Designing a Custom Button

The text of a command, submit, or reset button is determined by the `value` attribute. You are only allowed to specify the text displayed on the button; you can't add other elements such as an inline image to the button value. For more control over a button's appearance, you can use the `button` element as follows

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

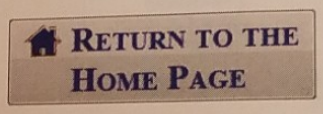
where the `type` attribute specifies the button type (`submit`, `reset`, or `button`—for creating a command button) and *`content`* is page elements displayed within the button. The page content can include formatted text, inline images, and other design elements supported by HTML. Figure 6-66 shows an example of a button that contains both formatted text and an inline image.

ure 6-66 Creating a custom button

HTML code

```
<button type="button">
   <img src="home.png" />
   Return to the <br /> Home Page
</button>
```

custom button



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

Validating a Web Form

The final part of your work on the survey form is to test your ability to submit values from the form. Data values often need to be tested or **validated** before they can be used. Validation can occur after the data is sent to the server with **server-side validation**, and it also can be tested on a user's own computer via **client-side validation** before sending it to the server. Whenever possible, you should supplement your server-side validation with client-side validation to reduce the workload on the server. For example, in a payment form, you'll want to verify that a user has entered a valid credit card number and completed all of the fields required for payment before submitting the data to the server. Sending an invalid payment to the server slows down the process and puts an extra burden on a server that may be dealing with hundreds or thousands of transactions every hour.

HTML5 includes several attributes that can be used to perform client-side validation. At the time of this writing, Opera, Firefox, and Google Chrome support client-side validation under HTML5; Internet Explorer and Safari do not. Once again, if client-side validation is an essential part of your Web site design, use a JavaScript library like jQuery or a program of your own creation to perform the validation on the user's computer. However, you also can supplement these programs with HTML5's built-in validation attributes.

The first HTML5 attribute you'll examine can be used to ensure that the user completes all required fields.

Indicating Required Values

In the survey form, Alice has indicated that the `custname`, `email`, and `receipt` fields all must be completed for the survey to be valid. If any of those fields are left blank, she wants the browser to notify the user of the missing data and cancel the submission. You can indicate that a field is required by adding the following `required` attribute to the control element:

```
required="required"
```

If a required field is left blank and the submit button is clicked, the browser will cancel the submission and display an error message.

REFERENCE

Validating Field Values

- To indicate that a field is required, add the `required="required"` attribute to the control element.
- To validate an e-mail 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.

To see how validation works with HTML5, add the `required` attribute to the `custname`, `email`, and `receipt` fields.

To apply and test the `required` attribute:

1. Return to the **survey.htm** file in your text editor.
2. Add the attribute **`required="required"`** to the `input` element for the `custname` field at the top of the form.
3. Scroll down to the `input` element for the `email` field and then add the attribute **`required="required"`**.
4. Go to the `receipt` field and then add the attribute **`required="required"`** to the `input` element. Figure 6-67 highlights the newly added code.

Figure 6-67 Marking required fields

```
<fieldset id="custInfo">
  <legend>Customer Information</legend>

  <label for="custname">Name *</label>
  <input name="custname" id="custname"
         placeholder="first and last name"
         required="required" />

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

  <label for="city">City</label>
  <input name="city" id="city" value="Ormond Beach" />

  <label for="state">State (abbr.)</label>
  <input name="state" id="state" value="FL" />

  <label for="zip">Postal code</label>
  <input name="zip" id="zip"
         placeholder="nnnnn (-nnnn)" />

  <label for="phone">Phone number</label>
  <input name="phone" id="phone" type="tel"
         placeholder="(nnn) nnn-nnnn" />

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

the custname field is required

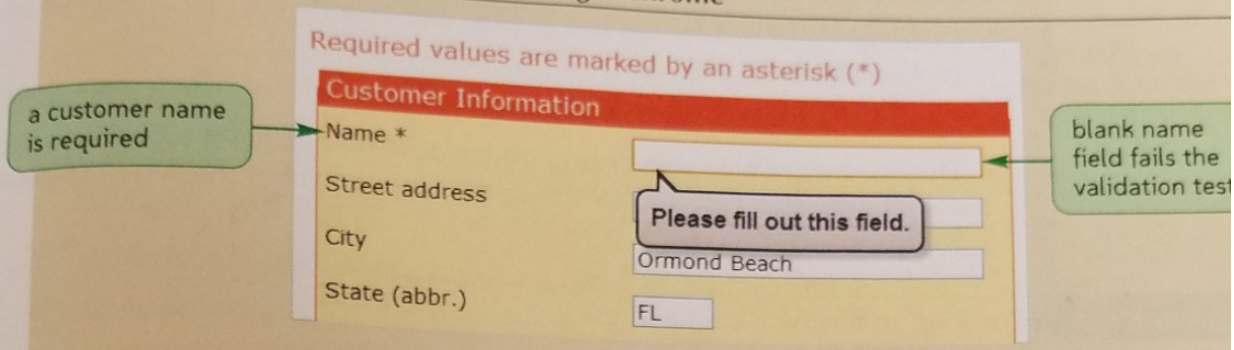
```
<fieldset id="experience">
  <legend>Share Your Experience at Red Ball Pizza</legend>

