



Lab 1-1: Creating a JavaScript-enabled page

In this lab, you will create your first JavaScript page, which will introduce two JavaScript objects using a method of one and two properties of the other. The first object is the `document` object and will use its `write` method. The second object is the `navigator` object and will use its `appName` and `appVersion` properties.

1. **Editor:** Open the **lab1-1.htm** file from the Lesson 1 folder of the Student_Files directory. Enter the code indicated in bold:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Lab 1-1</title>
</head>

<body>
<h3>CIW JavaScript Specialist</h3>
<hr />

<script type="text/javascript">
<!--

document.write(navigator.appName);
document.write("<p>");
document.write(navigator.appVersion);
document.write("</p>");

//-->
</script>

</body>
</html>
```

2. **Editor:** Save **lab1-1.htm**.
3. **Browser:** Open the **lab1-1.htm** file in your browser. Your screen should resemble Figure 1-2, depending on the browser you use (the figure shows the file displayed in Internet Explorer 8). You can see that this simple script determines and displays the type and version of browser used to display it.

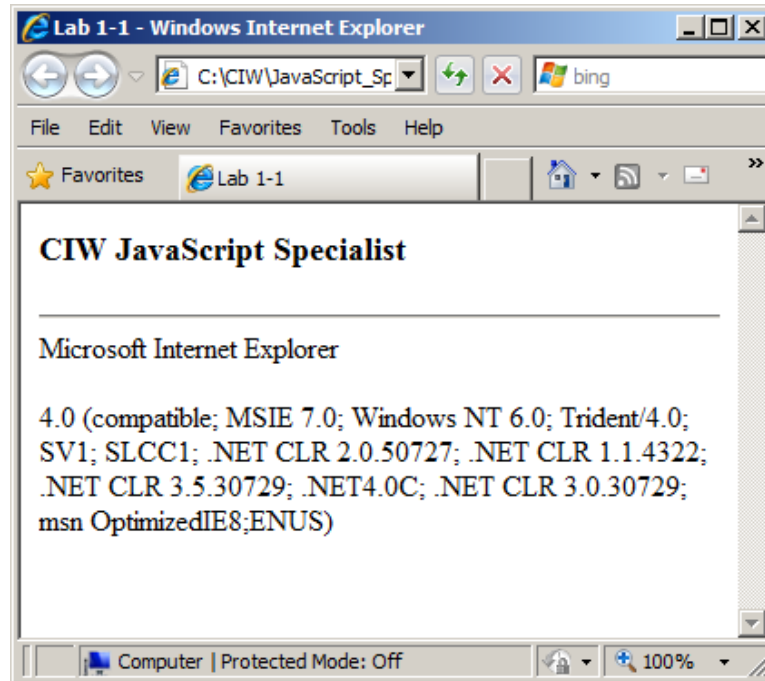


Figure 1-2: File lab1-1.htm displayed in Internet Explorer v8 browser

- 4. Browser:** Now launch a different browser, and use it to open the **lab1-1.htm** file. Your screen may resemble Figure 1-3, depending on the browser you use (the figure shows the file displayed in Mozilla Firefox). You can see that this simple script determines and displays the type and version of browser used to display it.

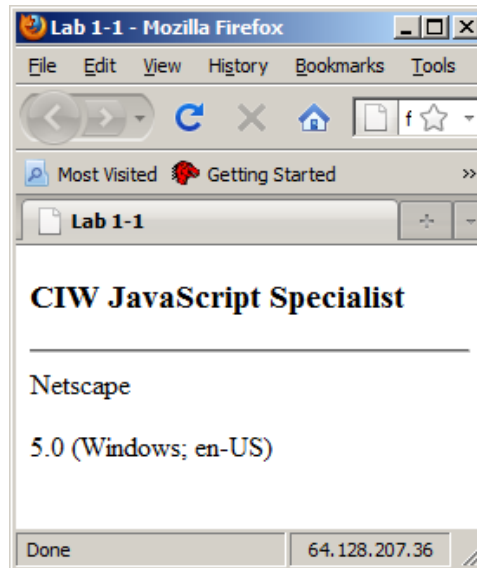


Figure 1-3: File lab1-1.htm displayed in Mozilla Firefox browser

- 5. Browser:** Study the display in the browser. As you can see, differences exist in the format that each browser uses for the output of the JavaScript statements. This example indicates the differences in implementing JavaScript from browser to browser.

- 6. Editor:** Review the code you wrote in the lab1-1.htm file. In this lab you used a `document.write()` statement. The document object's `write()` method is used to output data to the X/HTML stream. You also used the navigator object's `appName` and `appVersion` properties. The `appName` property returns a string value indicating the name of the client browser. The `appVersion` property returns a string value indicating the version number of the client browser, as well as the client's platform.

Notice that in the `document.write()` statements, the code `navigator.appName` and `navigator.appVersion` was not typed inside quotation marks, whereas the X/HTML `<p>` tag was inside quotation marks. The two lines of code using the `navigator` object are evaluated text. In other words, the JavaScript interpreter dynamically supplies the appropriate text when the script is executed. Therefore, that text was not inside quotation marks. The literal `<p>` tag is static text. Its value is known before the script runs, so it is placed inside quotation marks.



Lab 2-1: Using the JavaScript `alert()` method

In this lab, you will use the JavaScript `alert()` method to display a message to the user.

1. **Editor:** Open the **lab2-1.htm** file from the Lesson 2 folder of the Student_Files directory.
2. **Editor:** Locate the `<script type="text/javascript"></script>` block in the `<head>` section of the document. Within the block, add an `alert()` method with the message "Good Morning!" as the text within the alert box.
3. **Editor:** Save **lab2-1.htm**.
4. **Browser:** Open **lab2-1.htm**. You should see a dialog box that resembles Figure 2-1. If you do not, verify that the source code you entered is correct.

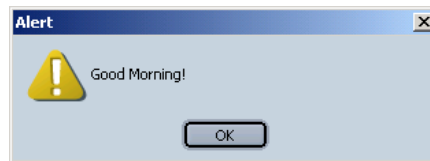


Figure 2-1: Alert message

5. **Browser:** After you click **OK**, your screen should resemble Figure 2-2.

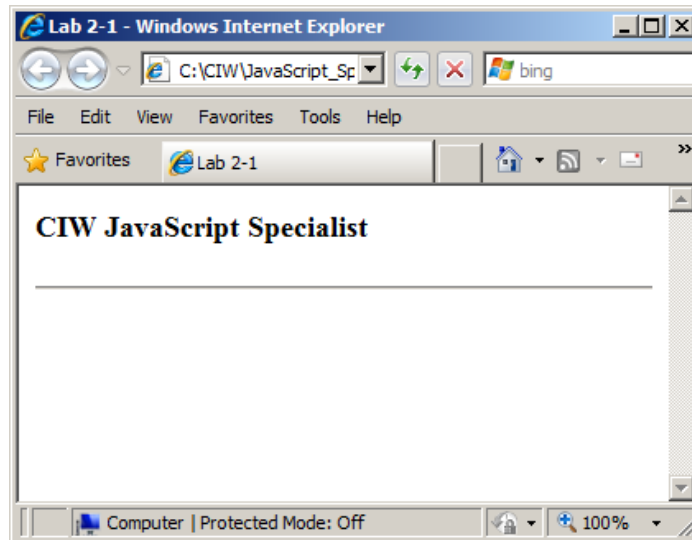


Figure 2-2: File lab2-1.htm displayed following JavaScript statement



Lab 2-2: Using the JavaScript `prompt()` method

In this lab, you will use the JavaScript `prompt()` method with concatenation to request and capture user input.

- 1. Editor:** Open **lab2-2.htm** from the Lesson 2 folder of the Student_Files directory.
- 2. Editor:** Locate the `alert()` method that has been defined for you. Modify the source code by adding a `prompt()` method that asks for the user's name. Concatenate the user's input with the existing text and add a closing period after the user input.
- 3. Editor:** Save **lab2-2.htm**.
- 4. Browser:** Open **lab2-2.htm**. When the page loads, you should see a prompt dialog box that resembles Figure 2-3. If not, verify that the source code you entered is correct.

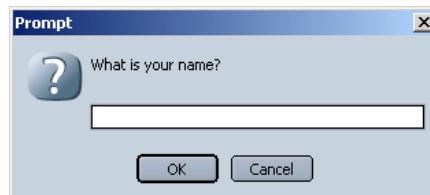


Figure 2-3: User prompt dialog box

- 5. Browser:** Enter your name in the text field, and then click **OK**. Your screen should display a message that resembles the one shown in Figure 2-4.



Figure 2-4: Alert message box

- 6. Browser:** When you click **OK**, your screen should resemble Figure 2-2 (from the previous lab).
- 7. Browser:** Reload the page to run the script again. This time, do not enter any text in the prompt, then click **OK**. The alert will display the message "Good morning, ." This is evidence of an empty string, which is a string that contains no characters.
- 8. Browser:** Reload the page again. This time, click **Cancel** (with or without first entering any text). The alert will display the message "Good morning, null." When no data is entered by the user, the `prompt()` method returns a null value, which is converted to the string "null" in this return display. Be sure to consider how the user's actions might affect any JavaScript methods you use that incorporate user input.

In this lab, the `prompt()` method is processed first and the user's input is then concatenated into the expression. In other words, the `prompt()` method will take precedence over the `alert()` method in a JavaScript statement. In fact, JavaScript statements always execute from the inside out.



Lab 2-3: Using the JavaScript `confirm()` method

In this lab, you will use the `confirm()` method to elicit a true or false return value from the user.

- 1. Editor:** Open **lab2-3.htm** from the Lesson 2 folder of the Student_Files directory.
- 2. Editor:** Locate the `alert()` method that has been defined for you. Modify the code to use the return value of a `confirm()` dialog box as the text for an `alert()` dialog box. In other words, concatenate the result of a `confirm()` method into an `alert()` method, just as you previously concatenated the result of a `prompt()` method into an `alert()`. The argument for the `confirm()` method should read, **"Do you want to proceed?"**
- 3. Editor:** Save **lab2-3.htm**.
- 4. Browser:** Open **lab2-3.htm**. When the page loads, you should see a confirm dialog box that resembles Figure 2-5. If not, verify that the source code you entered is correct.

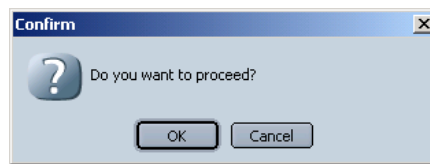


Figure 2-5: Confirm dialog box

- 5. Browser:** Click **OK**. Your screen should resemble Figure 2-6.

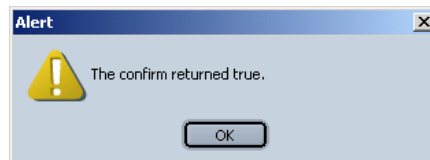


Figure 2-6: Result of `confirm()` method after clicking OK

- 6. Browser:** Reload **lab2-3.htm**. Click **Cancel**. Your screen should resemble Figure 2-7.

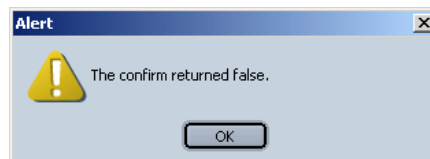


Figure 2-7: Result of `confirm()` method after clicking Cancel

Be aware that you can use the true or false result of the `confirm()` method to initiate further action. For example, a return value of true could launch another pop-up with additional information or redirect users to a new page on the site.



Lab 2-4: Using the JavaScript `document.write()` method

In this lab, you will use the `document.write()` method to customize a page for the user.

- 1. Editor:** Open **lab2-4.htm** from the Lesson 2 folder of the Student_Files directory.
- 2. Editor:** Locate the `prompt()` method that has been defined for you. Modify the source code to use a `document.write()` statement. Concatenate the results of the `prompt()` method into the `document.write()` expression. Designate the output of the `document.write()` as an `<h4>` level greeting that displays the following:

Welcome, user's name.

The text "user's name" will be the return value from the `prompt()` method. Be sure to end with a period and close the `<h4>` tag after inserting the user's name.

- 3. Editor:** Save **lab2-4.htm**.
- 4. Browser:** Open **lab2-4.htm**. You should see a dialog box that resembles Figure 2-8.



Figure 2-8: User prompt

- 5. Browser:** Type your name in the dialog box, and then click **OK**. Your screen should resemble Figure 2-9.

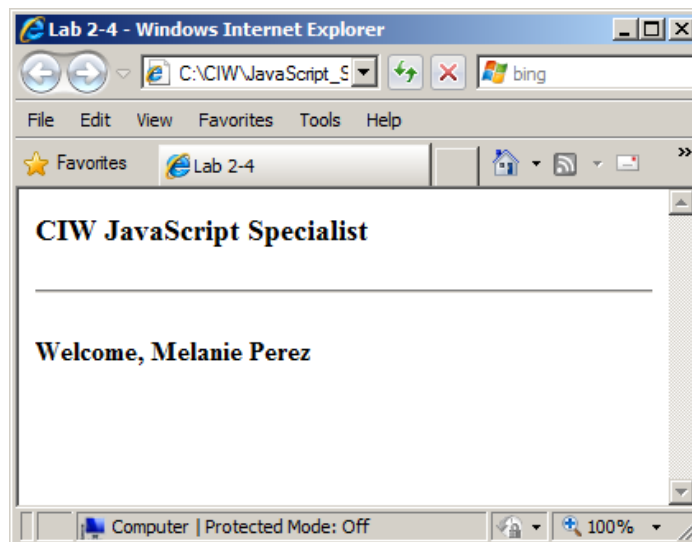



Figure 2-9: Page for lab2-4.htm with customized welcome message

- 6.** Edit the initial prompt to contain a message within the text entry field. Place the text string you want inside of the now-empty quotation marks. For example, you could insert the following text shown in bold:

```
prompt("What is your name?","Thank you   
for entering your name here");
```

This addition would alter the initial alert box as shown in Figure 2-10.



Figure 2-10: Customizing initial prompt

Note that in this lab, you were able to include an XHTML heading tag as part of the text that was written to the screen. X/HTML can be freely interspersed with text when using the `document.write()` method. Note also that the `prompt()` method takes processing precedence over the `document.write()` method when both are used in the same expression.



Lab 2-5: Storing user data in a JavaScript variable

In this lab, you will store the user's name in a variable so you can use it as needed.

1. **Editor:** Open **lab2-5.htm** from the Lesson 2 folder of the Student_Files directory.
2. **Editor:** Locate the `prompt()` method that has been defined for you. Modify the source code to add a variable named **userName**. Assign the result of the `prompt()` method as the value for `userName`.
3. **Editor:** Concatenate the `userName` variable into the `alert()` and `document.write()` expressions that have been provided for you. Make sure to concatenate closing periods in both the `alert()` and `document.write()` statements.
4. **Editor:** Save **lab2-5.htm**.
5. **Browser:** Open **lab2-5.htm**. Compare the prompt dialog box displayed in your browser with Figure 2-11. They should be similar.

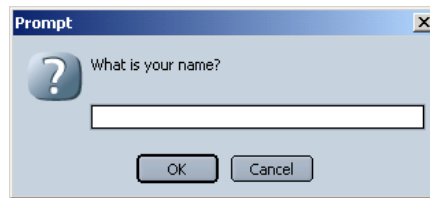


Figure 2-11: User prompt dialog box

6. **Browser:** Enter your name in the text field and click **OK**. Your screen should display a message similar to Figure 2-12.

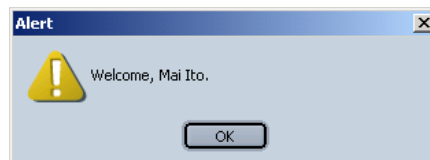


Figure 2-12: Alert message box

7. **Browser:** Click **OK** again, and your screen should resemble Figure 2-13.

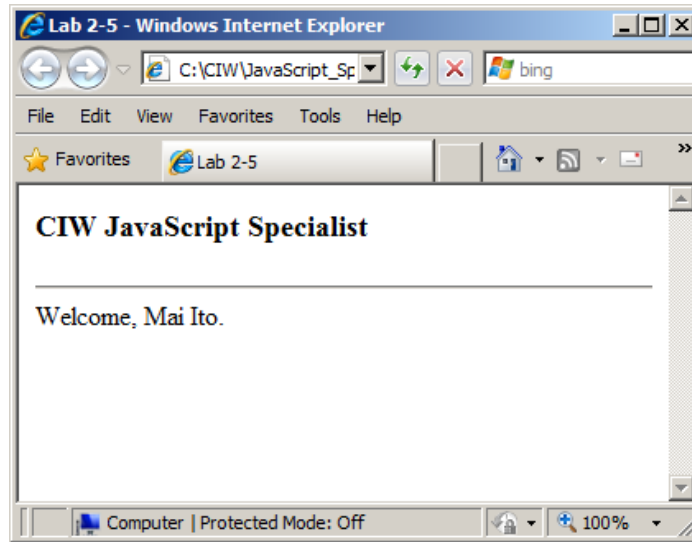


Figure 2-13: Page for lab2-5.htm with welcome message recalling user's name

Tech Note: If you try to edit an X/HTML file with an alert message present in the browser, you may not be able to save your file until you acknowledge the message by clicking the OK button. If you do not acknowledge the alert, you might receive a "file sharing" violation from your operating system.

In this lab, you declared a variable called `userName`. In that variable, you stored the result of the `prompt()` command. The `=` character was used as the assignment operator.

You then concatenated the phrase "Welcome, " with the `userName` variable using the `+` operator, and displayed the result in an alert box. After the user acknowledged the first alert, you generated a `document.write()` statement, with text before and after the variable `userName`.



Lab 2-6: Assigning and adding variables in JavaScript

In this lab, you will assign variables and use the addition operator to add them together.

- 1. Editor:** Open the **lab2-6.htm** file from the Lesson 2 folder of the Student_Files directory.
- 2. Editor:** Add source code inside the `<script type="text/javascript"></script>` block. Create two variables named **x** and **y**. Assign the numerical value of 4 to the first variable and 9 to the second variable.
- 3. Editor:** Create a third variable named **z**. Assign to **z** the result of adding together **x** and **y**. Display the variable **z** in an `alert()` dialog box.
- 4. Editor:** Save **lab2-6.htm**.
- 5. Browser:** Open **lab2-6.htm**. Your screen should immediately display an alert box, as shown in Figure 2-14.

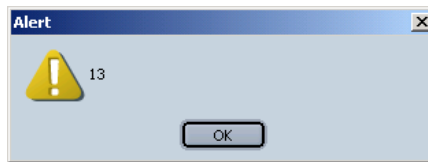


Figure 2-14: Result of JavaScript addition: z=13

- 6. Editor:** Immediately after the `alert(z)` line, add the following source code:

```
alert("4 + 9 = " + x + y);
```
- 7. Editor:** Save **lab2-6.htm**.
- 8. Browser:** Reload **lab2-6.htm**. Your screen should display the alert box showing 13, as seen in Figure 2-14. When you click OK, you should then see another alert box as shown in Figure 2-15.

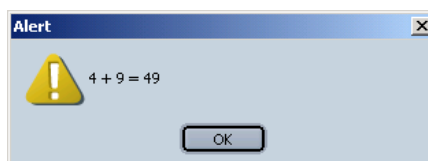


Figure 2-15: Concatenation, not sum

This alert shows that the `+` symbol was interpreted as an operator for concatenation. Because the values were concatenated, the script returned a result of 49. Note that this behavior is caused by the fact that the `alert()` method normally takes a string value as its argument. So the JavaScript interpreter treated the values as strings instead of numerical values.

It is common for the JavaScript interpreter to mistake numbers for characters and treat them accordingly. Thus, the interpreter is not recognizing **x** and **y** as numbers, but as characters. You would expect this alert to show "13" ($x + y$), but the script sees "4 + 9" as a string. Therefore, instead of performing addition, it performs concatenation, (working left to right, following mathematical precedence), for a result of 49, instead of adding them for a result of 13.

Tech Note: Mathematical precedence dictates the order in which operations will be worked: First, the script will work within the parentheses, doing multiplication and division first, then addition and subtraction next. Then the script will go outside the parentheses and work multiplication/division first, then work left to right doing addition and subtraction. This fact is important to know when you are creating complex formulas using JavaScript.

9. **Editor:** Modify the source code as indicated in bold, then save the file:

```
alert("4 + 9 = " + (x + y));
```

By adding parentheses around the $x + y$ section of the expression, you are asking JavaScript to first calculate $(x + y)$ and then attach it to the preceding text string. So the two integers are added before interacting with the string.

10. **Browser:** Reload **lab2-5.htm**. Now, as you click **OK**, you should see the alert boxes shown in Figure 2-14 and Figure 2-16, in sequence.

Note: Depending on the browser used, you may see another alert box appear in between these two that says, "Explicit declaration: $4 + 9 = 13$."

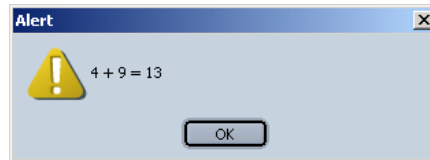


Figure 2-16: Correct sum is indicated



Lab 2-7: Using the JavaScript *onunload* event handler and inline scripting

In this lab, you will obtain the user's name as he or she loads the page, then use this data to personalize the user's visit in multiple ways, even as the page is unloaded. You will use both inline scripting and the `onunload` event handler.

1. **Editor:** Open **lab2-7.htm** from the Lesson 2 folder of the Student_Files directory.
2. **Editor:** Enter source code in the `<body>` tag. Add an **onunload** event handler that calls an `alert()` box. The text should read *"Goodbye, "* with the `userName` variable concatenated into the string, then the text *"Hurry back!"*
3. **Editor:** Save **lab2-7.htm**.
4. **Browser:** Open **lab2-7.htm**. You should see a series of dialog and alert boxes. Enter your name in the prompt dialog box and click **OK** as shown in Figure 2-17.

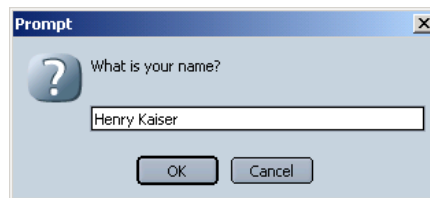


Figure 2-17: Prompt dialog box

5. **Browser:** The greeting alert box will appear, as shown in Figure 2-18. Click **OK**.

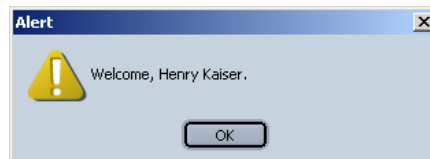


Figure 2-18: Alert box

6. **Browser:** The lab2-7.htm page will appear with a personalized greeting, as shown in Figure 2-19.

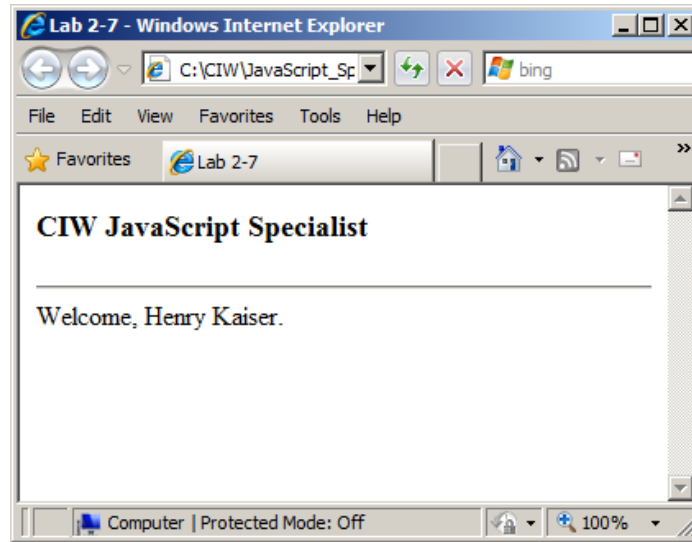


Figure 2-19: Page for lab 2-7.htm with welcome message

- 7. Browser:** Click the **Back** button or the **Home** button from your browser's tool bar to navigate to a different page. You should see the alert box shown in Figure 2-20.

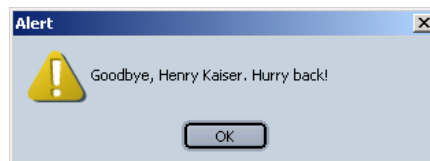


Figure 2-20: Goodbye alert box

- 8. Browser:** After clicking **OK** on the alert dialog box, your screen will show the page to which you navigated.

This lab provided an opportunity to add inline scripting using the `onunload` event handler. You will learn more about inline scripting, events and event handlers in later lessons.



Lab 3-1: Creating a user-defined function in JavaScript

In this lab, you will create a simple function that displays an alert dialog box. The function will be invoked by the `onload` event handler.

1. **Editor:** Open **lab3-1.htm** from the Lesson 3 folder of the Student_Files directory.
2. **Editor:** Create a function named **myFunction()** in the `<script>` block of the document's `<head>` section. After entering the **function** keyword and the name of the function, be sure to properly open the function definition with a curly brace.
3. **Editor:** Inside the function, create an alert dialog box. For the alert's message, use: ***The HTML page has loaded.***
4. **Editor:** You will now add an `onload` event handler to the document's `<body>` tag. The `onload` event handler will be used to call the function that you just created. Locate the `<body>` tag near the top of the document. Modify the `<body>` tag as indicated in bold:

