# 10 Creating Pop-Up Windows, Adding Scrolling Messages, and Validating Forms


## Objectives

You will have mastered the material in this chapter when you can:

- Write a JavaScript user-defined function to display a scrolling message
- Write a JavaScript user-defined function to validate form data
- Write a JavaScript user-defined function to calculate loan payments
- Define if and if...else statements, conditionals, and operands
- Write a JavaScript user-defined function to format output in a text field
- Describe how to display a pop-up window

# 10 Creating Pop-Up Windows, Adding Scrolling Messages, and Validating Forms

## Introduction

In Chapter 9, you learned how to integrate JavaScript into an HTML document using variables and objects, and how to write JavaScript user-defined functions called by event handlers. This chapter reinforces these skills and shows you how to create a scrolling message that displays a text message in a form text field, using JavaScript to validate the data users enter into forms. The validation techniques discussed in this chapter use the if…else statement; parseInt(), parseFloat(), and isNaN() built-in functions; the Math object's pow() method; and the Number object's toFixed() method. Finally, the chapter discusses how to display a pop-up window.

## Project — Statewide Realty Mortgage Loan Calculator

Many bank and real estate Web sites include monthly payment calculators that help buyers determine what their monthly payments will be for a car, mortgage, or other type of loan. These calculators generally allow buyers to input basic loan information — total amount, interest rate, and number of years — that is used to determine the monthly payment.

Recently, Statewide Realty has decided to improve their Web site based on a recent customer-satisfaction survey. One of the most-requested items was a simple loan calculator that would allow customers to estimate their monthly mortgage payments. As one of the Web developers, you have been assigned to create this new Web page.

You decide to create an interactive form that allows customers to enter the mortgage amount, interest rate, and number of years for the loan. After entering the information, users click a Calculate button to display the monthly mortgage payment or the Reset button to clear the text boxes. You suggest adding a simple scrolling message box that urges customers to take advantage of current low mortgage rates. You also suggest adding a pop-up window to promote Statewide's online home finder service. Figure 10–1 shows the pop-up window, the scrolling message, and the user input form for the mortgage calculator.

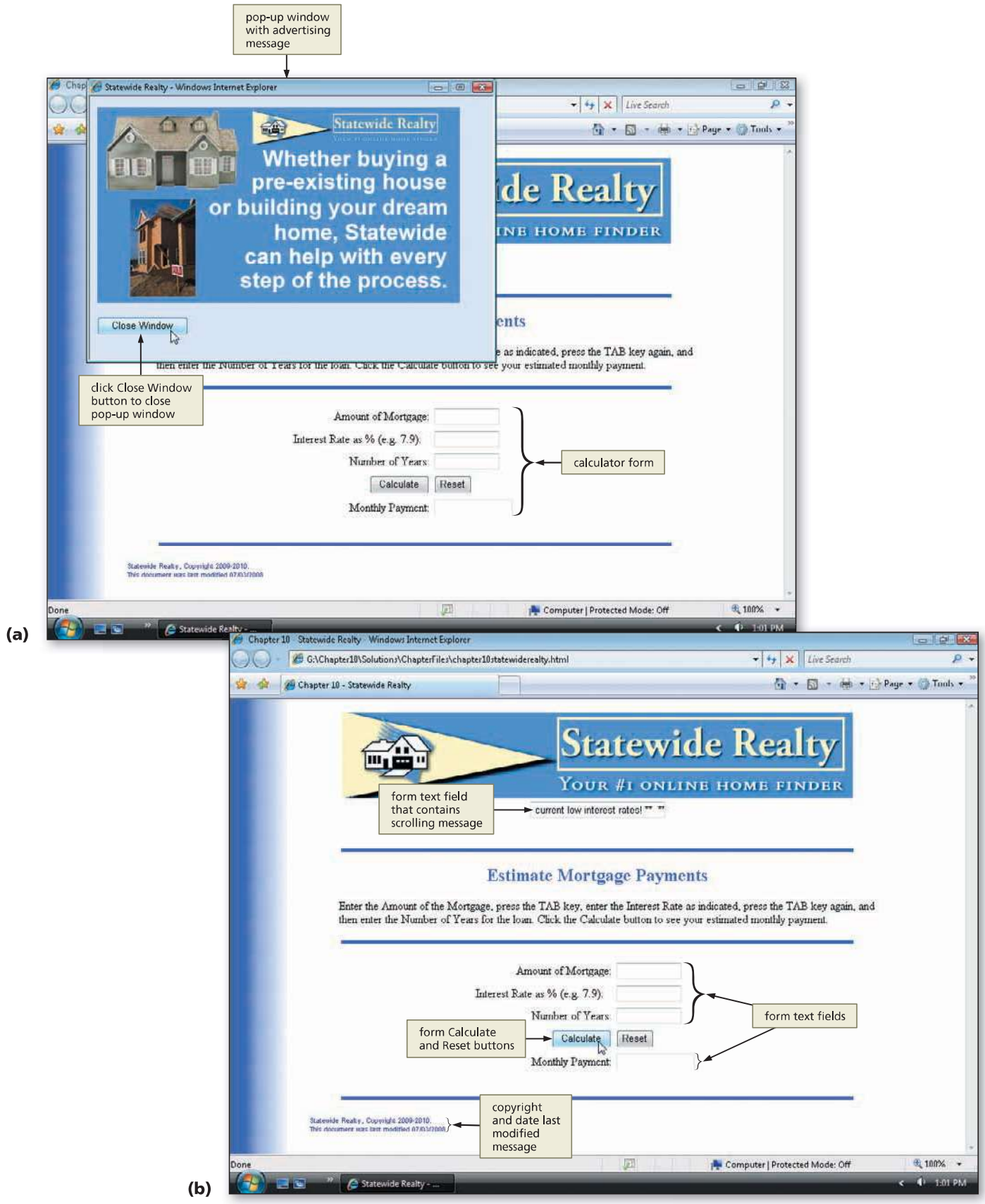
pop-up window with advertising message
Statewide Realty - Windows Internet Explorer
Statewide Realty
Whether buying a pre-existing house or building your dream home, Statewide can help with every step of the process.
Close Window
click Close Window button to close pop-up window
Amount of Mortgage:
Interest Rate as % (e.g. 7.9):
Number of Years:
Calculate
Reset
Monthly Payment:
calculator form
Statewide Realty, Copyright 2009-2010.
(a)
Chapter 10 - Statewide Realty - Windows Internet Explorer
G:\Chapter10\Solutions\ChapterFiles\chapter10statewiderealty.html
Statewide Realty
YOUR #1 ONLINE HOME FINDER
form text field that contains scrolling message
current low interest rates!
Estimate Mortgage Payments
Enter the Amount of the Mortgage, press the TAB key, enter the Interest Rate as indicated, press the TAB key again, and then enter the Number of Years for the loan. Click the Calculate button to see your estimated monthly payment.
Amount of Mortgage:
Interest Rate as % (e.g. 7.9):
Number of Years:
Calculate
Reset
Monthly Payment:
form text fields
form Calculate and Reset buttons
copyright and date last modified message
(b)

**Figure 10–1**

## Overview

As you read this chapter, you will learn how to write embedded JavaScript code to create the Web pages shown in Figures 10–1a and 10–1b by performing these general tasks:

- Open an existing HTML file and add JavaScript code.
- Create a scrolling message in a text field.
- Calculate a mortgage payment based on loan amount, interest rate, and number of years.
- Validate data entered into a form.
- Format the monthly payment to display as currency.
- Open a pop-up window when the Web page initially loads.
- Convert text to numeric values using built-in functions.
- Display the date the Web page was last modified.
- Save, validate, and test the Web pages.
- Print the HTML code and Web pages.

**Plan Ahead**

**General Project Guidelines**

When adding JavaScript or any scripting language to a Web page document, the actions you perform and decisions you make will affect the appearance and characteristics of the finished Web page. Before you write the JavaScript code, you should follow these general guidelines:

- **Determine what you want the code to accomplish.** For this chapter's project, you want to create a scrolling message in a text field, add a pop-up window, create a form for user input, validate the user input, perform a calculation based on the user input, output a result formatted as currency, and display the date the Web page was last modified.
- **Determine where in the Web page you want the code to appear.** All the JavaScript code in this chapter will be placed in the <head> section of the HTML code in user-defined functions. Event handlers will call these functions as needed.
- **Determine the overall Web page appearance.** When the Web page first loads, a pop-up window is displayed. The Web page also includes a text message that scrolls continuously. Data for the mortgage calculation is entered in a form, validated, and the results are displayed in currency format. The date the page was last modified displays at the bottom of the page.
- **Determine the data validation requirements.** Before the monthly payment can be calculated, the data entered in the form must be validated. The loan amount, interest rate, and the number of years for the loan must be numeric, not blank, and greater than zero. If the data does not meet these criteria, an alert message box notifies the user and positions the insertion point in the appropriate text field.
- **Determine the calculations needed.** You will need a formula for calculating the monthly payment. This formula is given later in the chapter.

When necessary, more specific details concerning the above guidelines are presented at appropriate points in the chapter. The chapter also will identify the actions performed and decisions made regarding these guidelines during the creation of the Web page shown in Figure 10–1 on the previous page.

# Inserting a Scrolling Message on a Web Page

A simple way to provide a Web site visitor with information is to add a scrolling text message to a Web page. Companies often use scrolling messages on their Web sites to highlight breaking news, key products, or special promotions. A scrolling text message can appear either in a text field within the Web page or on the status bar in the browser window. Because visitors to a Web page often do not look at the status bar, most Web developers agree that a scrolling message in a text field on the Web page is a better location.

A scrolling message has four basic components:

- The display object (a form text field)
- The text message to scroll in the text field
- The position of the next character in the text message
- A time delay

The **display object** identifies where the scrolling message is displayed, which, in this project, is in a form text field. The scrolling **message** is a text string assigned to a variable. The text string is what the user sees when the message is displayed. The **position** is the location of the next character in the text string. The **delay** regulates the speed in which the characters display in the text field.

BTW

**Placement of Scrolling Text**

Another reason to avoid placing scrolling text on the status bar is that it can be missed easily by the visitor.

The first step in creating the scrolling message for the Statewide Realty Web page is to create the display object (the text field). The text field is part of a simple form containing only the text field to display the scrolling message. The form and text field for the scrolling message are positioned below the title image. You begin by opening an existing HTML document, and adding the code to create a form and text field.

You must name the form and the form text field objects. These names serve as the object and properties used in the JavaScript code to assign the message string to the text field. The size attribute of the text field indicates the display width of the text field. Table 10–1 shows the HTML code to create the form and a text field for the scrolling message.

**Table 10–1 Code to Create a Form and a Text Field**

| Line | Code |
|---|---|
| 32 | `<form name="msgForm" action="">` |
| 33 | `<input type="text" name="scrollingMsg" size="25" />` |
| 34 | `</form>` |

Line 32 starts the form and uses the name attribute to give the form the unique name, msgForm. Line 33 indicates the input box is a text type, which means it can receive data. The text field is named scrollingMsg and is set to a size of 25. Line 34 is the closing <form> tag.

## To Open an Existing HTML File

As in Chapter 9, you will integrate JavaScript into an existing HTML document. The following step shows how to open the chapter10.html file included in the Data Files for Students.

1

- Start Notepad, and, if necessary, maximize the window. If Word Wrap is not enabled, click Format on the menu bar and then click Word Wrap to enable it.
- With a USB drive plugged in to your computer, click File on the menu bar and then click Open.

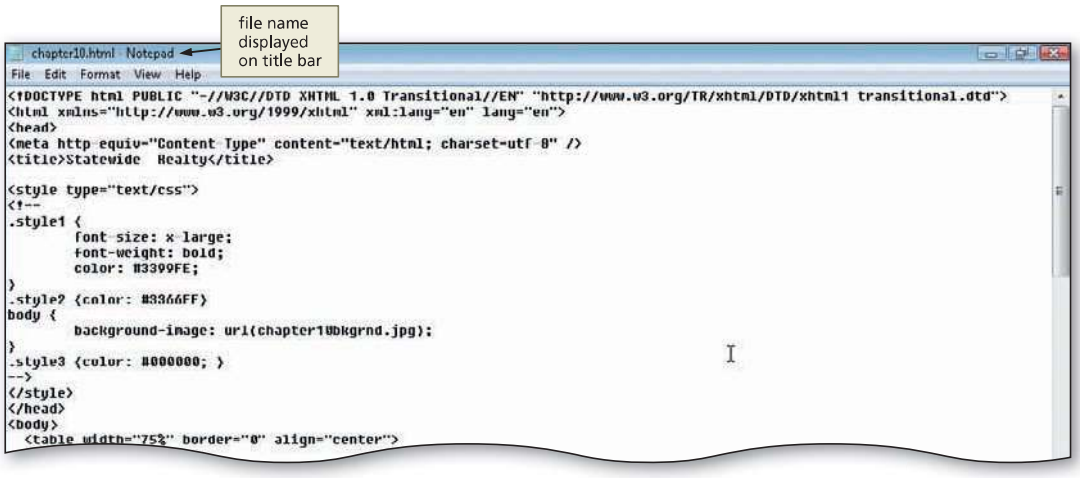


**Figure 10–2**

- If necessary, navigate to the Chapter10\ChapterFiles folder on the USB drive.
- If necessary, click the Look in box arrow, and then click All Files to display all files in the Chapter10\ChapterFiles folder.
- Click chapter10.html in the list of files.
- Click the Open button to open the chapter10.html file in Notepad (Figure 10–2).

## To Create a Form Text Field to Display a Scrolling Message

The following step illustrates how to create a form and a form text field to display a scrolling message.

1

- Click line 32 below the closing <p align="center"> tag.
- Enter the JavaScript code shown in Table 10–1 to enter the HTML code to create the form and text field (Figure 10–3).

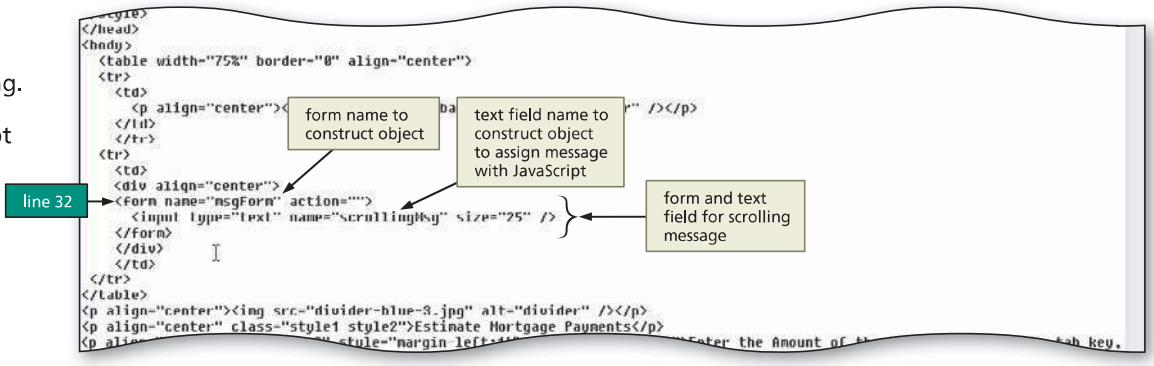


**Figure 10–3**

**Q&A** Can more than one scrolling message be placed on a Web page?

Generally not, especially with older browsers. Too many recursive calls can overflow the programming stack and crash the browser.

## Creating the scrollingMsg() User-defined Function

The scrollingMsg() function requires two variables and performs five tasks. The two variables are:

- The scrollMsg variable represents the message
- The beginPos variable represents the current character in the text message.

The five tasks the scrollingMsg() function performs are:

- Assigns the string message to the display object (which, in this project, is the text field)
- Increments the position variable by 1 to place the next character in the text message in the display object
- Uses an if statement to test for the end of the message
- If the text has scrolled to the end of the message, starts over with the first character
- Makes the display continuous and regulates the speed of the display using the setTimeout() method set to 200 milliseconds

Table 10–2 shows the code to begin the JavaScript section, declare and initialize the scrollMsg and beginPos variables, declare the scrollingMsg() function, and assign the first characters of the message to the text field. *Note: Because of the limitations of this textbook page, Lines 8 and 11 look like more than one line. However, each will be entered as a single line, pressing Enter only at the end of the entire numbered line.*

**Table 10–2 Code to Begin the scrollingMsg() Function**

| Line | Code |
|---|---|
| 6 | `<script type="text/javascript">` |
| 7 | `<!--Hide from old browsers` |
| 8 | `var scrollMsg = " ** Take advantage of the current low interest rates! ** "` |
| 9 | `var beginPos = 0` |
| 10 | `function scrollingMsg() {` |
| 11 | `document.msgForm.scrollingMsg.value = scrollMsg.substring (beginPos,scrollMsg.length)+scrollMsg.substring(0,beginPos)` |

Lines 6 and 7 start the <script> section of the Web page file. Line 8 declares the scrollMsg variable and assigns the message string, ** Take advantage of the current low interest rates! **, to it. The spaces at the beginning and end of the message string ensure that spaces appear at both ends of the message. Line 9 declares the beginPos variable, used to indicate the beginning position of the text string, and initializes it to zero. Line 10 declares the function scrollingMsg(). Line 11 assigns the message string to the text field using the object document.msgForm.scrollingMsg.value, which is derived from the form object and the input object. Figure 10–4 illustrates the relationship between these objects and how the statement is derived.

BTW

**The Marquee <marquee> Tag**

To make it easier to build scrolling messages, Microsoft developed the <marquee> tag. The direction attribute in the <marquee> tag controls scrolling up, down, left, or right. Internet Explorer recognizes the <marquee> tag, but other browsers do not. To create a scrolling message that works with Internet Explorer and other browsers, use a form text field, as discussed in this chapter.

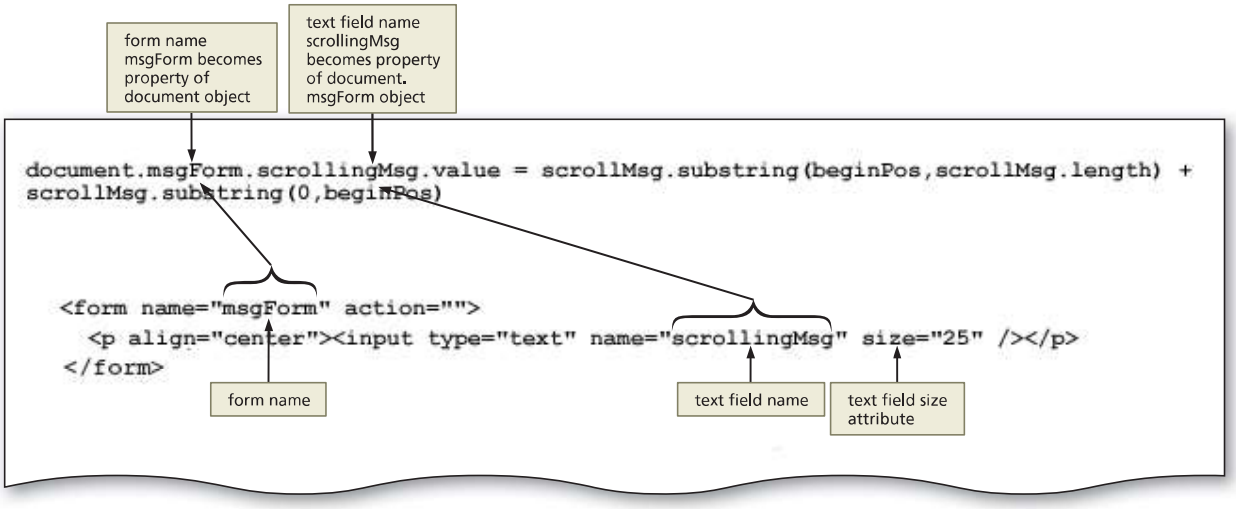


**Figure 10–4**

Line 11 also assigns the next character to the form text field object. The text field object is constructed using the form (msgForm) as an attribute of the document and the text field object (scrollingMsg) as an attribute of the msgForm object. The JavaScript code then assigns the string message to the input text field object (scrollingMsg) using the value attribute.