  <label for="visitdate">Date of visit</label>
  <input name="visitdate" id="visitdate" type="date" />

  <label for="receipt">Receipt number *</label>
  <input name="receipt" id="receipt"
         placeholder="re-nnnnnn"
         required="required" />
```

the email and receipt fields are both required

5. Save your changes to the file and then refresh **survey.htm** in the Opera or C Chrome browser.

6. Leaving the form blank, click the **Submit My Survey** button. As shown in Figure 6-68, in the Google Chrome browser for Windows, the browser does not submit the form but displays an error message for the first invalid field encounters.

Data validation error message in Google Chrome



Trouble? If you are running Internet Explorer or Safari, you might not see an error message.

7. Enter **Alice Nichols** in the Name box and then click the **Submit My Survey** button again. Verify that the browser displays an error message next to the blank input box for the `email` field.
8. Enter **alice.nichols@redballpizza.com** in the E-mail box and submit the form again. Verify that an error message now appears next to the input box for the `receipt` field.
9. Enter **re-123456** in the Receipt number box and submit the form one last time. Verify that no error messages are displayed by the browser, and that the browse displays a dialog box with the message *No invalid data detected. Will retain data for further testing.*

   The dialog box you encountered in Step 9 is not part of HTML5 or your Web browser; it comes from the *formsubmit.js* JavaScript file you linked to back in th first session. The message appears only once you've passed all of the HTML5 val dation checks.
10. Click the **OK** button to close the dialog box and return to the Web page.

Note that all of your data values have been preserved in the survey form. This is als feature of the *formsubmit.js* JavaScript file to avoid re-typing field values as you contin to test the Web page.

Validating Based on Data Type

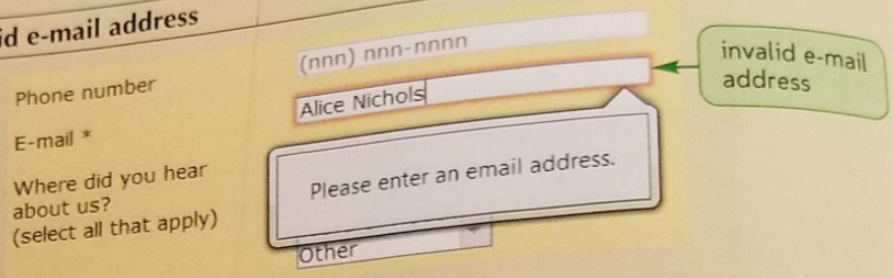
The new data types supported by HTML5 also can be used for data validation. For example, a data field with the `number` data type will be rejected if non-numeric data entered. Similarly, fields marked using the `email` and `url` fields will be rejected if a u provides an invalid e-mail address or Web site URL.

E-mail addresses must be entered in the form *username@domain*. Verify that your browser rejects an invalid e-mail address by attempting to enter erroneous data in the `email` field.

To test the email field:

1. Click the input box for the email field and change the text of the e-mail address to **Alice Nichols**.
2. Click the **Submit My Survey** button.
3. The browser rejects your text entry because it does not match the pattern of an e-mail message. Figure 6-69 shows the appearance of the error message in the Firefox browser.

69 **Entering an invalid e-mail address**



4. Change the text of the input box back to **alice.nichols@redballpizza.com** and re-submit the form. Verify that no validation errors are reported.

A browser that supports the `date` data type also rejects invalid dates if they are not in the form *yyyy-mm-dd*. However, a browser that supports the `date` data type also provides a calendar widget to allow for the easy submission of valid dates. Currently, only the Google Chrome and Opera browsers provide validation checks on date values.

Testing for a Valid Pattern

Several fields in the survey form are required to follow a specified pattern of characters. For example, the receipt numbers from Red Ball Pizza all follow the pattern `re-nnnnnn` where *n* is a single digit. Thus, a receipt labeled re-123456 would be considered valid, but receipt numbers such as 123456 or re-1234 would not. Other field values are limited to a set of possible character patterns. A U.S. phone number might be entered as (386) 555 - 7499, or 555-7499, or 3865557499. United States postal codes can be entered as 32175, or in the nine-digit format as 32175-6316. However, a phone number would not be valid without at least seven numbers, and a postal code would not be valid if it were written with other than five or nine numbers.

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 attribute

