.

Applying Inline Validation

One disadvantage with the validation tests you have applied is that they all occur after a user has completed and submitted the form. It is extremely annoying for the user to go back to an already completed form to fix an error. Studies have shown that users are less likely to make errors and can complete a form faster if they are informed of data entry errors as they occur. The technique of immediate data validation and reporting of errors is known as inline validation.

Using the focus Pseudo-Class

One way of integrating inline validation with a web form is to change the display style of fields that currently contain invalid data. This can be done using some of the pseudo-classes described in Figure 7–54.

Figure 7–54 Pseudo-classes for form controls and fields

Pseudo-Class	Matches
`checked`	A check box or option button that is selected or checked
`default`	A default control, such as the default option in a selection list
`disabled`	A control that is disabled
`enabled`	A control that is enabled
`focus`	A control that has the focus (is actively selected) in the form
`indeterminate`	A check box or option button whose toggle states (checked or unchecked) cannot be determined
`in-range`	A field whose value lies within the allowed range (between the `min` and `max` attribute values)
`invalid`	A field whose value fails the validation test
`optional`	A field that is optional (not required) in the form
`out-of-range`	A field whose value lies outside the allowed range (outside the `min` and `max` attribute values)
`required`	A field that is required in the form
`valid`	A field whose value passes the validation test

For example, to create styles for all of the checked option buttons in the form, you could apply the `checked` pseudo-class, as in the following style rule

```
input[type="radio"]:checked {
   styles
}
```

where *styles* are the CSS styles applied to checked option buttons. Note that option buttons that are not checked will not receive these styles.

The first pseudo-class you will apply to the survey form will be used to change the background color of any element that has the focus. **Focus** refers to the state in which an element has been clicked by the user, making it the active control on the form. You may have noticed that some browsers highlight or add a glowing border around input boxes that have the focus.

Alice would like the input boxes, selection lists, and text area boxes that have the focus to be displayed with a light green background color.

To create style rules for elements that have the focus:

1. Return to the **rb_forms.css** file in your editor and scroll down to the Validation Styles section.
2. Add the following style rule to change the background color to light green for all `input`, `select`, and `textarea` elements that have the focus.

```
input:focus, select:focus, textarea:focus {
   background-color: rgb(220, 255, 220);
}
```

Figure 7–55 highlights the style rule to change the background color.

Figure 7–55 Creating a style rule for the focus pseudo-class

change the background color to light green when the control element has the focus

```
/* Validation Styles */

input:focus, select:focus, textarea:focus {
   background-color: rgb(220, 255, 220);
}
```

3. Save your changes to the file and then reload rb_survey.html in your browser.
4. Click the input box for the customer name and verify that the background color changes to a light green as shown in Figure 7–56.

Figure 7–56 Text box with the focus

5. Press the **Tab** key repeatedly to change the focus to the remaining input controls. Verify the background color changes to light green when an input control has the focus and has a background.

Pseudo-Classes for Valid and Invalid Data

The `valid` and `invalid` pseudo-classes are used to format controls based on whether their field values pass a validation test or not. For example, the following style rule displays all `input` elements containing invalid data with a light red background

```
input:invalid {
   background-color: rgb(255, 232, 233);
}
```

while the following style rule displays all `input` elements containing valid data with a light green background:

```
input:valid {
   background-color: rgb(220, 255, 220);
}
```

Both of these style rules set the background color whether the `input` element has the focus or not. Displaying a form full of input backgrounds with different background colors can be confusing and distracting to the user. As a result, it is better practice to highlight invalid field values only when those input controls have the focus, as in the following style rule that combines both the `focus` and `invalid` pseudo-classes:

```
input:focus:invalid {
   background-color: rgb(255, 232, 233);
}
```

Alice suggests that the form perform inline validation for the input boxes with IDs of "name", "zip", "phone", and "mail". For valid data, she wants those input boxes to be displayed with a light green background along with a green check mark image. For invalid data, she wants the background to be light red with a red X image. Use the rb_valid.png and rb_invalid.png image files for the green check mark and red X images.