The rest of the assignment statement in line 11 uses the substring() method and concatenates the remainder of the scrollMsg variable to the beginning of the scrollMsg variable. As you learned in Chapter 9, the substring() method needs two parameters (x,y), where x is the starting point of the string and y is the location of the last character needed. This statement tells the scrollingMsg() function to assign the next character in the string message to the text field, to make the message appear as if it is scrolling.

## To Create the scrollingMsg() User-Defined Function

The following step shows how to create the scrollingMsg() user-defined function and define its variables.

1

- Click line 6, the blank line below the <title> tag.
- Enter the JavaScript code shown in Table 10–2 to enter the beginning script tags and define the variables used in the scrolling message, using the SPACEBAR to indent as shown, and then press the ENTER key (Figure 10–5).

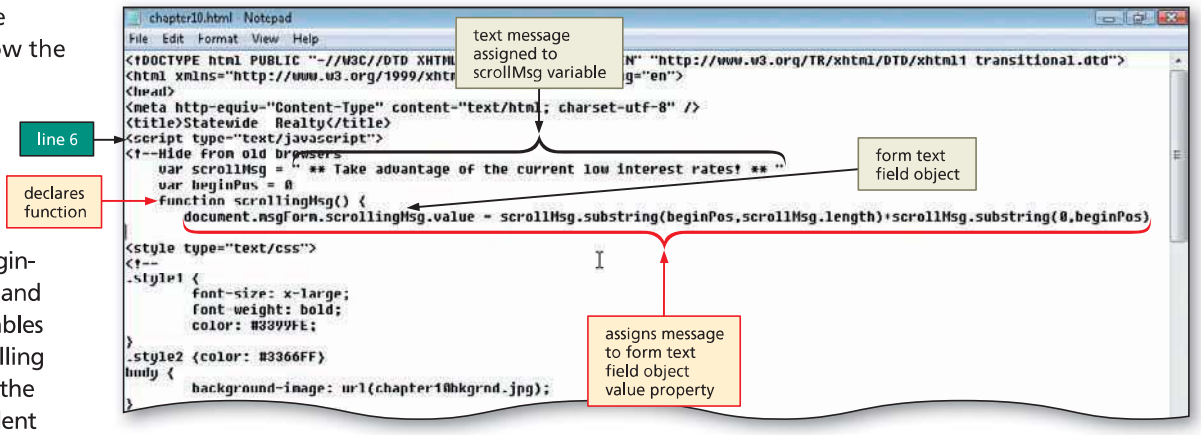


**Figure 10–5**

**Incrementing the Position Locator Variable** After declaring the scrollingMsg() function, the next step is to increment the beginPos variable and append the next character from the message string to the text field. To cause the message to scroll in the text field, the position locator variable (beginPos) must be incremented by one. Table 10–3 describes the various ways JavaScript statements can be used to increment variables.

**Table 10–3 Incrementing a Variable**

| Statement | Explanation |
|---|---|
| `variable=variable+1` | Executes the expression on the right side of the equal sign and assigns the result to a variable on the left side |
| `variable+=1` | Adds the number after the equal sign to a variable |
| `variable++` | Adds 1 to a variable, increments after the assignment |
| `++variable` | Adds 1 to a variable before the assignment |

Once incremented, the new value of the position locator variable, beginPos, allows the substring() method in line 11 to extract the next character in the message string and append it to the end of the message in the text field.

## To Enter the Code to Increment the Position Locator Variable

The following step illustrates how to enter the code to increment the position counter.

1

- Click line 12.
- Press SPACEBAR to indent under the previous line, then type `beginPos = beginPos + 1` to increment the position locator by one, and then press the ENTER key (Figure 10–6).

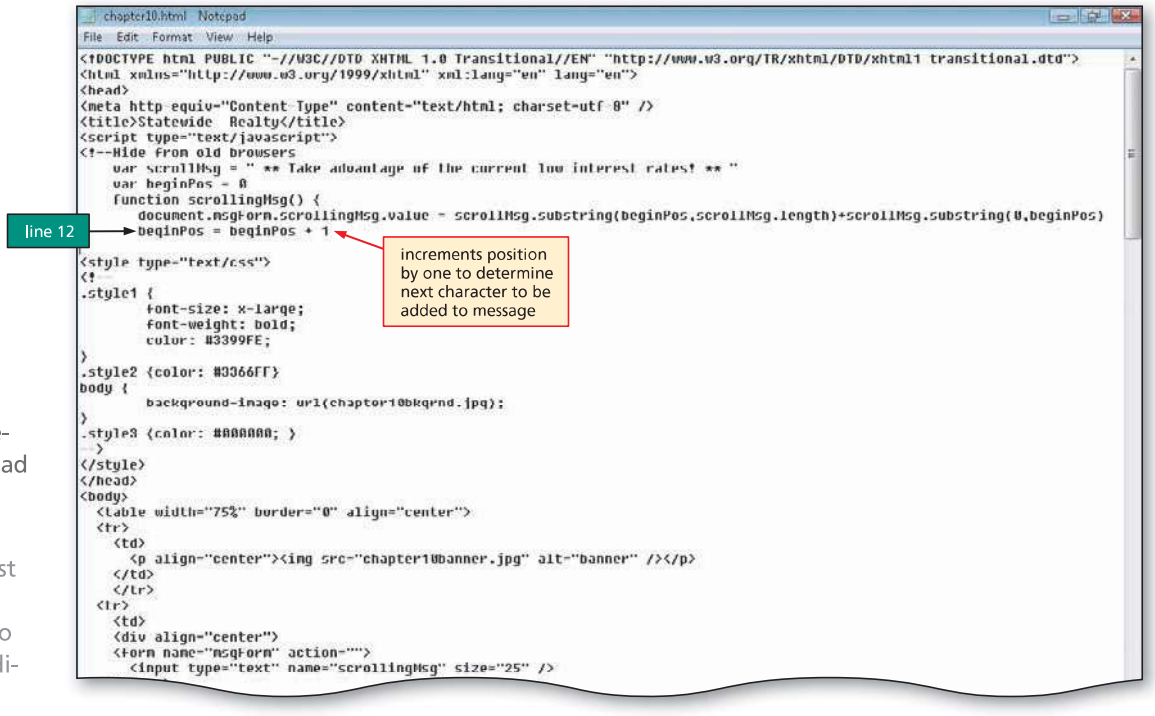


**Figure 10–6**

**Q&A** Why did we write the increment statement this way instead of using one of the other methods?

This way is the most common and easiest for beginners to understand. In addition, developers should use a format that can be recognized by anyone who might have to modify their code after the initial implementation.

**Entering an if Statement** After incrementing the position location variable (beginPos) by one, the JavaScript code must determine if the current value of beginPos exceeds the length of the message string. The loan payment calculator will use an if statement to determine if the current value of the beginPos variable is greater than the length of the message. An **if statement** is used to test a condition and then take one or more actions, based on the results of the test. The general form of the if statement is shown in Table 10–4. The if statement tests a **condition**, which is any comparison of values that evaluates to true or false. If the result of the comparison is true, the JavaScript code within the braces is executed. If the result of the comparison is false, the code after the closing brace is executed. Figure 10–7 shows the flowchart that corresponds to an if statement.

**Table 10–4 If Statement**

| | |
|---|---|
| General form: | `if (condition) {`<br>`  JavaScript statements if condition true`<br>`}` |
| Comment: | where condition is the comparison of values. All conditions must be placed in parentheses. If the result of the comparison is true, JavaScript executes the statements between the curly braces. If the result of the comparison is false, the JavaScript statements after the closing brace are executed. |
| Example: | `if (beginPos>scrollMsg.length) {`<br>`  beginPos=0`<br>`}` |

BTW

**The if Statement**
JavaScript if statements are an integral part of the programming language. They are used to define one or more statements that only should be executed based on the result of a conditional test, which controls the flow of logic.

As shown in the example in Table 10–4, the conditions use symbols called operators to indicate what type of comparisons should be made between the values. Table 10–5 shows the conditional operands used for comparisons. For more information about conditional operands, see the JavaScript Quick Reference in Appendix E.

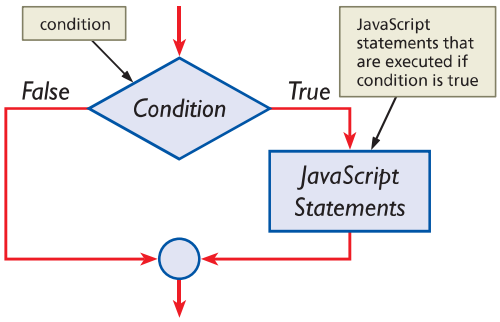


**Figure 10–7**

**Table 10–5 Conditional Operators**

| Operand | Example | Results |
|---|---|---|
| == | (a==b) | True if a equals b |
| === | (a=== b) | True if a equals b and the data are of the same type |
| != | (a!=b) | True if a does not equal b |
| !== | (a!==b) | True if a does not equal b and/or the data are not of the same type |
| > | (a>b) | True if a is greater than b |
| < | (a<b) | True if a is less than b |
| >= | (a>=b) | True if a is greater than or equal to b |
| <= | (a<=b) | True if a is less than or equal to b |
| && | (a==b) && (x<y) | True if both conditions are true (a equals b and x is less than y) |
| \|\| | (a!=b) \|\| (x>=a) | True if either condition is true (a does not equal b or x is greater than or equal to a) |

BTW

**Conditional Operators**
Conditional operators are symbols used to compare two values, called operands. The operator compares the two operands and returns a logical value based on whether the comparison is true.

To make the scrolling message work properly, an if statement is used to determine if the current value of beginPos is greater than the number of characters in the message string. The flowchart and sample code shown in Figure 10–8 illustrate how the if statement compares the beginning position variable (beginPos) with the overall length of the message (scrollMsg.length).

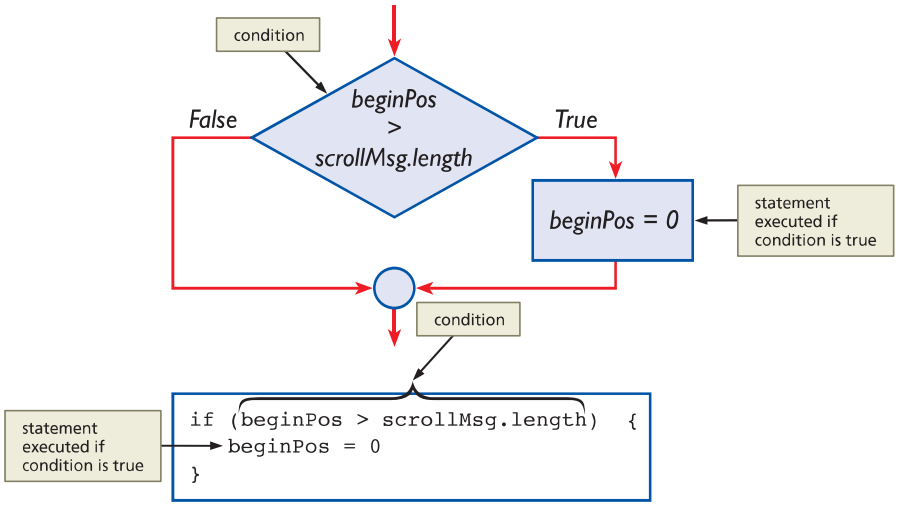


**Figure 10–8**

BTW

**Operands**
An operand is a numerical, string, logical, or object data type or value. Operands must be of the same data type, or you will not get a true comparison result.

If the current value of the beginPos variable exceeds the length of the scrollMsg variable, the statement assigns the value zero to the beginPos variable. By setting beginPos to zero, the code sets the string message so the first character of the string appears in the text field.

## To Enter an if Statement

The following step illustrates how to enter an if statement.

1

- Click line 13.
- Press the SPACEBAR to indent under the line above.
- Type `if (beginPos > scrollMsg.length) {` to enter the if statement and then press the ENTER key.
- Press the SPACEBAR to indent under the open parenthesis in (beginPos.
- Type `beginPos=0` to reset the variable to zero if the condition is true, and then press the ENTER key.
- Press the SPACEBAR to indent under and line up with the if statement.

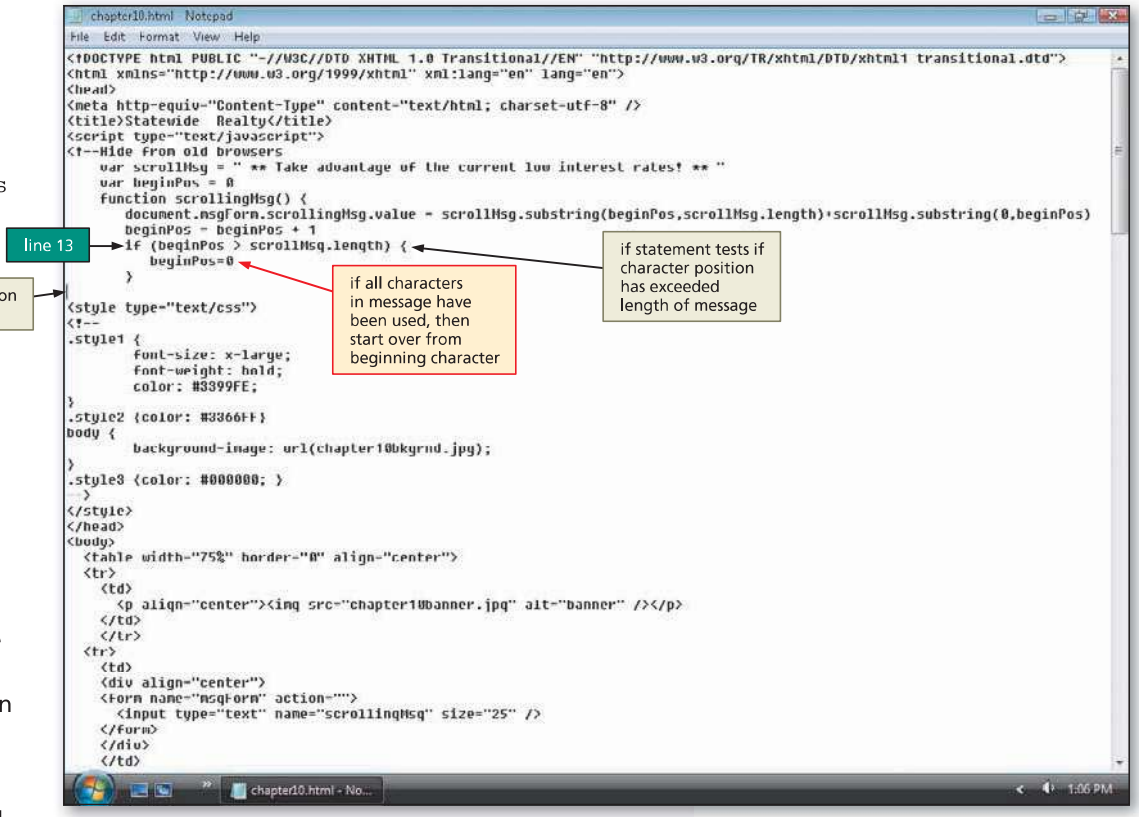


**Figure 10–9**

- Type `}` to close the if statement and then press the ENTER key (Figure 10–9).

Q&A

Do all JavaScript if statements have to be written in this format?

When only one statement follows the condition, like in this example, the statement could have been written as follows: `if (beginPos > scrollMsg.length) beginPos=0`. Note that the braces have been dropped for just one statement to be executed if the condition is true. If more than one statement needs to be executed, then the braces must be used to create the block of statements.

**Using the setTimeout() Method to Create a Recursive Call** To have the message text scroll continuously in the text field, you use a programming technique called **recursion**, in which a function is called within itself, creating an "endless loop." The setTimeout() method calls a function or evaluates an expression after a specified amount of time has elapsed, which is measured in milliseconds. The general form of the setTimeout() method is shown in Table 10–6.

BTW

**Recursion**
In this chapter's project, recursion is used to keep a routine going indefinitely or until some other function is called to stop it. Normally, recursive functions should have a mechanism that terminates the function when it completes its task.

**Table 10–6 setTimeout() Method**

| | |
|---|---|
| General form: | `setTimeout("instruction", time delay in milliseconds)` |
| Comment: | where instruction is any valid JavaScript statement and time delay is expressed in number of milliseconds |
| Example: | `window.setTimeout("scrollingMsg()",200)` |

In the scrollingMsg() user-defined function, the setTimeout() method continuously displays characters and regulates the speed of the characters displaying in the text field. The setTimeout() method calls the scrollingMsg() user-defined function from within the scrollingMsg() user-defined function. This recursive call to the scrollingMsg() function is what makes the message scroll in the text field continually.

## To Add the setTimeout() Method to Create a Recursive Call

The following step illustrates how to add the setTimeout() method to create a recursive call to the scrollingMsg() function.

1

- If necessary, click line 16.
- Press the SPACEBAR to indent under the closing brace, then type `window.setTimeout("scrollingMsg()",200)` to call scrollingMsg() from within itself and then press the ENTER key.
- Press the SPACEBAR to indent and then type `}` to close the function and then press the ENTER key two times (Figure 10–10).

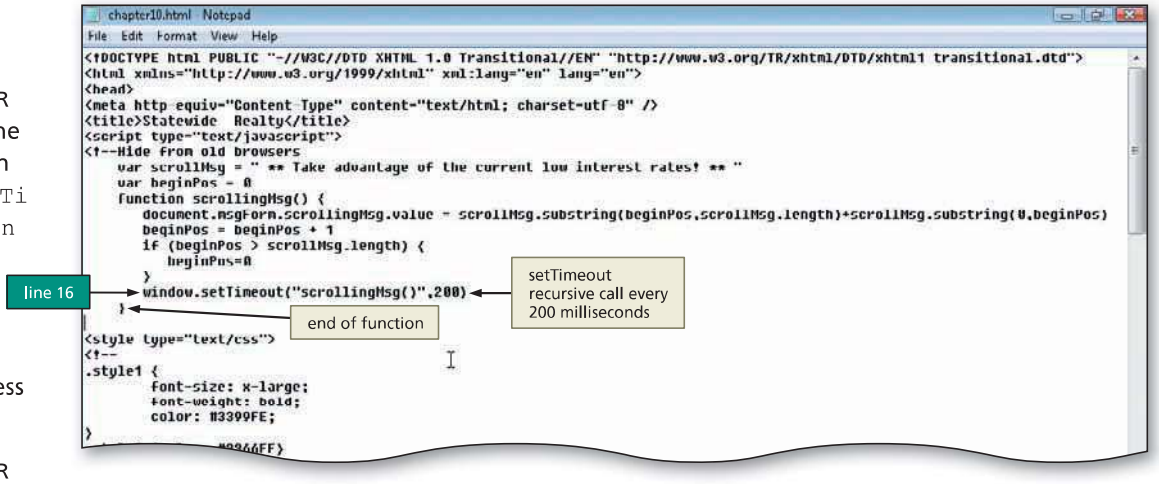


**Figure 10–10**

Q&A How do we know how fast to make the scrolling?