```
<body onload="myFunction();">
```

5. **Editor:** Save **lab3-1.htm**.
6. **Browser:** Open **lab3-1.htm**. Your screen should resemble Figure 3-1.

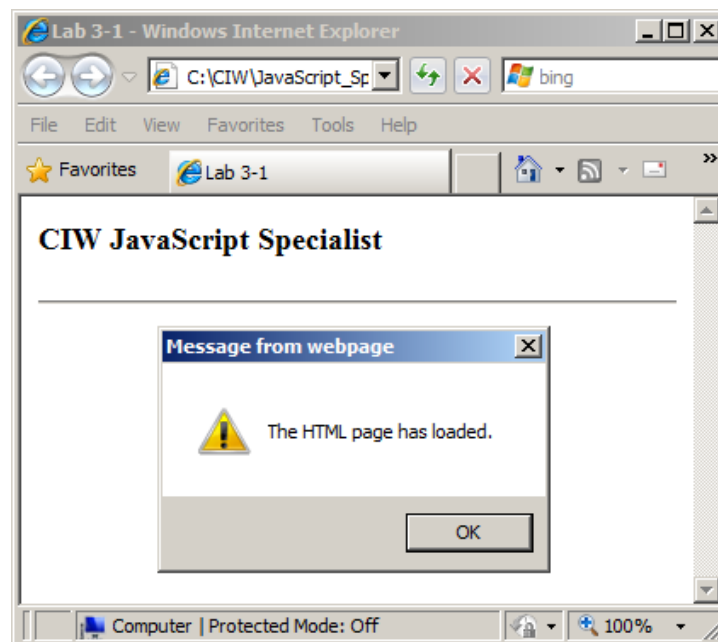


Figure 3-1: Page for lab3-1.htm with alert

7. **Browser:** After clicking **OK**, your screen should render the CIW JavaScript Specialist XHTML page seen behind the alert box in the preceding figure.

In this lab, you successfully created a simple function that displayed an alert dialog box. In addition, you launched your function using the `onload` event handler and inline scripting.



Lab 3-2: Using functions, arguments and return values in JavaScript

In this lab, you will create a simple function, pass arguments to that function when it is called, and return a value to the calling statement.

1. **Editor:** Open **lab3-2.htm** from the Lesson 3 folder of the Student_Files directory.
2. **Editor:** A simple function has been created for you. The lab3-2.htm file contains a form button. Add an **onclick** event handler to the **<input>** tag. Use this event handler to invoke **myFunction()**.
3. **Editor:** Save **lab3-2.htm**.
4. **Browser:** Open **lab3-2.htm**. Your screen should resemble Figure 3-2.

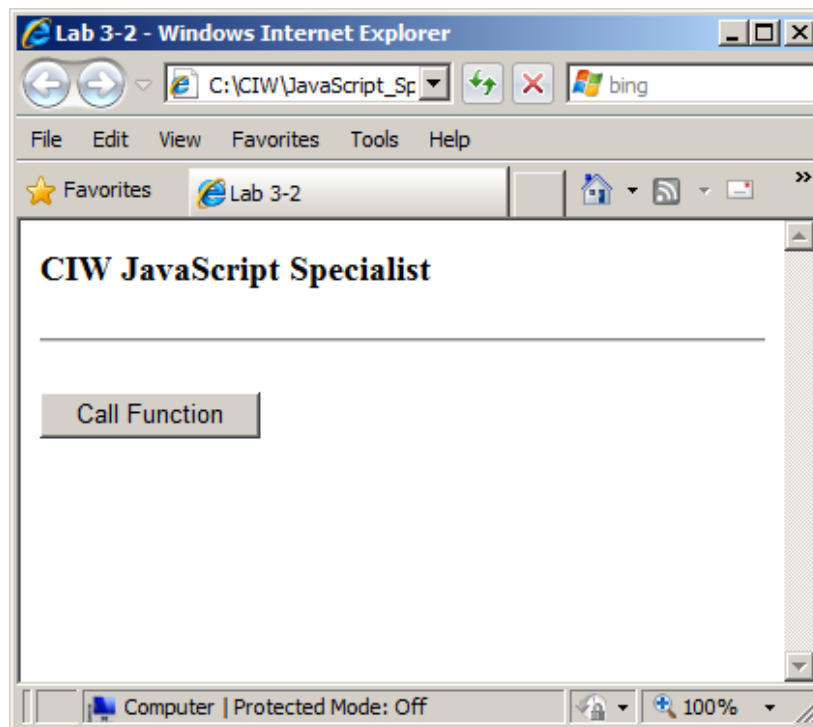


Figure 3-2: Page for lab3-2.htm

5. **Browser:** Click **Call Function**. You should see an alert as shown in Figure 3-3.

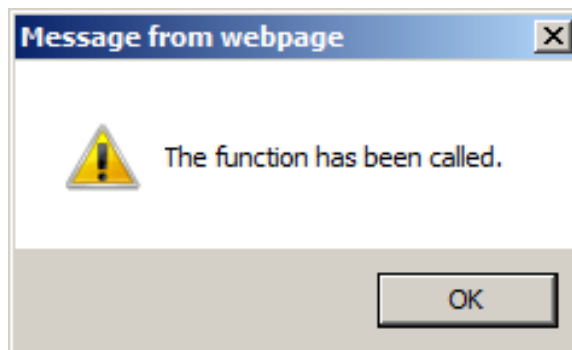


Figure 3-3: Result of function call

6. **Browser:** Click **OK**.
7. **Editor:** Open **lab3-2a.htm**. You will now add an argument to a function call, then use that argument in the function. Locate the `<input>` tag in the XHTML form. Add this string:

"a string of text"

in the parentheses after the function name in the `onclick` expression. You are passing literal text to the function, so be sure to include the quotation marks around the string.

8. **Editor:** Add an argument named **arg1** in the parentheses after the function name. Concatenate **arg1** into the argument of the **alert()** method. Save **lab3-2a.htm**.
9. **Browser:** Open **lab3-2a.htm**. Your screen should resemble Figure 3-2. Click **Call Function**. You should see an alert as shown in Figure 3-4.

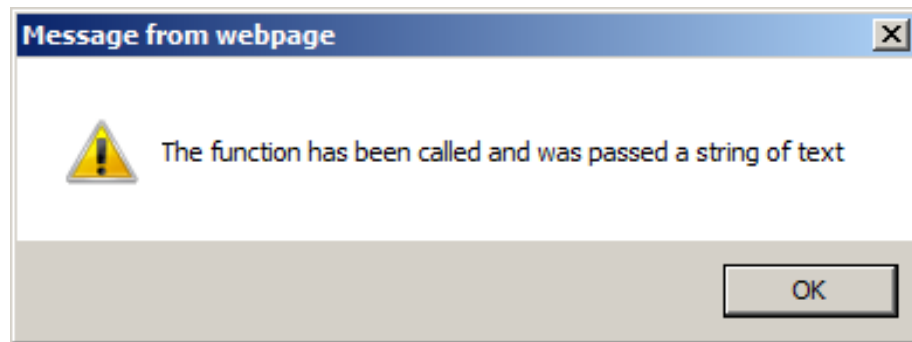


Figure 3-4: Result of function call

10. **Browser:** Click **OK**.
11. **Editor:** Open **lab3-2b.htm**. Add a `return` statement to `myFunction()` after the `alert()` statement. Add this text:

"myFunction() return value"

after the `return` keyword.
12. **Editor:** In order to see the return value from the function, you will add an `alert()` method to the calling statement. Locate the `onclick` event handler defined in the `<input>` tag near the bottom of the file. Wrap an `alert()` method around the call to `myFunction()`. Save **lab3-2b.htm**.
13. **Browser:** Open **lab3-2b.htm**. Your screen should resemble Figure 3-2. Click **Call Function**. You should see an alert as shown in Figure 3-4. Click **OK**. You should then see an alert as shown in Figure 3-5.

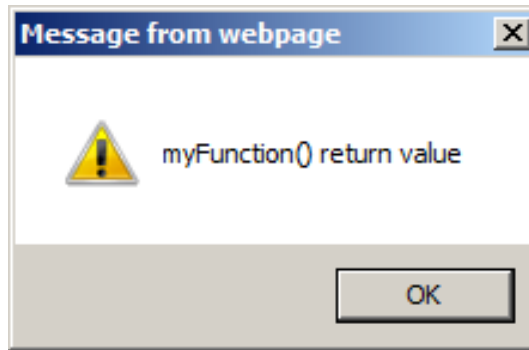


Figure 3-5: Return value from myFunction()

In this lab, you created a function that received an argument. When you created the calling statement, you passed a value to the function as an argument. You then returned a value to the calling statement. When a value is passed into a function and changed, the value of the original data is not changed. In programming, this mode is known as **pass by value**. The exception to this rule is if an object reference is passed as an argument to a function. JavaScript uses **pass by reference** for object arguments. For more information about pass by reference, see the related appendix.



Lab 3-3: Calling a function from within another function in JavaScript

In this lab, you will use one function to call another. It will also pass a value to that function, and then return a value to the calling statement.

- 1. Editor:** Open **lab3-3.htm** from the Lesson 3 folder of the Student_Files directory.
- 2. Editor:** Examine the following source code:

```
<script type="text/javascript"><!--
var numCorrect = 0;

function takeTest() {
var response = "";
var points = 0;

var q1 = "What company developed JavaScript?";
var a1 = "NETSCAPE";

var q2 = "Using JavaScript operator precedence,\n what is the
result of the following expression? 2 + 4 * 6";
var a2 = 26;

var q3 = "With what object-oriented programming language\n is JavaScript often
compared and confused?";
var a3 = "JAVA";

response = prompt(q1,"");
if (response) points= runningTotal((response.toUpperCase() == a1) ? 1 : 0);
alert(points);

response = prompt(q2,"");
if(response) points= runningTotal((response == a2) ? 1 : 0);
alert(points);

response = prompt(q3,"");
if (response) points=runningTotal((response.toUpperCase() == a3) ? 1 : 0);
alert("You answered a total of " + points + " correctly.");

numCorrect = 0;
points = 0;

}

function runningTotal(i) {
    numCorrect += i;
    return numCorrect;
}

// -->
</script>
```

- 3. Editor:** Locate the `<form>` tag near the bottom of the document. Examine the `<input>` tag. Note especially the code indicated in bold:

```
<form>
<input type="button" value="take quiz" onclick="takeTest();">
</form>
```

- 4. Editor:** Close **lab3-3.htm**.

5. **Browser:** Open **lab3-3.htm**. Your screen should resemble Figure 3-6.

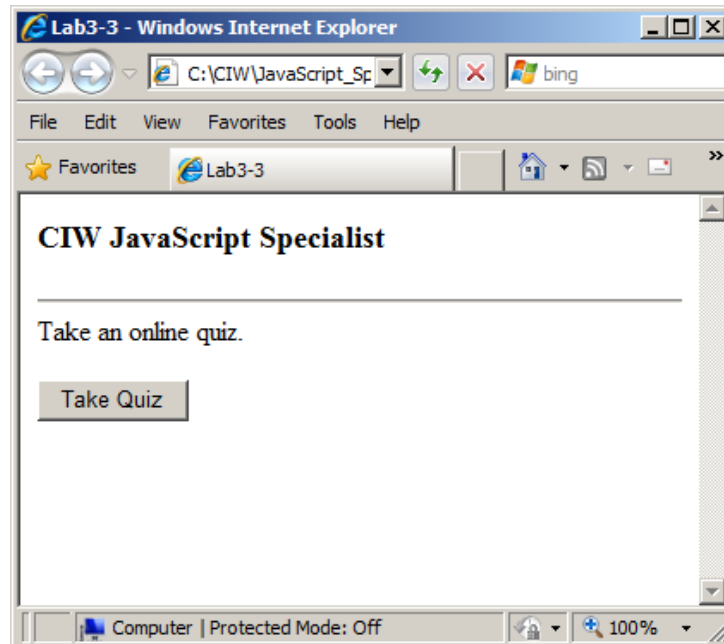


Figure 3-6: Page for lab3-3.htm

6. **Browser:** Click the **Take Quiz** button. Respond to each of the prompts. Enter both correct and incorrect answers to check the logic of the quiz.



Lab 4-1: Using *if* statements

In this lab, you will create an `if` statement. The `if` statement will test an integer for a particular range of values. An alert dialog box will provide pertinent information if the integer falls within this range.

1. **Editor:** Open **lab4-1.htm** from the Lesson 4 folder of the Student_Files directory.
2. **Editor:** A function named `checkgrade()` has been started for you. A variable named `myGrade` is assigned an integer value received from a `select` object in an XHTML form.
3. **Editor:** Following the `myGrade` initialization statement, create an `if` statement. As the condition for the `if` statement, test the `myGrade` variable for a value greater than or equal to 91. If the value is greater than or equal to 91, output an alert that informs the user that his or her grade is an A.
4. **Editor:** Save **lab4-1.htm**.
5. **Browser:** Open **lab4-1.htm**. Your screen should resemble Figure 4-1.

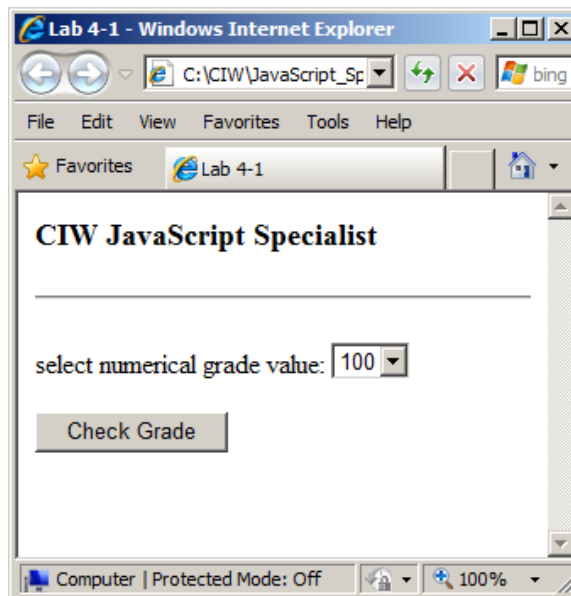


Figure 4-1: Page for lab4-1.htm

6. **Browser:** Select a numerical grade value of 91 or greater from the drop-down menu. You should see an alert resembling Figure 4-2.

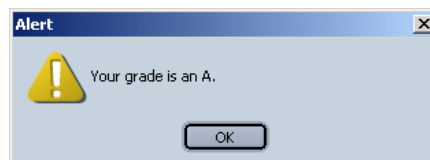


Figure 4-2: Alert dialog box

7. **Browser:** Click **OK** to close the alert.

8. **Editor:** Add an `else if` clause to the `if` statement. As the condition for the `else if` clause, test the `myGrade` variable for a value greater than or equal to 81 and less than or equal to 90. Hint: Use the logical AND operator (`&&`) to perform this test. If the `myGrade` variable falls within this range, output an alert dialog box informing the user that his or her grade is a B.
9. **Editor:** Save `lab4-1.htm`.
10. **Browser:** Refresh `lab4-1.htm`. Select a numerical grade value between 81 and 90 from the drop-down menu. You should see an alert resembling Figure 4-3.

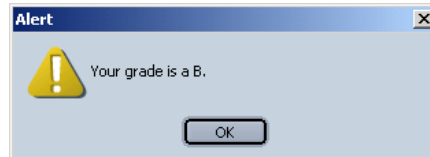


Figure 4-3: Alert dialog box

11. **Browser:** Click **OK** to close the alert.
12. **Editor:** You could continue adding `else if` clauses to test, then map all possible ranges of integer values to a letter grade. However, for now, add an `else` clause to the `if` statement. The `else` clause will simply output an alert dialog box informing the user that his or her grade is a C or lower.
13. **Editor:** Save `lab4-1.htm`.
14. **Browser:** Refresh `lab4-1.htm`. Select a numerical grade value of 80 from the drop-down menu. You should see an alert resembling Figure 4-4.

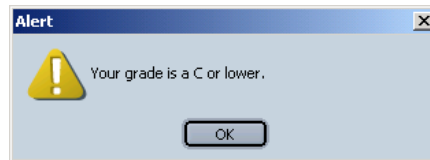


Figure 4-4: Alert dialog box

15. **Browser:** Click **OK** to close the alert.

This lab provided an opportunity to create and use an `if...else if...else` statement. You tested an integer's value and displayed an appropriate message when the value fell into a particular range.



Lab 4-2: Using a *while* statement

In this lab, you will use a `while` loop to test the user's input.

1. **Editor:** Open **lab4-2.htm** from the Lesson 4 folder of the Student_Files directory.
2. **Editor:** Locate the `checkGrade()` function in the `<head>` section of the document. This function contains essentially the same code as `lab4-1.htm`. However, the user enters a numerical grade value in a text box instead of selecting one from a drop-down menu.
3. **Editor:** Create a `while` loop after the `myGrade` initialization statement. In the condition for the loop, use the `isNaN()` function to test the `myGrade` variable. In pseudo-code, the first line of the `while` statement should read: *while is not a number (myGrade)*.
4. **Editor:** One line of code is needed inside the loop. If `myGrade` is not a number, reassign `myGrade` the return value of a prompt dialog box that asks the user to input a numerical value. Do not forget to use the `parseInt()` function to convert the string value returned by the prompt to an integer. The `while` loop will continue until the user enters a number.
5. **Editor:** Save **lab4-2.htm**.
6. **Browser:** Open **lab4-2.htm**. Your screen should resemble Figure 4-5.

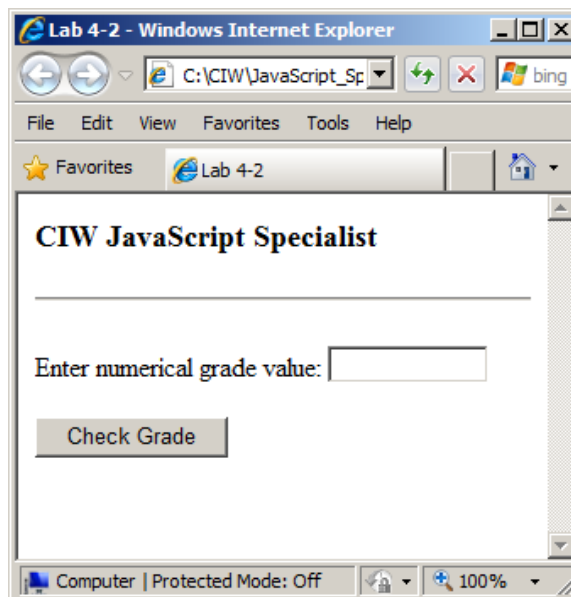


Figure 4-5: Page for `lab4-2.htm`

7. **Browser:** Entering a numerical value between 0 and 100 in the text box and clicking the Check Grade button will result in the same output as `lab4-1.htm`. Enter a non-numerical value in the text box and click the **Check Grade** button. You should see a prompt similar to Figure 4-6.

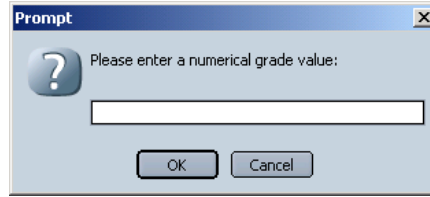


Figure 4-6: Prompt dialog box

- 8. Browser:** Click **OK** or **Cancel** without entering any information, or enter a non-numerical value. You should see the prompt again. In fact, the prompt dialog box should repeatedly appear until a numerical value is entered. After a numerical value is entered, you should see the same output as produced by lab4-1.htm.

This lab is intended to demonstrate a `while` loop, and you have seen the effect of using a loop to repeatedly ask for user input. However, good JavaScript programming practices suggest that you do not prevent a user from canceling a prompt dialog box as shown in lab4-2.htm. A well-designed program would ask for user input a predetermined number of times. If the user repeatedly clicks Cancel, at some point the program should exit. The next portion of this lab will demonstrate this concept.

- 9. Editor:** Open **lab4-2a.htm**. This is the same program as lab4-2.htm with a minor improvement. A variable named `attempts` has been declared and initialized with a value of 1.
- 10. Editor:** In the conditional expression for the `while` loop, also test the `attempts` variable for a value less than or equal to 2. Use the logical AND operator to perform this task.
- 11. Editor:** Add a statement inside the loop that increments the `attempts` variable each time through the loop.
- 12. Editor:** Note that an additional statement has been provided for you after the `while` loop. An `if` statement tests the `myGrade` and `attempts` variables. If `myGrade` is still not a number, and the `attempts` variable equals 3, a `return` statement is used to exit the function.
- 13. Editor:** Save **lab4-2a.htm**.
- 14. Browser:** Open **lab4-2a.htm**. Experiment with the page. If the user does not enter the required data in two attempts, the prompt dialog boxes should disappear with no further output from the program.



Lab 4-3: Using a for statement

In this lab, you will use a `for` statement to output XHTML and the value of the loop counter variable. You will use a `for` loop to create a drop-down menu that contains the values 100 decreasing to zero.

- 1. Editor:** Open **lab4-3.htm** from the Lesson 4 folder of the Student_Files directory.
- 2. Editor:** This file is essentially the same as lab4-1.htm. Locate the second `<script>` block in the `<body>` section of the document. Create a `for` statement. Create a variable named `i` for the loop counter variable. Assign an initial value of 100 to `i`. The loop will continue as long as `i` is greater than or equal to zero. Use the decrement operator (`--`) to decrement `i` by one each time through the loop.
- 3. Editor:** Inside the loop, use a `document.write()` statement to output an `<option>` tag. Concatenate the value of `i` into the expression. The value of the loop counter variable will supply the text for the drop-down menu.
- 4. Editor:** Save **lab4-3.htm**.
- 5. Browser:** Open **lab4-3.htm**. Your screen should resemble Figure 4-1. Click on the drop-down menu to open it, and scroll through the items in the drop-down menu. Your screen should resemble Figure 4-11.

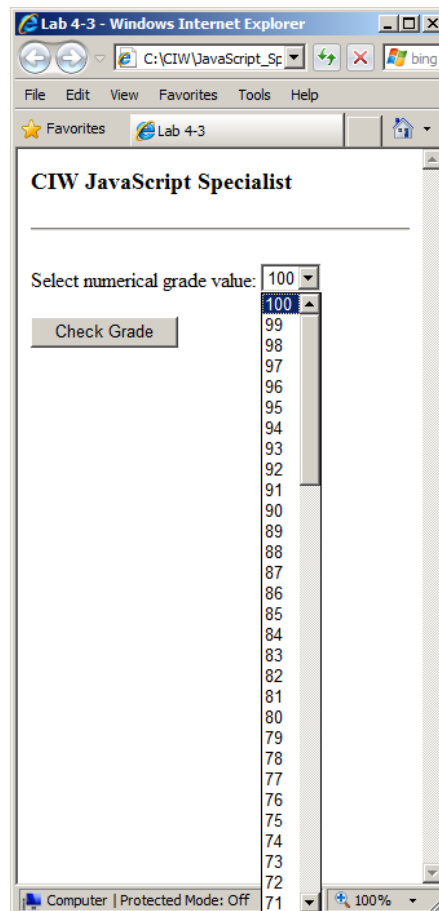



Figure 4-11: Page for lab4-3.htm

- 6. Browser:** The program should respond in a manner similar to lab4-1.htm when a numerical grade value is selected from the drop-down menu.

This lab demonstrated the ease with which a few lines of JavaScript code generated 100 lines of XHTML code. The `for` statement can be used to automate many tasks in many different situations.





Lab 4-4: Nesting *if* and *break* statements inside a *while* loop

- 1. Editor:** Open the **lab4-4.htm** file from the Lesson 4 folder of the Student_Files directory.
- 2. Editor:** Locate the `<script>` block in the `<head>` section of the document. A function named `breakTest()` has been started for you. Two variables have been declared: `loopBoolean` and `myValue`. The `loopBoolean` variable is set to `true` and is used to determine when the `while` loop will end. The `myValue` variable is assigned an empty string in anticipation of the user's input.
- 3. Editor:** After the variable declarations, a `while` loop is started. After the `myValue` variable receives its value, add an `if` statement that tests `myValue` for a `null` value. If it is `null`, invoke a `break` statement.
- 4. Editor:** Save **lab4-4.htm**.
- 5. Browser:** Open **lab4-4.htm**. Your screen should resemble Figure 4-12.

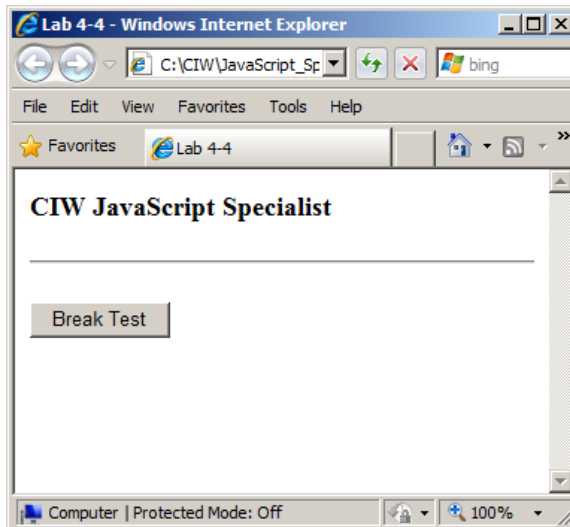


Figure 4-12: Page for lab4-4.htm

- 6. Browser:** Click the **Break Test** button. You should see a prompt dialog box as shown in Figure 4-13.

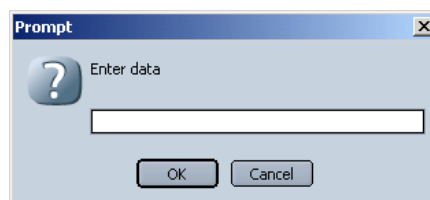


Figure 4-13: Prompt dialog box

- 7. Browser:** Click **OK** without entering any data. The prompt dialog box should reappear whenever this action is taken. Click **Cancel**. The program should recognize the `null` value returned when this action is taken and the `break` statement should end the program.

- 8. Browser:** Any data entered in the prompt dialog box should appear in an alert dialog box.



Lab 4-5: Using a *continue* statement

In this lab, you will use the `continue` statement to manipulate a `for` loop.

- 1. Editor:** Open **lab4-5.htm** from the Lesson 4 folder of the Student_Files directory.
- 2. Editor:** A `for` loop has been defined that outputs the numbers 1 through 100. Before the `document.write()` statement, create an `if` statement that tests the loop counter variable `i`. If `i` divided by 7 returns a remainder, invoke a `continue` statement. The only numbers that should be output are those evenly divisible by 7. Hint: Use the modulus operator (`%`) to perform this task.
- 3. Editor:** Save **lab4-5.htm**.
- 4. Browser:** Open **lab4-5.htm**. Your screen should resemble Figure 4-15.

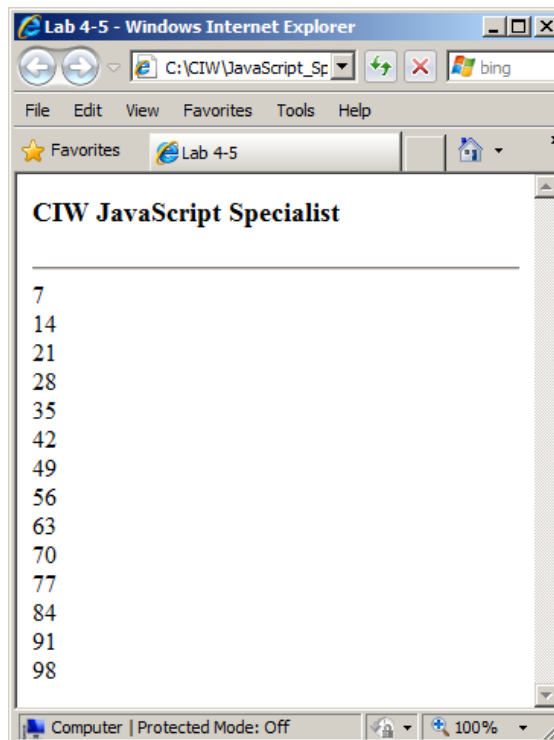


Figure 4-15: Page for lab4-5.htm



Lab 4-6: Using a *switch* statement

In this lab you will use a `switch` statement.

- 1. Editor:** Open **lab4-6.htm** from the Lesson 4 folder of the Student_Files directory.
- 2. Editor:** Study the code in the file, which appears as follows:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Lab 4-6</title>

<script language="JavaScript" type="text/javascript">
function switchTest(SetOption) {
OptionsSelect = document.frmOne.SetOption.value

alert(OptionsSelect);
}
</script>
<body>
<h3>CIW JavaScript Specialist</h3>
<form name = "frmOne">

<select name= "SetOption" onchange="switchTest(this)">
<option value = "0">Select a U.S. City - See the State!</option>
<option value = "1">Phoenix</option>
<option value = "2">Roswell</option>
<option value = "3">Sacramento</option>
<option value = "4">My city</option>
<option value = "5">Your city</option>
</select>

</form>

</body>
</html>
```

- 3. Browser:** Open **lab4-6.htm**. The page should resemble Figure 4-16. You will see that selecting an option automatically displays an alert (without clicking a button) that tells the value of the select statement. Notice the U.S. city choices are assigned values 1, 2 and 3, and notice the additional selections ("my city" and "your city") are defaults that are assigned the values are 4 and 5. We will discuss the default values shortly.