```
pattern="regex"
```

to the control element, where *regex* is the regular expression pattern. For example, the following code tests the value of the `receipt` field against the regular expression `^re\-\d{6}$`:

```
<input name="receipt" pattern="^re\-\d{6}$" />
```

This regular expression will cause browsers to reject any value for the `receipt` field that is not exactly in the form `re-nnnnnn`, where *n* is a single digit.

TIP

The pattern attribute can be used only with input elements that store text. You cannot use it with other data types.

Alice has obtained some regular expressions for phone numbers, postal codes, and Red Ball Pizza receipt numbers. You'll add these regular expression patterns to the phone, zip, and receipt fields, and then test them in your browser. Note that some of the 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 *regex.txt* file in the tutorial.06/tutorial folder.

To apply and test regular expression patterns:

1. Return to the **survey.htm** file in your text editor.
2. Within the `input` element for the `zip` field, insert the following regular expression pattern that tests for the presence of a five- or nine-digit postal code:

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

3. Go to the `input` element for the `phone` field. Change the data type from `type="tel"` to `type="text"` and add the following attribute that tests for a valid phone number pattern:

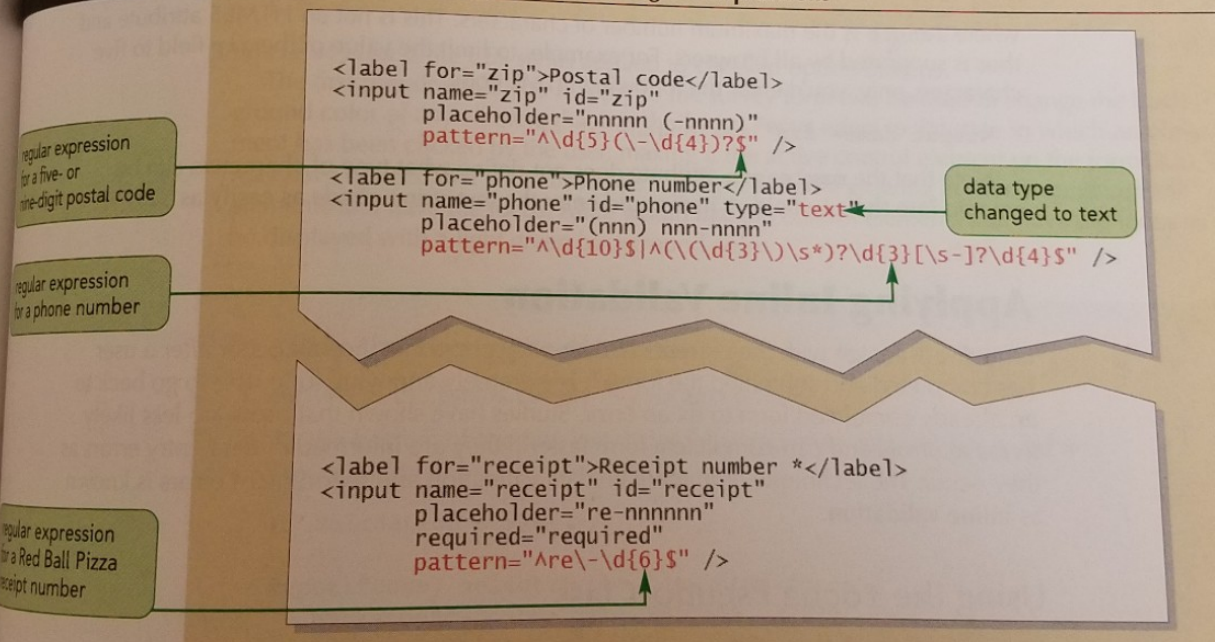
   ```
   pattern="^\d{10}$|^(\(\d{3}\)\s*)?\d{3}[\s-]?\d{4}$"
   ```

4. Scroll down to the `input` element for the `receipt` field and insert the following attribute that tests for a valid receipt number:

   ```
   pattern="^re\-\d{6}$"
   ```

   Figure 6-70 highlights the revised text in the file.

Figure 6-70 Specifying character patterns with regular expressions

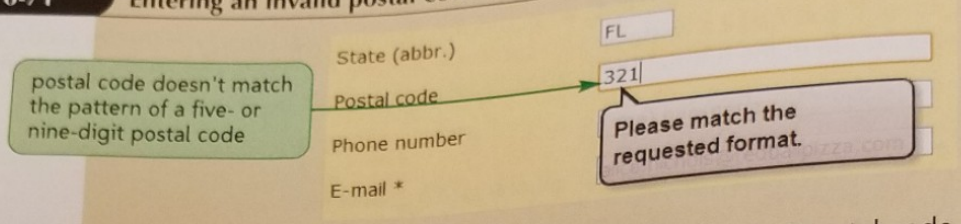


TIP

You can provide a more descriptive error message by adding the `title` attribute to the `input` element along with the text of your error message.

5. Save your changes to the file and then refresh **survey.htm** in your Web browser. Enter some text in the Name input box.
6. Type **321** in the input box for the postal code and then submit the form. As shown in Figure 6-71, the browser rejects the field value because it does not match the pattern of either a five-digit or a nine-digit postal code.

igure 6-71 Entering an invalid postal code



Trouble? If your browser does not reject the invalid postal code, it might not support character pattern validation. Currently, only some versions of the Firefox, Opera, and Google Chrome browsers support validation of character patterns.

7. Change the value of the `zip` field to **32175** and then resubmit the form. Verify that the form is submitted without error.
8. Try entering values for the `phone` and `receipt` fields that do not match the character patterns for phone numbers or receipt numbers, verifying that the form is rejected when invalid values are entered.
9. Enter **(386) 555-7499** for the phone field and **re-123456** for the receipt field and submit the form.

You also can use HTML to define the maximum number of characters in a field using the `maxlength` attribute