The best way is to try several different speeds and ask potential users to look at it and indicate their preference.

Q&A What if we changed the number from 200 to 2000?

The text would display one character every two seconds and would be so boring to watch, you would lose the interest of your user.

## To Complete a JavaScript Section

The following step shows how to enter the JavaScript code to complete the <script> section.

1

- If necessary, click line 19.
- Type `//-->` to close the comment to hide the JavaScript and then press the ENTER key.
- Type `</script>` to close the <script> section and then do not press the ENTER key (Figure 10–11).

Q&A

Can I use one hyphen and the greater than sign to end the comment?

No, it must be two hyphens and the greater than sign to close the comment started above.

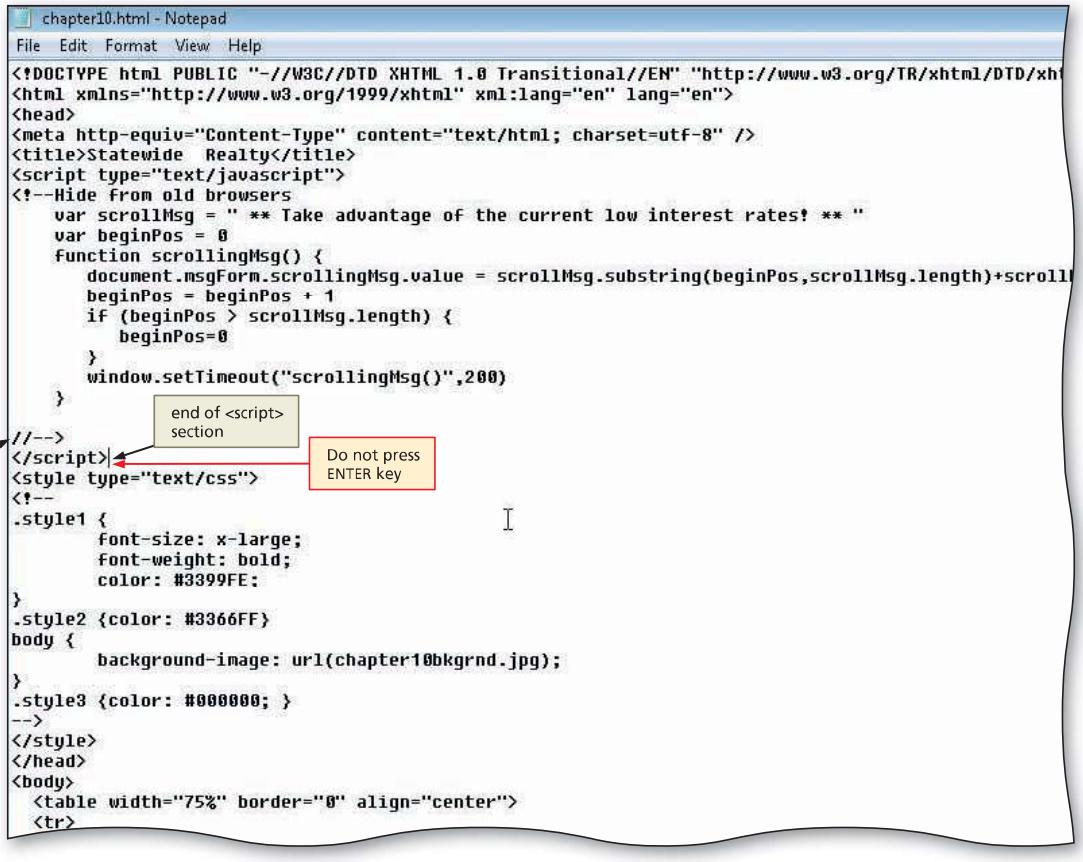


**Figure 10–11**

## Adding an Onload Event Handler

BTW

**Event Handlers**
Some older browsers have a problem recognizing mixed case event handlers. Although the newer versions of Web browsers have fixed the problem, they still recognize event handlers using all lowercase characters.

The last step in adding a scrolling message to a Web page is to add an event handler to start the scrolling message when the Web page loads. As discussed in Chapter 9, an event is an action, such as a mouse click or a window loading. An event handler is a way to associate that action with a function. The event handler to start the scrolling message is the onload event handler.

The JavaScript standard uses both upper- and lowercase in spelling event handlers, as shown in Table 10–7. In this text, however, to be XHTML-compliant and to pass XML validation, developers spell the event handlers in all lowercase characters because XHTML treats event handlers as tag attributes. The XHTML standard requires all tag attributes to be lowercase.

Table 10–7 shows some of the event handlers and the associated objects. As the table indicates, event handlers can be used only with certain objects. For example, the onclick event handler is used to trigger JavaScript code when a user clicks a button or link, while the onload event handler is used to trigger JavaScript code when a document is loaded into the browser window. For more information about event handlers, see the JavaScript Quick Reference in Appendix E.

**Table 10–7 Objects and Associated Event Handlers**

| Object | Event Handler |
|---|---|
| button | onClick, onDblClick |
| document | onLoad, onUnload |
| form | onSubmit, onReset, onBlur, onKeydown, onKeypress, onKeyup |
| hyperlink | onClick, onMouseover, onMouseout, onDblClick, onMousemove, onMousedown |
| image | onLoad, onAbort, onError, onMousemove, onMousedown |
| input box | onBlur, onChange, onFocus, onKeypress, onKeyup, onKeydown |
| Submit button | onClick |
| window | onLoad, onUnload, onBlur, onFocus |

In this chapter, the onload event handler calls the scrollingMsg() function, using the following statement:

```
onload="scrollingMsg()"
```

where onload is the event handler and the scrollingMsg() function is the code that is executed as the result of the event. The statement is entered in the <body> tag to indicate that the onload event handler should call the scrollingMsg() function when the Web page loads.

## To Enter the onload Event Handler to Call the scrollingMsg() Function

The following step illustrates how to enter the onload event handler to call the scrollingMsg() function.

1

- Click to the right of the y in the body in line 36.
- Press the SPACEBAR once.
- Type `onload="scrollingMsg()"` to add the event handler and do not press the ENTER key (Figure 10–12).

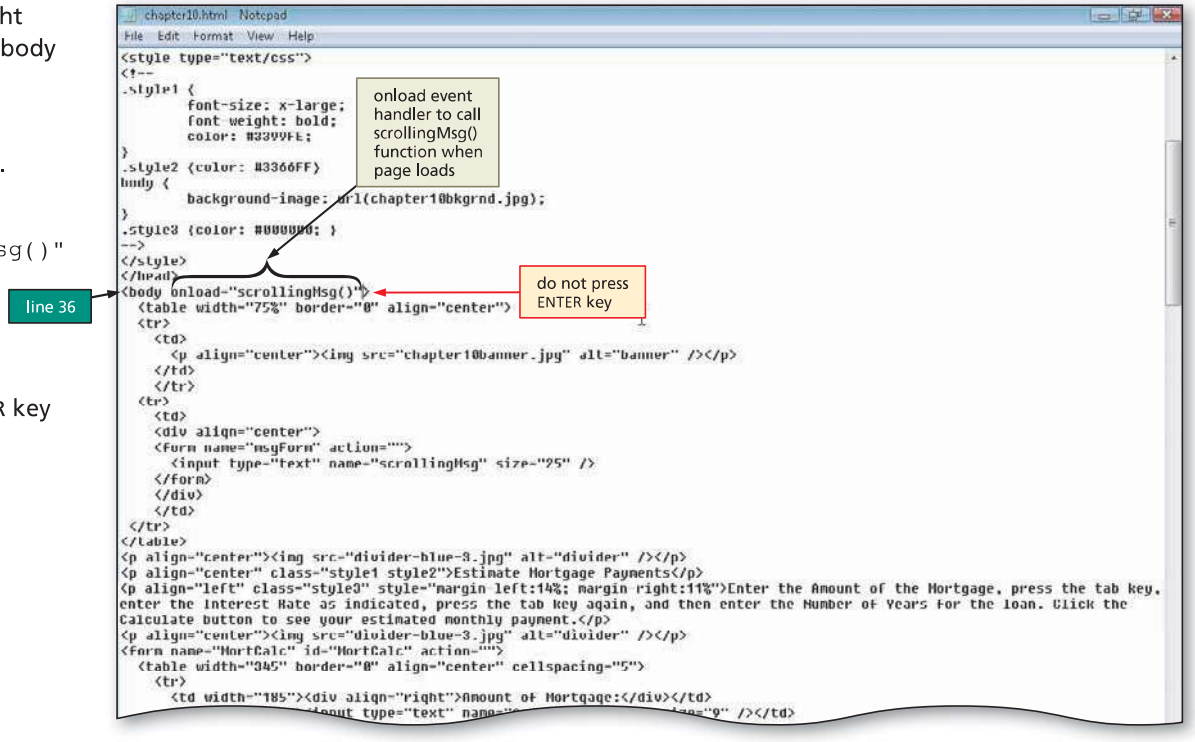


**Figure 10–12**

## To Save an HTML File and Test a Web Page

With the code for the scrollingMsg() function complete and the onload event handler added to call the function when the Web page loads, you should save the HTML file and test the Web page. The following step illustrates how to save the HTML file and then test the Web page.

**1**

- With a USB drive plugged into your computer, click File on the Notepad menu bar and then click Save As. Type `chapter10 statewiderealty.html` in the File name text box (do not press the ENTER key).
- If necessary, browse to the USB drive and open the Chapter10\ChapterFiles folder.
- Click the Save button in the Save As dialog box to save the file on the USB drive with the name chapter10 statewiderealty.html.
- Start your browser. If necessary, click the Maximize button.
- Type `g:\chapter10\ChapterFiles\chapter10 statewiderealty.html` in the Address box and then press the ENTER key.
- If necessary, click the security bar under the tabs, click Allow Blocked Content... (Figure 10–13a), and if necessary click Yes in the Security Warning dialog box to display the scrolling message (Figure 10–13b).

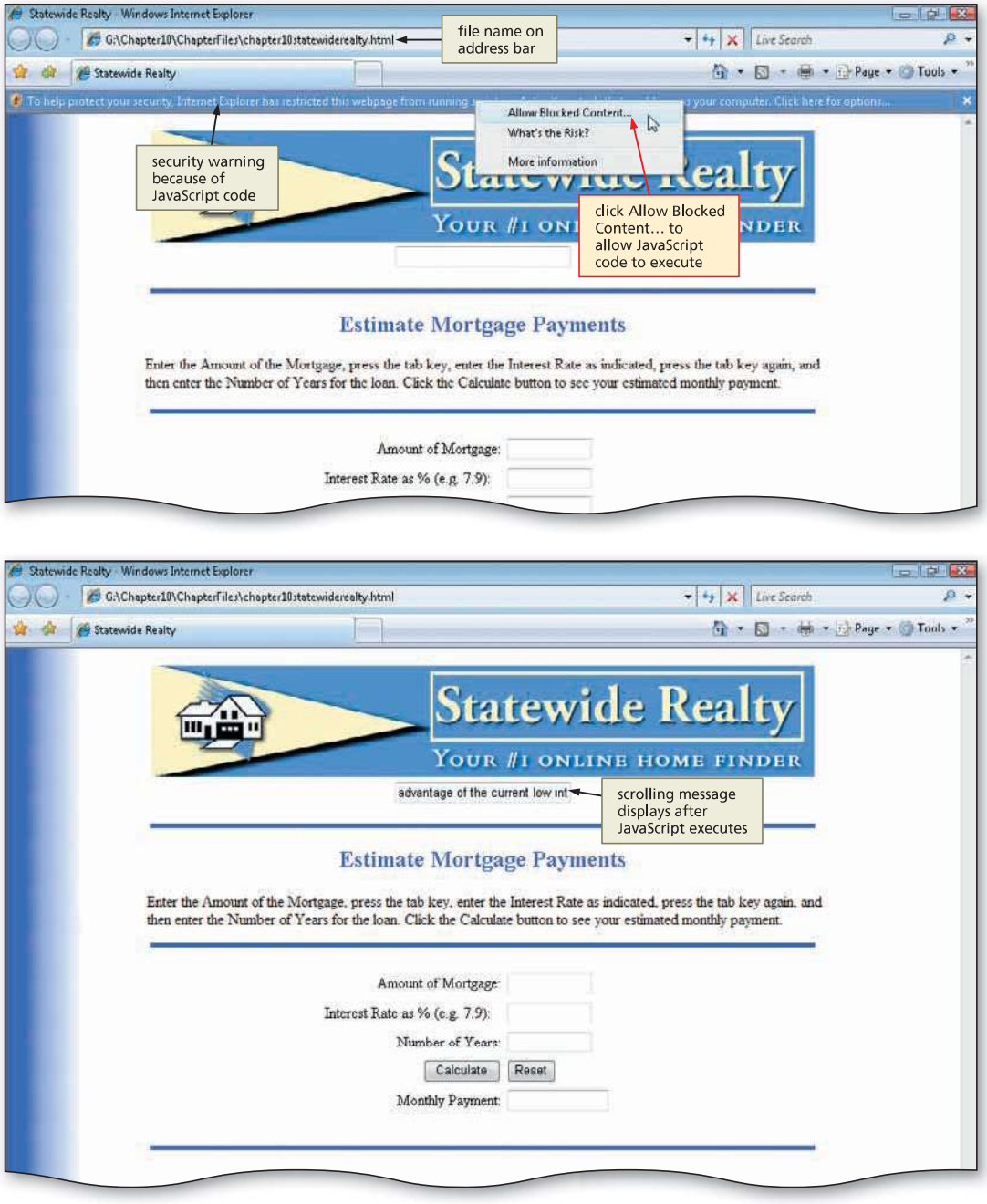


**Figure 10–13**

**Q&A** What if I do not see a security bar?

That simply means that tight restrictions and security are not set on your browser.

**Validating a form.**
In order to calculate the monthly payment, the values entered into the text fields must be valid numbers. A user-defined function called Calc() follows these steps to validate the text field entries:

- Convert the text field value to a numeric value using the parseInt() or parseFloat() function
- Test the value to be numeric using the isNaN (is Not a Number) function and checking that the value is greater than zero
- If the value is not a number or is zero or less, display a message, clear the text field, and position the insertion point in that text field

To call the Calc() validation function, an onClick event handler is added to the form.

**Plan Ahead**

# Adding a Loan Payment Calculator

The mortgage loan payment calculator form shown in Figure 10–14 requests user input. The form, which is named MortCalc, already has been created in the HTML file. JavaScript code must be added to validate the input, calculate the monthly payment, and display the results in the MortCalc form. In order for the calculator to work, each text field must have a valid data entry. You will write a user-defined function called Calc() to perform these three tasks.

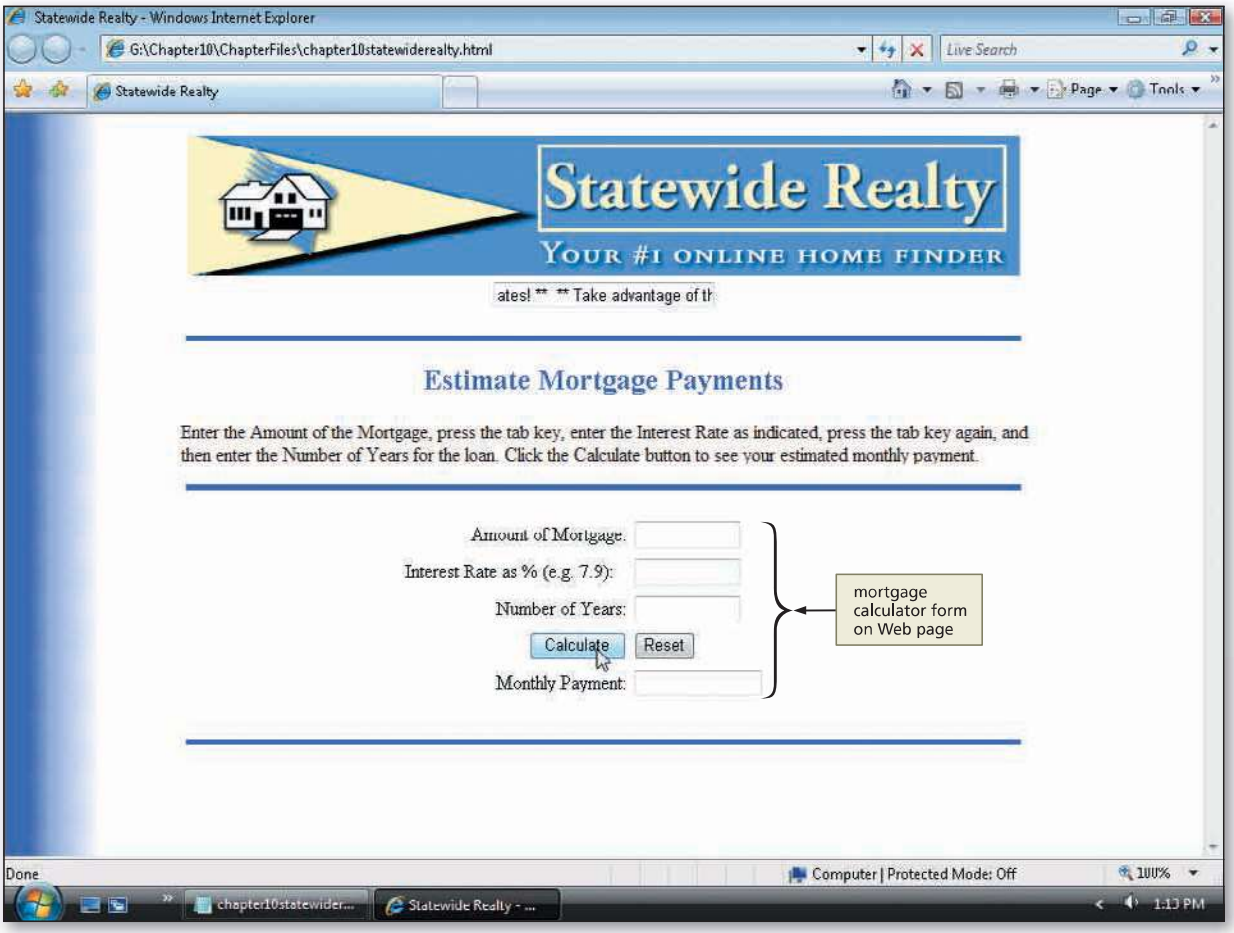


**Figure 10–14**

BTW

**Validating Web Form Data**

Web developers use multiple techniques to validate Web forms using JavaScript. Some developers choose to validate each item as it is entered by using a combination of onchange event handlers and user-defined functions. It is important to validate Web forms because some Web databases are sensitive to invalid data and may crash, or mathematical formulas may cease to function when using invalid data.

## Validating Forms Using Nested if…else Statements

You can use different techniques to validate forms. This chapter uses a series of nested if…else statements, which is like the if statement except that it specifies statements to execute if the condition is false, as shown in the flowchart in Figure 10–15. Much like the if statement, an if…else statement tests a condition. If the condition is true, the statements between the curly braces after the if statement execute. If the condition is false, the statements between the braces after the else statement execute.