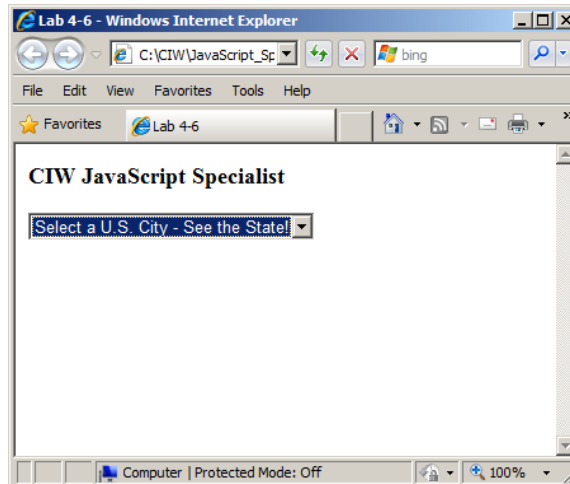


Figure 4-16: Page for lab4-6.htm

4. **Editor:** Open the file **lab4-6.htm**. You will change the code between the `<script></script>` tags as follows. Be sure to overwrite the previous code in the `<script></script>` section so these commands do not duplicate. Change the code as shown in bold, then save the file. (The file **lab4-6_newCode.txt** provides this new code for you to use.)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Lab 4-6</title>
```

```
<script language="JavaScript" type="text/javascript">
function switchTest(SetOption) {
OptionsSelect = document.frmOne.SetOption.value
```

```
alert(OptionsSelect);

switch (OptionsSelect) {
case "1":
alert("Arizona")
break
case "2":
alert("New Mexico")
break
case "3":
alert("California")
break
default:
alert("Cannot be determined")
}
}
</script>
```

```
<body>
<h3>CIW JavaScript Specialist</h3>
<form name = "frmOne">
```

```
<select name= "SetOption" onchange="switchTest(this)">
<option value = "0">Select a City - See the State!</option>
<option value = "1">Phoenix</option>
<option value = "2">Roswell</option>
<option value = "3">Sacramento</option>
<option value = "4">My city</option>
```

```
<option value = "5">Your city</option>
</select>

</form>
</body>
</html>
```

This code demonstrates one of a JavaScript developer's favorite tools. The `select` statement will trigger the `switchtest` function with a `switch` statement. Then the script will react based on the user's selection. This type of script can be written to take the user to a different page, or to show an alert box (as in this example), and often it will be used in a quiz to activate the next question.

Tech Note: Each value must have a corresponding case statement. It is good practice to put in a default case statement that will catch everything outside of the intended results. In this way, all situations are covered, even unexpected results. This example uses "cannot be determined" as the default case statement.

- 5. Editor:** Save **lab4-6.htm**.
- 6. Browser:** Open **lab4-6.htm**. As you select the various options, you will see alert boxes similar to those shown in Figure 4-17.

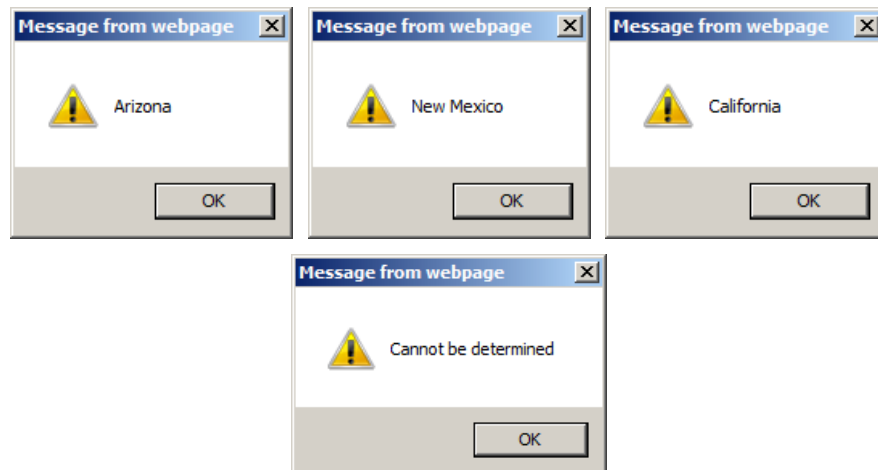


Figure 4-17: Alert boxes

- 7. Browser:** When you select any of the first three options from the list (the U.S. city names), you will see the corresponding alert listing that city's state. This is because the `switch` statement recognizes the number (1, 2 or 3), matches it in the proper case and runs the `alert`, and then the `break` statement stops it from continuing. If the option selected is outside the `switch`'s case statement (0, 4, 5), then it will display the default statement ("Cannot be determined"). Then the `break` statement will stop the script. The `switch` statement will check each case in order and check for a match. If there are no matches, then the last statement (the default statement) will catch the `switch` statement, and the script will not have issues.

Tech Note: The `break` statement will stop the script — not just for `switch` statements like this one, but anywhere in JavaScript. The default case statement will activate only if no other case statement matches the choices. The `switch` statement is one of the easiest ways to utilize multiple options without using nested `if` statements (which is considered messy programming). Typically, if there are more than three choices, the developer should consider using the `switch` statement.



Lab 5-1: Launching a new window with the `open()` method

In this lab, you will open a new window. This lab launches a new instance of the browser that points to another Web site.

- 1. Editor:** Open **lab5-1.htm** from the Lesson 5 folder of the Student_Files directory.
- 2. Editor:** Locate the existing `<script>` tags. A function named `newWindow()` has been started for you. Inside this function, use the `open()` method of the `window` object to open a new window to the CIW Web site, using `http://www.CIWcertified.com` for the URL.
- 3. Editor:** Experiment with the various attributes, such as `toolbar`, `location` and `scrollbars`, in the `open()` method for the new window. For this portion of the lab, omit any height or width attributes to demonstrate the effect when these attributes are not included in the `open()` method's arguments.
- 4. Editor:** A form button is used to call the `newWindow()` function. This button has been scripted for you.
- 5. Editor:** Save **lab5-1.htm**.
- 6. Browser:** Open **lab5-1.htm**. Your screen should resemble Figure 5-2.

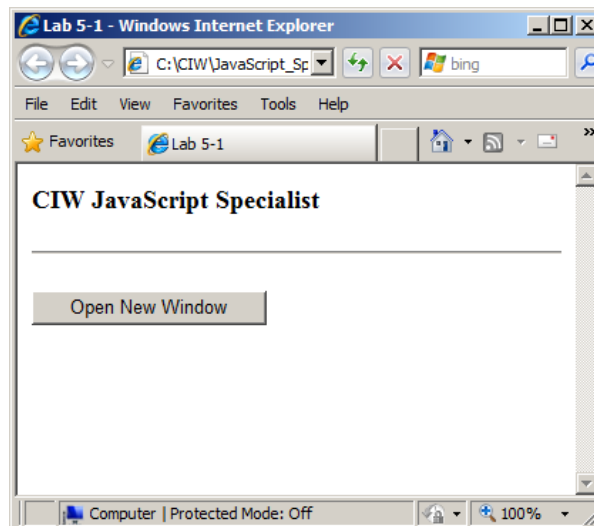


Figure 5-2: Page for lab5-1.htm

- 7. Browser:** Click the **Open New Window** button to launch a new window. Because you did not specify a size, the new window might match the size of the existing window, or it might be a different size. The new window should resemble Figure 5-3, depending on the attributes you used in the `open()` method.

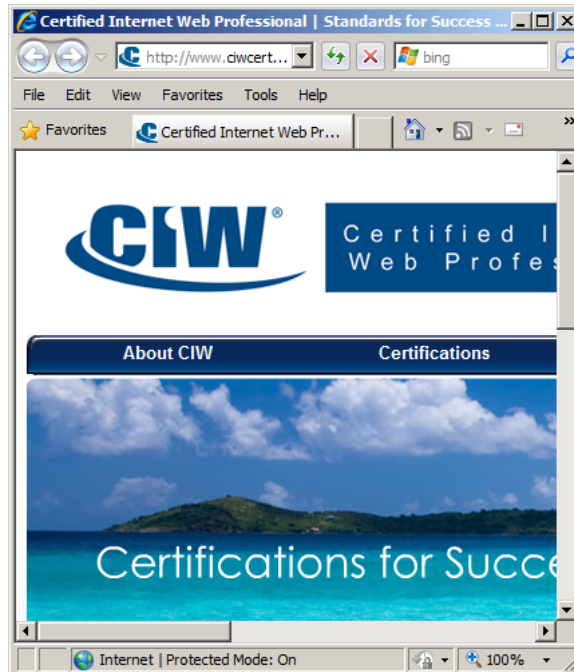


Figure 5-3: CIW Web site

8. **Browser:** Close the CIW window.
9. **Editor:** Add **height** and **width** attributes to the `open()` method. Use **400** for the width and **300** for the height. Save **lab5-1.htm**.
10. **Browser:** Refresh **lab5-1.htm**. Click the **Open New Window** button. The new window should resemble Figure 5-4, depending on its attributes.

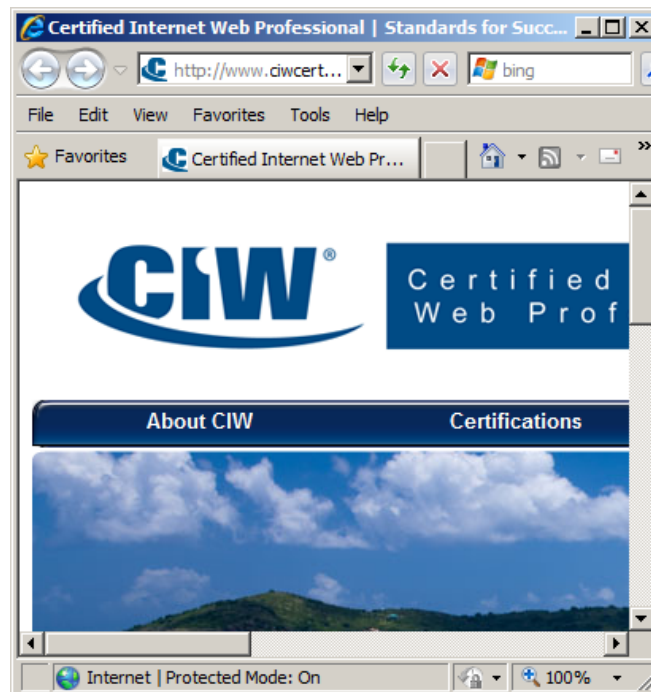
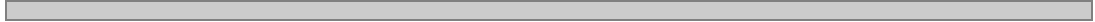


Figure 5-4: New window with height and width attributes

11. Browser: The outer frame of the new window should have a width of 400 pixels and a height of 300 pixels.

Note that browsers will not consistently interpret the height and width attributes. However, if the browser does not create a new window with the exact measurements specified, it will create a new window that is proportional to the dimensions in the attribute list.

In this lab, you used the window object's `open()` method to open a new window. You targeted a specific URL, which filled the new window. The function `newWindow()` was called using the `onClick` event handler of the button object.





Lab 5-2: Writing content to new windows

In this lab, you will open a window, and assign it a name that will be used to refer to its objects and properties. You will then write content to the new window, and learn how to close the window programmatically.

- 1. Editor:** Open **lab5-2.htm** from the Lesson 5 folder of the Student_Files directory.
- 2. Editor:** Locate the existing `<script>` tags. A function named `smallWindow()` has been started for you. Inside the function, create a variable named `myWindow`. Assign to `myWindow` the result of the `open()` method of the window object. Do not assign a URL in the `open()` method's arguments. Insert an empty set of quotation marks in the URL's place. Remember to give the window a name. For the attributes list, assign only a **height** of **100** and a **width** of **100**.
- 3. Editor:** Add the `smallWindow()` function to add content to the new window. Use the reference to the new window held in the `myWindow` variable to write to the new window's document using `document.write()`. In the new window, create `<form>` tags. Within the form, use an `<input>` tag to create a **button** object. Set the button's **value** attribute equal to **Close**. Add an **onclick** event handler to call the `window.close()` method. This will provide a button for the user to close the newly opened window.
- 4. Editor:** In the `lab5-2.htm` file, a form button is used to call the `smallWindow()` function. This button has been scripted for you.
- 5. Editor:** Save **lab5-2.htm**.
- 6. Browser:** Open **lab5-2.htm**. Your screen should resemble Figure 5-2.
- 7. Browser:** Click the **Open New Window** button. You should now see a small window, as shown in Figure 5-5. If not, verify that the source code you entered is correct.

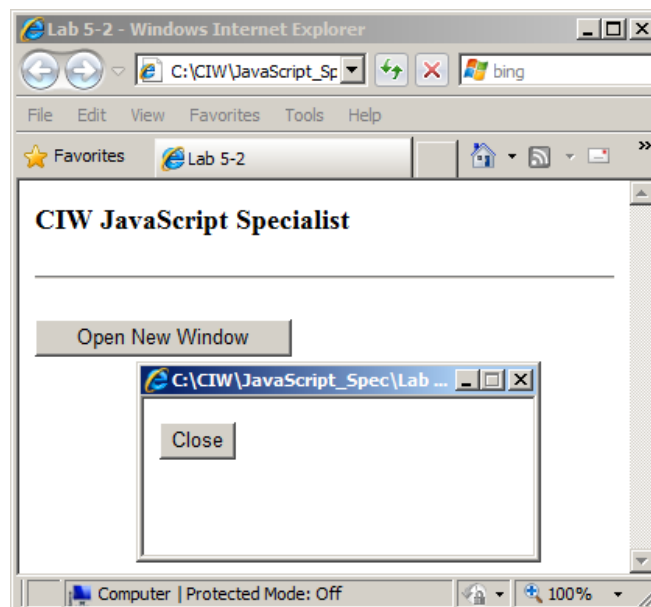


Figure 5-5: Small window opened

- 8. Browser:** Click **Close** in the new window. The new window should close. If it does not, verify that the source code you entered is correct.

Tech Note: The browser may attempt to stop the script, or it may force you to allow ActiveX controls in order to run this script.

If you are encountering errors, first look for misspellings. Next, verify that you are using the proper case for each command. Careless typing in your script can lead to hours of troubleshooting. Get in the habit of checking for misspellings and case-sensitivity before you look for other errors.

In this lab, you learned that you can give a window an identity. In this case, the new window was given the name `myWindow`. Because the window had a name, you could access the window's document, write to the new window, and use the window object's `close()` method to close that particular window. Writing dynamic content to a new window is an important concept, and will be explored further in this lesson.



Lab 5-3: Changing status bar text

In this lab, you will change the status bar text using the `onmouseover` event handler for the `<a>` tag. You will then change the status bar text to an empty string (or its default state, depending on the browser) using the `onmouseout` event handler.

1. **Editor:** Open **lab5-3.htm** from the Lesson 5 folder of the Student_Files directory.
2. **Editor:** Locate the `<a href>` that is defined before the `` tag. Inside the anchor tag, add an **onmouseover** event handler that changes the status bar text to the following:

Visit the CIW Web site!

Also, add an **onmouseout** event handler that changes the status text bar to an empty string.

3. **Editor:** Save **lab5-3.htm**.
4. **Internet Explorer Browser:** Open **lab5-3.htm** in Internet Explorer. Your screen should resemble Figure 5-7.

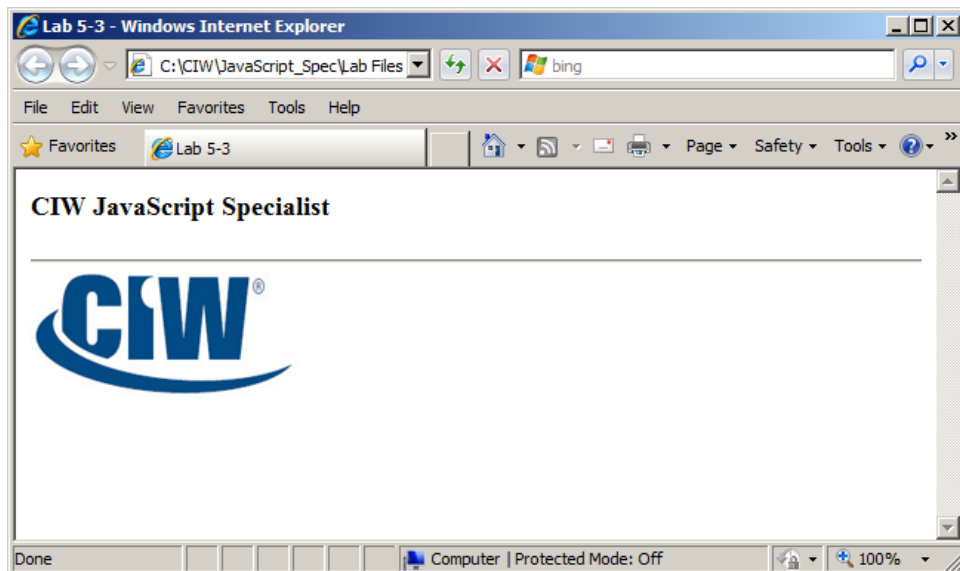


Figure 5-7: Page for lab5-3.htm

5. **Internet Explorer Browser:** Move the mouse pointer over the CIW image. The message you created should appear in the status bar as shown in Figure 5-8. If it does not, verify that the source code you entered is correct.

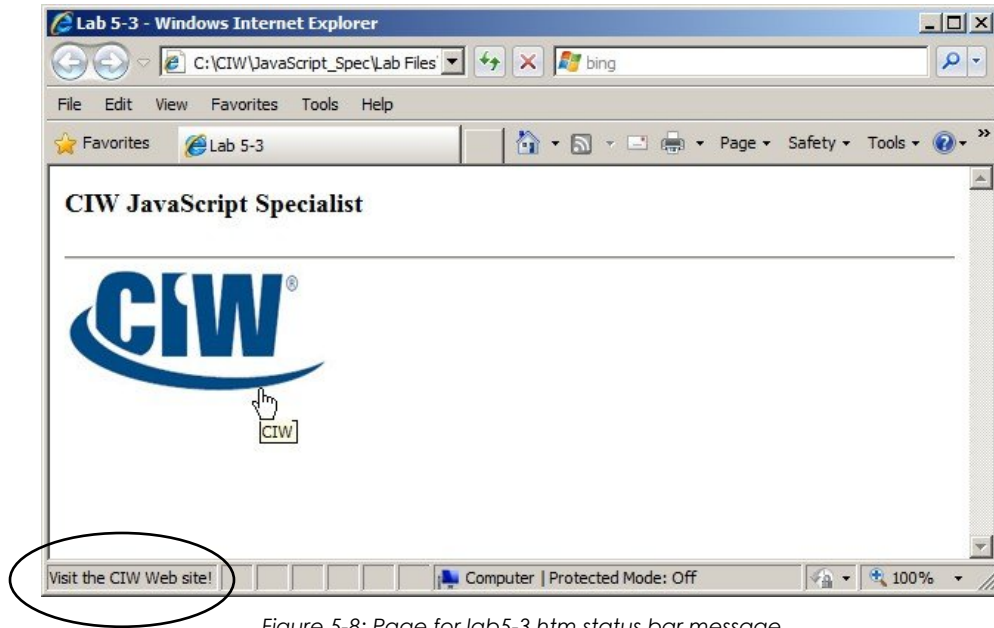


Figure 5-8: Page for lab5-3.htm status bar message

- 6. Firefox Browser:** Open **lab5-3.htm** in Firefox. Move the mouse pointer over the CIW image. Do you see your status bar message?
- 7. Firefox Browser:** You should see text in the status bar, but it is not the same message you saw in Internet Explorer. In Firefox, you should see only the URL for the CIW site, which is the target of the link. Why does this happen?

For security reasons, some browsers are beginning to disallow any change in the status bar message to ensure that you know exactly where a link is pointing before you click it. This helps prevent phishing and pharming attacks, in which a look-alike site impersonates a trusted site, such as a bank or online payment service. The imposter site can use a status bar message to mask a link's URL so you will click it, because you may not be able to detect the imposter once at the site. The imposter then gains your trust, and you may enter sensitive information, such as a password or credit card number. For this reason, the `status` property may eventually be deprecated.

In this lab, you learned to manipulate the `status` property of the `window` object to change the text in the window's status bar. Now that you have worked with the `window` object, you will learn about the `document` object.



Lab 5-4: Assigning properties to a remote document dynamically

In this lab, you will see how to create a remote document and write dynamic content to that document. The user will be asked several questions about look and feel, as well as the content intended for the new document. After the user supplies the information, the dynamic content is written to the remote document from the script contained in the lab file lab5-4.htm.

- Editor:** Open **lab5-4.htm** from the Lesson 5 folder of the Student_Files directory.
- Editor:** Examine the `writeToDocument()` function:

```
<script language="JavaScript" type="text/javascript">
<!--
function writeToDocument(){
    var textColor, backColor, pageTitle;
    var yourText, pageContent, docWindow;

    textColor =prompt("Please enter a text color:", "white");
    backColor =prompt("Please enter a background color:", "black");
    pageTitle =prompt("The page will be titled: ",
        "Default Title");
    yourText = prompt("Add content to the new document:",
        "page content");

    pageContent = "<html><head><title>";
    pageContent += pageTitle + "</title>";
    pageContent += "<script>alert('The page ' +
        document.title + ' was created: ' +
        + document.lastModified);</script>";
    pageContent += "</head><body " + "bgcolor=" + backColor + "><strong>" +
        "<font color= " + textColor + ">" + yourText + "</font>";
    pageContent += "</strong></body></html>";

    docWindow = open("", "docWin", "width=250,height=150,
        resizable=1,status=1");