1. Return to the **rb_forms.css** file in your editor and scroll to the bottom of the file.

Include the focus pseudo-class so that the validation style is only applied when the control is active in the form.

2. Add the following style rule to display a light green background and a green check mark image when valid data is entered in the custName, custZip, custPhone, and custEmail fields:

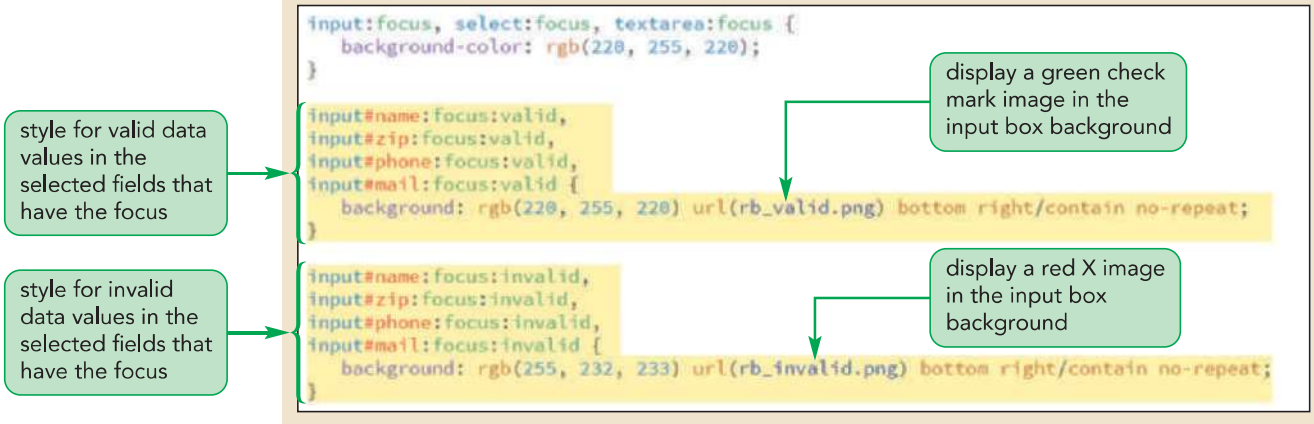
```
input#name:focus:valid,
input#zip:focus:valid,
input#phone:focus:valid,
input#mail:focus:valid {
   background: rgb(220, 255, 220) url(rb_valid.png) bottom
right/contain no-repeat;
}
```

3. Add the following style rule to display a light red background and a red X image when invalid data is entered in those same fields:

```
input#name:focus:invalid,
input#zip:focus:invalid,
input#phone:focus:invalid,
input#mail:focus:invalid {
   background: rgb(255, 232, 233) url(rb_invalid.png) bottom
right/contain no-repeat;
}
```

Figure 7–57 highlights the style rules to style valid and invalid data.

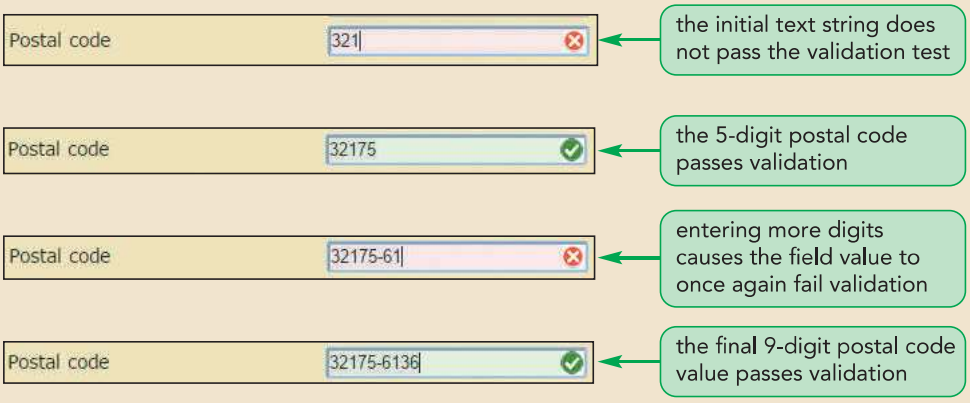
Figure 7–57 Creating styles for valid and invalid field values