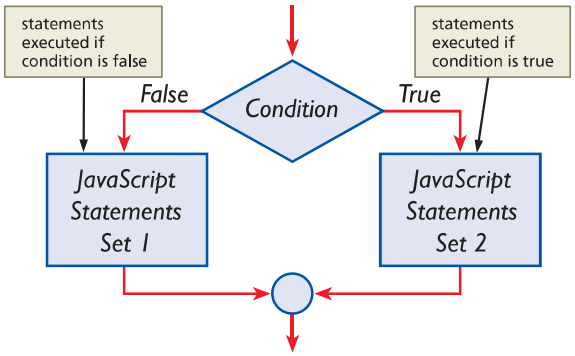


**Figure 10–15**

The validation algorithm begins by converting the text field value to a number. The if…else statement tests if the value entered in a text field is invalid (a true condition). If true, an error message is displayed, the text field is cleared, and the insertion point is placed back in the text field. This occurs until the user enters valid data in the text field. If the value entered in the text field is valid (a false condition), the next text field is examined until all text fields are validated. The validation process is shown in the flowchart in Figure 10–16.

BTW

**Domains**

A domain is a range of acceptable values for a field or column in a database. Using an HTML select list can ensure that accurate values are entered.

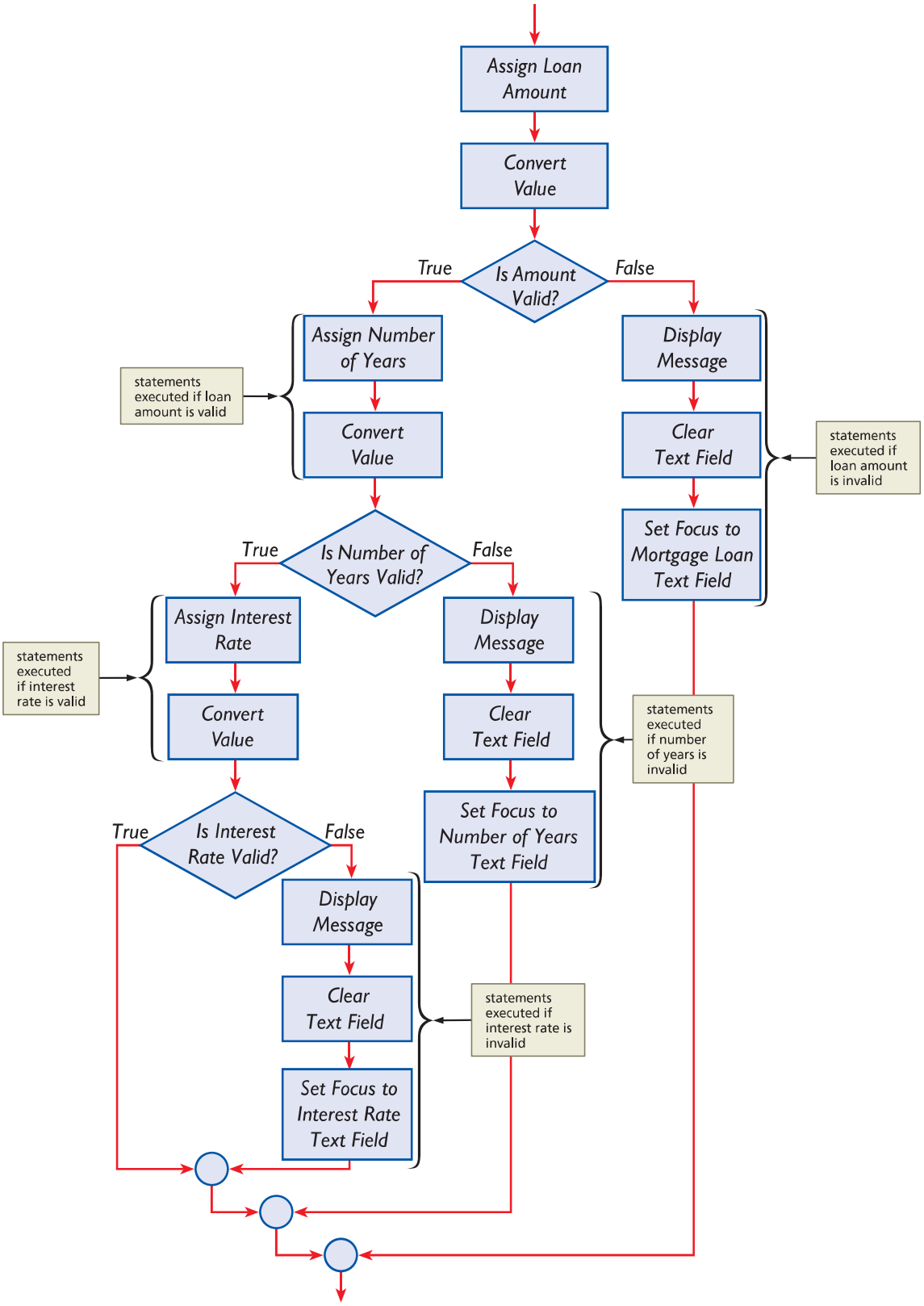
Assign Loan Amount
Convert Value
Is Amount Valid?
True
False
Assign Number of Years
Convert Value
statements executed if loan amount is valid
Display Message
Clear Text Field
Set Focus to Mortgage Loan Text Field
statements executed if loan amount is invalid
Is Number of Years Valid?
True
False
Assign Interest Rate
Convert Value
statements executed if interest rate is valid
Display Message
Clear Text Field
Set Focus to Number of Years Text Field
statements executed if number of years is invalid
Is Interest Rate Valid?
True
False
Display Message
Clear Text Field
Set Focus to Interest Rate Text Field
statements executed if interest rate is invalid

**Figure 10–16**

This validation design is necessary because of the event-driven nature of JavaScript. When a user triggers an event that calls a function, processing stays within that function until all statements execute. Because all the statements execute in a function, the form validation routine uses nested if…else statements to ensure each text field is validated correctly. By nesting if…else statements, you can place an if…else statement inside another as shown in Figure 10–17.

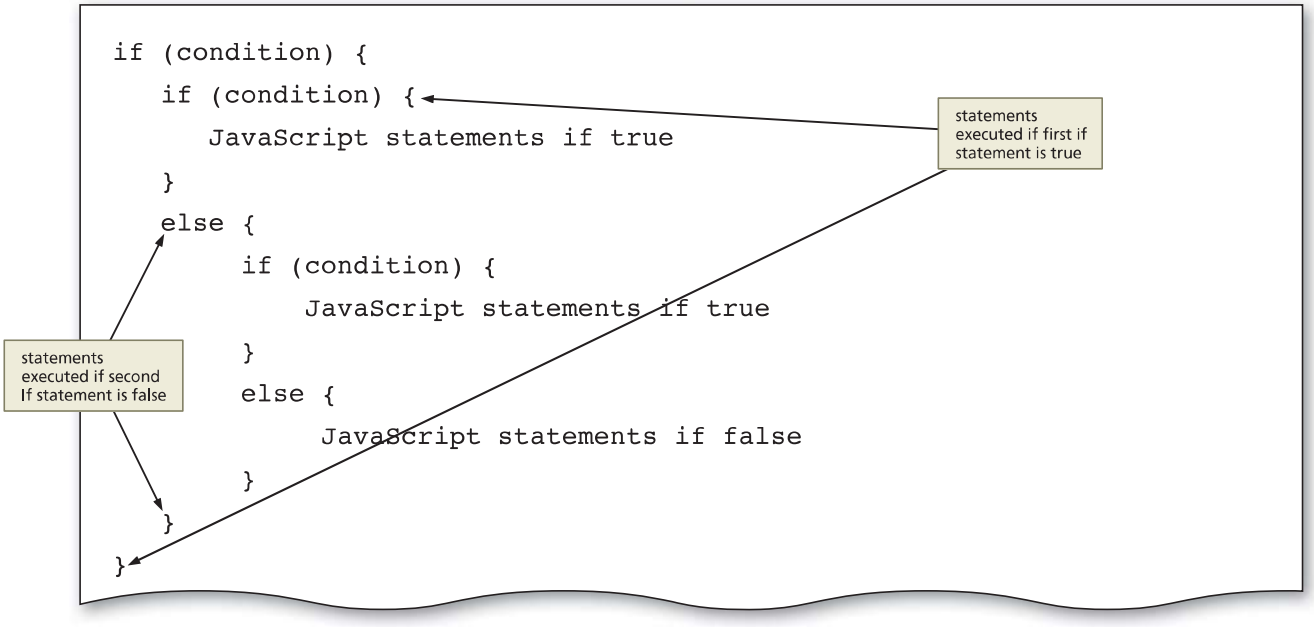


**Figure 10–17**

BTW

**Radix or Number Base**
Radix is the number base to which the integer value should be converted. The use of the numeral 2 represents binary, 8 represents octal, and 16 represents hexadecimal.

**Using Built-In Functions to Validate Data** When validating data, the JavaScript code may have to evaluate several criteria — for example, to ensure that a text field is not blank or that it contains numeric data (not text or characters). JavaScript accepts data entered into a text field as text character data, which means that the values must be converted to a number before they can be tested or validated. Table 10–8 describes the two built-in functions (parseInt() and parseFloat()) used to convert values and one function (isNaN()) used to test if the converted value is a number.

BTW

**The parseFloat() Function**
The parseFloat() built-in function parses a string argument and converts the value into a decimal floating-point number. If the first character cannot be converted to a number, the result is NaN, which means "not a number."

**Table 10–8 Built-In Functions: parseInt(), parseFloat(), isNaN()**

| | |
|---|---|
| General form: | `variable = parseInt(value, base)` |
| Comment: | converts to an integer. Value is any string, which can be a variable or literal; base is the number base to which you want the string converted. A base of 2 means binary base number, an 8 means octal, and a 10 means decimal. The function returns an integer value, stripping the value after the decimal point. |
| Example: | `parseInt(loanAmount,10)` |
| General form: | `variable = parseFloat(value)` |
| Comment: | converts to a floating point number. Value is any string, which can be a variable or literal, representing a floating-point number. A floating-point number is one with a fractional or decimal value (including percentages). The function returns the value as a floating-point number. |
| Example: | `parseFloat(loanAmount)` |
| General form: | `isNaN(value)` |
| Comment: | isNaN means is Not a Number. Value is any value, which can be a variable or literal. The function returns a Boolean condition of true or false. |
| Example: | `isNaN(loanAmount)` |

Figure 10–18 shows how the values of the form are passed to the Calc() user-defined function. Table 10–9 shows the general form of the JavaScript statement used to assign a null, or other, value to a text field object within a form.

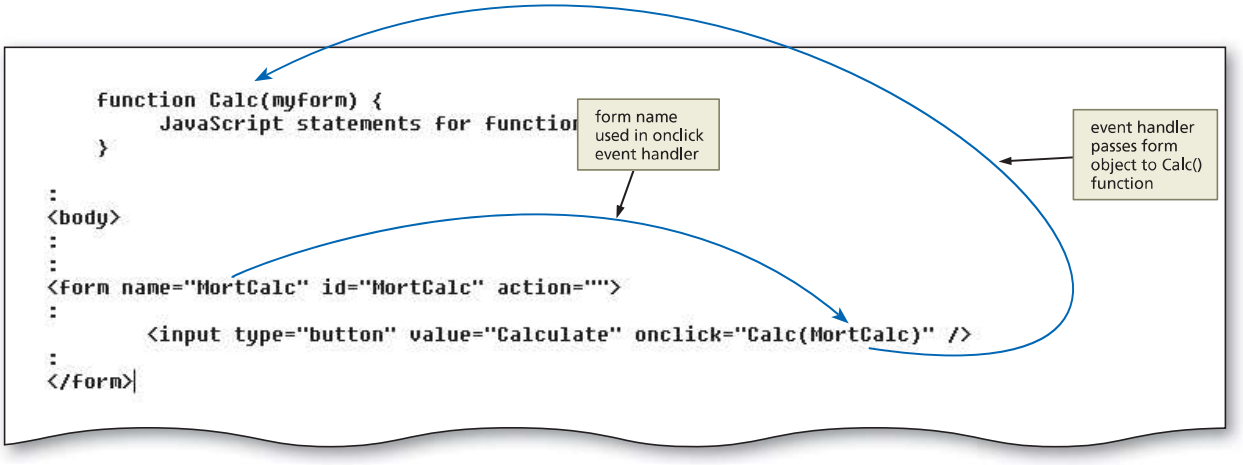


**Figure 10–18**

**Table 10–9 Assignment Statement**

| | |
|---|---|
| General form: | `document.formname.textfieldname.value=variable_or_literal` |
| Comment: | where formname is the name of the form; textfieldname is the name of a text field in the form; value is the attribute; and the variable_or_literal is the value assigned to the text field. |
| Examples: | `document.MortCalc.Amount.value=LoanAmt`<br>`document.MortCalc.Amount.value="12500"`<br>`document.MortCalc.Amount.value=""` |

To place the insertion point back in a specific text field in the form, the focus must be set for that text field. Setting the focus means giving attention to an object. JavaScript uses the **focus() method** to give attention to an object. When the focus is set to an object, such as the Amount text field, the JavaScript statement automatically positions the insertion point in the text field. Table 10–10 shows the general form of the focus() method.

**Table 10–10 focus() Method**

| | |
|---|---|
| General form: | `document.formname.objectname.focus()` |
| Comment: | where formname is the name of the form that contains the object; and objectname identifies the object to which focus should be set. |
| Examples: | `document.MortCalc.Amount.focus()` |

Table 10–11 shows the code to enter the Calc() user-defined function and the statements necessary to validate the mortgage loan amount using the parseInt() and isNaN() functions.

**BTW**

**The isNaN() Built-in Function**

The isNaN() function to test whether a value is not a number is the only function that tests a numeric value as the argument. The test uses the NOT operator and returns a Boolean value of true or false.

**Table 10–11 Code for Calc() Function to Validate the Loan Amount**

| Line | Code |
|---|---|
| 19 | function Calc(myForm) { |
| 20 | var mortAmount=document.MortCalc.Amount.value |
| 21 | var mortAmount=parseInt(mortAmount,10) |
| 22 | if (isNaN(mortAmount) \|\| (mortAmount<=0)) { |
| 23 | alert("The loan amount is not a valid number!") |
| 24 | document.MortCalc.Amount.value=" " |
| 25 | document.MortCalc.Amount.focus() |
| 26 | } |

Line 19 declares the Calc() function and passes any values entered or selected in the MortCalc form to the function. Line 20 can assign the data entered in the Amount of Mortgage text field to the mortAmount variable. Line 21 converts that value to an integer. The if statement beginning on line 22 checks the condition to see if the value for the mortAmount variable is not a number or if the value entered is less than or equal to zero. If the result of the condition is true (that is, the value entered is not a number or it is a negative number), the function notifies the user with an alert message (line 23) that the loan amount is not valid; clears the data entered in the Amount of Mortgage text field (line 24); and then sets the focus back to the Amount of Mortgage text field (line 25). The brace in line 26 closes the if statement.

## To Start the Calc() Function and Nested if…else Statements to Validate Form Data

The following step illustrates how to enter the Calc() user-defined function and the if statement that validates the loan amount value entered in the Amount of Mortgage text field.

1

- If necessary, click the chapter10 statewiderealty.html - Notepad button on the taskbar to display the Notepad window.
- Click line 19.
- Press the ENTER key once to create a blank line, and then position the insertion point on the blank line (line 19).

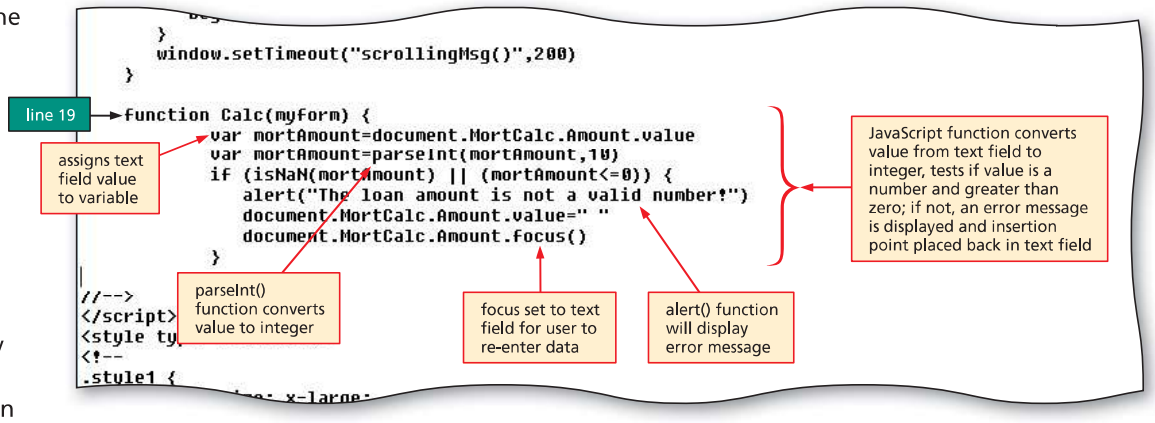


**Figure 10–19**

- Enter the JavaScript code shown in Table 10–11 using the SPACEBAR to indent as shown to start the Calc() user-defined function, define the variables for the loan amount, convert the loan amount to numeric values, and validate that the values are numeric (Figure 10–19).

## Completing the Validation and Adding the Event Handler

Because an event must be executed until completion, the Calc() function validates all the entered values. Table 10–12 shows the code to validate the Interest Rate as % text field using the parseFloat() and isNaN() functions. If the interest rate data is valid, the function proceeds to validate the value in the Number of Years field and convert it to a floating-point number.

**Table 10–12 Code to Validate the Interest Rate**

| Line | Code |
|---|---|
| 27 | `else {` |
| 28 | `var mortRate=document.MortCalc.Rate.value` |
| 29 | `var mortRate=parseFloat(mortRate)` |
| 30 | `if (isNaN(mortRate) \|\| (mortRate<=0)) {` |
| 31 | `alert("The interest rate is not a valid number!")` |
| 32 | `document.MortCalc.Rate.value=" "` |
| 33 | `document.MortCalc.Rate.focus()` |
| 34 | `}` |

Line 27 is an else statement that executes if the mortAmount data is valid and the function should proceed to validate the data entered in the Interest Rate as % text field. Line 29 passes the value in the Interest Rate as % text field to the mortRate variable, and converts it to a floating-point number using the parseFloat() function. Because the interest rate is a floating-point number, you must use the parseFloat() function to keep the interest rate a floating-point number.

The if statement in line 30 tests the mortRate variable to determine if it is a number or if the value is less than or equal to zero. If the result of the condition is true, an alert message (line 31) notifies the user that the interest rate is not valid. Line 32 clears the data entered in the Interest Rate as % text field, and then line 33 sets the focus back to the Interest Rate as % text field. The brace in line 34 closes the if statement. If the mortRate data is valid, the function then proceeds to validate the Number of Years.

Table 10–13 shows the code used to validate the value entered in the Number of Years text field.

**Table 10–13 Code to Convert and Validate the Years Entered Value**

| Line | Code |
|---|---|
| 35 | `else {` |
| 36 | `var mortYears=document.MortCalc.Years.value` |
| 37 | `var mortYears=parseInt(mortYears,10)` |
| 38 | `if (isNaN(mortYears) \|\| (mortYears<=0)) {` |
| 39 | `alert("The number of years is not a valid number!")` |
| 40 | `document.MortCalc.Years.value=" "` |
| 41 | `document.MortCalc.Years.focus()` |
| 42 | `}` |
| 43 | `}` |
| 44 | `}` |
| 45 | `}` |

Line 35 is an else statement that executes the statements if the if condition on line 30 is false. Line 37 converts years to an integer, using the parseInt() function. The if statement beginning in line 38 checks the condition to determine if the mortYears value is greater than zero. If the number of years is not valid, line 39 displays a message and line 40 places the focus back in the Years text field. The braces in lines 42 through 45 close the nested if…else statements and the function.