```
<input name="name" maxlength="value" />
```

where *`length`* is the maximum number of characters. This is not an HTML5 attribute and thus *is* supported by all browsers. For example, to limit the value of the `zip` field to five characters only, you'd enter the following HTML code:

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

Note that the `maxlength` attribute does not define what type of characters can be entered into the `zip` field. A user could enter the text string *abcde* as easily as *32175*.

Applying Inline Validation

One disadvantage with the current validation checks is that they all occur *after* a user has completed and submitted the form. It is extremely annoying for a 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 into a Web form is to create style rules that change the appearance of each control element based on the validity of the data it contains. This can be done using some of the CSS3 pseudo-classes described in Figure 6-72.

.72 **Pseudo-classes for Web form controls**

Pseudo-Class	Matches
`checked`	Check boxes or options that are checked
`default`	The default user control element
`disabled`	Control elements that are disabled
`enabled`	Control elements that are enabled
`focus`	Control elements that have the focus (are actively selected) in the form
`indeterminate`	Check boxes or option buttons whose toggle states (checked or unchecked) cannot be determined
`in-range`	Control elements whose values are within each field's range of values (between the `min` and `max` attribute values)
`invalid`	Control elements whose values fail validation tests
`optional`	Control elements that are optional (not required) in the Web form
`out-of-range`	Control elements whose values are outside each field's range of values (outside of the `min` and `max` attribute values)
`required`	Control elements that are required in the Web form
`valid`	Control elements whose values pass validation tests

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

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

where *styles* is the CSS styles applied to checked option buttons.

The first pseudo-class you'll 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 element on the form. You may have noticed that some browsers highlight or add a glowing border around control elements that have the focus. Alice would like the control elements that have the focus be displayed with a light green background color.

To apply the `focus` pseudo-class:

1. Return to the **forms.css** file in your text editor.
2. At the bottom of the file, insert the following style rule, as displayed in Figure 6-73:

```
/* Validation styles */

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

Figure 6-73 Style rule for elements that have the focus

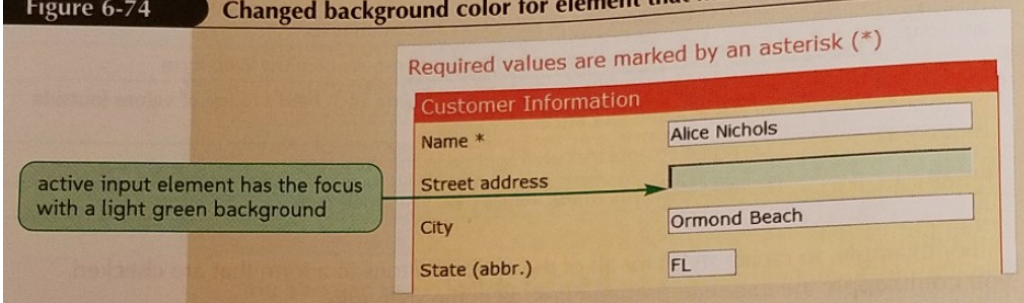
selector for input, select, and textarea elements that have the focus

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

changes the background color to light green

3. Save your changes to the file and then refresh **survey.htm** in your Web browser.
4. Click the input box for the `street` field and verify that the background color changes to a light green as the input box receives the focus (see Figure 6-74).

Figure 6-74 Changed background color for element that has the focus



Note that the `focus` pseudo-class is supported by all current browsers, so you can feel confident in adding it to your Web forms.

Pseudo-Classes for Valid and Invalid Data

The `valid` and `invalid` pseudo-classes can be used to create styles for control elements based on whether the field value associated with a control is valid or invalid. For example, the style rule

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

displays all `input` elements for invalid data with a light red background, while the style rule

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

displays all `input` elements for valid data with a light green background. Both of these style rules set the background color whether the `input` element has the focus or not. Displaying a multitude of input boxes with different background colors can be confusing and distracting to the user, however. To set the background color only when a field is valid or invalid and the input box has the focus, you combine the two pseudo-classes as follows:

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

Alice wants you to display all invalid data with a light red background when the `input` element has the focus. In addition, she wants you to add an image of a red x to the input box background. For valid data, she wants the input box to be displayed with a light green background along with a green check mark image.

To set styles for the valid and invalid data:

1. Return to the **forms.css** file in your text editor. At the bottom of the file, add the following style rule for input boxes containing valid data:

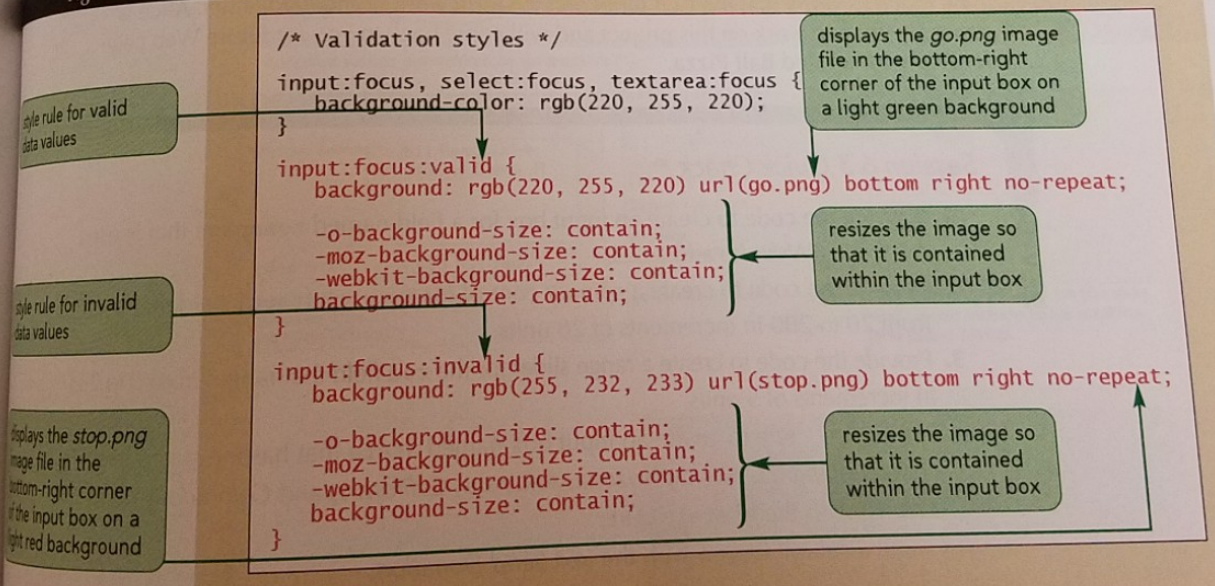
```
input:focus:valid {
   background: rgb(220, 255, 220) url(go.png) bottom right
no-repeat;

   -o-background-size: contain;
   -moz-background-size: contain;
   -webkit-background-size: contain;
   background-size: contain;
}
```