    // Create a with statement

    }
//-->
</script>
</head>
```

- Editor:** Note that several variables have been created. The variables `textColor`, `backColor`, `pageTitle` and `yourText` are all assigned the result of user input via `prompt()` methods. The `pageContent` variable is assigned a combination of XHTML, JavaScript code, document object properties and variable values. The `docWindow` variable receives a reference to a new window via the `window.open()` method.
- Editor:** Locate the comment that reads **//Create a with statement**. Modify the `writeToDocument()` function to add a **with** statement. The **with** statement should use the `docWindow` document as its target object. Be sure to open the XHTML data stream to the new document. Add code that will set the **bgColor** and the **fgColor**. For these, use the values in the `backColor` and `textColor` variables. Use the `write()` method to output the `pageContent` variable. Make sure to close the XHTML data stream.
- Editor:** Examine the rest of the code on the page. Save **lab5-4.htm**.

6. **Browser:** Open **lab5-4.htm** in Internet Explorer. Your screen should resemble Figure 5-10.

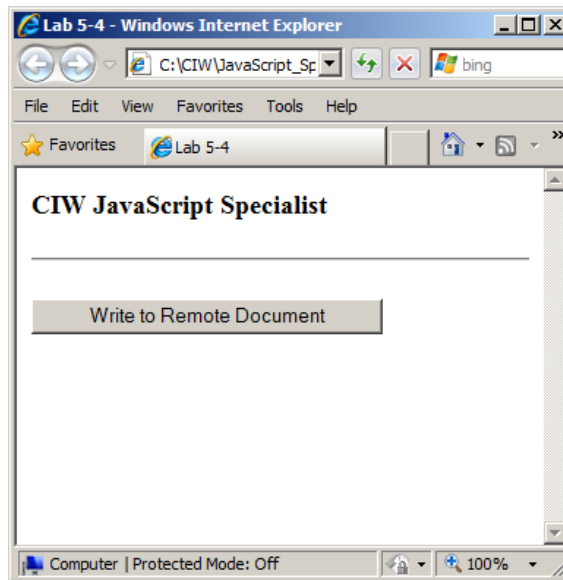


Figure 5-10: Page for lab5-4.htm

7. **Browser:** Click the **Write To Remote Document** button. This button has been scripted to call the `writeToDocument()` function. You will see a series of prompt dialog boxes, which will each appear as you enter data into the previous one (see Figure 5-11).

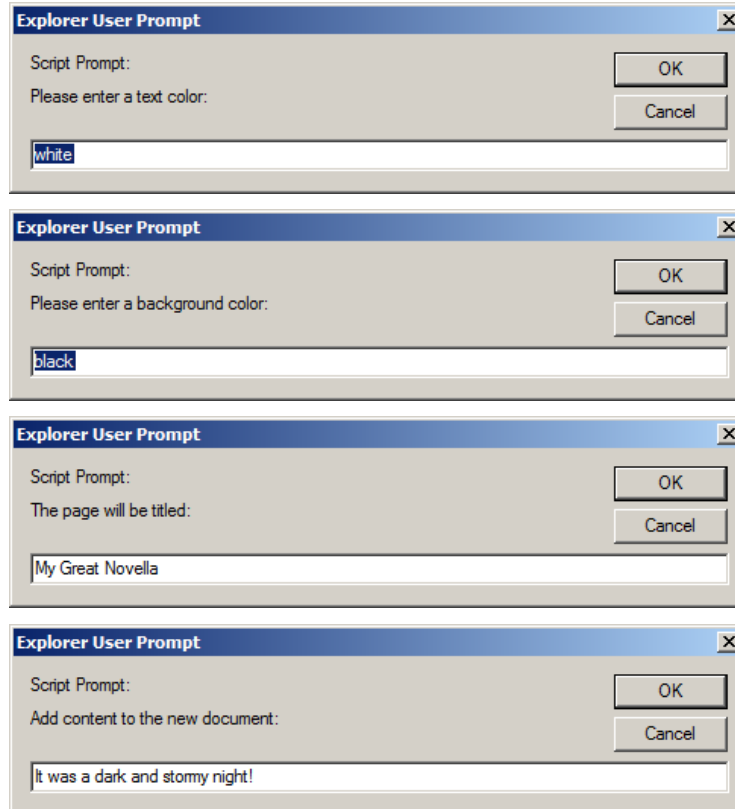


Figure 5-11: Prompt dialog boxes

8. **Browser:** After entering data into all dialog boxes and clicking **OK**, you should see an alert dialog box similar to that shown in Figure 5-12.

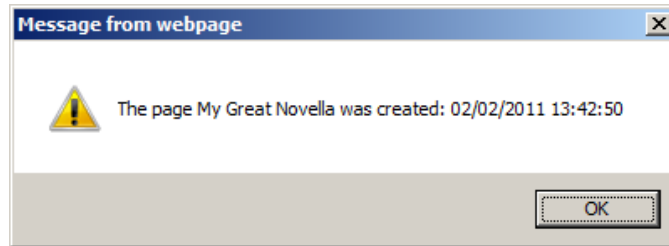


Figure 5-12: Alert dialog box

9. **Browser:** Click **OK**. Your screen should resemble Figure 5-13, depending on your input.

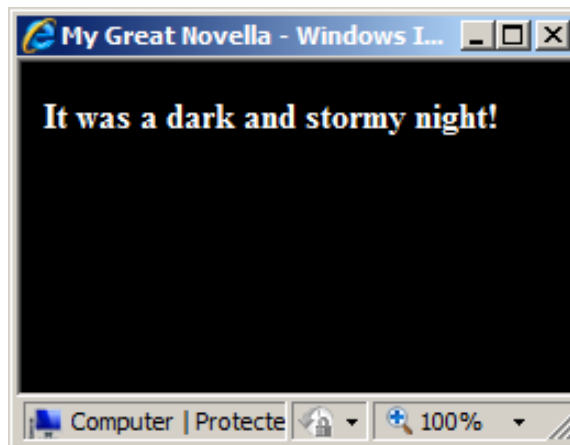


Figure 5-13: New window

10. **Browser:** If time permits, try running this script in another browser, such as Firefox. How do the results differ? Remember that perfectly valid script often renders differently in different browsers and browser versions. Be sure to always test your code in as many browsers as possible.

This lab demonstrates that your scripts can reflect your X/HTML proficiency. The more sophisticated your knowledge of X/HTML, the more creative you can be when designing dynamically generated Web pages.



Lab 5-5: Preloading and swapping images to create an active link

In this lab, you will use the `image` object to replace a default image with another, using the `onmouseover` and `onmouseout` event handlers discussed elsewhere in the course. You will add code to preload two images for the XHTML document. These images will be used to create rollover effects with each image used as a link on the page. The navigational button has two images associated with it: one for when the user hovers the mouse pointer over the image (on position), and one for when the user moves the mouse pointer away from the image (off position). The `src` properties will be assigned values based on the two image files provided in the images directory of the Lesson 5 folder of the Student_Files directory: `images/ciw_on.gif` and `images/ciw_off.gif`.

- 1. Editor:** Open the `lab5-5.htm` file from the Lesson 5 folder of the Student_Files directory.
- 2. Editor:** Locate the `<script>` tag in the document's `<head>` section. Locate the comment that reads as follows:

```
//Add image preloading code
```

- 3. Editor:** Add code to ensure that the target browser supports the image object. Then add code that will preload the images provided. Create four new `Image()` objects using variables named:

- `ciw_on`
- `ciw_off`
- `ciw1_on`
- `ciw1_off`

Assign the `src` properties as follows:

- Assign `images/ciw_on.gif` for `ciw_on`
- Assign `images/ciw_off.gif` for `ciw_off`.
- Assign `images/ciw1_on.gif` for `ciw1_on`
- Assign `images/ciw1_off.gif` for `ciw1_off`

- 4. Editor:** Your code should resemble the following (before adding the preloading code):

```
<script type="text/javascript">
<!--

    //Add image preloading code

    function imageOn(imageName){
        if (document.images){
            (imageName=="ciw") ? document.images[0].src ↴
                = ciw_on.src : "";
            (imageName=="ciw1") ? document.images[1].src ↴
                = ciw1_on.src : "";
        }
    }

    function imageOff(imageName){
        if (document.images){
```

```

        (imageName=="ciw") ? document.images[0].src
                          = ciw_off.src : "";
        (imageName=="ciw1") ? document.images[1].src
                          = ciw1_off.src : "";
    }
}
//-->
</script>

```

5. **Editor:** Examine the `imageOn()` and `imageOff()` functions that have been provided for you. These functions will be discussed later in this lesson.
6. **Editor:** Scroll to the bottom of the document and locate the `` tag. Note that the `name` attribute inside the `` tag is set to "ciw".
7. **Editor:** Examine the `<a>` tag that encloses the `` tag. An **onmouseover** event handler has been scripted to call the **imageOn()** function. When the function is called, the value of the `name` attribute for the `` tag is passed as an argument.
8. **Editor:** Still inside the `<a>` tag, an **onmouseout** event handler has been scripted to call the **imageOff()** function. When the function is called, the value of the `name` attribute for the `` tag is passed as an argument.
9. **Editor:** The `` tag and its associated `<a>` tag are constructed in the same manner.
10. **Editor:** Save **lab5-5.htm**.
11. **Browser:** Open **lab5-5.htm**. Your screen should resemble Figure 5-14.

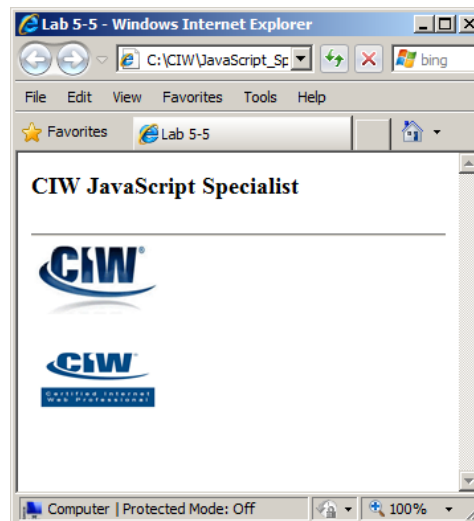


Figure 5-14: Page for lab5-5.htm

12. **Browser:** Move your cursor over the **CIW** image. Your screen should resemble Figure 5-15. If it does not, verify that the source code you entered is correct.

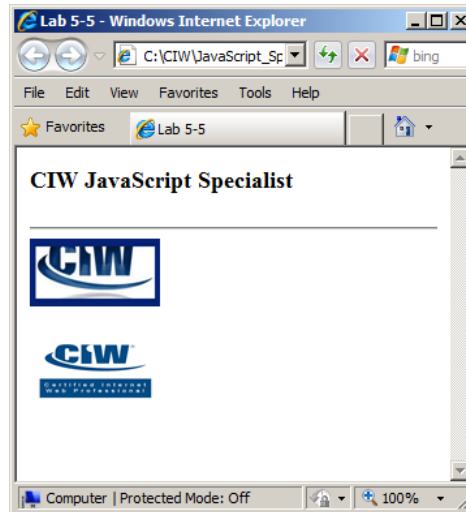


Figure 5-15: Page for lab5-5.htm with image swap

- 13. Browser:** You should see the CIW image change when the mouse pointer hovers over it, then change back to the original image when the mouse pointer is moved away. The second image on the page should function in the same manner.
- 14. Editor:** If time permits, alter your code to change the `status` property of the `window` object so that it indicates the page to which the link will take the user, as shown in the preceding figure.

In this lab, you used the `image` object to create a rollover effect for navigational images on an XHTML page.



Lab 5-6: Identifying browser properties with the navigator object

In this lab, you will learn about the various types of information you can detect from several `navigator` object properties.

- Editor:** Open the file **lab5-6.htm** from the Lesson 5 folder of the Student_Files directory.
- Editor:** Locate the `<script>` block in the `<head>` section of the file, as shown in bold. Add code to the `openWindow()` function. Concatenate the `appCodeName`, `appName`, `appVersion` and `userAgent` properties of the `navigator` object into the `info` variable. The following code shows this function before your changes:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head><title>Lab 5-7</title>

<script language="JavaScript" type="text/javascript">
<!--

function openWindow() {
  var info="";

  info += "Welcome, " +           ; //Add appCodeName here
  info += " user!\nYou are using the ";
  info +=      + " browser,\nversion "; //Add appName here
  info +=      + ".\nYour user agent "; //Add appVersion here
  info += "information is " +     ; //Add userAgent here

  alert(info);
}
//-->
</script>
</head>

<body>
<h3>CIW JavaScript Specialist</h3>
<hr />

<script type="text/javascript">

    //Add userAgent here to write into browser window

</script>

<form name="myForm">
<input type="button" name="buttonClick" value="Click for More Browser
Information" onclick="openWindow()" />
</form>
</body>
</html>

```

- Editor:** Locate the `<script>` block in the `<body>` section of the file. Add code to write the information from the `userAgent` property of the `navigator` object into the browser window, as shown in bold:

```

<script type="text/javascript">
document.write(
"<p>The property <strong><em>navigator.userAgent</em></strong> "
  + "returns:<br /><strong><font color='#FF0000'>"
  + navigator.userAgent + "</font></strong></p>"
)
</script>

```

- 4. Editor:** Save **lab5-6.htm**.
- 5. Browser:** Open **lab5-6.htm** in the Firefox browser. Your screen should resemble Figure 5-16.

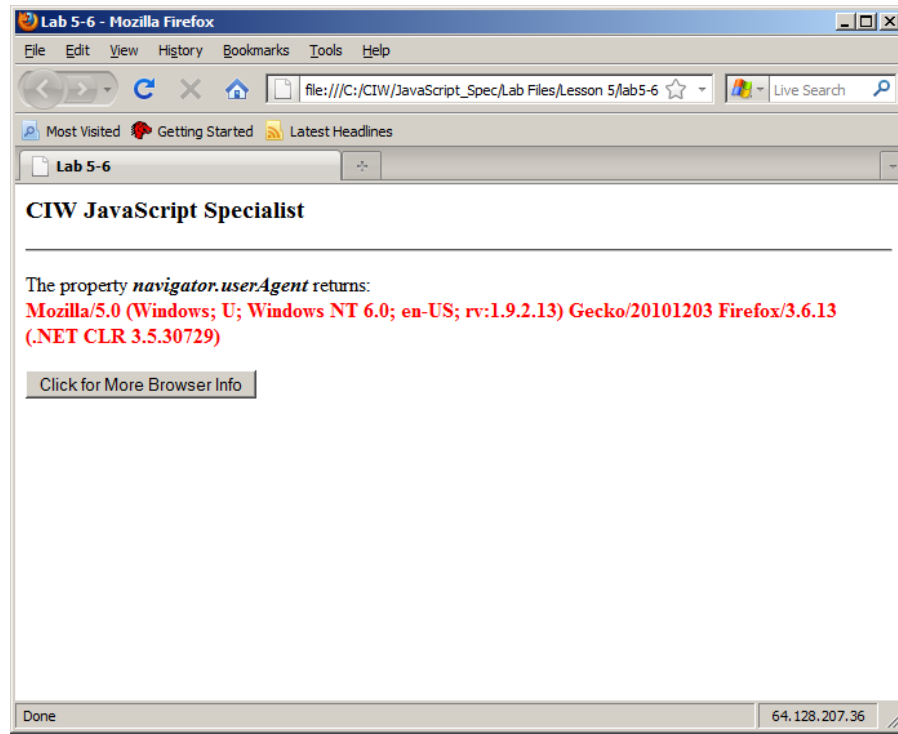


Figure 5-16: Page for lab5-6.htm

- 6. Browser:** Click the **Click For More Browser Info** button. You should see an alert similar to Figure 5-17.

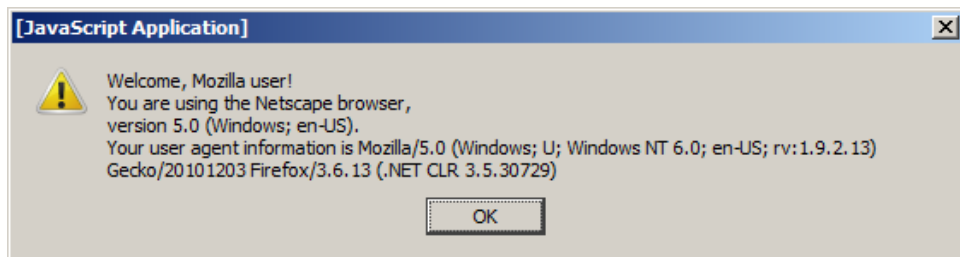


Figure 5-17 Alert in Mozilla Firefox 3.5

- 7. Browser:** Now open the file **lab5-6.htm** in the Internet Explorer browser. You should see an alert similar to Figure 5-18.

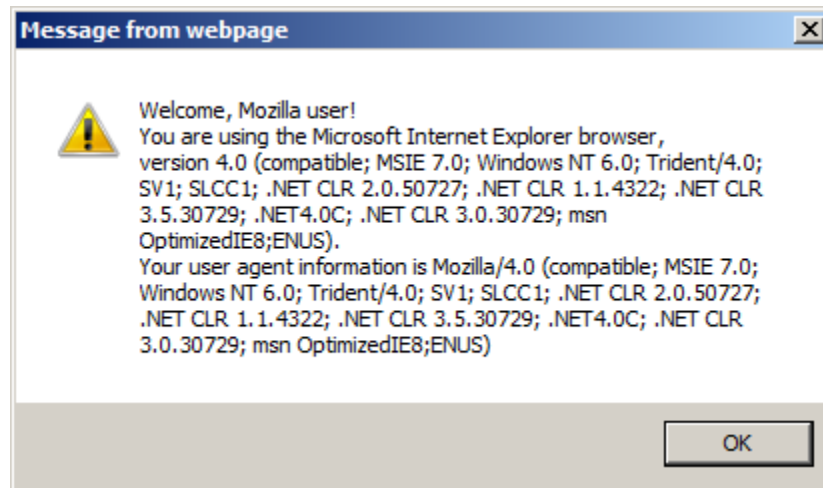


Figure 5-18: Alert in Microsoft Internet Explorer 7.0

- 8. Browser:** Notice that for both browsers, the `appName` property returns Mozilla as the browser's code name. As time permits, try opening this page in other browsers and versions to see how the browser information differs.

Tech Note: The reason that Internet Explorer returns Mozilla for `appName` is that many years ago, `appName` was used by Web developers to allow only the Netscape (Mozilla) browser to access frames-based pages, because other browsers could not render them. Many developers continued to exclude non-Mozilla browsers in this way when Netscape was dominating the market. To thwart this exclusion, Microsoft began giving its Internet Explorer browsers a Mozilla `appName` as well. This trend has persisted among many browsers, and even today the `appName` will return Mozilla for all Mozilla, Internet Explorer and Google Chrome browsers. Therefore, be sure to use the `appName` property instead of `appName` when you want to display the accurate browser name.

This lab demonstrated the various types of information you can access using the `navigator` object. One of the most challenging aspects of Internet application development is creating code that will function in all browsers. When coding for different browsers is a mission-critical concern, the information that the `navigator` object contains is an essential element. As your application development experience grows, you will find many uses for the `navigator` object.



Lab 5-7: Redirecting to a page based on browser type

In this lab, you will create a script that will redirect to a different page depending on the type of browser used.

- 1. Editor:** Open the file **lab5-7.html**.
- 2. Editor:** Study the code. Add the following script in the body of the document, as shown in bold, then save **lab5-7.html**:

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Lab 5-6</title>
</head>
<body>
<h3>CIW JavaScript Specialist</h3>
<hr />

<script language="JavaScript" type="text/javascript">
  if(navigator.appName == "WebTV")
  {
    alert("You're using the WebTV browser.")
    window.location="http://www.webtv.com"
  }
  if(navigator.appName == "Netscape")
  {
    alert("You're using a Mozilla browser.")
    window.location="http://www.mozilla.com/"
  }
  if(navigator.appName == "Opera")
  {
    alert("You're using an Opera Browser.")
    window.location="http://www.opera.com/"
  }
  if(navigator.appName == "Microsoft Internet Explorer")
  {
    alert("You're using the Internet Explorer browser.")
    window.location=" http://www.microsoft.com "
  }
</script>

</body>
</html>

```

- 3. Browser:** Open the file **lab5-7.html** in at least two different types of browsers. You should see an alert informing you which type of browser you used to view the page. Click **OK** in the alert box. The browser will redirect to another site, as instructed by the JavaScript code.



Lab 6-1: Using *String* object formatting methods

In this lab, you will use some formatting methods of the `String` object to manipulate text in a new window.

- Editor:** Open the file `lab6-1.htm` from the Lesson 6 folder of the `Student_Files` directory.
- Editor:** Locate the existing `<script>` tag in the `<head>` section of the document. Inside the existing `linksFun()` function, locate the comment that reads:

```
//Add the alert() shown in Figure 6-1.
```

Create an `alert()` method that outputs text as shown in Figure 6-1. Use the `\n` special character to create the line breaks.

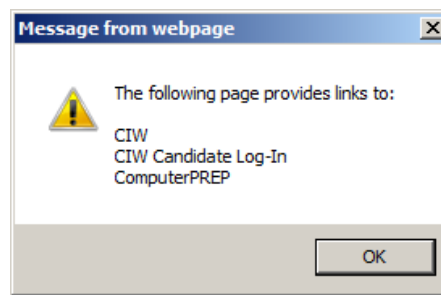


Figure 6-1: Alert with line breaks in text

- Editor:** Locate the comment that reads:

```
//Add big() and fontcolor() methods.
```

Add the `big()` and `fontcolor()` methods to the text **CIW Sites**. Choose any color as the argument for the `fontcolor()` method.

- Editor:** Locate the comment that reads:

```
//Add italics() method.
```

Add the `italics()` method to the text **CIW**.

- Editor:** Locate the comments that read:

```
//Add link() method
//Use http://www.CIWcertified.com.
```

Add the `link()` method to the text **CIW**, **CIW Candidate Log-In**, and **ComputerPREP**. Use the URLs included in the comments for the `link()` method arguments as shown in the following code:

```
<script type="text/Javascript">
<!--

function linksFun() {
    var content="", linkWindow;
    //Add the alert() shown in Figure 6-1
    alert("The following page provides links to: \n\nCIW\nCIW Candidate Log-
In \nComputerPREP");
```

```

content += "<html><head><base target='_blank'></head><body>";
content += "";

//Add big() and fontcolor() methods
content += "CIW Sites".big().fontcolor("blue");
content += "<p>\nThese are sites of interest to ";

//Add italics() method
content += "CIW".italics();
content += " candidates.</p>\n";
content += "<ul><li>";

//Add link() method
//Use http://www.CIWcertified.com
content += "CIW".link("http://www.CIWcertified.com");
content += "\n</li>";

content += "\n<li>";
//Add link() method
//Use http://www.CIWcertified.com/CandidateLogin
content += "CIW Candidate Log-
In".link("http://www.CIWcertified.com/CandidateLogin");
content += "\n</li>";
content += "\n<li>";
//Add link() method
//Use http://www.ComputerPREP.com/
content += "ComputerPREP".link("http://www.ComputerPREP.com/");
content += "\n</li></ul></body></html>";

linkWindow=open("", "Links", "width=350,height=200,resizable=1");

with(linkWindow.document) {
open();
write(content);
close();
}
}

//-->
</script>

```

6. **Editor:** Examine the rest of the code in the `linksFun()` function.
7. **Editor:** In the body of the document, locate the comment that reads:

```
<!--Add HREF code -->.
```

Modify the `<a>` tag as indicated in bold (this code will be discussed following the lab):

```

<a href ="javascript:void(linksFun());"
onmouseover="status='CIW Links';return true;"
onmouseout="status='';return true;">
</a>

```

8. **Editor:** Save **lab6-1.htm**.
9. **Browser:** Open **lab6-1.htm**. Your screen should resemble Figure 6-2.

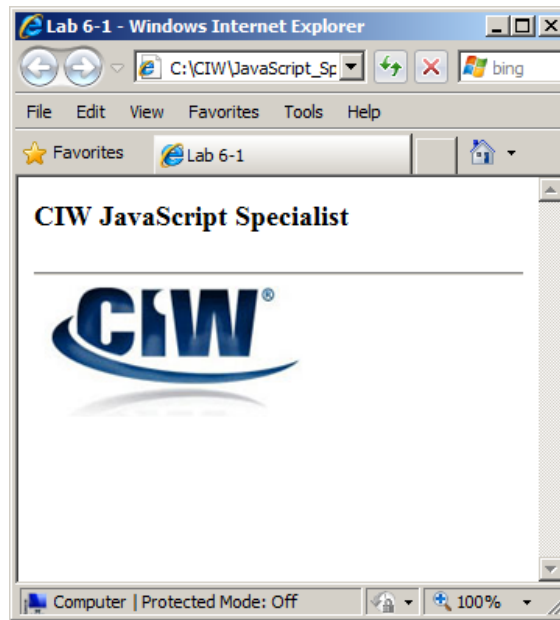


Figure 6-2: Page for lab6-1.htm

- 10. Browser:** Click the **CIW** image. You should first see an alert dialog box as shown in Figure 6-1. You should then see a smaller window appear. The small window should resemble Figure 6-3. If it does not, verify that the source code you entered is correct.



Figure 6-3: Links window

- 11. Browser:** Click the links to make sure they access the appropriate sites.

In this lab, you learned how to use some of the formatting methods of the **String** object to manipulate text in a new window. As time permits, try editing the source code file to incorporate other formatting methods and other special characters.



Lab 6-2: Applying *String* methods to text

In this lab, you will use some methods of the `String` object to examine and manipulate the contents of several text boxes. In addition, this lab reviews another method: the `toUpperCase()` string-conversion method. You will learn more about form validation later in the course.

- Editor:** Open the file **lab6-2.htm** from the Lesson 6 folder of the `Student_Files` directory.
- Editor:** Examine the following JavaScript code (this code will be discussed in detail following the lab):

```
<script type="text/javascript">
<!--

function showUpper(checked) {
    var showThis;
    if (checked) {
        showThis = document.myForm.name.value.toUpperCase();
    } else {
        showThis = document. myForm.name.value.toLowerCase();
    }
    document.myForm.name.value = showThis;
}

function emailTest(form) {
    if (form.email_address.value.indexOf("@", 0) < 0) {
        alert("This is not a valid e-mail address!");
    } else {
        alert("This could be a valid e-mail address");
    }
}

function showFirst2(form) {
    var first2Chars;
    first2Chars = form.phone_number.value.substring(0, 2);
    alert(first2Chars);
}
//-->
</script>
```

- Editor:** Locate the `<form>` tag in the source code and examine the following (particularly the code in bold):

```
<form name="myForm" id="myForm">Name:<br />
<input type="text" size="30" name="name" />
<input type="checkbox" name="upperCheckbox"
onclick="showUpper(this.checked);"/>
Convert string to uppercase or lowercase<br />