## To End the Nested if…else Statements to Validate Form Data

The following step shows how to enter the else portions of the nested if…else statements in the Calc() function to validate the Interest Rate as % text field and the Number of Years.

1

- If necessary, click line 27.
- Enter the JavaScript code shown in Table 10–12 to validate the interest rate, using the SPACEBAR to indent the code as shown in Figure 10–20.
- Press the ENTER key.
- Continue on line 35.
- Enter the JavaScript code shown in Table 10–13 to validate the number of years for the loan, using the SPACEBAR to align the code as shown.
- Press the ENTER key to finish the else portions of the nested if...else statements (Figure 10–20).

```
chapter10statewiderealty.html - Notepad
File  Edit  Format  View  Help
<script type="text/javascript">
<!--Hide from old browsers
    var scrollMsg = " ** Take advantage of the current low interest rates! ** "
    var beginPos = 0
    function scrollingMsg() {
       document.msgForm.scrollingMsg.value = scrollMsg.substring(beginPos,scrollMsg.length)+scrol
       beginPos = beginPos + 1
       if (beginPos > scrollMsg.length) {
          beginPos=0
       }
       window.setTimeout("scrollingMsg()",200)
    }

    function Calc(myform) {
            var mortAmount=document.MortCalc.Amount.value
            var mortAmount=parseInt(mortAmount,10)
            if (isNaN(mortAmount) || (mortAmount<=0)) {
               alert("The loan amount is not a valid number!")
               document.MortCalc.Amount.value=" "
               document.MortCalc.Amount.focus()
            }
 line 27 ->else {
               var mortRate=document.MortCalc.Rate.value
               var mortRate=parseFloat(mortRate)
               if (isNaN(mortRate) || (mortRate<=0)) {
                  alert("The interest rate is not a valid number!")
                  document.MortCalc.Rate.value=" "
                  document.MortCalc.Rate.focus()
               }
      line 35 ->else {
                     var mortYears=document.MortCalc.Years.value
                     var mortYears=parseInt(mortYears,10)
                     if (isNaN(mortYears) || (mortYears<=0)) {
                        alert("The number of years is not a valid number!")
                        document.MortCalc.Years.value=" "
                        document.MortCalc.Years.focus()
                     }
               }
            }
    }

//-->
</script>
<style type="text/css">
<!--
```

code to validate that interest rate is a number and greater than zero

code to validate that number of years is a number and greater than zero

braces close if and else statements

**Figure 10–20**

**Q&A** Why is the year not converted to a floating point number?

Most loans, especially mortgage loans, are NOT made on part of a year, so the number of years should be an integer.

## To Enter an onclick Event Handler to Call the Calc() Function

The last step in adding form validation to the mortgage loan payment calculator is to add an event handler to trigger the Calc() function when the user clicks the Calculate button. After entering data in the form, a user clicks the Calculate button, which triggers the Calc() function to validate the data entered in the form, using the if…else statements and built-in functions entered in previous steps. The following step shows how to enter the onclick event handler to call the Calc() function.

**1**

- Scroll down to the HTML code for the form and then click line 101, right after the closing quote in "Calculate" and before the rightmost /> bracket.
- Press the SPACEBAR once.
- Type `onclick="Calc(MortCalc)"` to add the event handler to the Calculate button, but do not press the ENTER key (Figure 10–21).

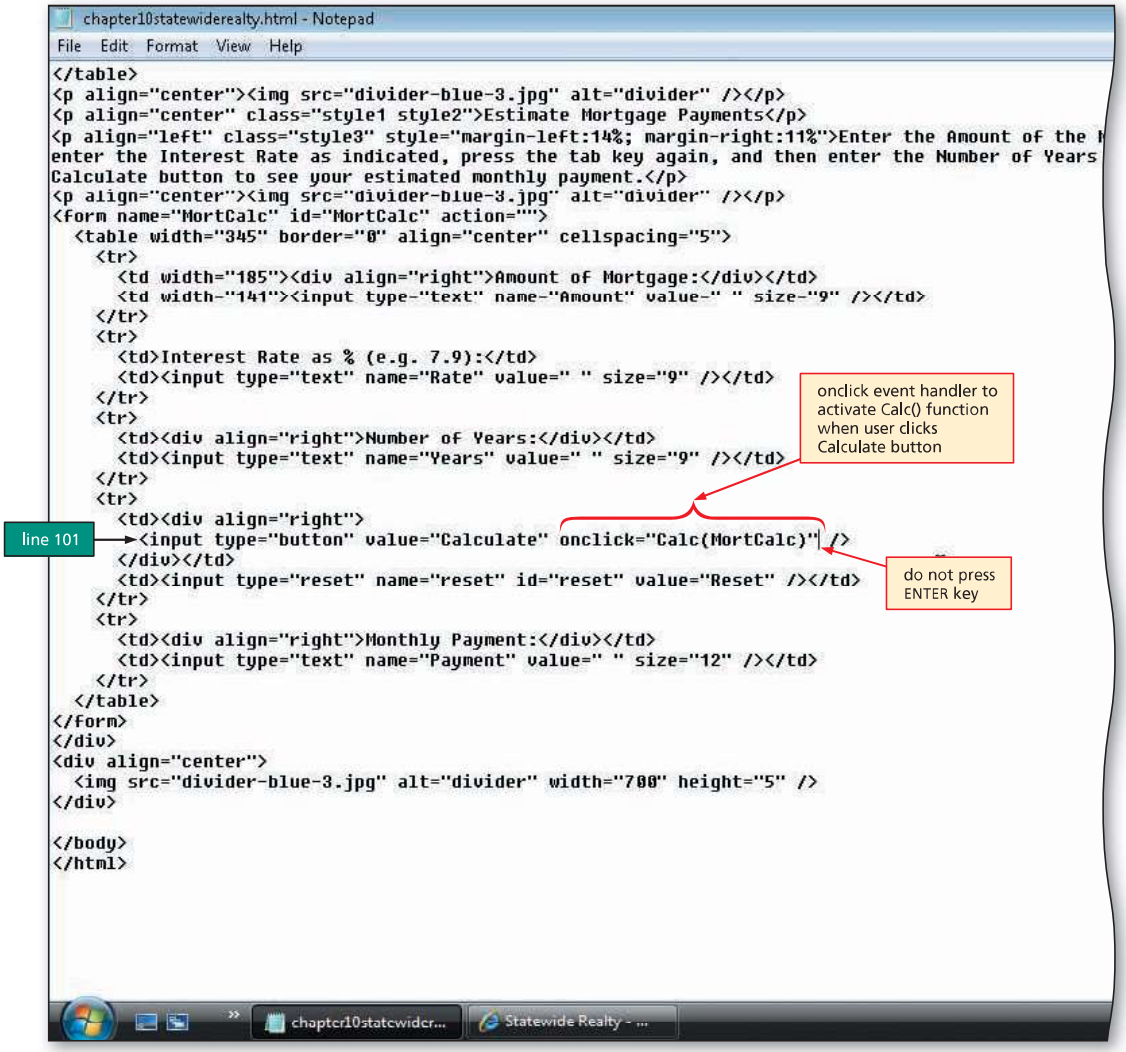


**Figure 10–21**

## To Save an HTML File and Test a Web Page

With the JavaScript code for the form validation entered, the Web page can be saved and tested in a browser. The Calc() function will validate the text field entries, but will not yet calculate the monthly payment. The following step shows how to save the HTML file and test the Web page.

1

- With the USB drive plugged into your computer, click File on the menu bar and then click Save.
- Click the browser button on the taskbar.
- Click the Refresh button on the browser toolbar.
- When the Web page is displayed, click the Amount of Mortgage text field.
- Enter test data set 1, as shown in Table 10–14.
- Press the TAB key to move the insertion point to the next text field.
- When you have entered test data set 1, click the Calculate button at the bottom of the form (Figure 10–22).

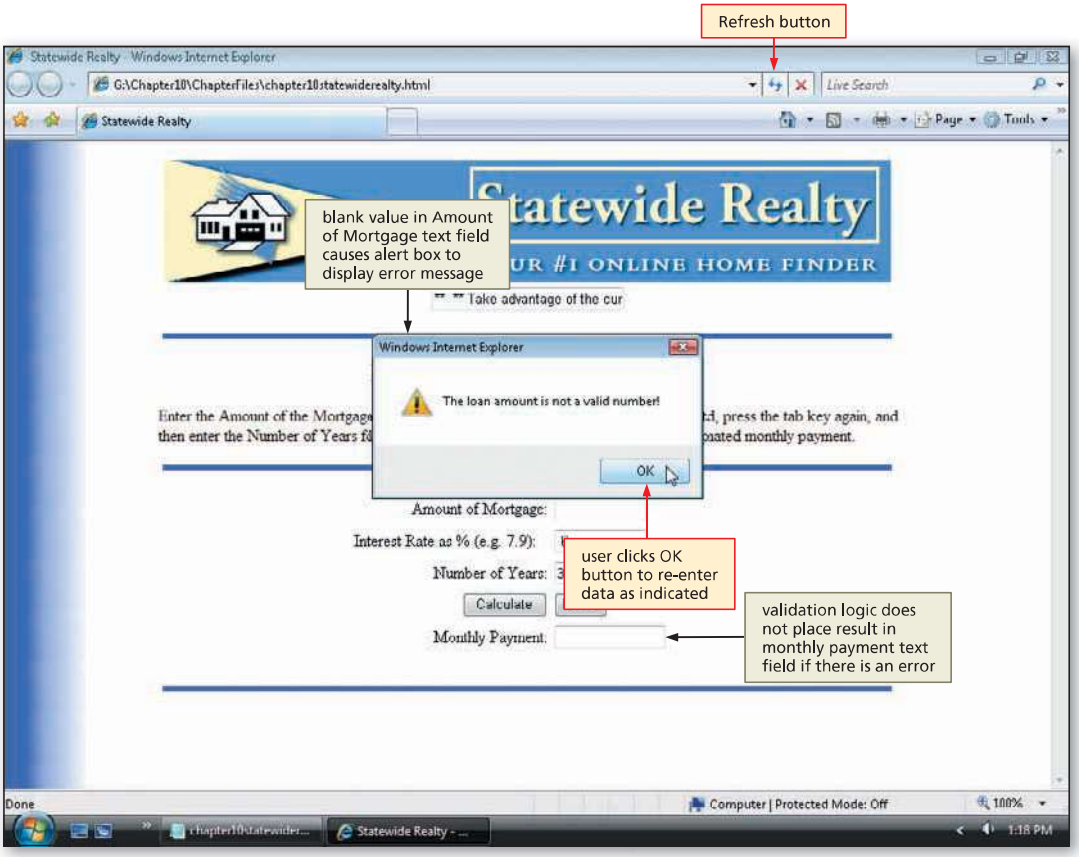


**Figure 10–22**

- When the message box is displayed, click the OK button.
- Click the Reset button at the bottom of the form.
- Repeat Steps 5 through 9, using test data sets 2, 3, and 4, as shown in Table 10–14.

**Table 10–14 Test Data Set**

| Data Set | Amount of Mortgage | Interest Rate % | Number of Years | Comment |
|---|---|---|---|---|
| 1 | | 6 | 30 | The loan amount is not a valid number! |
| 2 | 109000 | A | 4 | The interest rate is not a valid number! |
| 3 | 99000 | 6 | 30 | No error messages |
| 4 | 193000 | 5.9 | -30 | The number of years is not a valid number! |

**Plan Ahead**

**Calculating the monthly payment.**
The monthly() function requires three parameters: the mortgage loan amount (mortAmount), the interest rate (mortRate), and the number of years that the payments will be made (mortYears). These values are passed from the Calc() user-defined function to the monthly() function. The steps to calculate the monthly payment are as follows:

- The function call statement passes the three variables — mortAmount, mortRate, and mortYears — to the monthly() function.
- Convert the monthly interest rate to lrate by dividing mortRate by 1200
- Convert the number of years to Pmts by multiplying mortYears by 12
- Calculate the monthly payment with the following formula:

  mortAmount * (lrate / (1 - (1 / Math.pow(1+lrate,Pmts))))
- Return the monthly payment as a fixed decimal value to two decimal places using the toFixed(2) method.

# Adding the Monthly Payment Calculation

With the JavaScript code for the form validation complete, the next step is to add code to the Calc() function to calculate the monthly payment. First, a statement must be added to the Calc() function to call a user-defined function, named monthly(), which calculates the monthly payment. The monthly() function uses the valid data in the form and calculates the monthly payment. The result is the monthly payment, which is returned as a floating-point value.

The placement of the monthly() function within the Calc() function is important so that, if a value in a text field is invalid, the function does not attempt to process invalid data and return an undefined result. To place this function properly, one more else statement must be added to the Calc() function, as shown in Table 10–15.

**Table 10–15 Code to Call the monthly() Function**

| Line | Code |
|---|---|
| 43 | `else {` |
| 44 | `var mortPayment=monthly(mortAmount,mortRate,mortYears)` |
| 45 | `document.MortCalc.Payment.value=mortPayment` |
| 46 | `}` |

Line 43 adds an additional else statement to the nested if…else statements. Line 44 calls the monthly() function and passes the loan amount, interest rate, and number of years for the loan as variables: mortAmount, mortRate, and mortYears. The result is stored in a temporary variable named monthlyPmt. Line 45 assigns the result to the Monthly Payment text field on the form. Line 46 is the closing brace for the additional else statement.

## To Enter Code to Call the monthly() Function

The following step illustrates how to enter the final else statement and the function call that passes the required values to the monthly() function.

1

- Click line 43 and then press the ENTER key to insert a blank line.
- Click the blank line just inserted (line 43).
- Press the SPACEBAR to indent under the closing brace in line 42, then enter the JavaScript code shown in Table 10–15 to call the monthly user-defined function and assign the result to the payment text field but do not press the ENTER key (Figure 10–23).

**Q&A**

If I try to execute this Web page now, will an error occur?

Yes, because the monthly() user-defined function has not been written and entered.

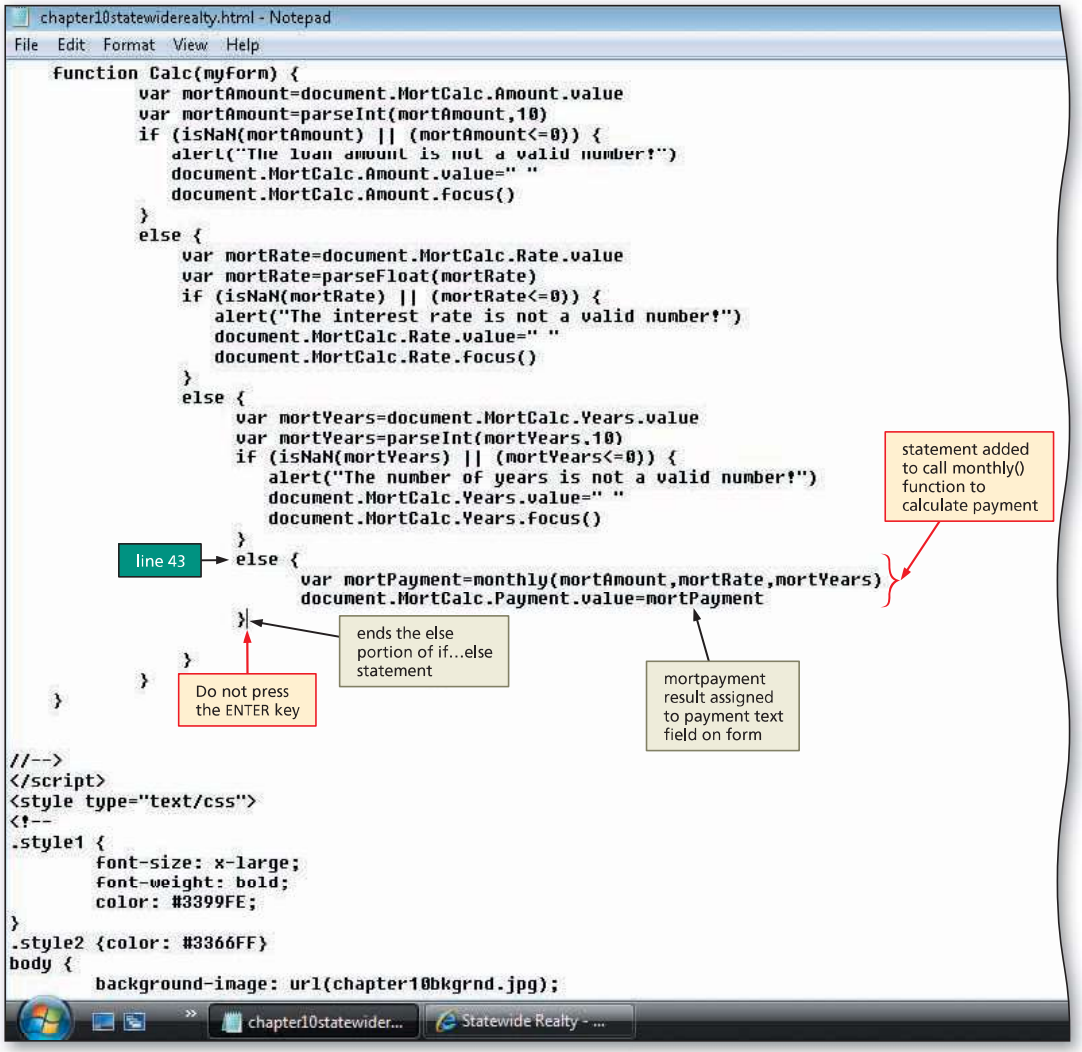


**Figure 10–23**

**Creating the monthly() User-Defined Function** The monthly() function is a user-defined function to calculate the monthly payment amount. The JavaScript code for the monthly() function is shown in Table 10–16.

**Table 10–16 Code for monthly() User-Defined Function**

| Line | Code |
|---|---|
| 51 | function monthly(mortAmount,mortRate,mortYears) { |
| 52 | var Irate=mortRate/1200 |
| 53 | var Pmts=mortYears*12 |
| 54 | var Amnt=mortAmount * (Irate / (1 - (1 / Math.pow(1+Irate,Pmts)))) |
| 55 | return Amnt.toFixed(2) |
| 56 | } |

Line 51 declares the monthly() function. Line 52 determines the monthly interest rate percentage by dividing the annual rate by 1200. The result is assigned to the Irate (interest rate) variable. Line 53 determines the number of monthly payments on the loan, by multiplying the number of years in the loan by 12. The resulting value is assigned to the Pmts variable.

Line 54 is the formula for calculating a monthly payment based on the amount of the loan, the monthly interest percentage, and the number of monthly payments. The mathematical representation of the formula is:

$$\text{loan amount} * (\text{monthly interest rate} / (1 - (1 / (1 + \text{monthly interest rate})^{\text{number of payments}})))$$

JavaScript, however, does not use typical programming language symbols to represent exponentiation in code. Instead, to calculate the expression $(1 + \text{monthly interest rate})^{\text{number of payments}}$, JavaScript uses the pow() method associated with the Math object. Table 10–17 shows the general form of the pow() method.

**Table 10–17 Math.pow() Method**

| | |
|---|---|
| General form: | Math.pow(number, exponent) |
| Comment: | where number is the value raised to the power of the exponent value. The pow() method accepts variables (X,n), constants (2,3), or both (Sidelength,2). |
| Examples: | Math.pow(2,3)<br>Math.pow(X,n)<br>Math.pow(Sidelength,2) |

BTW

**The Math Object**
The Math object cannot be used to create other objects. Most of the properties of the Math object return preset values. Other properties really are methods and act as functions.