2. Add the following style rule for invalid data:

```
input:focus:invalid {
   background: rgb(255, 232, 233) url(stop.png) bottom right
no-repeat;

   -o-background-size: contain;
   -moz-background-size: contain;
   -webkit-background-size: contain;
   background-size: contain;
}
```

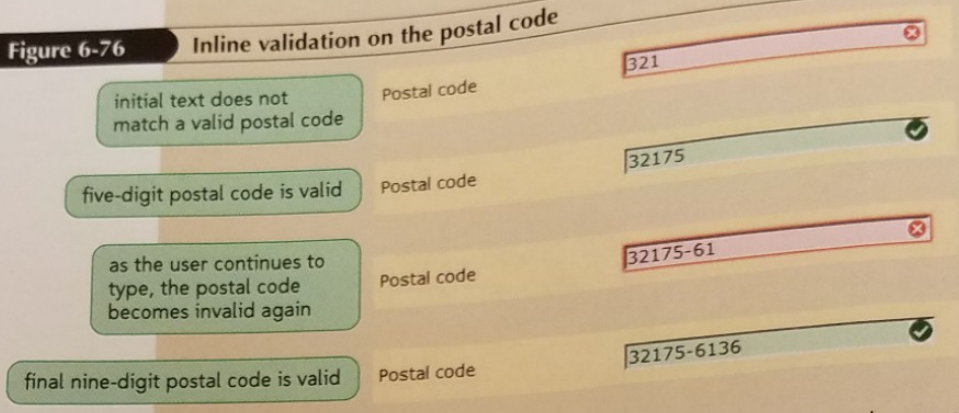
Figure 6-75 displays the content of the new style rules.

Figure 6-75 Style rules for valid and invalid field values



3. Save your changes to the file and then refresh **survey.htm** in your Web browser.
4. Test the inline validation by typing the postal code value **32175-6136** into the `zip` field. Note that the background of the input box provides immediate visual feedback on whether the data value you enter is currently valid or invalid (see Figure 6-76).

Figure 6-76 Inline validation on the postal code



5. Continue to enter different values into the different input boxes and verify the effectiveness of the inline validation tools in providing immediate feedback on the validity of your data.

   Trouble? If you're using Internet Explorer or Safari, you might not see a change in the background of the different input boxes as you enter the field values. If you are using Opera, the background image will change but the background color will not.

You've finished the initial work on the survey form. Alice has placed a copy of your files in a folder on the company's Web server, and from there the Web form can continue to be tested to verify that the CGI script and the form work properly together. Alice is pleased with your work on this project and will come back to you for future Web page development for Red Ball Pizza.

REVIEW

Session 6.3 Quick Check

1. Provide the code to create an input box for a field named `homepage` that is used for storing Web site addresses.
2. Provide the code to create a spinner control for the `withdrawal` field that ranges from 20 to 200 in increments of 20 units.
3. Provide the code to create a range slider for the `red` field that ranges from 0 to 255 in increments of 5 units.
4. Provide the code to create an input box named `state` that has the suggested options *Alabama*, *Alaska*, *Arizona*, *Arkansas*, *California*, and *Colorado* from a data list with the id `statelist`.
5. Create a submit button with the text *Send Donation*.
6. Provide the code to create a text input box for the `socSecNum` field and make the field required.
7. The `userAccount` field must follow the regular expression pattern `^\user\-d{4}$`. Provide the code for a text input box that can be validated for this field.
8. Provide a style rule to display all `textarea` elements with the background color `rgb(220, 220, 255)` when they have the focus.