E-mail address:<br />
<input type="text" size="30" name="email_address" />
<input type="checkbox" onclick="(this.checked) ? emailTest(this.form) : '';/>
Test for e-mail address<br />

Phone number:<br />
<input type="text" size="30" name="phone_number" /><br />
Fax number:<br />
<input type="text" size="30" name="fax_number" />
<p>
```

```

<input type="button" value=
"First Two Characters of Phone Number" onclick=
"showFirst2(this.form);" />
</p>
<p>
<input type="button" value="Fax Number Length" onclick=
" var strLength = document.myForm.fax_number.value.length; alert('That string
is ' + strLength + ' characters long');" />
</p>
</form>

```

4. **Editor:** Close the file.
5. **Browser:** Open **lab6-2.htm**. Your screen should resemble Figure 6-5. Enter some text and test your options.

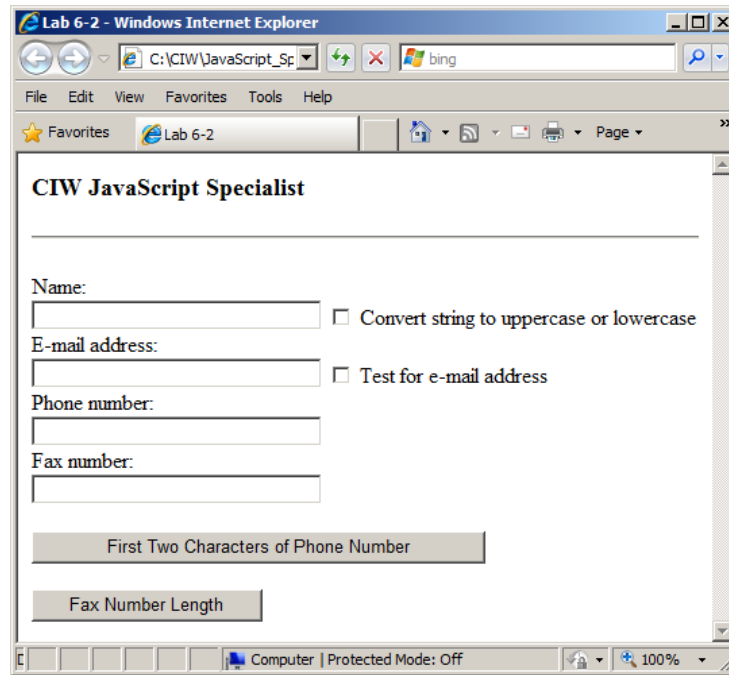


Figure 6-5: Evaluating strings with string methods

6. **Editor:** As time permits, modify the **emailTest()** function to include a test that ensures that the e-mail address is at least six characters in length. A suggested solution named **lab6-2a.htm** is included in the Lesson 6 student files.

This lab demonstrated several `String` object methods.



Lab 6-3: Creating an Array object

In this lab, you will create an Array object and add elements to the array. You will also use the `length` property of Array with a `for` statement to write a few lines of code that will generate three lines of output.

- 1. Editor:** Open the file **lab6-3.htm** from the Lesson 6 folder of the Student_Files directory.
- 2. Editor:** Locate the existing `<script>` tag in the `<head>` section of the document. Create an array named `citiesArray`. Add three elements to `citiesArray`: **New York**, **Los Angeles** and **Chicago** (in that order). Use any valid syntax previously discussed in this lesson to add the array elements to the array.
- 3. Editor:** Locate the existing `<script>` tag in the `<body>` section of the document. Create a variable name `len`. Assign the result of the `citiesArray` `length` property as the value for `len`.
- 4. Editor:** Next, create a **for** loop with a loop counter variable initialized to **0**. The second argument in the `for` loop will use the `len` variable to determine the number of times that the loop will execute. Inside the loop, use a `document.write()` statement to output the elements of the `citiesArray` with two line breaks (`
`) between each one.
- 5. Editor:** Save **lab6-3.htm**.
- 6. Browser:** Open **lab6-3.htm**. Your screen should resemble Figure 6-8. If not, verify that the source code you entered is correct.

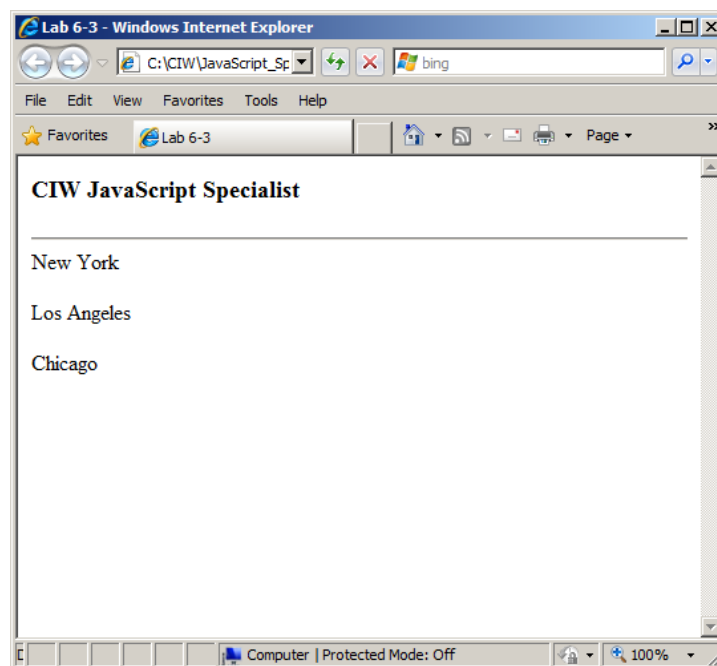


Figure 6-8: Page for lab6-3.htm

- 7. Editor:** Use the `sort()` method to sort `citiesArray`. Save **lab6-3.htm**.

8. **Browser:** Refresh **lab6-3.htm**. Your screen should resemble Figure 6-9.

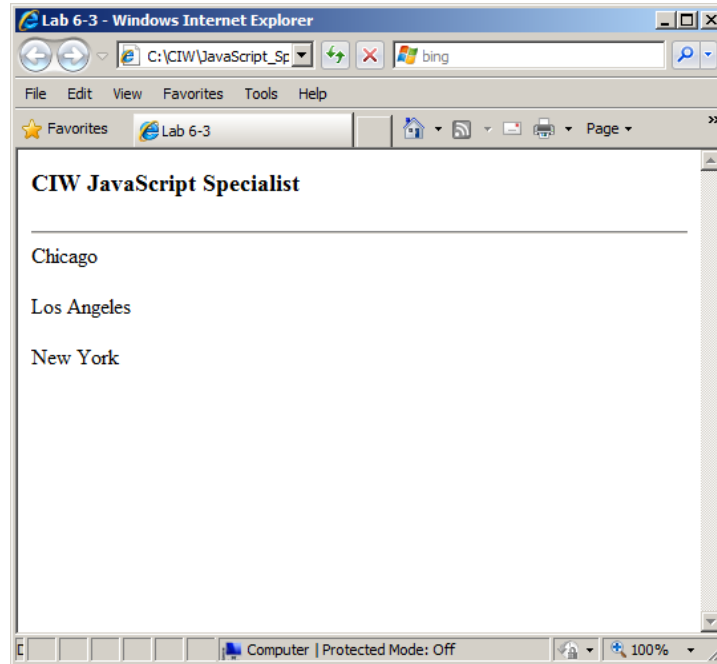


Figure 6-9: Page for Llb6-3.htm with sorted array

9. **Editor:** After the `for` loop, add an `alert` that outputs the value of `citiesArray[0]`. This will illustrate the fact that the array has been rearranged, not just temporarily copied. Save **lab6-3.htm**.
10. **Browser:** Refresh **lab6-3.htm**. You should see an alert as shown in Figure 6-10.

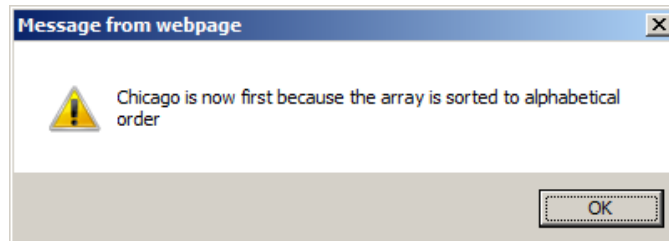


Figure 6-10: Alert dialog box

11. As time allows, try changing the city names in this page to see the sort method in action. Also try adding additional names to the list, which you can make as long as you like.

In this lab, you created an `Array` object and added elements to the array. You also saw that using the `Array` object and its `length` property with a `for` statement allowed you to write a few lines of code to generate three lines of output. However, had `citiesArray` contained 1,000 elements, the same few lines of code would have generated 1,000 lines of output. Using a loop construct to access array values is a very common operation in JavaScript. The index numbers for arrays begin with 0, and a loop counter variable is typically initialized to 0. Therefore, a `for` loop is a natural tool for accessing an array's values.



Lab 6-4: Accessing and using dates with the Date object

In this lab, you will use the `Date` object and the `Array` object to determine and use information about the current date. You will also use a simple formula to add the appropriate suffix to the number for the day of the month (changing the cardinal number to an ordinal number, as commonly used for dates).

- Editor:** Open the file `lab6-4.htm` from the Lesson 6 Folder of the `Student_Files` directory. The code appears as follows. Notice the bold code, where you will create a date object:

```
<script type="text/javascript">
<!--

// Create Date object

var monthName = new Array();
monthName[0] = "January";
monthName[1] = "February";
monthName[2] = "March";
monthName[3] = "April";
monthName[4] = "May";
monthName[5] = "June";
monthName[6] = "July";
monthName[7] = "August";
monthName[8] = "September";
monthName[9] = "October";
monthName[10] = "November";
monthName[11] = "December";

var myYear = today.getFullYear();
var myDate = today.getDate();

var dayExt = "th";

if ((myDate == 1) || (myDate == 21) || (myDate == 31)) dayExt= "st";
if ((myDate == 2) || (myDate == 22)) dayExt = "nd";
if ((myDate == 3) || (myDate == 23)) dayExt = "rd";

var extDate = myDate + dayExt;

alert("The month number is: " + (today.getMonth() + 1));
alert("The date number is: " + today.getDate());
alert("The year number is: " + today.getFullYear());

// -->
</script>

</head>
<body>
<h3>CIW JavaScript Specialist</h3>
<hr />
<h4>Today is the
<script type="text/javascript">
<!--
document.write(extDate + " day of ");
document.write(monthName[today.getMonth()] + " in the year ");
document.write(myYear + ".");
// -->
</script>
</h4>
```


- 2. Editor:** Locate the comment that reads:

```
//Create Date object.
```

Create a variable named **today** and assign it the result of creating a **Date** object.

- 3. Editor:** Examine the rest of the source code with your instructor, then save **lab6-4.htm**.
- 4. Browser:** Open **lab6-4.htm**. Three alert dialog boxes should appear indicating the month number, date number and year number, respectively, as shown in Figure 6-12.

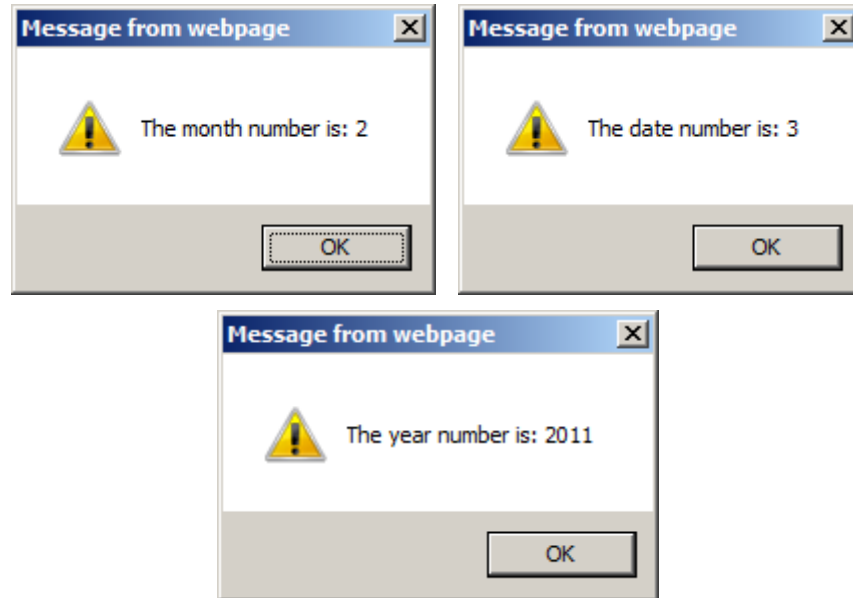


Figure 6-12: Alerts with date information

- 5. Browser:** After closing the alerts, your screen should resemble Figure 6-13, except for differences in the day, month and year displayed.

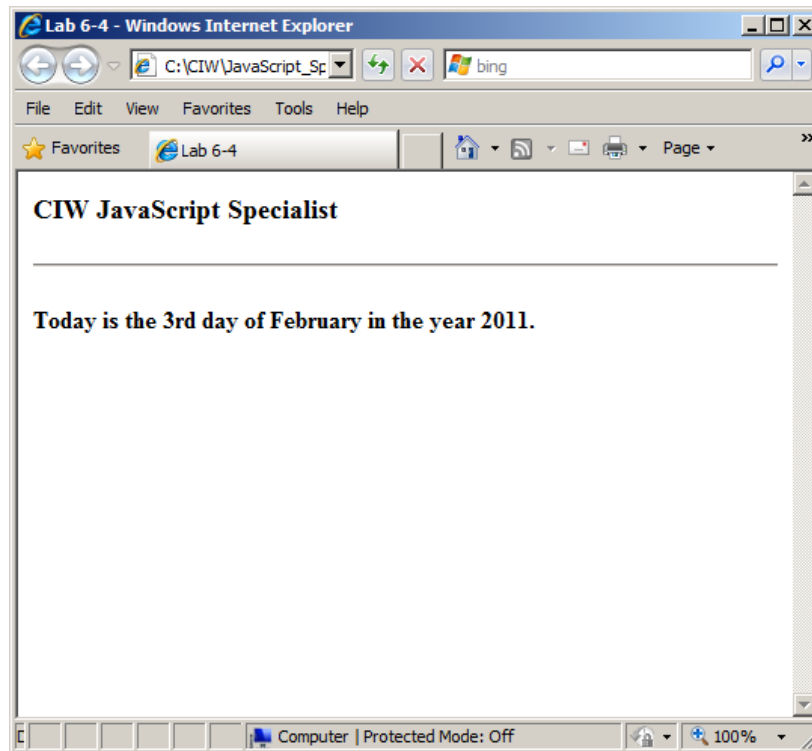


Figure 6-13: Date information calculated through script

In this lab, you used the `Array` object as well as the `Date` object to determine and use information about the current date. You also used a simple formula to add the appropriate suffix to the number for the day of the month (i.e., changing the cardinal number to an ordinal number, as commonly used for dates).



Lab 6-5: Creating an onscreen clock

In this lab, you will review code that creates a clock. This program uses the `setTimeout()` method to update the clock every second by recursively calling a function that outputs the time to a text box.

- 1. Editor:** Open the file **lab6-5.htm** file from the Lesson 6 folder of the Student_Files directory.
- 2. Editor:** Examine the following code with your instructor:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<script type="text/javascript">
<!--

function runClock() {
    var timeNow = new Date();
    var hours = timeNow.getHours();
    var minutes = timeNow.getMinutes();
    var seconds = timeNow.getSeconds();
    var ampm = "";

    (seconds < 10) ? seconds = "0" + seconds : seconds;
    (minutes < 10) ? minutes = "0" + minutes : minutes;
    (hours < 12) ? ampm = "AM" : ampm = "PM";
    (hours > 12) ? hours = hours - 12 : hours;
    (hours == 0) ? hours = 12 : hours;

    var stringTime = " " + hours + ":" + minutes + ":" + seconds + " " + ampm;

    document.clockForm.clockBox.value = stringTime;

    setTimeout("runClock()", 1000);
}

//-->
</script>
<title>Lab 6-5</title>
</head>
<body onload="runClock();">
<h3>CIW JavaScript Specialist</h3>
<hr />
<form name="clockForm" id="clockForm">
<h4>Time Watcher</h4>
<input type="text" name="clockBox" size="11"
onfocus="blur();" />
</form>
</body>
</html>
```

- 3. Editor:** Close the file.
- 4. Browser:** Open **lab6-5.htm**. Your screen should resemble Figure 6-14.

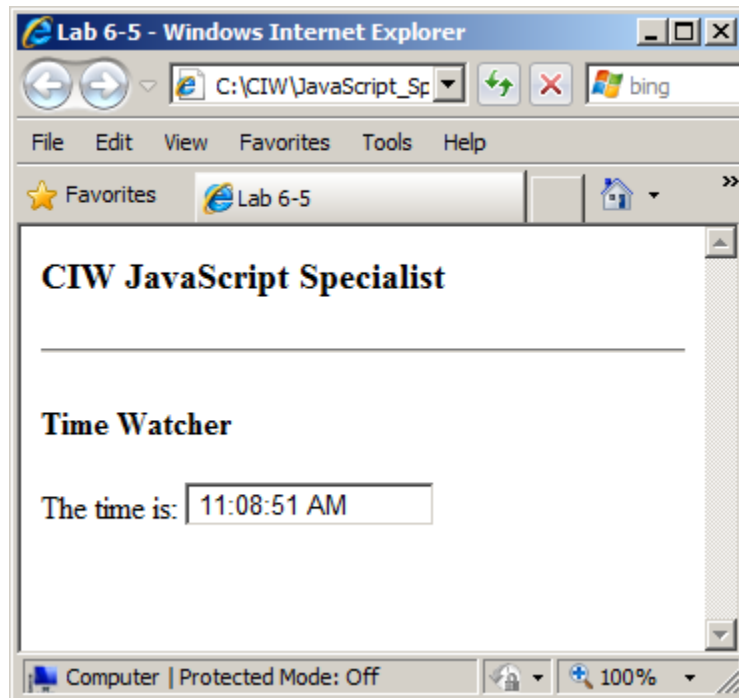


Figure 6-14: Page for lab6-5.htm



Lab 6-6: Using the *Math* object to generate a random quotation

In this lab, you will use the *Math* object and two of its methods. The XHTML page used for this lab will display random quotations, using the *Math* object to determine which quotation will appear.

- Editor:** Open the file **lab6-6.htm** from the Lesson 6 folder of the Student_Files directory. Examine the code, which appears as follows:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/><title>Lab 6-6</title>
</head>

<body onload="quotes()">
<h3>CIW Web Languages</h3>
<hr />
<script language="JavaScript" type="text/javascript">
var randomnumber=Math.floor(Math.random()*9);
alert(randomnumber);
<!--

// -->
</script>
</body>
</html>
```

- Editor:** Notice the code `floor(math.random()*9)`. This will create a variable from 0 to 9 by creating a number less than 1 (`math.random()`), then multiplying it by 9 so the most it can be is 9.99, which will round to 9. (The `floor()` method rounds down, to the bottom (or floor) of the number.)
- Browser:** Run the script. It will display an alert box with a value from 1 to 9. Refresh the browser several times. Different numbers will appear.
- Editor:** Remove the `alert()`, and place the following code between the `<script></script>` tags as shown in bold:

```
<script language="JavaScript" type="text/javascript">
<!--
// Make sure these quotes don't wrap in your file

function quotes(){
var quotes = new Array()
quotes[0] = "Every time history repeats itself the price goes up.<br
/><small><em>- Anonymous</em></small>"
quotes[1] = "The moment you think you understand a great work of art, it's
dead for you.<br /><small><em>- Robert Wilson</em></small>"
quotes[2] = "To love one person with a private love is poor and miserable;
to love all is glorious.<br /><small><em>- Thomas Traherne</em></small>"
quotes[3] = "Every violation of truth is not only a sort of suicide in the
liar, but is a stab at the health of human society.<br /><small><em>- Ralph
Waldo Emerson</em></small>"
quotes[4] = "Man is to be found in reason, God in the passions.<br
/><small><em>- G. C. Lichtenberg</em></small>"
quotes[5] = "Great innovations should not be forced on slender
majorities.<br /><small><em>- Thomas Jefferson</em></small>"
quotes[6] = "In this world nothing can be said to be certain, except death
and taxes.<br /><small><em>- Benjamin Franklin</em></small>"
}
```

```

    quotes[7] = "Nine-tenths of wisdom consists in being wise in time.<br
/><small><em>- Theodore Roosevelt</em></small>"
    quotes[8] = "We have no more right to consume happiness without producing
it than to consume wealth without producing it.<br /><small><em>- George
Bernard Shaw</em></small>"
    quotes[9] = "So little done, so much to do.<br /><small><em>- Cecil
Rhodes</em></small>"
    document.write("<h3>CIW JavaScript Specialist</h3>");
    document.write("<hr />");
    document.write(quotes[randomnumber]);
}

// Use Math object

// -->:
</script>

```

5. **Editor:** Save **lab6-6.htm**.

6. **Firefox Browser:** Open **lab6-6.htm** in Firefox. Your screen should resemble Figure 6-16. If it does not, verify that the source code you entered is correct.

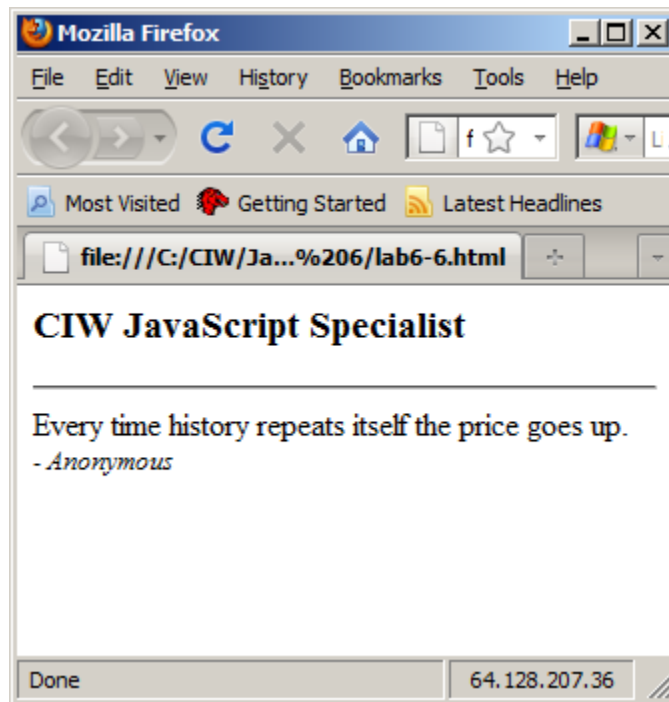


Figure 6-16: Page for lab6-6.htm

7. **Firefox Browser:** Run the script in Firefox and refresh the page. You will see a different quotation each time you refresh. Reload the file several times and you will see different quotations. Because the generator is truly random, you may see the same quotation more than once, even subsequently. If you reload enough times, you will eventually see all the quotations. This script works the same in Google Chrome.
8. **Internet Explorer Browser:** Now open the file **lab6-6.htm** in Internet Explorer and run the script. What happens when you refresh the page? This is another example of code that is interpreted differently by different browsers. In Internet Explorer, the `document.write()` does not reload. You can fix this.
9. **Editor:** Open **lab6-6a_IE_complete.htm**. Study the code, and notice the changes made to this script (shown in bold below). Notice the differences in the `<body>` tag, as

well as the placement of the <h3> page heading into the document.write. Notice also the changes to the function and to its placement. These modifications to the code will cause Internet Explorer to refresh the DOM. This difference demonstrates one way to write code that will work in several browsers.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Lab 6-6a</title>
</head>

<body>
<script language="JavaScript">
var quotes=new Array()

quotes[0] = "Every time history repeats itself the price goes up.<br
/><small><em>- Anonymous</em></small>"
quotes[1] = "The moment you think you understand a great work of art, it's
dead for you.<br /><small><em>- Robert Wilson</em></small></em>"
quotes[2] = "To love one person with a private love is poor and miserable; to
love all is glorious.<br /><small><em>- Thomas Traherne</em></small></em>"
quotes[3] = "Every violation of truth is not only a sort of suicide in the
liar, but is a stab at the health of human society.<br /><small><em>- Ralph
Waldo Emerson</em></small>"
quotes[4] = "Man is to be found in reason, God in the passions.<br
/><small><em>- G. C. Lichtenberg</em></small>"
quotes[5] = "Great innovations should not be forced on slender majorities.<br
/><small><em>- Thomas Jefferson</em></small>"
quotes[6] = "In this world nothing can be said to be certain, except death
and taxes.<br /><small><em>- Benjamin Franklin</em></small>"
quotes[7] = "Nine-tenths of wisdom consists in being wise in time.<br
/><small><em>- Theodore Roosevelt</em></small>"
quotes[8] = "We have no more right to consume happiness without producing it
than to consume wealth without producing it.<br /><small><em>- George Bernard
Shaw</em></small>"
quotes[9] = "So little done, so much to do.<br /><small><em>- Cecil
Rhodes</em></small>"
var i = quotes.length;
var whichquotes=Math.round(Math.random()*(i-1));
function showquotes(){document.write(quotes[whichquotes]);}
document.write("<h3>CIW JavaScript Specialist</h3>");
document.write("<hr />");
showquotes();
</script>

</body>
</html>
```

- 10. Internet Explorer Browser:** Open the file **lab6-6a.htm** in Internet Explorer, and try refreshing the page. This script will refresh properly in Internet Explorer, Firefox and Chrome. Note that both versions of this script use valid JavaScript code, but the first script renders as expected in only two browsers, whereas the second script renders as expected in all the major browsers. This demonstrates again why it is important to check your code in multiple browsers.

In this lab, you used the Math object's random() method to generate a random number between 0 and 1. You multiplied that random number by 9, then used the floor() method to create an integer used as the subscript for the quotes array. You then used a document.write() statement to output a random element from the array. You also explored ways to modify code that does not work as expected in all browsers.