The return statement in line 55 tells the function to send the results of the expression back as a fixed decimal value with a length of two. The Number object's toFixed() method returns a value set to a specific decimal length, as shown in Table 10–18.

**Table 10–18 Number.toFixed() Method**

| | | | |
|---|---|---|---|
| General form: | Number.toFixed(digits) | | |
| Comment: | where digits is the exact number of digits after the decimal point. The number is rounded or padded with zeros if necessary. | | |
| Examples: | Pmt = 234.8932<br>Amt = 843.6778 | Pmt.toFixed(3)<br>Amt.toFixed(2) | Result: 234.893<br>Result: 843.68 |

## To Create the monthly() Function

The following step illustrates how to enter the monthly() user-defined function to calculate the monthly payment on a mortgage loan.

1

- Click line 51.
- Position the insertion point on the blank line directly above the //--> tag. (If necessary, insert a blank line above the tag.)
- Enter the JavaScript code shown in Table 10–16 to write the code to calculate the monthly payment and then press the ENTER key twice (Figure 10–24).

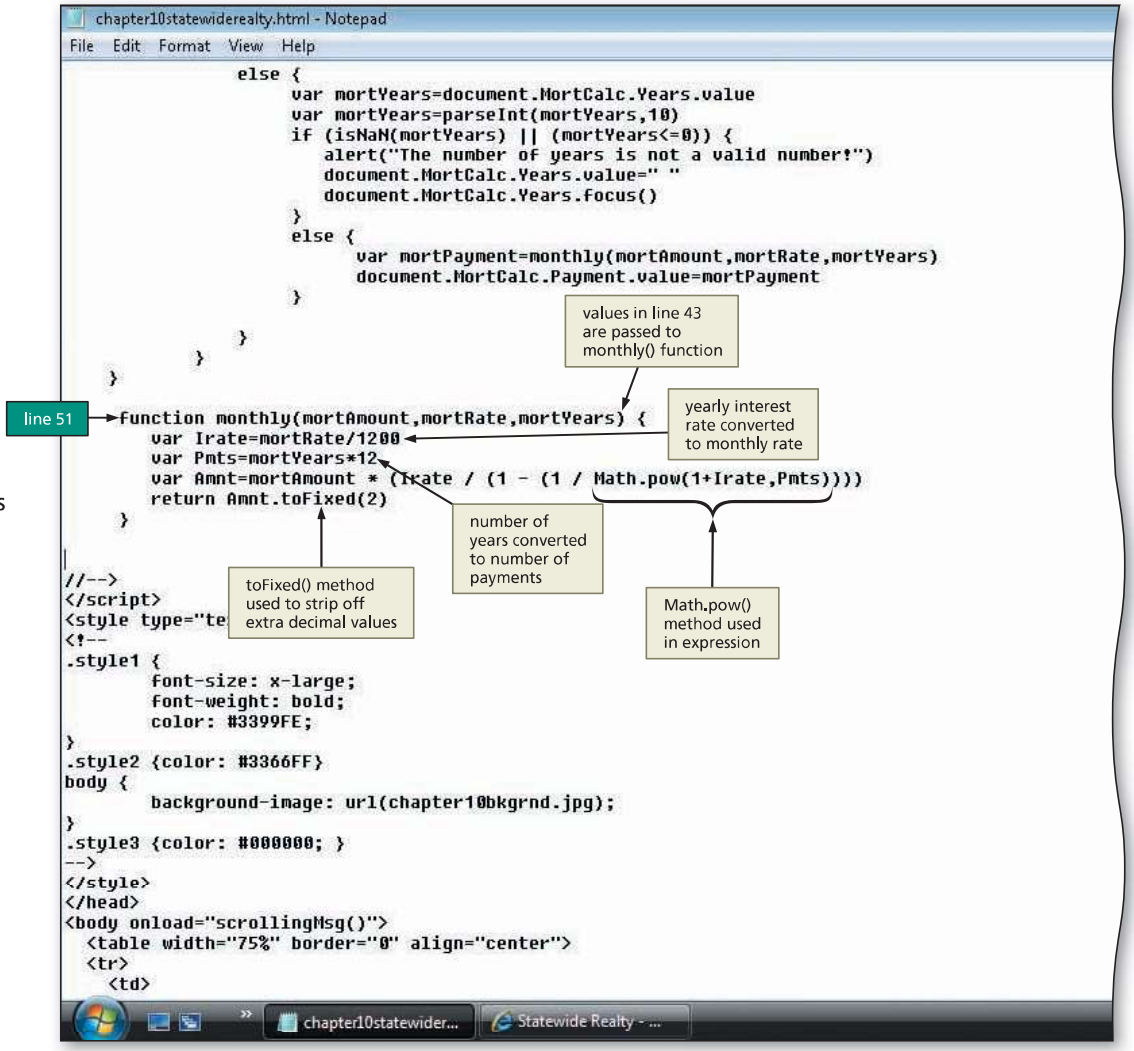


**Figure 10–24**

BTW

**Math.round() Method**

The Math.round() method returns the nearest integer value of a floating-point number. Thus, 28.453 will return 28. By multiplying the original number by 10 raised to the power of the number of decimals needed and then dividing by the same number, an original floating-point number also can be changed. For example, Math.round(9.453*100)/100 returns 9.45 and Math.round(9.454*10)/10 returns 9.5.

## To Save an HTML File and Test a Web Page

Now that the monthly payment can be calculated, this is a good place to test the Web page. The following step shows how to save the HTML file and test the Web page.

1

- With the USB drive plugged into the computer, click File on the menu bar and then click Save.
- Click the browser button on the taskbar.
- Click the Refresh button on the browser toolbar.
- If necessary, click the Amount of Mortgage text field to place the insertion point in the text field.
- Type 73000 in the Amount of Mortgage text field and then press the TAB key.
- Type 6 in the Interest Rate as % (e.g. 7.9) text field and then press the TAB key.

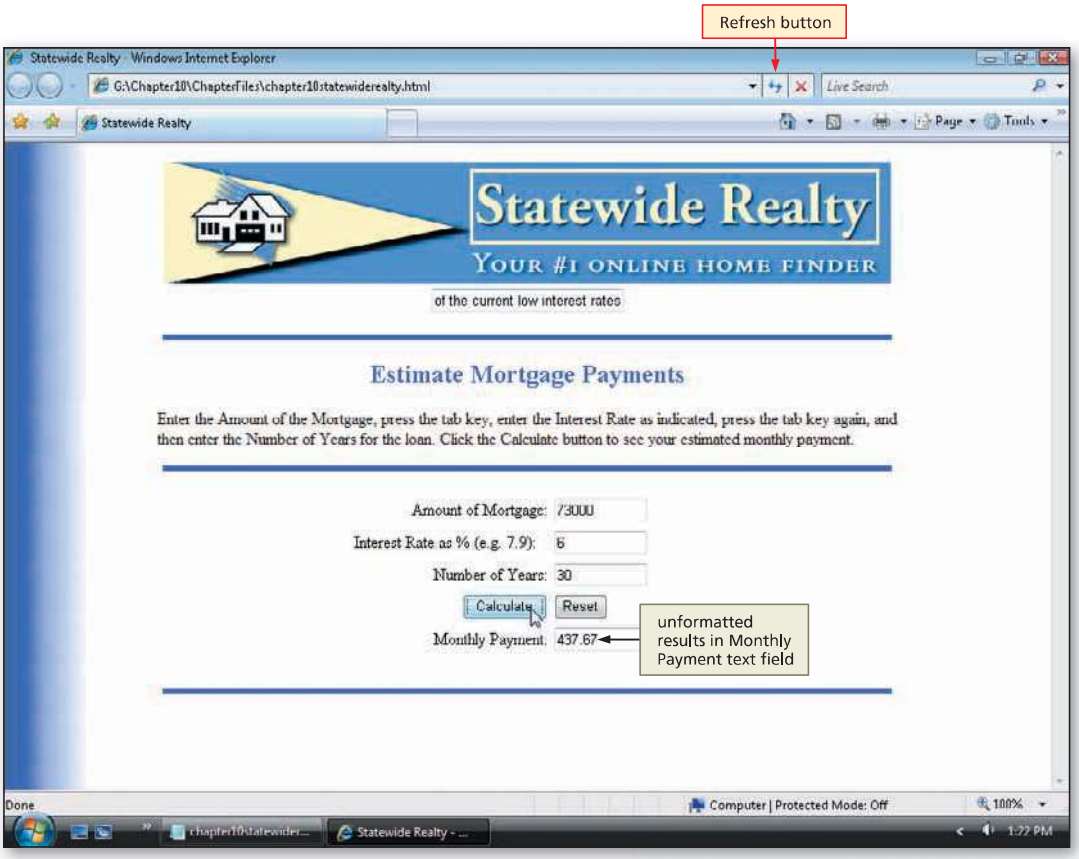


**Figure 10–25**

- Type 30 in the Number of Years text field and then click the Calculate button (Figure 10–25).

Q&A

What if my result was slightly different? Did I do something wrong?

If your result is only a few pennies different, it is probably just a difference in the math processor on your computer. If the value is hundreds or even thousands of dollars off, check your formulas on lines 52 through 54.

## Formatting the Monthly Payment Output as Currency

As shown in Figure 10–25, the mortgage loan payment calculator currently displays the monthly payment amount as a value with two decimal places. To set the form to display the monthly payment amount in a currency format with a dollar sign, the dollarFormat() function is used. First, you must enter a statement that passes the resulting string object of the monthly payment Calc() function to the dollarFormat() function. The dollarFormat() function then analyzes the string, adds commas, and returns the number with a dollar sign and two decimal places.

**Plan Ahead**

**Formatting output results.**
To format the result, the dollarFormat() function performs these seven basic steps:

1. Use the string value passed to the function, which separates the dollars from the cents based on the position of the decimal point
2. Determine the location of the decimal point using the indexOf() method
3. Separate the value to the left of the decimal point as the dollar amount and the value to the right of the decimal point as the cents amount
4. Insert commas every three positions in dollar amounts exceeding 999
5. Reconstruct the result string value with two decimal places
6. Insert a dollar sign immediately to the left of the first digit without spaces
7. Return the completed formatted value

The value is assigned to the Monthly Payment text field in the form.

**Using the indexOf() Method** The **indexOf()** method is used to search a string for a particular value and returns the relative location of that value within the string. The indexOf() method searches the string object for the desired value, which is enclosed within the quotation marks. Table 10–19 shows the general form of the indexOf() method.

**Table 10–19 indexOf() Method**

| | |
|---|---|
| General form: | `var position = stringname.indexOf("c")` |
| Comment: | where position is a variable; stringname is any string object; and "c" is the value for which the function searches. |
| Example: | `var decipos = valuein.indexOf(".")` |

If the search value is found in the string object, the indexOf() method returns the relative position of the value within the string object. If the search value is not found, the indexOf() method returns a negative one (–1). In this chapter, the indexOf() method is used to search for a decimal point in the monthly payment amount. Figure 10–26 provides an example of how the indexOf() method works.

```
<form name="regForm">
    <input type="text" name="emailAddr" value="" />

value of emailAddr: someName@some_emailer.com
var tEmail=document.regForm.emailAddr.value
var atPos=tEmail.indexOf("@")
value of atPos: 8
```

value entered in text field

value passed to variable

indexOf() method searches for position of @ sign

result returned to calling statement

**Figure 10–26**

**Beginning the dollarFormat() Function and Formatting the Dollars Portion** The dollarFormat() function initializes the variable that will return the formatted value and the variable used to manipulate the unformatted value. Most programmers agree it is a good programming practice to clear and initialize variables to ensure the data is valid. Table 10–20 shows the JavaScript code used to add the dollarFormat() function.

**Table 10–20 Code for the dollarFormat() Function**

| Line | Code |
|---|---|
| 58 | `function dollarFormat(valuein) {` |
| 59 | `var formatStr=""` |
| 60 | `var Outdollars=""` |
| 61 | `var decipos=valuein.indexOf(".")` |
| 62 | `if (decipos==-1)` |
| 63 | `decipos=valuein.length` |

Line 58 declares the dollarFormat() function and the valuein variable. Lines 59 and 60 clear the variables used to assemble the formatted output by assigning null (or empty) values. The indexOf() method in line 61 returns a value indicating the location of the decimal point — a value stored in the decipos variable. This value indicates at what position to concatenate the decimal values. Line 62 tests the condition of the decipos variable. If the value of decipos is –1, then decipos is set equal to the length of the string, as shown in line 63. If the value of decipos is 0, then the input value (valuein) is an integer and no decimal values need to be modified.

## To Enter the dollarFormat() Function

The following step shows how to enter the dollarFormat() function and initialize the variables.

1

- Click line 58 above the closing comment //-->.
- Enter the JavaScript code from Table 10–20 to begin the dollarFormat() function and then press the ENTER key (Figure 10–27).

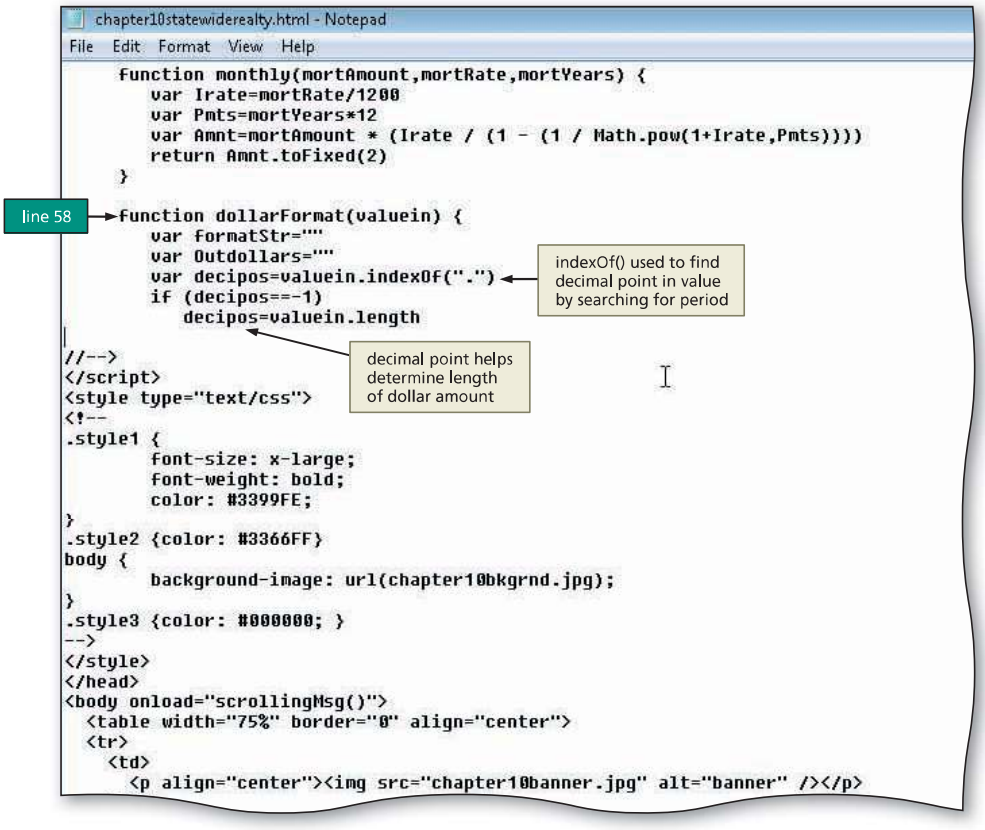


**Figure 10–27**

**Plan Ahead**

**Using a while loop to insert commas every three digits in a number.**
To place a comma every three digits, use a while loop. The following steps describe the logic of the while loop:

1. Extract three digits from the dollar value, starting from the right by subtracting 3 from the length of the dollar value (dollen).
2. Verify three digits have been subtracted and then insert a comma in the output string.
3. Decrement the length of the dollar value to look for the next group of three digits.
4. The process (loop) is complete when no more groups of three digits exist and the length of dollen is zero.

**Using an if...else Statement and while Loop to Extract the Dollars Portion and Insert Commas** A **loop** is a series of statements that executes repeatedly until it satisfies a condition. JavaScript has two types of loops: for loops and while loops. Both types of loops use the logic illustrated by the flowchart in Figure 10–28. Both loops first test a condition to determine if the instructions in the loop are to be executed.

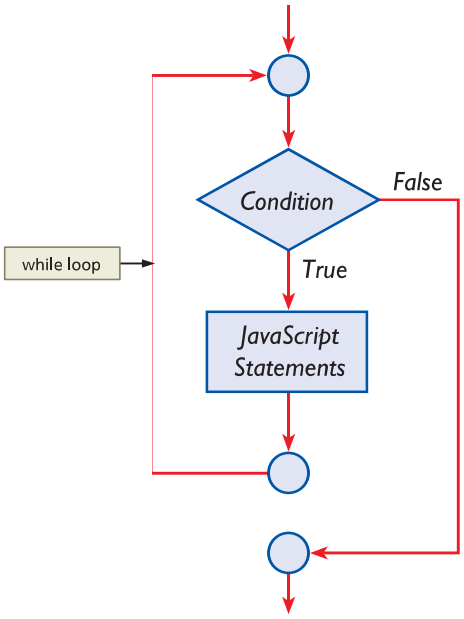


**Figure 10–28**

The **for loop** relies on a conditional statement using numeric values and thus often is referred to as a counter-controlled loop. Table 10–21 shows the general form of the for loop.

**Table 10–21 for Loop**

| | |
|---|---|
| General form: | `for (start; stop; counter-control) {`<br>`  JavaScript statements`<br>`}` |
| Comment: | where start is a variable initialized to a beginning value; stop is an expression indicating the condition at which the loop should terminate; and the counter-control is an expression indicating how to increment or decrement the counter. Semicolons separate the three variables. |
| Examples: | `for (j=1; j<5; j++) {`<br>`for (ctr=6; ctr>0; ctr--) {`<br>`for (itemx=1; itemx<10; itemx=itemx+2) {` |

The while loop relies on a conditional statement that either can use a numeric value or a string. Table 10–22 shows the general form of the while loop.

**Table 10–22 while Loop**

| | |
|---|---|
| General form: | `while (condition) {`<br>`  JavaScript statements`<br>`}` |
| Comment: | where condition is either a numeric value or a string; and the JavaScript statements execute while the result of the condition is true. |
| Examples: | `while (ctr < 6) {`<br>`while (isNaN(temp)) {`<br>`while (Response != "Done") {` |

In this chapter, the while loop is used in formatting the dollar value of the Monthly Payment value. The dollars portion is represented by the digits to the left of the decimal point. If the dollars portion of the mortgage loan payment contains more than three digits, commas need to be inserted. Table 10–23 shows the JavaScript statements used to determine the length of the dollar value and placement of the commas.