4. Save your changes to the style sheet and then reload rb_survey.html in your browser.

5. Test inline validation by typing the zip code value **32175-6136** into the input box for the customer's zip code. Note that the input box provides immediate visual feedback on whether the current field value passes the validation test. See Figure 7–58.

Figure 7–58 Inline validation on the customer postal code



6. Continue to test the web form by entering data into the other input boxes, noting how the form automatically performs a validation test on your data values.

PROSKILLS

Problem Solving: Using Form Building Tools

One of the limitations of CSS is that it does not provide an easy way to format the form controls other than basic styles for text and background colors. To gain more control over your form controls, you may want to explore third party frameworks that provide customized widgets and form design tools. Some popular frameworks include:

- Google Forms (*docs.google.com/forms*): A free service for form design that also automatically tabulates the user responses in an online spreadsheet
- Wufoo (*wufoo.com*): A paid service that supplies a powerful form builder engine and tools for uploading documents and images
- Jotform (*www.jotform.com*): A paid service with form tools and the ability to automatically upload completed forms to your website
- Form Stack (*www.formstack.com*): A paid service with form building software and tools to manage workflow, data analysis, and tabulation

Form building tools can speed up the process of designing and testing your web forms. However, like all frameworks they are best used when you have a good understanding of the underlying HTML and CSS code that they employ.

You have finished your work on the survey form. Alice will place a copy of your files in a folder on the company's web server and from there the form can continue to be tested to verify that the server program and the web form work well together. Alice is pleased with your work on this project and will get back to you to create other web forms for the Red Ball Pizza website.

REVIEW

Session 7.3 Quick Check

1. To create a spinner control that goes from 0 to 100 in steps of 10 with a default value of 50, which of the following attributes would you *not* include in the <input /> tag?
   a. `min = "0"`
   b. `max = "100"`
   c. `value = "50"`
   d. `type = "spinner"`
2. What attribute do you add to an `<input />` tag to create a range slider?
   a. `type = "range"`
   b. `type = "slider"`
   c. `control = "range"`
   d. `number = "slider"`
3. What attribute do you add to an `<input />` tag to apply the data list "cityList" to the input box control?
   a. `list = "cityList"`
   b. `data = "cityList"`
   c. `datalist = "cityList"`
   d. `values = "cityList"`
4. The code to create a submit button containing the text "Add Order" is:
   a. `<input type="button">Add Order</input>`
   b. `<input type="button" value="Add Order" />`
   c. `<input type="submit" value="Add Order" />`
   d. `<input type="submit" label="Add Order" />`
5. What attribute do you add to an `input` element to force the user to enter a value?
   a. `type = "required"`
   b. `required`
   c. `type = "autocomplete"`
   d. `autocomplete`
6. What attribute do you add to an `input` element to validate the element's value against the regular expression "^\user\-d{4}$"?
   a. `regular="^\user\-d{4}$"`
   b. `valid="^\user\-d{4}$"`
   c. `value="^\user\-d{4}$"`
   d. `pattern="^\user\-d{4}$"`
7. What selector do you use to match all `input` elements that have the focus?
   a. `input:focus`
   b. `input#focus`
   c. `input:hasFocus`
   d. `input.hasFocus`
8. What is inline validation?
   a. Validation applied to inline elements
   b. Validation with the error message displayed inline with the text
   c. Validation that is applied immediately to the element prior to the form submission
   d. All of the above