Lab 7-1: Working with text boxes, check boxes and buttons

In this lab, you will work with form elements in your JavaScript code and perform some basic form field validation.

- 1. Editor:** Open the file **lab7-1.htm** from the Lesson_7 folder of the Student_Files directory.
- 2. Editor:** An XHTML form has been created for you. Note that the **text** object in the form is named **myText**. Note also that a **button** object is scripted to call a function named **checkText()**. As previously discussed, the argument **this.form** passes the form's name/value pairs to the function.
- 3. Editor:** Locate the existing **<script>** tag in the document's **<head>** section. Locate the **checkText()** function that has been started for you. You will add two lines of code to this function. Locate the comment that reads as follows:

```
// Check myText element value
```

Create a variable named **myValue** and assign as its value the text entered in the **myText** form element. Then create an alert dialog box that reflects this information back to the user.

- 4. Editor:** Save **lab7-1.htm**.
- 5. Browser:** Open **lab7-1.htm**. Your screen should resemble Figure 7-2.

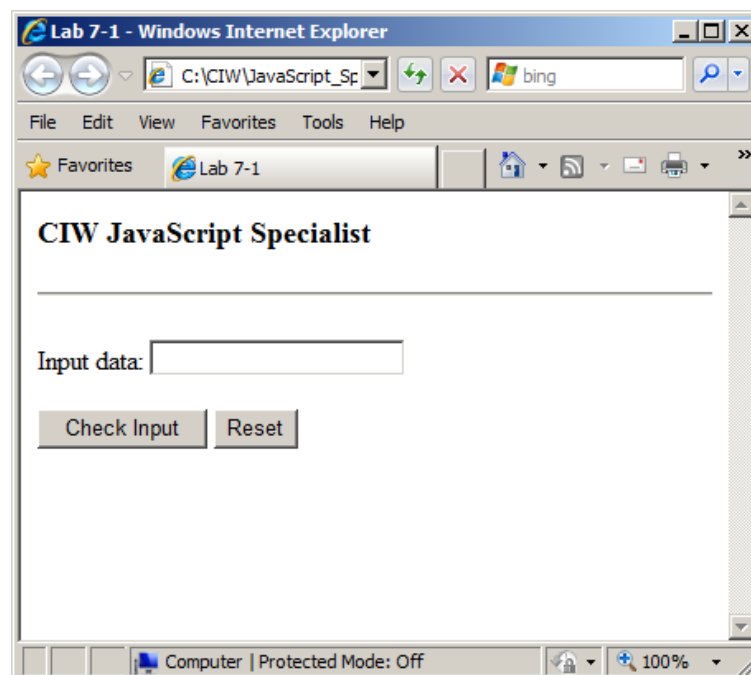


Figure 7-2: Page for lab7-1.htm

- 6. Browser:** Enter data in the text field. Click the **Check Input** button. You should see an alert dialog box containing the text entered in the text field. If you do not, verify that the source code you entered is correct.

7. **Editor:** Save **lab7-1.htm** as **lab7-1a.htm**. You will now add code to **lab7-1a.htm** that performs simple form validation. Following the `myValue` variable declaration, create an `if` statement that checks `myValue` for a value. If `myValue` is empty, create an alert dialog box that asks the user to enter data. Then use the `focus()` method to place the user's cursor in the text box. As the next line of code, use the `return` keyword to end the function. Remember to properly close the `if` statement. As in **lab7-1.htm**, if user input is present, it should appear in an alert dialog box.
8. **Editor:** Save **lab7-1a.htm**.
9. **Browser:** Open **lab7-1a.htm**. Your screen should resemble Figure 7-2. Without entering data, click the **Check Input** button. You should see an alert similar to Figure 7-3. If you do not, verify that the source code you entered is correct.

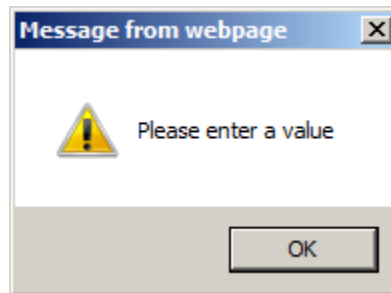


Figure 7-3: Alert dialog box

10. **Editor:** Open the file **lab7-1b.htm** from the `Lesson_7` folder of the `Student_Files` directory. This file contains an XHTML form with a `checkbox` object named `myCheckBox`. A `button` object is scripted to call a function named `isChecked()`. The argument `this.form` passes the form's name/value pairs to the function.
11. **Editor:** In the `isChecked()` function that has been started for you, locate the comment that reads as follows:

```
// Create booleanChecked variable
```

Create a variable name `booleanChecked`. Assign as its value the return value of the `checked` property for `myCheckBox`. An `if` statement has been created for you that reflects to the user the state of the check box.
12. **Editor:** Save **lab7-1b.htm**.
13. **Browser:** Open **lab7-1b.htm**. Your screen should resemble Figure 7-4.

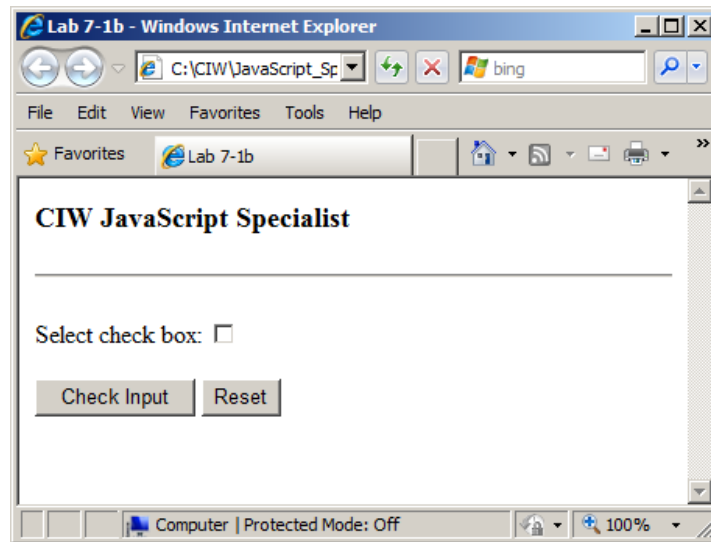


Figure 7-4: Page for lab7-1b.htm

- 14. Browser:** Test the page to ensure that the proper alert dialog box appears when the check box is selected. If the proper alert dialog box does not appear, verify that the source code you entered is correct.



Lab 7-2: Working with radio buttons

In this lab, you will work with radio buttons in your JavaScript code. You will capture the state of `radio` objects and use `radio` object properties in a `for` loop to detect which radio button is selected.

- 1. Editor:** Open the file **lab7-2.htm** from the `Lesson_7` folder of the `Student_Files` directory.
- 2. Editor:** This file contains an XHTML form with a group of `radio` objects named `myRadio`. A `button` object is scripted to call a function named `isChecked()`. The argument `this.form` passes the form's name/value pairs to the function.
- 3. Editor:** In the `isChecked()` function that has been started for you, a variable named `len` has been created. This variable is assigned the value returned by the `length` property for the `myRadio` group. Locate the comment that reads as follows:

```
// Create for loop
```

Create a `for` loop. Use a loop counter variable that starts at zero. Use the `len` variable in the loop's logical expression. Inside the loop, determine whether a particular radio object is selected. Create an alert dialog box that reflects to the user the value of any selected radio button.

- 4. Editor:** Save **lab7-2.htm**.
- 5. Browser:** Open **lab7-2.htm**. Your screen should resemble Figure 7-5.

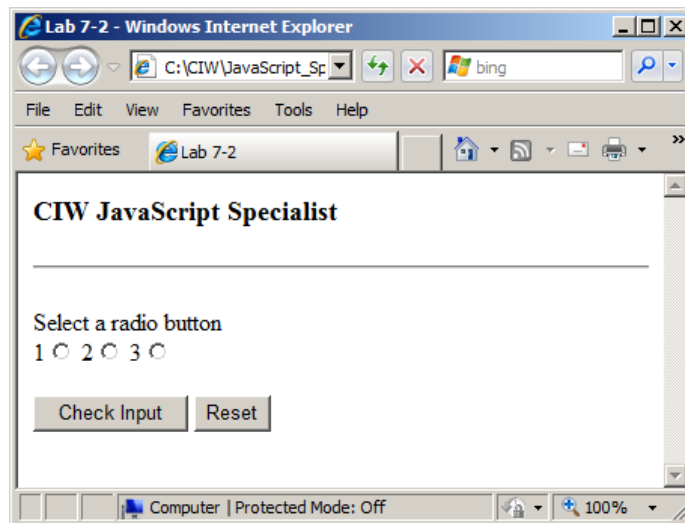


Figure 7-5: Page for lab7-2.htm

- 6. Browser:** Select a radio button, then click the **Check Input** button. If the first radio button is selected, you should see an alert similar to Figure 7-6. If you do not, verify that the source code you entered is correct.

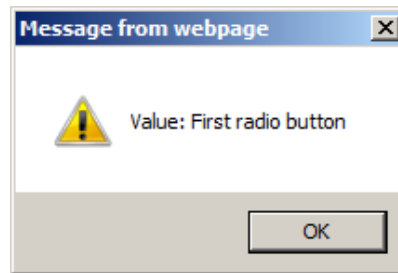


Figure 7-6: Alert dialog box

7. **Browser:** Ensure that the proper message is received for each radio button selection.
8. **Editor:** Save **lab7-2.htm** as **lab7-2a.htm**. Determine a way to give the user an appropriate message if no radio button is selected when the check input button is clicked.
9. **Editor:** Save **lab7-2a.htm**.
10. **Browser:** Open **lab7-2a.htm**. Your screen should resemble Figure 7-5. Click the **Check Input** button without selecting a radio button. You should see an alert dialog box as shown in Figure 7-7. If you do not, check the logic of your source code.

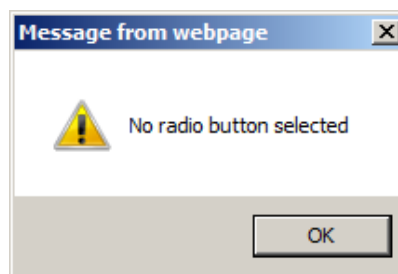


Figure 7-7: Alert dialog box



Lab 7-3: Working with selection lists

In this lab, you will create a drop-down selection list and receive the contents of the form on the page, to determine which option was selected from the drop-down menu and reflect the option's value back to the user.

- 1. Editor:** Open the file **lab7-3.htm** from Lesson_7 folder of the Student_Files directory.
- 2. Editor:** This file contains an X/HTML form with a `select` object named `mySelect`. A button object is scripted to call a function named `isSelected()`. The argument `this.form` passes the form's name/value pairs to the function.
- 3. Editor:** In the `isSelected()` function that has been started for you, a variable named `len` has been created. This variable is assigned the value returned by the `length` property for the `mySelect` object. Locate the comment that reads as follows:

```
// Create for loop
```

Create a `for` loop. Use a loop counter variable that starts at zero. Use the `len` variable in the loop's logical expression. Inside the loop, determine whether a particular option is selected. Create an alert dialog box that reflects to the user the `value` property of any selected option.

- 4. Editor:** Save **lab7-3.htm**.
- 5. Browser:** Open **lab7-3.htm**. Your screen should resemble Figure 7-8.

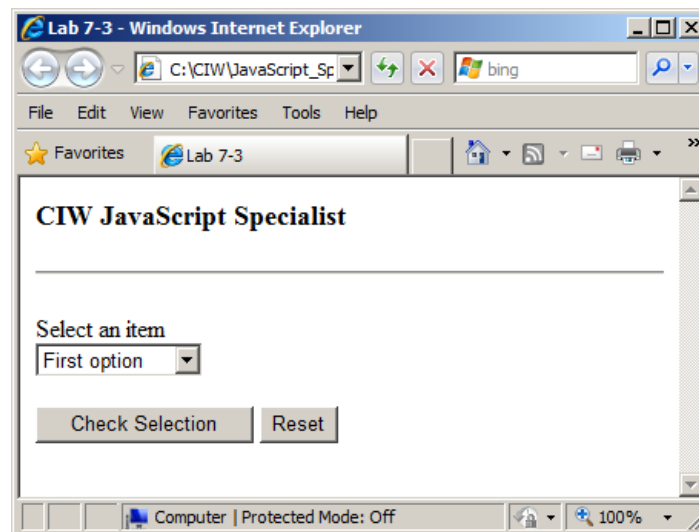


Figure 7-8: Page for lab7-3.htm

- 6. Browser:** Make a selection from the drop-down menu and click the **Check Selection** button. If you select the second option, you should see an alert dialog box that resembles Figure 7-9. If you do not, verify that the source code you entered is correct.



Figure 7-9: Alert dialog box

7. **Browser:** Test the page to ensure that the proper value is returned for each item in the drop-down menu.
8. **Editor:** Open **lab7-3a.htm**. In the body of the document, locate the comment that reads as follows:

```
<!-- Add onchange event handler -->
```

This file is a modified version of lab7-3.htm. An extra `<option>` tag is defined before the existing options. This option's text reads *Select An Item*. Also, no `button` object is scripted to invoke the function as in lab7-3.htm. In the XHTML `<select>` tag, add an `onchange` event handler that invokes the `isSelected()` function. Make sure to pass `this.form` as an argument in the method invocation statement.

9. **Editor:** After the `isSelected()` method invocation statement, add a semicolon. You will now add another statement that will also execute via the `onchange` event handler. Add this line of code:

```
this.selectedIndex=0;
```

10. **Editor:** Save **lab7-3a.htm**.

11. **Browser:** Open **lab7-3a.htm**. Your screen should resemble Figure 7-10.

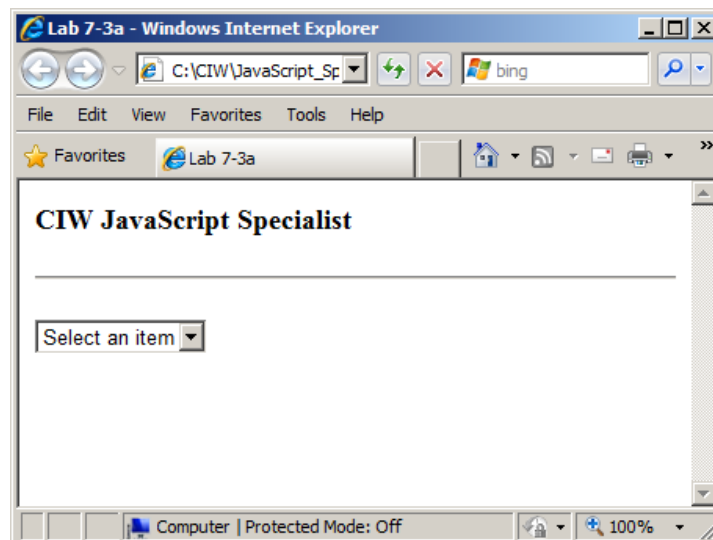


Figure 7-10: Page for lab7-3a.htm after adding onchange event handler

- 12. Browser:** Select an item from the drop-down menu. If you select the third item, you should see an alert similar to Figure 7-11. If you do not, verify that the source code you entered is correct.



Figure 7-11: Alert dialog box

- 13. Browser:** Note that the drop-down menu does not hold the user's selection after closing the alert dialog box. The drop-down menu should revert to its original state. If it does not, verify that the `this.selectedIndex=0` statement added in Step 9 is correct.

In this lab, you scripted the function `isSelected()` to receive the contents of the form on the page. Inside the function, you determined which option was selected from the drop-down menu and reflected the option's value to the user. As with all form elements, it is important to know how to access the value or text of an option chosen from a `select` object.

This lab also demonstrated the `onChange()` event handler used to invoke a function. The `onChange()` event handler was also used to invoke another line of JavaScript code. In this case, the `selectedIndex` property of the `select` object is used to reset the drop-down menu to its default state. Also, the programs were scripted to capture a single selection from a `select` object.



Lab 7-4: Working with a multiple-selection list box

- 1. Editor:** Open **lab7-4.htm** file from the Lesson_7 folder of the Student_Files directory.
- 2. Editor:** This file contains an XHTML form with a `select` object named `mySelect`. The `select` object is created as a multiple-selection list box by adding the `multiple` attribute in the `<select>` tag. A `button` object is scripted to call a function named `isSelected()`. The argument `this.form` passes the form's name/value pairs to the function. In the `isSelected()` function, two variables are created for you. The `len` variable is used as in the previous lab. The variable `theSelections` will receive the text of the user's choices.
- 3. Editor:** Locate the comment that reads as follows:

```
// Create for loop
```

Create a `for` loop. Use a loop counter variable that starts at zero. Use the `len` variable in the loop's logical expression. Inside the loop, determine whether a particular option is selected. If an option is selected, use the `theSelections` variable to build a string containing the text of the selected items. Add a newline character after the text for each selection.
- 4. Editor:** After the `for` loop, create an alert dialog box that displays the `theSelections` variable.
- 5. Editor:** Save **lab7-4.htm**.
- 6. Browser:** Open **lab7-4.htm**. Your screen should resemble Figure 7-12.

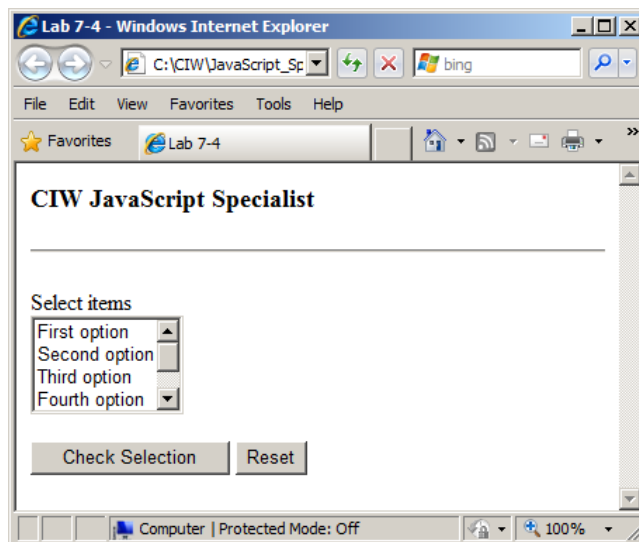


Figure 7-12: Page for lab7-4.htm

- 7. Browser:** Select several items from the list by pressing the CTRL button on your keyboard while clicking each choice. Then click the **Check Selections** button. If the third and sixth items were selected, you should see an alert dialog box similar to Figure 7-13. If you do not, verify that the source code you entered is correct.

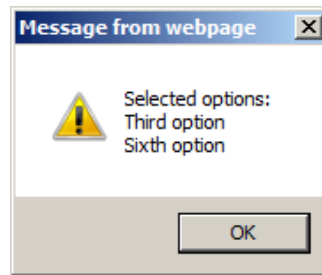


Figure 7-13: Alert dialog box

- 8. Browser:** Experiment with the page to ensure that the alert dialog box displays the proper information in various situations.



Lab 7-5: Conducting form validation

In this lab, you will use JavaScript to validate user input to an XHTML form.

1. **Editor:** Open the **lab7-5.htm** file from the Lesson_7 folder of the Student_Files directory. The XHTML code is already included to create a form that resembles Figure 7-14.

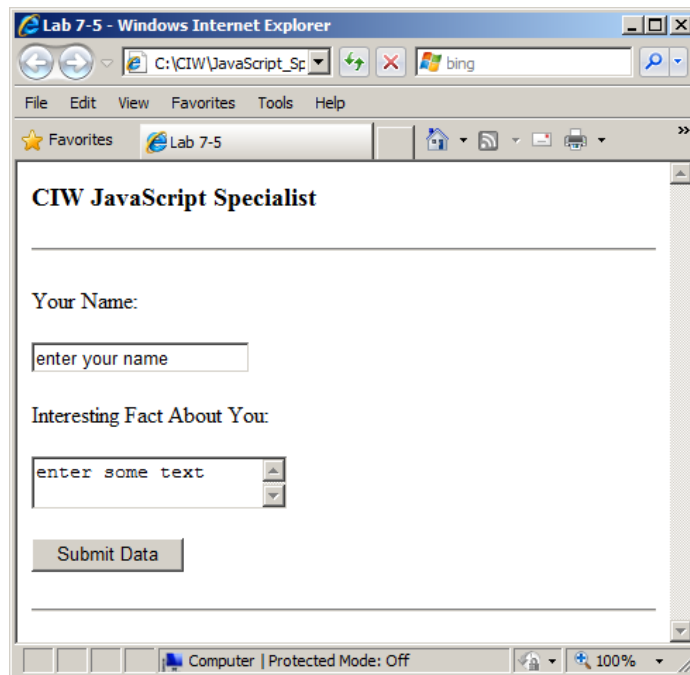


Figure 7-14: Interactive form

2. **Editor:** A submit object is scripted to call a function named `checkForm()`. Examine the function invocation statement:

```
<input type="submit" value="Submit Data" onclick="return
  checkForm(this.form);" />
```

3. **Editor:** Note the use of the `return` keyword. The `checkForm()` function will be written to return either a `true` or a `false` value. If a `true` value is returned, the form will be submitted. A `false` return value will cancel the form submission. The argument `this.form` passes the form's name/value pairs to the function.
4. **Editor:** In the `checkForm()` function, a variable is created for you. The `len` variable is assigned the return value of the form object's `length` property. This variable will be used in a `for` loop to determine how many times the loop will execute.
5. **Editor:** In the file, find the line that reads as follows:

```
//Insert for statement here//
```

Create a `for` loop. Use a loop counter variable that starts at zero. Use the `len` variable in the loop's logical expression. Inside the `for` loop, create an `if` statement. As the condition for the `if` statement, use the `elements` property of the `form` object to determine if each form element's length is less than 1. If an element's length is less

than 1, the function's value will return `false`, which will ensure that the page does not submit.

Tech Note: Another way to check for data in a field is to determine if the form element is null or empty, instead of checking for length. Remember that null and empty are not the same. An entered space is empty, whereas no data entered whatsoever is null. When checking for length, an entered space (empty) will count as a number because a space is a character. Checking for a length is the more typical approach.