**Table 10–23 Code for Determining the Length of the Dollar Value and Comma Placement**

| Line | Code |
|---|---|
| 64 | `var dollars=valuein.substring(0,decipos)` |
| 65 | `var dollen=dollars.length` |
| 66 | `if (dollen>3) {` |
| 67 | `while (dollen>0) {` |
| 68 | `tDollars=dollars.substring(dollen-3,dollen)` |
| 69 | `if (tDollars.length==3) {` |
| 70 | `Outdollars=","+tDollars+Outdollars` |
| 71 | `dollen=dollen-3` |
| 72 | `} else {` |
| 73 | `Outdollars=tDollars+Outdollars` |
| 74 | `dollen=0` |
| 75 | `}` |
| 76 | `}` |
| 77 | `if (Outdollars.substring(0,1)==",")` |
| 78 | `dollars=Outdollars.substring(1,Outdollars.length)` |
| 79 | `else` |
| 80 | `dollars=Outdollars` |
| 81 | `}` |

The substring() method in line 64 uses the decipos value (the location of the decimal point) to extract the dollar value (the variable dollars). Next, a series of statements determines the length of the dollar value and then assigns the length to the variable dollen (line 65). Line 66 begins the if statement that determines if the dollar portion of the value is longer than three digits. The while loop routine (lines 67 through 76) places a comma every three digits, while the length of the dollar value is greater than three digits. Line 68 extracts three digits starting from the right by subtracting 3 from the length of the dollar value (dollen). Line 69 verifies three digits and line 70 inserts a comma in the output string. Line 71 decrements the length of the dollar value to look for the next group of three digits. When

no more groups of three digits exist, the length of dollen is set to zero (line 74) and the loop terminates at line 76. The statements in lines 77 through 80 prevent the code from inserting a comma if only three digits are to the left of the decimal point.

## To Enter an if…else Statement and while Loop to Extract the Dollar Portion of the Output and Insert Commas

The following step illustrates how to enter the if…else statement and while loop to extract the dollar portion of the output and insert commas into the output if needed.

1

- If necessary, click line 64, the line directly below the statement, decipos=valuein.length.
- Enter the JavaScript code as shown in Table 10–23 on the previous page using the SPACEBAR to indent as shown to extract the dollar portion of the output and insert the appropriate commas and then press the ENTER key (Figure 10–29).

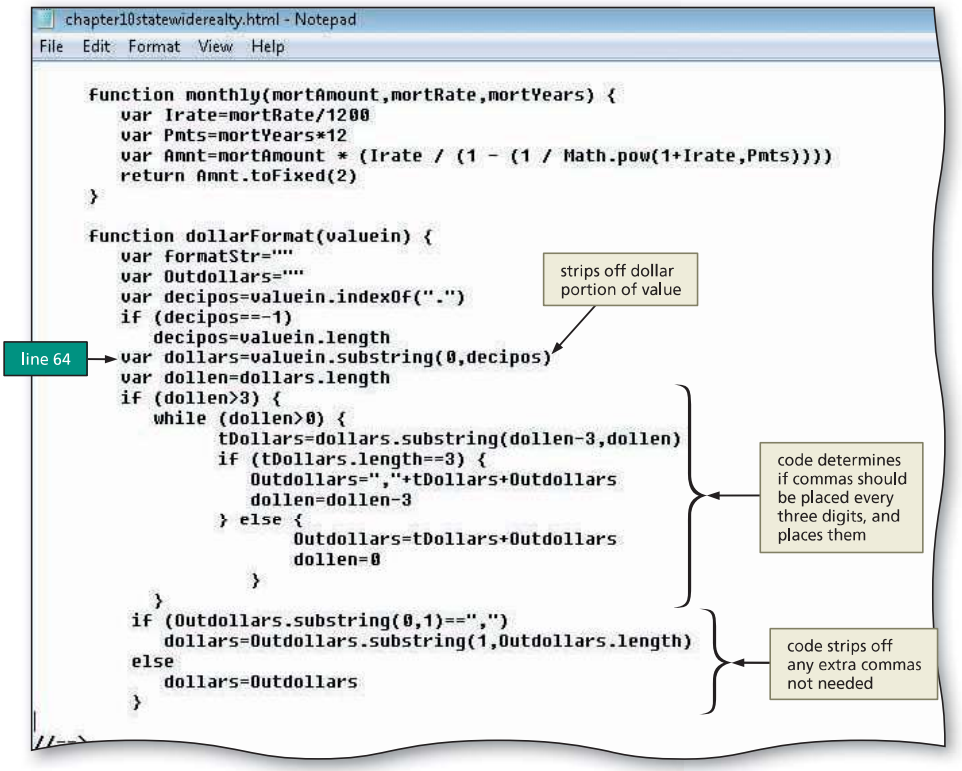


Figure 10–29

**Reconstructing the Formatted Output and Returning the Formatted Value** Next, the JavaScript statements must be written to reconstruct (concatenate) the formatted dollars and cents output into a formatted payment amount value, store the payment amount value in the formatStr variable, and return the formatStr value. Table 10–24 shows the statements needed to complete this task.

**Table 10–24 Code for Reconstructing the Formatted Output and Returning the Formatted Value**

| Line | Code |
|---|---|
| 82 | var cents=valuein.substring(decipos+1,decipos+3) |
| 83 | var formatStr="$"+dollars+"."+cents |
| 84 | return formatStr |
| 85 | } |

Line 82 extracts the two decimal places and assigns them to a variable to be used to reconstruct the formatted value. Line 83 reconstructs the values, concatenating a dollar sign ($) and the decimal value, and assigns the value to the formatStr variable. Line 84 returns the formatted value to the dollarFormat() function.

## To Reconstruct the Formatted Output and Return the Formatted Value

The following step illustrates how to reconstruct the formatted output and return the formatted value to the calling function.

1

- If necessary, click line 82.
- Enter the JavaScript code from Table 10–24 using the SPACEBAR to indent as shown to reconstruct the formatted output and return the formatted value and then press the ENTER key (Figure 10–30).

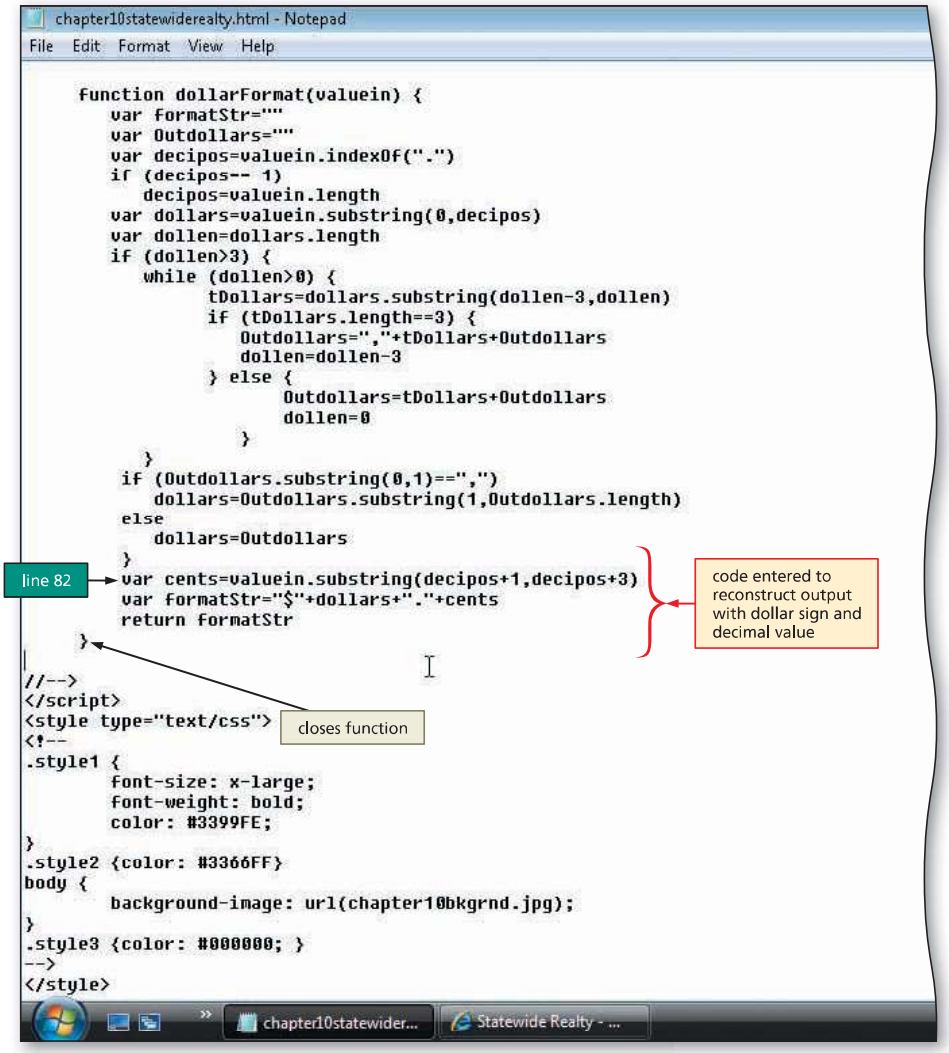


**Figure 10–30**

## To Pass the Monthly Payment Value to the dollarFormat() Function

To have the monthly payment value appear in the Monthly Payment text field formatted as currency, it must be passed to the dollarFormat() function. Because the dollarFormat() function manipulates a string value, the monthly payment result first must be converted to a string using the toString() method. In Chapter 9, the toString() method was used to convert a date value to a string. In this chapter, the toString() method is used to convert the monthly payment to a string that the dollarFormat() function can manipulate.

The following step shows how to enter the JavaScript statements needed to pass the monthly payment as a string object to the dollarFormat() function.

1

- Scroll up to and click line 45 (the line that starts document. MortCalc).
- Highlight and delete the variable, mortPayment, after the = symbol.
- Type `dollarFormat (mortPayment. toString())` to replace mortPayment with the dollarFormat() function using the mortPayment value. Do not press the ENTER key (Figure 10–31).

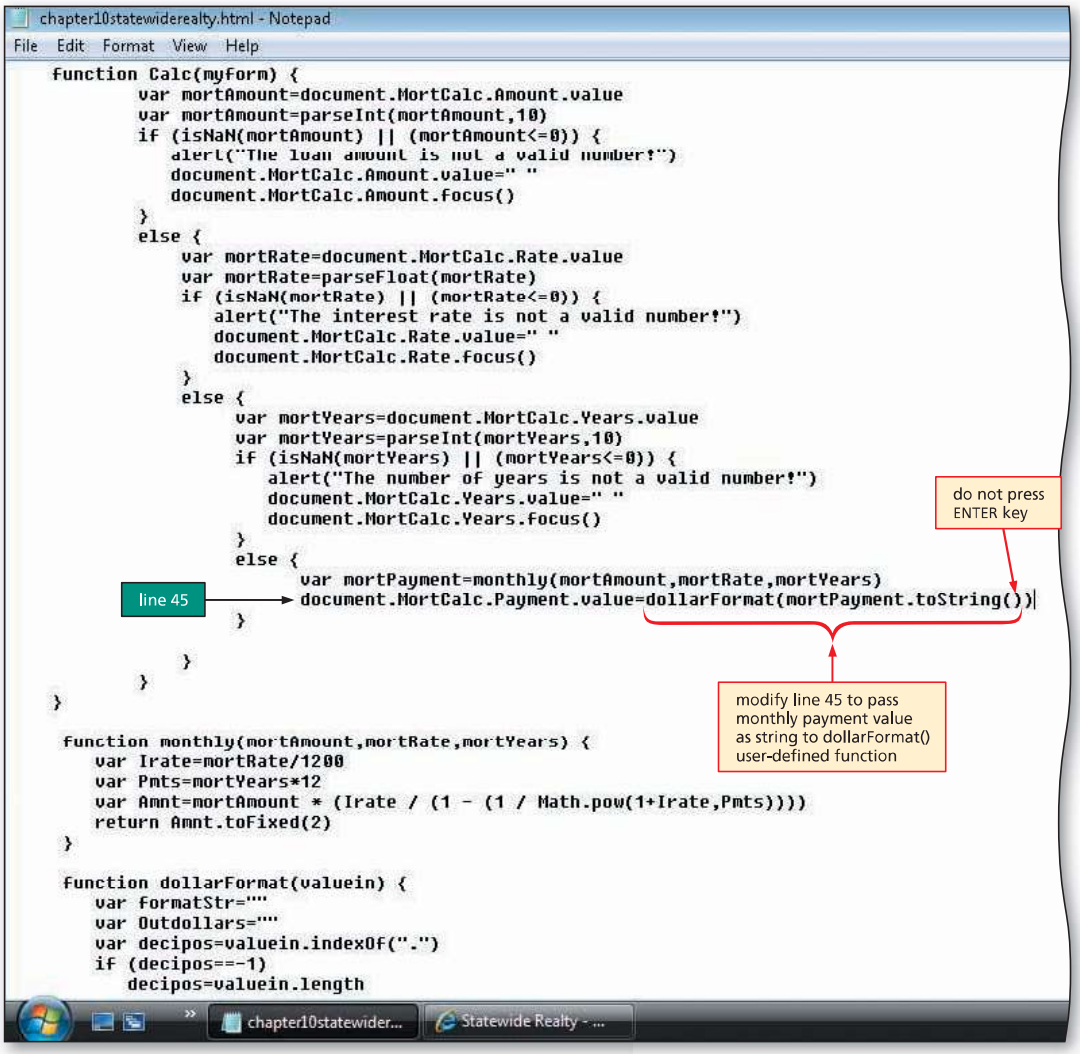


**Figure 10–31**

## To Save an HTML File and Test a Web Page

The following step shows how to save the HTML file and test the Web page.

1

- With the USB drive plugged into your computer, click File on the menu bar and then click Save.
- Click the browser button on the taskbar.
- Click the Refresh button on the browser toolbar.
- Enter the test data as follows: Amount of Mortgage: `173000`; Interest Rate as % (for example, 7.9): `6`; and Number of Years: `30`.
- Click the Calculate button. The result should be formatted as shown in Figure 10–32.

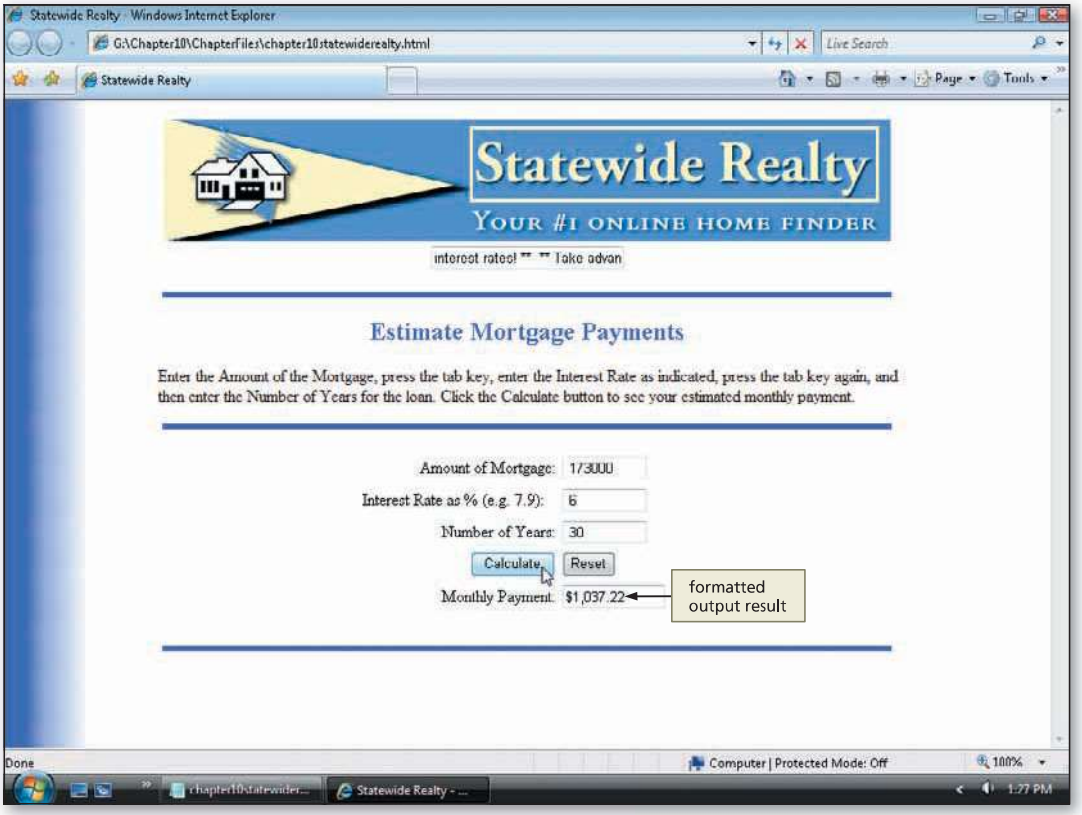


**Figure 10–32**

**Planning pop-up windows.**
A pop-up window is used to add additional information to Web page. You must decide what features you want the pop-up window to utilize. Ask yourself these questions:

- Do you want to include the status bar, title bar, address bar, scroll bars, toolbars, and menu bar?
- Do you want the user to be able to resize the window?
- What width and height should the window be?

Each of these properties can be set as properties in the open() method, creating a customized look for that particular pop-up window.

**Plan Ahead**

# Adding a Pop-Up Window

As you have learned in this chapter, the alert() method is one way to display messages to a user. These message boxes, however, only display text on a gray background. To create more visually interesting messages, you can use JavaScript to open and display another HTML file in a separate window that displays colors, graphics, animations, and other media. Such a window is called a **pop-up window**, because it appears over the previously opened browser window.

The **open() method** is used to create a pop-up window. Table 10–25 shows the general form of the open() method.

**Table 10–25 open() Method**

| | |
|---|---|
| General form: | `var windowname=open("window file name(URL)", "object name", "window features")` |
| Comment: | where windowname is an optional name of a window object (required only if you need to refer to the pop-up window in any other Web page); window file name is the name of the HTML file; and window features describe how the window should appear. |
| Examples: | `open("Adwindow.htm", "AdWin", "resize=off,titlebar=false")` |

BTW

**Pop-up Windows**
In addition to alert() message boxes, Web developers use pop-up windows to display more information to gain attention. Use pop-up windows sparingly, because having many pop-up windows can annoy visitors.

As shown in Table 10–25, when adding the open() method to create a pop-up window, all of the pop-up window features must be enclosed within one set of quotation marks. Table 10–26 describes the more commonly used attributes of the open() method, which are used to define pop-up window features. For more information about the open() method, see the JavaScript Quick Reference in Appendix E.

**Table 10–26 open() Method Attributes**

| Feature | Description | Written As | Comments |
|---|---|---|---|
| location | Includes address bar | "location=yes" | |
| menubar | Includes menu bar | "menubar=yes" | |
| resize | Allows user to resize | "resizeable=yes" | |
| scrollbars | Includes scroll bars | "scrollbars=yes" | |
| status | Includes status bar | "status=yes" | |
| titlebar | Removes title bar | "titlebar=false" | Default is true |
| toolbar | Includes toolbar | "toolbar=yes" | |
| width | States width in pixels | "width=220" | |
| height | States height in pixels | "height=450" | |

In this chapter, the open() method is used to open a pop-up window that will display information about Statewide Realty services. You insert code for the popupAd function and open() method, and then add an event handler to call the popupAd() function when the page is loaded. Earlier in the chapter, the onload event handler was associated with the scrollingMsg() function. You also will use the onload event handler for the popupAd() function. Multiple functions can be associated with the same event handler.

Table 10–27 shows the code to create the user-defined function popupAd(). The chapter10notice.html file already has been created and is stored in the Chapter10\ChapterFiles folder of the Data Files for Students.

**Table 10–27 Code to Open chapter10notice.html Pop-Up Window**

| Line | Code |
|---|---|
| 85 | `function popupAd() {` |
| 86 | `open("chapter10notice.html","noticeWin","width=550,height=360")` |
| 87 | `}` |

Line 85 declares the popupAd() function. The statement in line 86 opens the chapter10notice.html Web page as a pop-up window that is 550 pixels wide and 360 pixels high.

## To Enter the popupAd() Function to Open a Pop-Up Window

The following step illustrates how to enter the popupAd() user-defined function that contains the open() method to open the chapter10notice.html file in a pop-up window.

1

- Click line 85, the line directly above the //--> tag.
- Enter the JavaScript code from Table 10–27 to open a pop-up window and press the ENTER key (Figure 10–33).

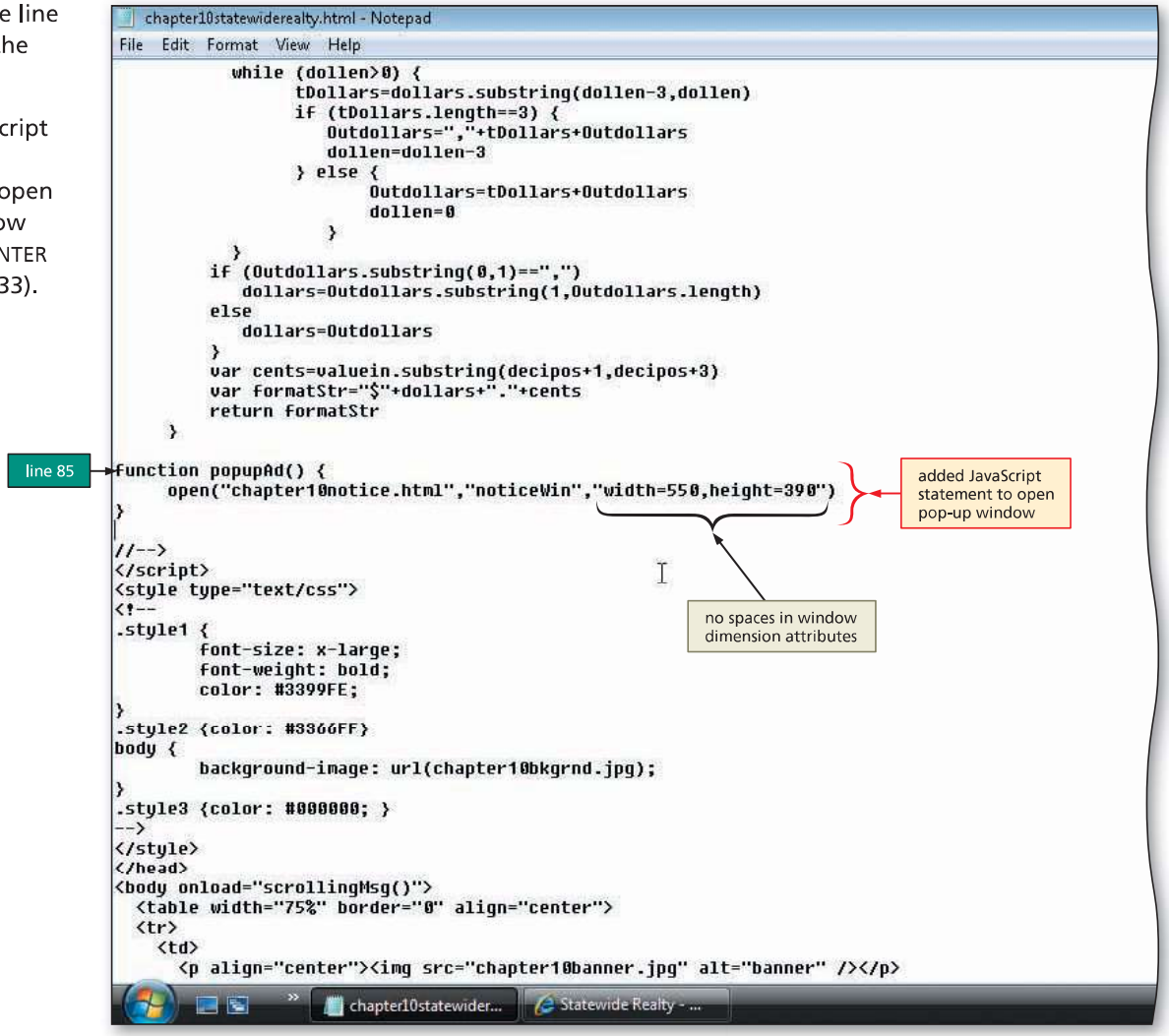


**Figure 10–33**

## To Add the Event Handler to Call the popupAd()Function

Now we need to add the onload event handler to open the pop-up window when the Web page loads. Each function name is enclosed in one set of quotation marks and separated by a semicolon. The following step illustrates how to add the second function call to the onload() event handler.

1

- Position the insertion point in front of the "> at the end of line 106 (the line that begins <body onload= ) (Figure 10–34).

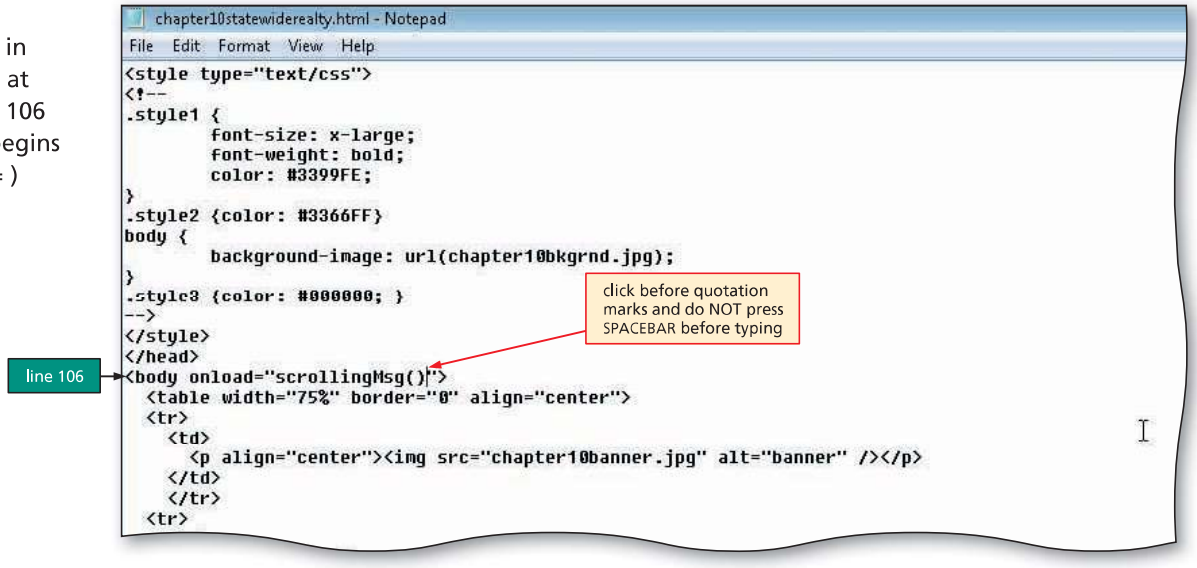


**Figure 10–34**

- Type ; popupAd() to add the call to the popupAd() user-defined function to the left of the second quotation mark (Figure 10–35).

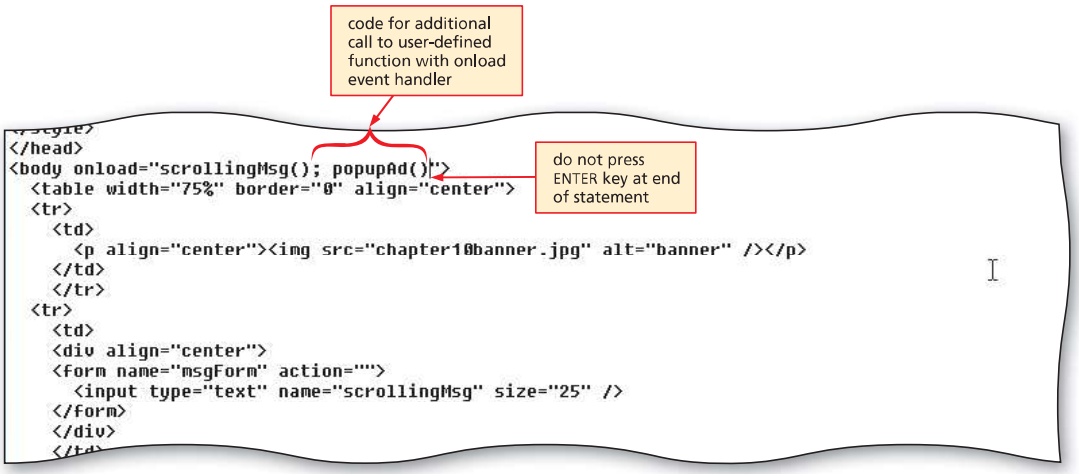


**Figure 10–35**

**Q&A**

**Is this the only way to open a pop-up window?**

An open() method can be placed anywhere in the <script> section in the <head> section. Once the <head> section loads in the browser, it will execute the "stand-alone" open() method, opening a pop-up window. The code must be in the <head> section, however, in order for this to work properly.

## Adding the Date Last Modified

As you learned in Chapter 9, the purpose of displaying the date a Web page was last modified is to make sure the user knows how current the information is contained on the Web page. You added JavaScript code to the Westlake Landscaping Web page to display the date and time a file was last modified. The Statewide Realty Web page should include similar JavaScript code to display just the date the Web page was modified, not

the time. To display just the date, the code in Table 10–28 uses the substring() method to grab just the date. The parameters used in the substring() method only return the date for Microsoft Internet Explorer properly.

**Table 10–28 Code to Display the Date Last Modified Using the substring() Method**

| Line | Code |
|---|---|
| 157 | `<script type="text/javascript">` |
| 158 | `<!--Hide from old browsers` |
| 159 | `document.write("<p style='margin-left:10%; font-family:Arial, sans-serif; font-size:xx-small; color:#0000ff'>")` |
| 160 | `document.write("Statewide Realty, Copyright 2009-2010. ")` |
| 161 | `document.write("<br />This document was last modified "+document.lastModified.substring(0,10)+"</p>")` |
| 162 | `//-->` |
| 163 | `</script>` |

Lines 157 and 158 are the beginning <script> section tags. Line 159 is the document.write() method with an embedded style in a <p> tag. Line 160 writes the copyright statement and line 161 writes the date last modified. The substring(0,10) method in line 161 extracts the first 10 characters of the date and time — mm/dd/yyyy — so only the date is displayed, not the time. Lines 162 and 163 complete the JavaScript section.

## To Display the Date Last Modified Using the substring() Method

The following step illustrates how to enter the JavaScript code to display the date the file was last modified using the substring() method.

1

- Click line 157 (the blank line between the  and </body> tags).
- Enter the JavaScript code from Table 10–28 to enter the copyright and date last modified code, pressing the ENTER key only at the end of each complete line. Do not press ENTER after the last </script> line (Figure 10–36).

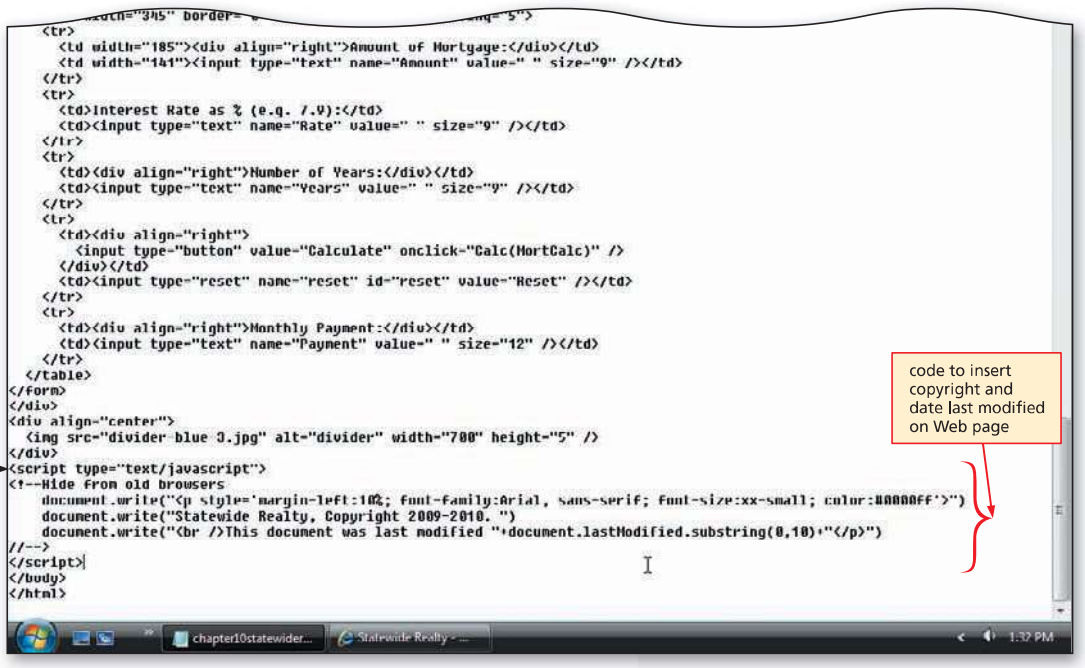


**Figure 10–36**

## To Save and Validate an HTML File, Test a Web Page, and Print the HTML File

The code for the Statewide Realty Web page with a mortgage loan payment calculator and pop-up window is complete. Now you should save the HTML file, test the JavaScript code using a Web browser, and then print the HTML file.

1

- With the USB drive plugged into your computer, click File on the menu bar and then click Save.
- Validate the Web page.
- Click the browser button on the taskbar.
- Click the Refresh button on the browser toolbar.
- Click the Close Window button to close the pop-up window.
- If necessary, scroll down to verify that the bottom of the Web page displays the date the page was last modified (the date the file was saved) as shown in Figure 10–37.

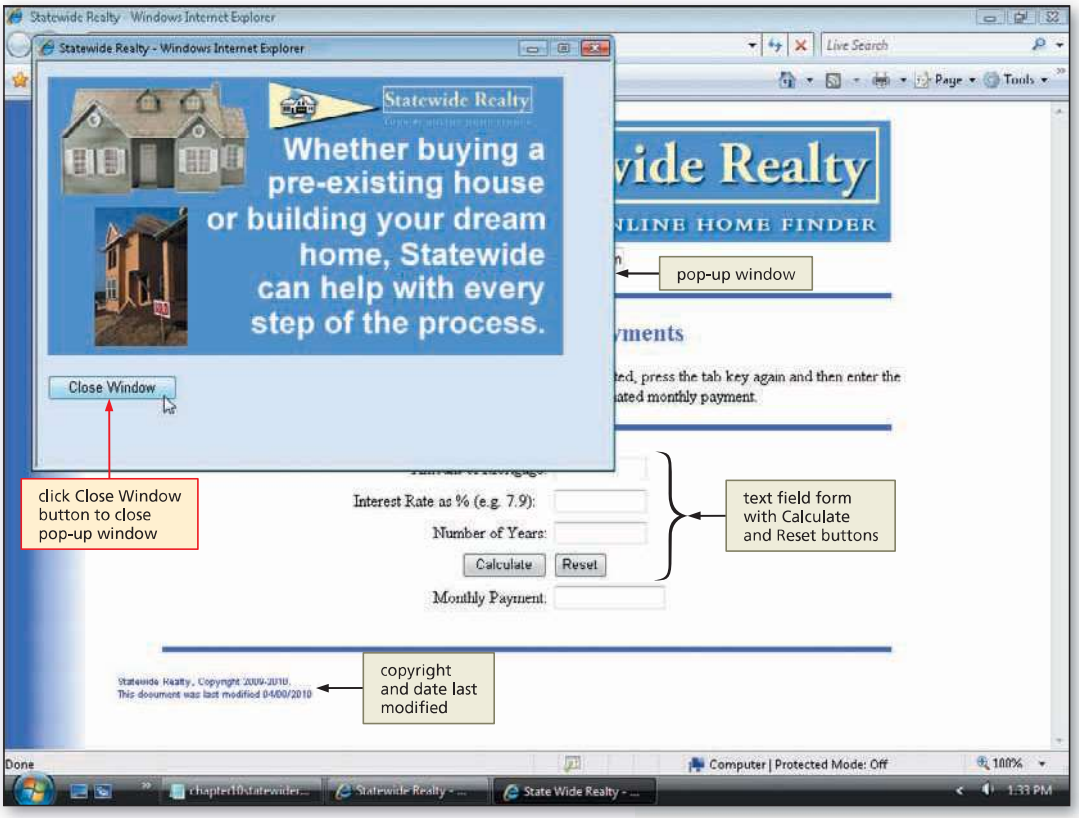


**Figure 10–37**

- If necessary, click the chapter10statewiderealty - Notepad button on the taskbar to activate the Notepad window.
- Click Print on the File menu to print the HTML file.

## To Close Notepad and a Browser

1

- Click the Close button on the browser title bar.
- Click the Close button on the Notepad window title bar.

BTW

**Quick Reference**
For a list of JavaScript statements and their associated attributes, see the JavaScript Quick Reference (Appendix E) at the back of this book, or visit the HTML Quick Reference Web page (scsite.com/HTML5e/qr).

## Chapter Summary

This chapter described how to write JavaScript to create a scrolling message, a pop-up window, if and if…else statements, pass values to a user-defined function, how to validate the data entered into a form and convert text to numeric values using the parseInt(), parseFloat(), and isNaN() built-in functions, and how to format string output results to display as currency. The items listed below include all the new JavaScript skills you have learned in this chapter.

1. Create a Form Text Field to Display a Scrolling Message (HTML 438)
2. Create the scrollingMsg() User-Defined Function (HTML 440)
3. Enter the Code to Increment the Position Locator Variable (HTML 441)
4. Enter an if Statement (HTML 444)
5. Add the setTimeout() Method to Create a Recursive Call (HTML 445)
6. Enter the onload Event Handler to Call the scrollingMsg() Function (HTML 447)
7. Start the Calc() Function and Nested if…else Statements to Validate Form Data (HTML 454)
8. End the Nested if...else Statements to Validate Form Data (HTML 456)
9. Enter an onclick() Event Handler to Call the Calc() Function (HTML 457)
10. Enter Code to Call the monthly() Function (HTML 460)
11. Create the monthly() Function (HTML 462)
12. Enter the dollarFormat() Function (HTML 465)
13. Enter an if…else Statement and while Loop to Extract the Dollar Portion of the Output and Insert Commas (HTML 468)
14. Reconstruct the Formatted Output and Return the Formatted Value (HTML 469)
15. Pass the Monthly Payment Value to the dollarFormat() Function (HTML 470)
16. Enter the popupAd() Function to Open a Pop-Up Window (HTML 473)
17. Add the Event Handler to Call the popupAd() Function (HTML 474)
18. Display the Date Last Modified Using the substring() Method (HTML 475)

## Learn It Online

Test your knowledge of chapter content and key terms.

*Instructions:* To complete the Learn It Online exercises, start your browser, click the address bar, and then enter the Web address `scsite.com/html5e/learn`. When the HTML Learn It Online page is displayed, click the link for the exercise you want to complete and read the instructions.

### Chapter Reinforcement TF, MC, and SA

A series of true/false, multiple choice, and short answer questions that test your knowledge of the chapter content.

### Flash Cards

An interactive learning environment where you identify chapter key terms associated with displayed definitions.

### Practice Test

A series of multiple choice questions that test your knowledge of chapter content and key terms.

### Who Wants To Be a Computer Genius?

An interactive game that challenges your knowledge of chapter content in the style of a television quiz show.

### Wheel of Terms

An interactive game that challenges your knowledge of chapter key terms in the style of the television show, *Wheel of Fortune*.

### Crossword Puzzle Challenge

A crossword puzzle that challenges your knowledge of key terms presented in the chapter.