6. **Editor:** If the form element is null (no data), create an alert dialog box that asks the user to enter information in the appropriate form element. (*Hint: Use the `name` property of each element in the argument for the `alert()` method.*) Then use the `focus()` method to place the user's cursor in the appropriate form element. Finally, create a `return false` statement. This statement will be returned to the function invocation statement and will cancel submission of the form if the function value returns false (for null data). Close the `if` statement. Close the `for` loop.
7. **Editor:** After the `for` loop, create a `return true` statement. This statement will execute only if all form elements pass the validation test.
8. **Editor:** Save **lab7-5.htm**. The code should now appear as follows with your changes, shown in bold:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Lab 7-5</title>
</head>

<script type="text/javascript">
    function checkForm()
    {
        var str = '';
        var elem = document.getElementById('myForm').elements;
        for(var i = 0; i < elem.length; i++)
        {
            if(elem[i].value.length < 1)
            {
                alert("Enter something for the field: " + elem[i].name);
                var mytext = document.getElementById(elem[i].name);
                mytext.focus();
                return false;}
            }
        }
    }
</script>
</head>

<body>
<h3>CIW JavaScript Specialist</h3>
<hr />

<form id="myForm" name="myForm" action="lab7-5a.htm">
<p>
Your Name: <br /><br />
<input type="text" name="Your_Name" value="enter your name" />
</p>
<p>
Interesting Fact About You: <br /><br />
<textarea name="Interesting_Fact">enter some text</textarea>
</p>
<input type="submit" value="Submit Data" onclick="return
checkForm(this.form);" />
</form>
```

```

</form>

<hr />
<div id="lblValues"></div>
</body>
</html>

```

- 9. Internet Explorer Browser:** Open **lab7-5.htm** in Internet Explorer. Your screen should resemble the preceding figure (Figure 7-14). Delete the text from the Name text box. Click the **Submit Data** button without entering any data in the form, not even a space. You should see an alert dialog box similar to Figure 7-15. If you do not, verify that the source code you entered is correct.

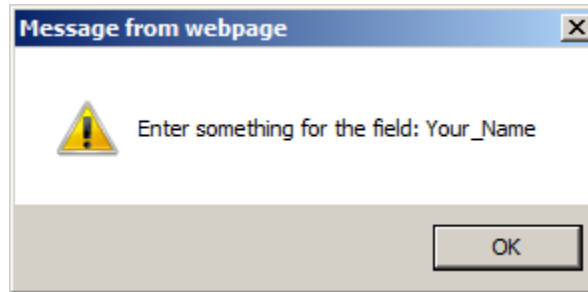


Figure 7-15: Alert dialog box

- 10. Browser:** After closing the alert dialog box, your cursor should be in the Name text box. If it is not, verify that the source code you entered is correct.

Tech Note: Perform this lab using Internet Explorer as your browser. When you are finished, you can try it again using Firefox or another browser. Notice that the script may not work as expected in other browsers. This is an example of how JavaScript code often renders differently in different browsers, and it provides a reminder of the importance of testing your pages in multiple browsers.

- 11. Browser:** Experiment with the form to ensure that the proper alert dialog boxes appear for each situation. Ensure that the user's cursor is in the proper text box if the text box is left empty.

Tech Note: Also notice that after submitting information correctly, the fields appear to reset to the defaults. But in fact, the page has reloaded to the next file (lab7-5a.htm) because that file name is specified in the `action` attribute of the `<form>` tag (`action="lab7-5a.htm"`), instead of posting the form data to a server.

- 12. Editor:** Open **lab7-5a.htm**. Notice the additions of the following code and the changes to the `getElementById` method, shown in bold:

```

<script type="text/javascript">
  function checkForm()
  {
    var str = '';
    var elem = document.getElementById('myForm').elements;
    for(var i = 0; i < elem.length; i++)
    {

      str += "<strong>Type:</strong>" + elem[i].type + "&nbsp;&nbsp;&nbsp;";
      str += "<strong>Name:</strong>" + elem[i].name + "&nbsp;&nbsp;&nbsp;";
      str += "<strong>Value:</strong><em>" + elem[i].value + "</em>&nbsp;&nbsp;&nbsp;";
      str += "<br />";

      if(elem[i].value.length < 1)
      {

```

```
        alert("Enter your name for the field: " + elem[i].name);  
        return false;}  
    }  
    document.getElementById('tblValues').innerHTML = str;  
}  
</script>
```

- 14. Browser:** Open **lab7-5a.htm**. Enter some information into both fields, and click the **Submit Data** button. Notice the information that you enter (or the default text if you do not change it) appears at the bottom of the page after you click the Submit Data button, as shown in Figure 7-16. As the form loops through the **for** loop, the program extracts the name and value of each field. It is important to note that the button properties are also shown in this information because the buttons are also form elements.

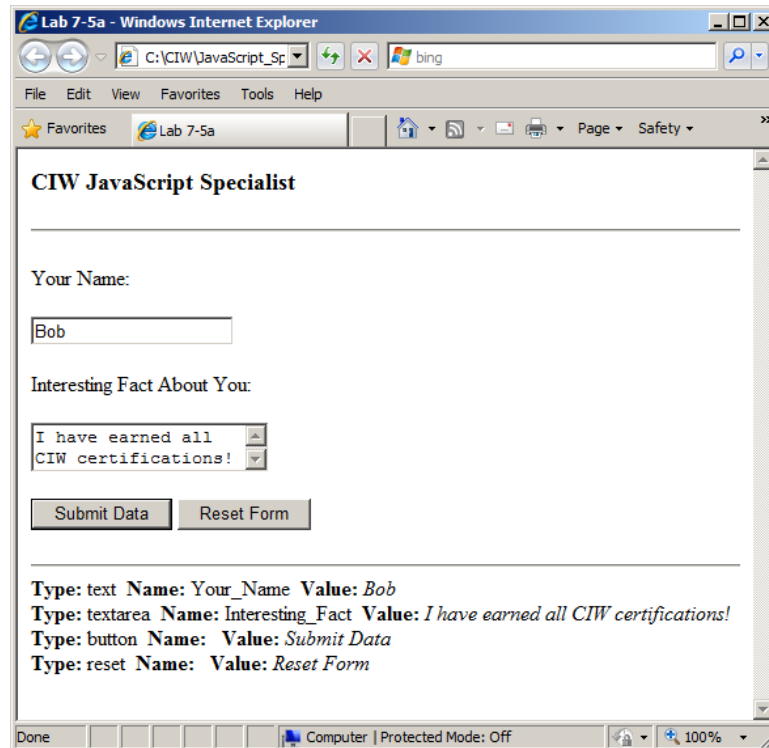


Figure 7-16: Page for lab7-5a.htm after submitting data



Lab 8-1: Performing client-side browser detection

In this lab, you will write a script that performs both basic and advanced browser detection. Most sites in the present time will check the type of browser the client is using first, and then redirect the user to the pages that have been optimized for the browser. Unfortunately, malicious scripters will specifically look for an exploitable browser (this is called zero-day vulnerability) and take advantage of it before the virus protection or browser vendor has a chance to create and distribute a patch.

- 1. Editor:** Open the file **lab8-1.htm**. This file provides the shell of a page you will further develop.
- 2. Editor:** Find the comments that direct you where to place the JavaScript code. In that location, enter the following code (provided in the file lab8-1code.htm):

```
<script type="text/javascript">
mactest=(navigator.userAgent.indexOf("Mac")!=-1) //My browser sniffers

is_chrome = navigator.userAgent.toLowerCase().indexOf('chrome') > -1

Netscape=(navigator.appName.indexOf("Netscape") != -1)

msafari=(navigator.userAgent.indexOf("Safari")!= -1)

wsafari=0; if(!mactest&&msafari){wsafari=1;msafari=0}

is_opera = 0; if(window.opera){is_opera=1}

is_ie_mac = 0; is_ie=0;if(document.all){is_ie=1}

if(is_ie&&mactest){is_ie_mac=1}

alert("mactest:"+mactest+", Netscape:"+Netscape+", wsafari:"+wsafari+",
msafari:"+msafari+", is_ie:"+is_ie+", is_ie_mac:"+is_ie_mac+",
is_opera:"+is_opera+", is_chrome:"+is_chrome)
</script>
```

- 3. Editor:** Save the file **lab8-1.htm**.
- 4. Browser:** Open the file **lab8-1.htm**. You will see a pop-up window similar to the one shown in Figure 8-1.

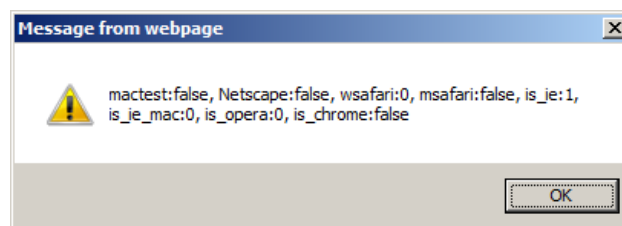


Figure 8-1: Pop-up describing browser type detected

- 5. Browser:** Notice that this pop-up lists several browser types, and gives a value after each type. The browser type you are using will be specified by a value of true (or a Boolean value of 1); all the browser types that were not detected will show values of false (or Boolean 0). This alert does not provide user-friendly readability, but rather a raw data string that would be used further in a program. This script identifies the type of browser you are using, but it does nothing further with this information. Many Web sites use the detection information to legitimately enhance your

experience by automatically directing you to pages that are optimized for your browser.

- 6. Editor:** Open **lab8-1a.htm**. Find the comments that direct you where to place the JavaScript code. In that location, enter the following code (provided in the file lab8-1a_code.htm) as shown in bold, then save the file:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Lab 8-1a</title>
</head>

<body>
<script language="javascript" type="text/javascript">
<!--
    // write out some introductory blurb and begin the table.

    document.writeln('<h3>CIW JavaScript Specialist</h3><hr />');

    document.writeln('<table width=470 align=center><tr><td><p
class="narrow">' +
    'The table below shows the principal properties of your '
+
    'browser, the name by which it identifies itself, and ' +
    'the plug-ins supported. The final part of the table ' +
    'lists all the document types for which a plug-in has ' +
    'been defined, together with the name of the plug-in ' +
    'that handles each type. Types for which no plug-in is '
+
    'available are not shown.<' + '/p><hr /><' +
    '/td><' + '/tr><' + '/table>');

    document.writeln('<table cellpadding=2 cellspacing=1 border=0 ' +
    'width=470 align=center><tr>');
    document.writeln('<th width=100 align=left class="small" ' +
    'bgcolor="#666699">Property<' + '/th>');
    document.writeln('<th width=100 align=left class="small" ' +
    'bgcolor="#666699">Variable<' + '/th>');
    document.writeln('<th width=100 align=left class="small" ' +
    'bgcolor="#666699">Value<' + '/th><' + '/tr>');

    // set up the variables we need to test the first few properties

    var number_of_values_to_test = 4;
    var name_array = new Array(number_of_values_to_test);
    var properties = new Array(number_of_values_to_test);
    name_array[0] = "Application Name";
    properties[0] = "appName";
    name_array[1] = "Code Name";
    properties[1] = "appCodeName";
    name_array[2] = "Version";
    properties[2] = "appVersion";
    name_array[3] = "User Agent";
    properties[3] = "userAgent";

    // Loop through the properties, outputting one table row for each one.

    for (var index=0;index < number_of_values_to_test;index++) {
        document.write('<tr><td bgcolor="#9999cc" align=left class="small">' +
            name_array[index] + '<' + '/td>');
        document.write('<td bgcolor="#ccccff" class="small">' +
properties[index] +
            '<' + '/td>');
        document.write('<td bgcolor="#ccccff" class="small">' +
```

```

        navigator[properties[index]] + '<' + '/td>');
    document.writeln('<' + '/tr>');
}

// Write out details about the user's installed plug-ins

document.write('<tr><td bgcolor="#9999cc" align=left class="small">Plug-
Ins<' +
    '</td>');
document.write('<td bgcolor="#ccccff" class="small">plugins<' + '/td>');
document.writeln('<td bgcolor="#ccccff" class="small">');
if (navigator.plugins != null) {
    for (var index=0;index < navigator.plugins.length; index++) {
        document.writeln(navigator.plugins[index].name + '<br />');
    }
}
else {
    document.writeln("Browser doesn't recognize " +
        "'navigator.plugins' property.");
}
document.writeln('<' + '/td><' + '/tr>');

// Do the same for the MIME types. This is more complex because we need to
// build a subtable within our main table, showing the types, their names
// and extensions, and the plug-in assigned to them.

document.write('<tr><td bgcolor="#9999cc" align=left class="small">' +
    'MIME Types<' + '/td>');
document.write('<td bgcolor="#ccccff" class="small">mimeTypes<' + '/td>');
document.writeln('<td bgcolor="#ccccff" class="small">');
if (navigator.mimeTypes != null) {
    document.writeln('<table>');

    document.writeln('<tr bgcolor="#9999cc">' +
        '<th class="small" align=left bgcolor="#9999cc">' +
        'mime type<' + '/th>' +
        '<th class="small" align=left bgcolor="#9999cc">' +
        'identifier<' + '/th>' +
        '<th class="small" align=left bgcolor="#9999cc">' +
        'extn.<' + '/th>' +
        '<th class="small" align=left bgcolor="#9999cc">' +
        'plug-in<' + '/th><' + '/tr>');

    for (var index=0;index < navigator.mimetypes.length; index++) {
        if (navigator.mimetypes[index].enabledplugin != null) {
            document.write('<tr><td class="small" valign=top>' +
                navigator.mimetypes[index].type +
                '<' + '/td><td class="small" valign=top>' +
                navigator.mimetypes[index].description +
                '<' + '/td><td class="small" valign=top>' +
                navigator.mimetypes[index].suffixes +
                '<' + '/td><td class="small" valign=top>' +
                navigator.mimetypes[index].enabledplugin.name +
                '<' + '/td><' + '/tr>');
        }
    }
    document.writeln('<' + '/table>');
}
else {
    document.writeln("Browser doesn't recognize " +
        "'navigator.mimeTypes' property.");
}
document.writeln('<' + '/td><' + '/tr>');

// finish up the table.

document.writeln('<' + '/table>');
// -->

```



```
</script>
```

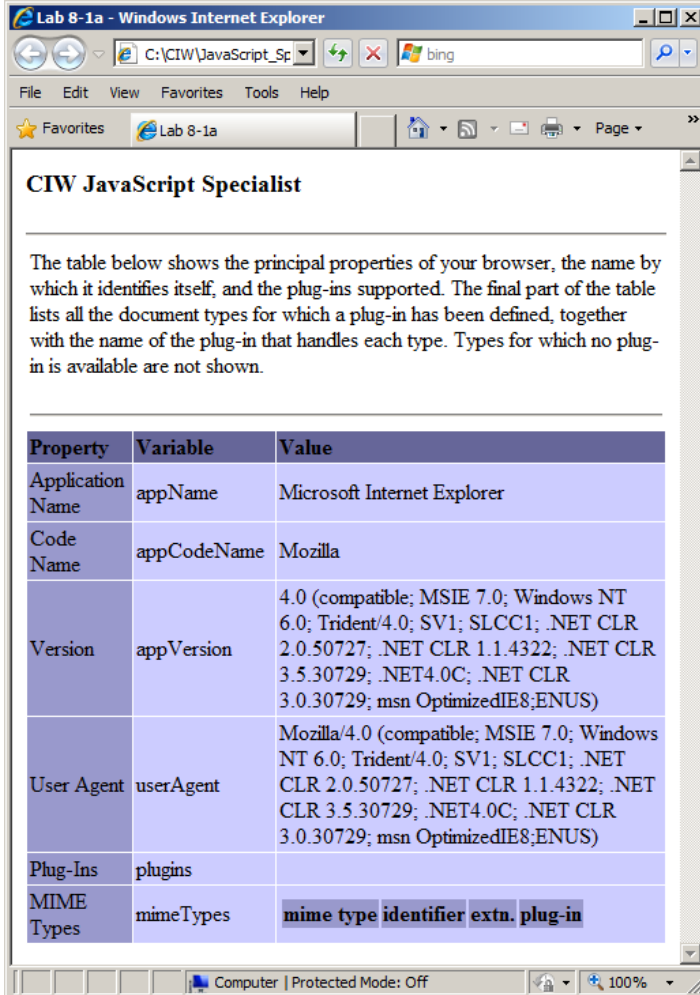
```
</body>
```

```
</html>
```

7. **Editor:** Save the file **lab8-1a.htm**.

8. **Browser:** Open the file **lab8-1a.htm** to run the script. You should see the page shown in Figure 8-2 with information about your browser. Notice that this script is using standard JavaScript to identify a host of information about your browser.

Tech Note: Be sure to view this file in multiple browsers. You will see plug-in information in Firefox.



The screenshot shows a Windows Internet Explorer window titled "Lab 8-1a - Windows Internet Explorer". The address bar shows "C:\CIW\JavaScript_Sp" and the search engine is set to "bing". The page content includes the title "CIW JavaScript Specialist" and a paragraph explaining that the following table shows browser properties. The table has three columns: Property, Variable, and Value.

Property	Variable	Value
Application Name	appName	Microsoft Internet Explorer
Code Name	appCodeName	Mozilla
Version	appVersion	4.0 (compatible; MSIE 7.0; Windows NT 6.0; Trident/4.0; SV1; SLCC1; .NET CLR 2.0.50727; .NET CLR 1.1.4322; .NET CLR 3.5.30729; .NET4.0C; .NET CLR 3.0.30729; msn OptimizedIE8;ENUS)
User Agent	userAgent	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; Trident/4.0; SV1; SLCC1; .NET CLR 2.0.50727; .NET CLR 1.1.4322; .NET CLR 3.5.30729; .NET4.0C; .NET CLR 3.0.30729; msn OptimizedIE8;ENUS)
Plug-Ins	plugins	
MIME Types	mimeTypes	mime type identifier extn. plug-in

Figure 8-2: Table detailing your browser properties

From a security point of view, consider that this script reveals a significant portion of your browser information. Although this may be useful for determining browser compatibility, it could also provide a hacker with enough information about your system to stage an attack.

This demonstration is a reminder that no computer or server is 100-percent safe. However, by keeping your browser, operating system and anti-virus software current, you can dramatically decrease the chances that your system or server will be compromised.



Lab 8-2: Locking the browser with malicious code

In this lab, you will observe JavaScript used to lock a user's browser.

- 1. Editor:** Open the file **lab8-2.htm** from the Lesson_8 folder of the Student_Files directory.
- 2. Editor:** Examine the following source code:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/><title>Lab 8-2</title>

<script language="JavaScript" type="text/javascript">
<!--

for (i=0; i >= 0; i++) {
    alert("Stop me if you can!");
}

//-->
</script>
</head>
<body>
This page demonstrates poorly written or malicious JavaScript code that locks
the browser.
</body>
</html>
```

- 3. Editor:** Close **lab8-2.htm**.
- 4. System:** Close any open programs.
- 5. Browser:** Open **lab8-2.htm**. You will see an alert dialog box as shown in Figure 8-4.

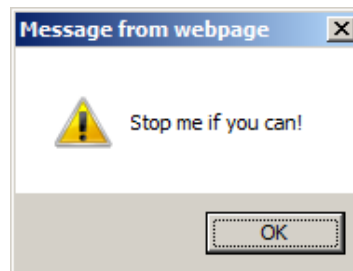


Figure 8-4: Alert dialog box

- 6. Browser:** You can click the **OK** button repeatedly, but the message will return.
- 7. System:** To stop execution of this script, you must close the browser. To do this, hold down the **CTRL + ALT** keys and press the **DELETE** key on your keyboard. This action will open a dialog box similar to Figure 8-5.

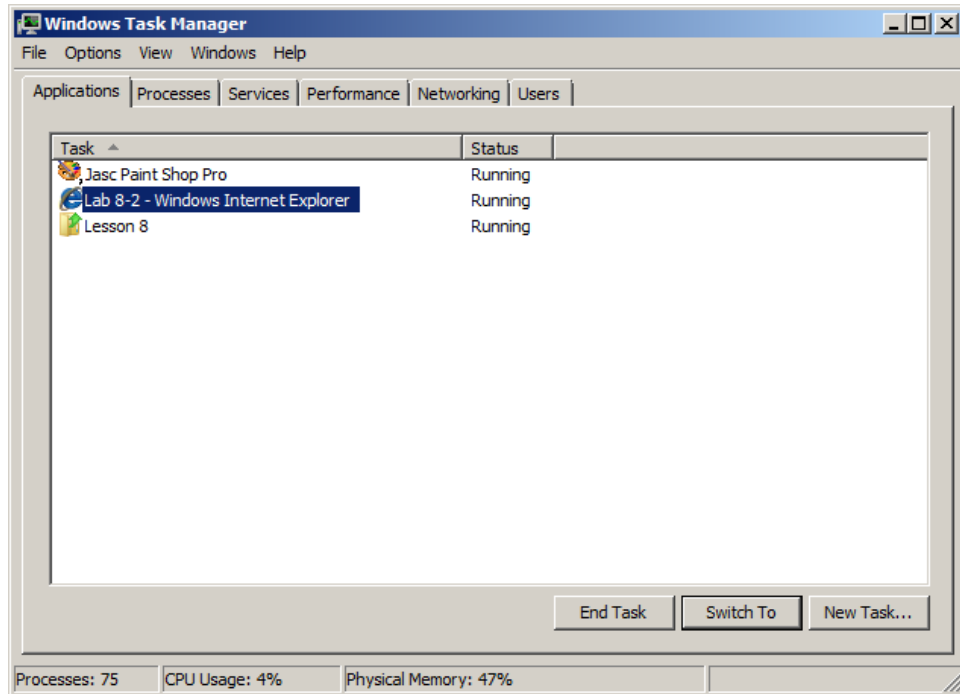


Figure 8-5: Close Program menu

8. **System:** Verify that the Internet Explorer task (or whichever browser you are using) is selected. Click the **End Task** button to close the browser. You will see the message shown in Figure 8-6.

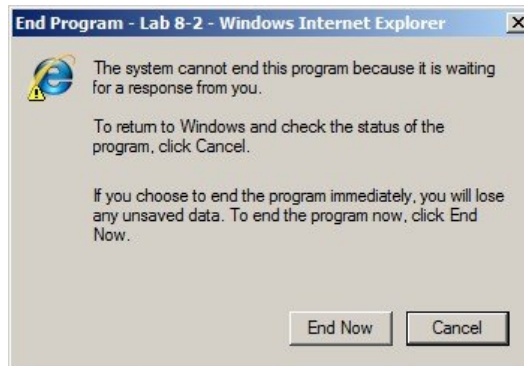


Figure 8-6: End Task message

9. **System:** Click **End Now** to complete the operation. You have now closed your browser with no damage done.
10. **Editor:** The browser lock is caused by an infinite loop in the for statement. This sort of loop is typical when a developer is first learning to script loops, but it is rarely found in production, because it is simple to stop and easy to fix. An infinite loop has no conditions for ending. You can correct it by creating a condition that will cause the program to finish the loop. Open the file **lab8-2.htm**. In the script section, change the code as shown in bold, then save the file:

```
<script language="JavaScript" type="text/javascript">
<!--
var i=0;
```

```
for (i=0;i<=5;i++)
{
alert("The number is " + i);
}

//-->
</script><body>
<h3>CIW JavaScript Specialist</h3>
<hr />
This page demonstrates repaired JavaScript code that will no longer loop
infinitely.
</body>
```

- 11. Browser:** Open the file **lab8-2.htm**. You should see that the alert box counts up and then stops as expected at the number 5. You have now performed your diligent troubleshooting and corrected the code to remove the infinite loop.



Lab 8-3: Setting, viewing and clearing a cookie with JavaScript

In this lab, you will set a cookie, delete a cookie and retrieve cookie values, which you can apply to an X/HTML document.

- Editor:** Open the file **lab8-3.htm** from the Lesson_8 folder of the Student_Files directory. Study the `<script>` portion of the code in the `<head>` section of the file, which appears as follows. This is where the cookie is created:

```
<script language="JavaScript" type="text/javascript">
<!--
    var myColor = prompt("Please enter a hexadecimal color code (must be 6
characters, including 0 thru 9 and A thru F):","FFFFFF");
    var myName = prompt("Please enter your name:","Cookie Monster");
    if ((myColor == null) || (myColor == "") || (myColor.length < 6 ||
myColor.length > 6)) myColor = "#ffffff";
    var keepCookie = (confirm("A cookie will be set. Delete cookie?")) ?
"delete" : "keep";
    if (keepCookie == "delete") {
        document.cookie="testCookie=" + "YColor=" + myColor + " " + "name=" +
myName + ";expires=20-May-2010";
        message= "Here is the cookie that was deleted: ";
        message += document.cookie;
        alert(message);
    }
    else {
        document.cookie="testCookie=" + "YColor=" + myColor + " " + "name="
+ myName + ";expires=20-May-2015";
        message= "Here is the cookie that was set on your system: ";
        message += document.cookie;
        alert(message);
    }

    var start = document.cookie.indexOf("YColor=");
    start=start + 7;
    var pos = "#" + document.cookie.substr(start,6);
    alert("Your background color will be: " + pos);

    var nameLengthS = document.cookie.indexOf("name=");
    nameLengthS=nameLengthS + 5;
    var PersonName = document.cookie.substr(nameLengthS,25);
    alert("Your name is: " + PersonName);

//-->
</script>
</head>
```

- Editor:** Now examine the following code after the `<body>` tag in the file, which will display the cookie contents:

```
<body>
<h3>CIW JavaScript Specialist</h3>
<hr />
<script language="JavaScript" type="text/javascript">
<!--
    if ((pos == null) || (pos == "")) pos = "#ffffff";
    document.bgColor = pos;
    document.write("Your background color is: " + document.bgColor +
"<br />");
    document.write("Your name is: " + window.PersonName + "<br />");
    document.write("Cookie contents: " + document.cookie);
    document.write();
//-->
</script>
```

```
</body>  
</html>
```

- 3. Editor:** Save **lab8-3.htm**.
- 4. Internet Explorer Browser:** Open **lab8-3.htm** in Internet Explorer.
- 5. Internet Explorer Browser:** You should first see a prompt dialog box asking for a color. You need to supply the color in hexadecimal code, as described in the prompt. The default is #FFFFFF, which is white. (Some example colors are #FF0000 for red, #00FF00 for green, and #0000FF for blue.)
- 6. Internet Explorer Browser:** The next dialog will ask you to enter your name. After you enter a name and click **OK**, you will see a dialog advising you that a cookie will be set when you proceed, and asking if you would rather delete it, as shown in Figure 8-16.

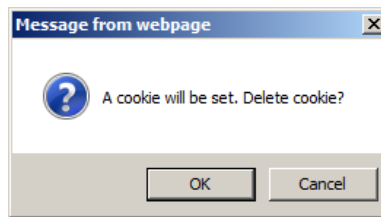


Figure 8-16: Dialog asking whether to delete cookie

- 7. Internet Explorer Browser:** For this step, click **Cancel** to allow the cookie to be set. The next message, as shown in Figure 8-17, will specify the cookie you just accepted based on your answers to the questions.



Figure 8-17: Cookie accepted

- 8. Internet Explorer Browser:** Click **OK** to continue. You will see two more dialogs specifying the name and color you entered, then you will see a landing page that resembles Fig 8-18 and uses the background color you entered.

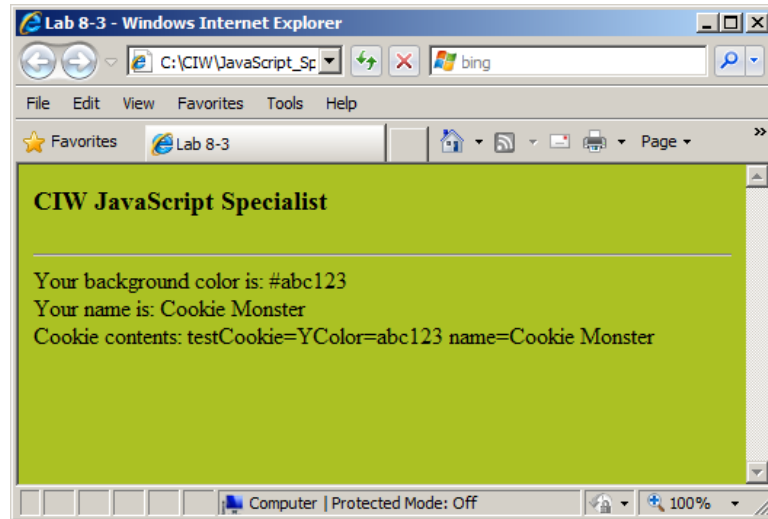


Figure 8-18: Landing page for lab8-3.htm

- 9. Internet Explorer Browser:** Reload the file. Repeat Steps 5 and 6 by entering the same information, but this time, when asked if you want to delete the cookie, click **OK** to delete it. You will see the dialog shown in Figure 8-19.

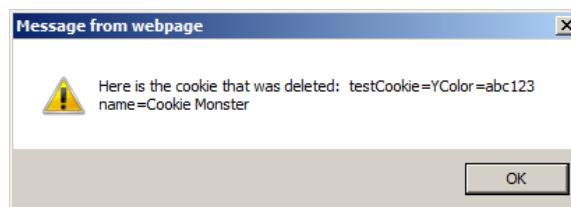


Figure 8-19: Cookie deleted

- 10. Internet Explorer Browser:** Click **OK** or **Yes**. This sets the expired cookie, which replaces (deletes) the existing cookie with the same name. Because this new cookie has an expired date, it is automatically deleted from your system when you restart Internet Explorer.
- 11. Internet Explorer Browser:** Click **OK** to continue. The cookie is gone. You will see the same dialogs recapping your entries, then the same landing page that specifies your information.
- 12. Firefox Browser:** Open **lab8-3.htm** in Firefox, and perform the same steps to accept the cookie. It should perform as it did in Internet Explorer. However, when you click **OK** to delete the cookie, notice that no cookie information appears in the prompts or the final page. The reason for this is that Firefox disposes of the cookie immediately; it will not run the expired cookie at all. By contrast, Internet Explorer will retain the expired cookie for the session, displaying its information now, then delete it when you close the browser.

In this lab, you learned how to set a cookie. You also learned that an easy method for deleting a cookie is to replace it with an expired one.



Lab 8-4: Setting passwords with cookies

In this lab, you will use a cookie that allows access to a second XHTML page. If the cookie is present in the password-protected page and the user enters the correct password value, then the user will be allowed to view the page. If the password is incorrect, the user will be returned to the previous page.

- 1. Editor:** Open the **lab8-4.htm** file from the Lesson_8 folder of the Student_Files directory.
- 2. Editor:** Scroll down in the source code and examine the following code:

```
<form name="myForm" id="myForm">
<input type="text" name="pWord" />
<p>
<input type="button" value="Submit" onclick="storePass(this.form);" />
<input type="Reset" />
</p>
</form>
```

- 3. Editor:** Locate the existing `<script>` tags in the `<head>` section of the document. Locate the comment that reads as follows:

```
// Create cookie here
```

Create a cookie named `password` and assign as its value the user's entry in the `pWord` text box.

- 4. Editor:** The following code shows the `storePass()` function before your changes:

```
<script language="JavaScript">
<!--
function storePass(form) {
    // Create cookie here

    alert(document.cookie);
    location.href = "lab8-4a.htm";
}
//-->
</script>
```

- 5. Editor:** Save **lab8-4.htm**.
- 6. Editor:** Open **lab8-4a.htm** from the Lesson_8 folder of the Student_Files directory. Examine the following code in the `<head>` section of the file:

```
<script language="JavaScript">
<!--
all = document.cookie;
if (all.indexOf("password=hello") != -1){
    alert("You entered the correct password. Proceed.");
} else {
    alert("Incorrect password. Return and re-enter.");
    location.href = "lab8-4.htm";
}
//-->
</script>
```

- 7. Editor:** Close **lab8-4a.htm**.

8. **Browser:** Open **lab8-4.htm**. Your screen should resemble Figure 8-20.

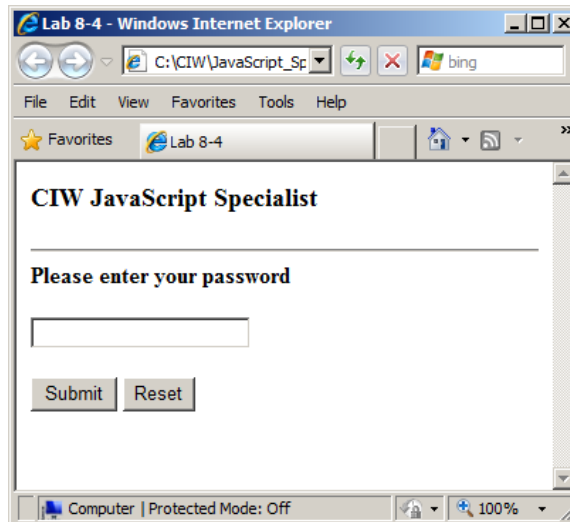


Figure 8-20: Page for lab8-4.htm

9. **Browser:** Enter the password **hello** and click the **Submit** button. This calls the `storePass()` function that stores your password into a cookie. Accept the cookie (if you are warned). You should then see the alert dialog box shown in Figure 8-21.



Figure 8-21 Alert dialog box

10. **Browser:** The alert shows the password `name=value` pair that you entered in the page. (It may also show information from a previous cookie used in another lab; that information will disappear after the associated cookie is cleared from memory.) Click **OK** to continue. You should see another alert with the cookie's `name=value` pair, and then the alert dialog box shown in Figure 8-22.

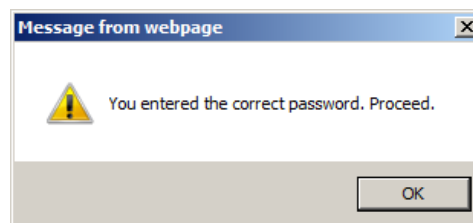


Figure 8-22: Alert dialog box

11. **Browser:** Click **OK** to continue. You will see the password-protected page, as shown in Figure 8-23.

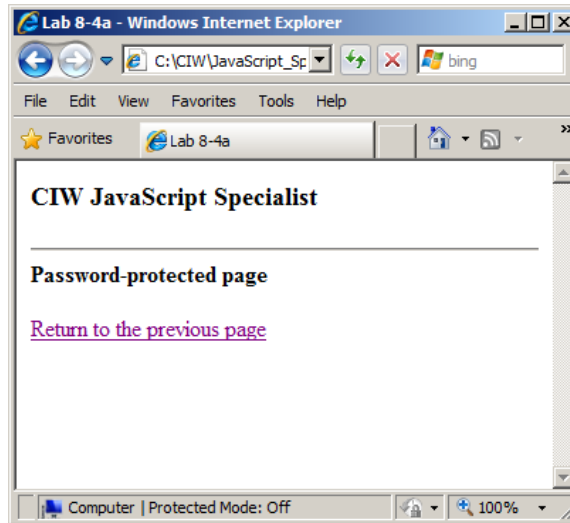


Figure 8-23: Password-protected page (lab8-4a.htm)

- 12. Browser:** To ensure that your script works for both correct and incorrect passwords, click the **Return** link to return to the previous page.
- 13. Browser:** Enter an incorrect password and click the **Submit** button. After the alert displaying the cookie's `name=value` pair, you should see the message shown in Figure 8-24.

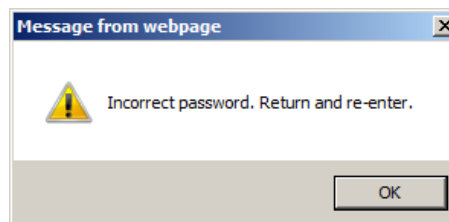


Figure 8-24: Alert dialog box

- 14. Browser:** Click **OK** to continue. Rather than seeing the password-protected page (file `lab8-4a.htm`), you should be returned to the page where you entered your password (file `lab8-4.htm`).

Tech Note: This lab is for demonstration purposes only. Any sensitive content in a Web application should not be protected with the mechanism shown in this lab.



Lab 9-1: Creating a custom object

In this lab, you will create and instantiate a custom object. You will then write code to display data pertaining to the custom object.

- 1. Editor:** Open the **lab9-1.htm** file from the Lesson_9 folder of the Student_Files directory.
- 2. Editor:** Locate the comment that reads as follows:

```
// Complete the employeeObject constructor
```
- 3. Editor:** Complete the `employeeObject` constructor function that has been started for you. Add the properties that you see listed in the `employeeObject` constructor signature. Also add a method named `showEmployee`.
- 4. Editor:** Locate the comment that reads as follows:

```
// Instantiate 3 instances of employeeObject
```
- 5. Editor:** Instantiate three instances of `employeeObject`. Use the `employees` array that has been declared for you. Note that the first element of the `employees` array has already been defined. Make sure you start your array index numbers with 1 for the first employee. The values for the `employeeObject` properties have been provided for you.
- 6. Editor:** Locate the comment that reads as follows:

```
// Complete the showEmployee() function
```
- 7. Editor:** Complete the `showEmployees()` function. Use the `info` variable that has been declared for you. Use the `+=` operator to build a string containing the text **Employee:**, **Department:** and **Extension:** with the appropriate `employeeObject` data concatenated in the appropriate locations. Concatenate a line break character after the name and department information.
- 8. Editor:** An `alert()` method has been defined in the `showEmployees()` function that will display the `info` variable.
- 9. Editor:** Examine the rest of the code, then save **lab9-1.htm**.
- 10. Browser:** Open **lab9-1.htm**. Your screen should resemble Figure 9-7.

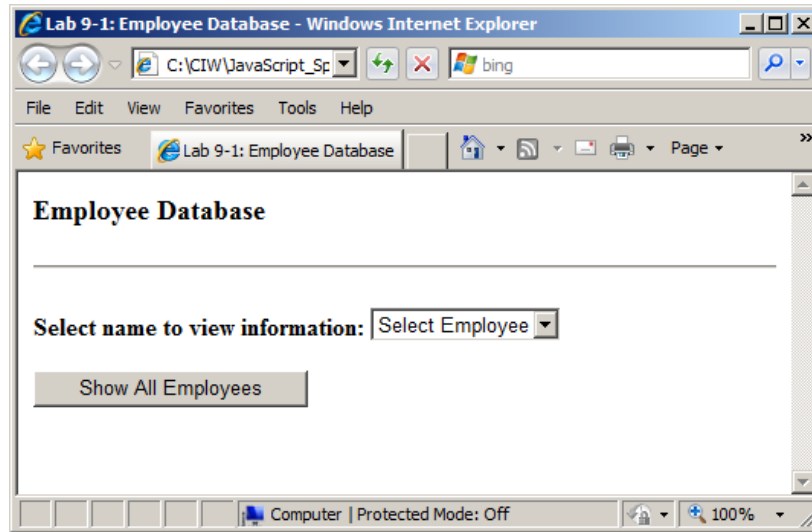


Figure 9-7: Page for Lab9-1.htm

- 11. Browser:** Select the first name from the drop-down menu. You should see an alert dialog box as shown in Figure 9-8. If you do not, check the code you entered in the lab9-1.htm file.

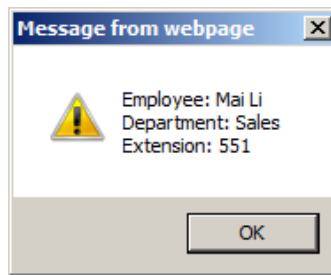


Figure 9-8: Alert displaying employee information

- 12. Browser:** Click the **Show All Employees** button. You should see an alert dialog box as shown in Figure 9-9.

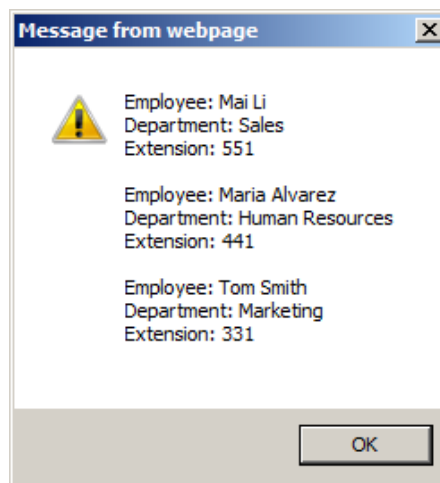


Figure 9-9: Alert showing all employees

13. Browser: Test all the names in the drop-down menu. Ensure that the proper information is displayed for each employee. If the proper data is not displayed, check the code you entered in the lab9-1.htm file.

In this lab, you gained hands-on experience in creating, instantiating and displaying the data for a JavaScript custom object. You added code to a constructor function to make it a template for your custom objects. You added code to instantiate instances of your custom object, populating the object's properties with actual values. You also added code that extracted and displayed the data from your custom object.



Lab 10-1: Redirecting a page based on user input with `getElementById`

In this lab, you will direct users to a Web page based on their input, create a global variable, and use the `getElementById` method in conjunction with radio buttons.

- Editor:** From the Lesson 10 folder in your student lab files, open **lab10-1.htm**. Study the code in the file, which should match the following:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Lab 10-1</title>

<script type="text/javascript">
<!--
function changeButton()
{
    if(document.getElementById('radiobuttonWebdev').checked === true)
    {
        radioButton = document.getElementById('radiobuttonWebdev').value;
        var header = document.getElementById('radiobuttonWebdev').value.innerHTML
= "Web Developer!";
        alert("Hello " + header);
    }
    if(document.getElementById('radiobuttonGovernment').checked === true)
    {
        radioButton = document.getElementById('radiobuttonGovernment').value;
        var header = document.getElementById('radiobuttonWebdev').value.innerHTML
= "Uncle Sam!";
        alert("Hello " + header);
    }
    getTest();
}
-->
</script>
</head>

<body>
<h3>CIW JavaScript Specialist</h3>
<hr />
<form>
<p>Please select your favorite type of Web site:</p>
<input id="radiobuttonWebdev" type="radio" name="group1"
value="Web_Development" checked>Web Development</input><br />

<input id="radiobuttonGovernment" type="radio" name="group1"
value="Government">Government</input><br />

<input id="a" type="radio" name="group1" value="A"> A </input><br />
<input id="b" type="radio" name="group1" value="B"> B </input><br />
<input id="c" type="radio" name="group1" value="C"> C </input><br />

<script type="text/javascript">
<!--
function getTest()
{
    document.write("<h3>CIW JavaScript Specialist</h3>");
    document.write("<hr />");

    if(radioButton == "Web_Development")
    {
```

```

        document.write("<p>Here's a great site for Web developers:</p><a
href=http://www.CIWcertified.com/>CIW Certified</a>");
    }
    if(radioButton == "Government")
    {
        document.write("<p>Here's a great site if you're interested in U.S.
government:</p><a href=http://www.whitehouse.gov/>The White House</a>");
    }
}
-->
</script>
<p>
<input type="button" onclick="changeButton()" value="Click Here" />
</p>
</form>
</body>
</html>

```

2. **Browser:** Now open the file **lab10-1.htm** in your browser. It should resemble Figure 10-1.

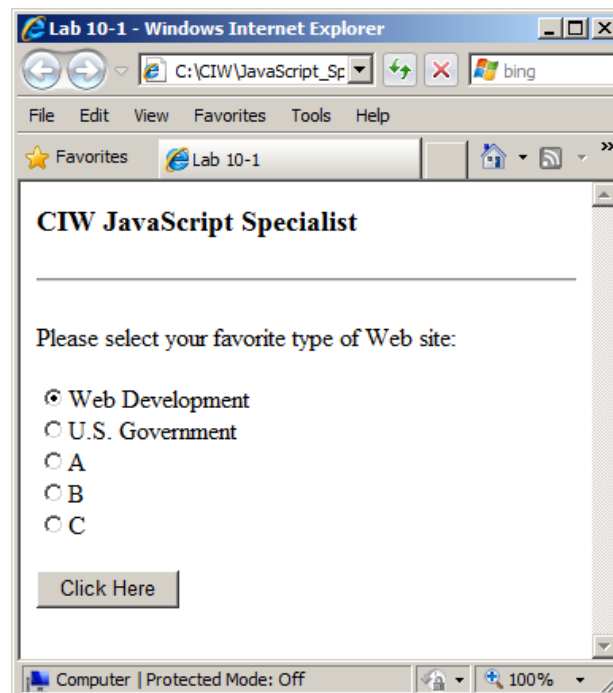


Figure 10-1: Page for lab10-1.htm

3. **Browser:** Select the first or second option (Web Development or U.S. Government) and click the button. You will be redirected to page with a link related to your choice.
4. Consider the following points about this code:
- You can create a global variable in JavaScript by assigning the variable to the window. When you do this, the variable is assigned to the entire window and not just to the function it is in. This practice is especially useful when using multiple functions.
 - When you look over the code, take careful note of the case-sensitivity of the methods. The most common error made in JavaScript programming is neglecting to verify and follow proper case-sensitivity.

5. **Editor:** Open the **lab10-1.htm** file. Notice the "a", "b" and "c" id values in the radio button code. Replace these with some of your favorite Web sites. Be sure to watch your letter case. Save the file after making changes to it.
6. **Editor:** For each redirect page, create an entire Web page from it using the `document.write` statements, as well as other useful items such as images and headings. You have learned to do this in previous lessons. Save the file.
7. **Browser:** Open your new file in at least two different browsers and test to see if your script works as expected.

This lab demonstrated some skills you will use throughout your Web development career. You used the `getElementById` method to gather input. You also learned a way to set global variables by simply assigning the window property to a variable. This allows the variable to be used in other functions, not just the function in which it was created and manipulated. Of course, you can still create global variables by creating a variable outside of a function, but this will reset to the default with each different function.



Lab 10-2: Changing the DOM using `getElementsByName`

In this lab, you will use the `getElementsByName` method to grade a quiz. Typically, after the quiz is graded, the scores would be uploaded to a database. Remember that array elements begin with the number 0.

1. **Editor:** Open file **lab10-2.htm**. Study the code, which appears as follows:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"␣http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Lab 10-2</title>

<script type="text/javascript">
function getValues(objName)
{
var rightanswers=0;
var txt=document.getElementsByName("txtField");
if(txt[0].value == "javascript")
{alert("You got Question 1 right")
rightanswers++;
};
}
</script>
</head>

<body>
<h3>CIW JavaScript Specialist</h3>
<hr />
<p>
Which certification are you seeking?
</p>
<p>
<select name="txtField" id=Text1>
<option value="javascript">JavaScript Specialist</option>
<option value="canoeing">Extreme Canoeing</option>
<option value="moviemaking">Movie Production</option>
</select>
</p>
<p>
Which organization awards the certification?
</p>
<p>
<select name="txtField" id=Text2>
<option value="CIW">CIW</option>
<option value="YMCA">YMCA</option>
<option value="DGA">DGA</option>
</select>
</p>
<p>
What is the best grade you can score?
</p>
<p>
<select name="txtField" id=Text3>
<option value="100">100</option>
<option value="B">B</option>
<option value="pass">Pass</option>
</select>
</p>
<p>
```

```



```

- 2. Browser:** Now open file **lab10-2.htm** in your browser to run the code. Your page should resemble Figure 10-3. Select answers to the questions, then click the Submit button to check your answers. You will find that only the first question gets scored.

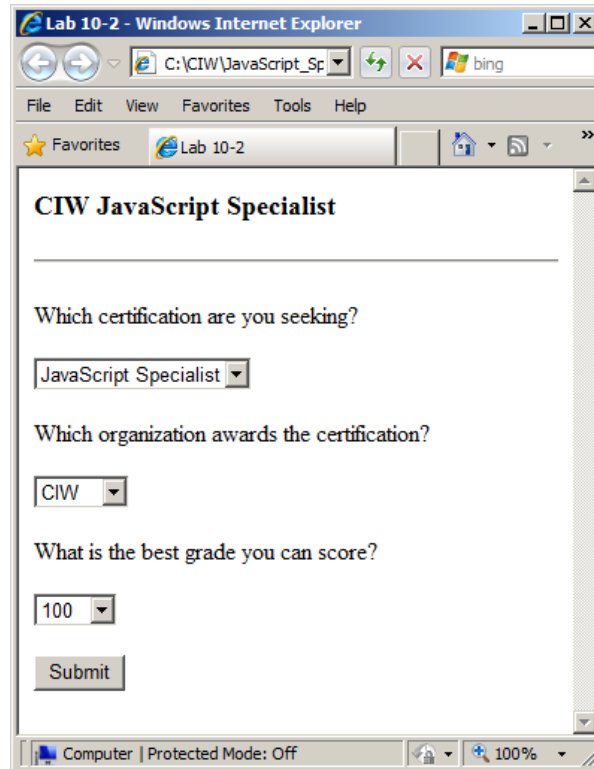


Figure 10-3: Quiz page for lab10-3.htm

- 3. Editor:** After the `if` statement that scores Question 1, enter the proper code to grade Questions 2 and 3. The correct answers are CIW and 100, respectively. Also add a `document.write` statement that will show the user's score on a new page. Try this step first on your own. The correct code follows in bold:

```

if(txt[0].value == "javascript")
    {alert("You got Question 1 right")
    rightanswers++;
    };
if(txt[1].value == "CIW")
    {alert("You got Question 2 right")
    rightanswers++;
    };
if(txt[2].value == 100)
    {alert("You got Question 3 right")
    rightanswers++;
    };
document.write
document.write("You got " + rightanswers + " out of 3 answers correct");

```

- 4. Browser:** Test your page in the browser after completing your code. Notice that you see an alert for questions that you answer correctly. Your score page should resemble Figure 10-4.

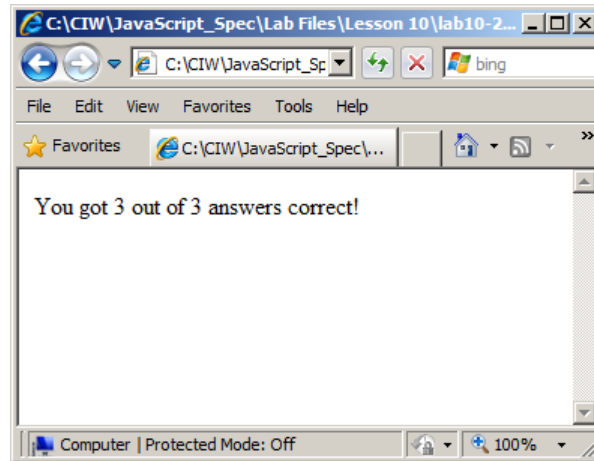


Figure 10-4: Score page generated by `document.write`

- 5. Editor:** Remember that this function runs after the page has rendered, so `document.write` will overwrite the current information (the quiz page) with the single line of text you specified, but no other page elements. However, you can use the `document.write` to ensure a full page for users to view, instead of just a single statement. When time permits, code a full page for the `document.write`. Be sure to properly format the `<html>`, `<body>` and `<title>` tags.

In this lab, you used a very powerful command — `getElementsByName` — and you learned how it can be used to separate data using an array.



Lab 10-3: Getting, setting and removing X/HTML attributes

In this lab, you will modify X/HTML attributes using the `getAttribute`, `setAttribute` and `removeAttribute` methods.

- Editor:** Open the file **lab10-3.htm**. It is the same code from the previous example.
- Editor:** Edit the script by adding the code shown in bold, then save the file:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"⤴http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Example: Getting Attributes</title>
</head>
<body>

<h3>CIW JavaScript Specialist</h3>
<hr />
<div id="SetDiv" special_attribute="CIW rules!" align="left">
<p>What are the attributes of this div tag?</p>
</div>
<input type="button" value="Press Me!" onclick="getValues()" />
<p>Please click the button</p>
<script type="text/javascript">
    var div = null;
    function getValues()
    {
        if (div == null)
        {
            div = document.getElementById("SetDiv");
        }
        alert(div.id);
        alert(div.special_attribute);
        alert(div.align);
        changeValues();
    }

    function changeValues(){
        if (div == null) {
            div = document.getElementById("SetDiv");
        }
        var d = document.getElementById("SetDiv");
        d.setAttribute("align", "center");
        alert(div.align);
        RemoveValues();
    }

    function RemoveValues(){
        if (div == null) {
            div = document.getElementById("SetDiv");
        }
        var d = document.getElementById("SetDiv");
        d.removeAttribute("align");
        alert(div.align);
        ChangeValuesBack();
    }

    function ChangeValuesBack(){
        if (div == null) {
            div = document.getElementById("SetDiv");
        }
        var d = document.getElementById("SetDiv");
        d.setAttribute("align", "left");
        alert(div.align);
    }
}
```

```
</script>  
</body>  
</html>
```

- 3. Browser:** Run this script in Internet Explorer. The alert boxes will show you where the attributes were changed, removed and reset. You can see the page content change as the attributes change where the text between the <div></div> tags dynamically moves to the center, as shown in Figure 10-7.

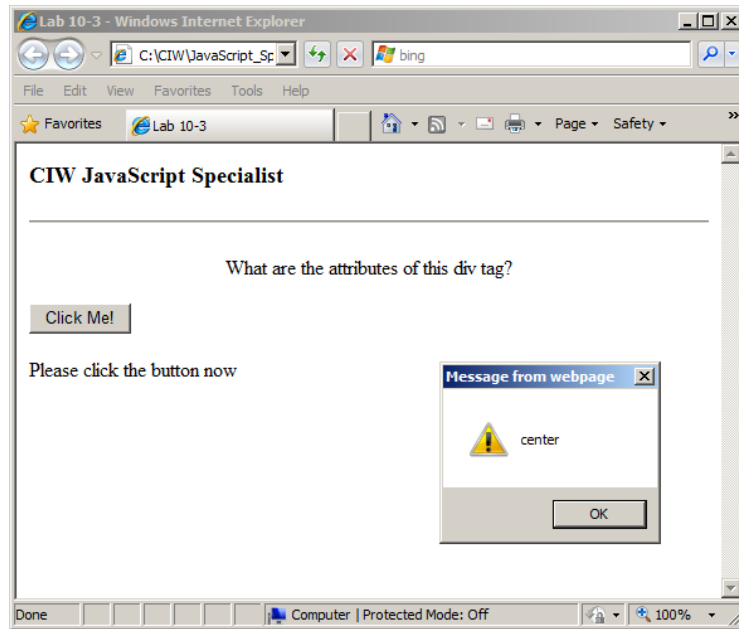


Figure 10-7: Page content with attributes accessed and set



Lab 11-1: Loading a JavaScript library and running a library script

In this lab, you will load jQuery in your X/HTML page and test it with a basic command. A jQuery library script is an external file so it needs to be linked to the X/HTML file via the `<script>` tag's `src` attribute. The `src` attribute indicates that the jQuery library script has been placed in the same directory as the X/HTML page.

- 1. Editor:** From your Lesson_11/Lab11-1 folder, open the **lab11-1.htm** file.
- 2. Editor:** Study the contents of this file. This is a copy of the full directory from the jQuery library that includes all dependent files so they are easy to find. It is good practice to use the entire library directory for this so you do not miss any files or pages your code might need.

Tech Note: You never make changes to the library file. You simply attach this file to your X/HTML page as a reference for the portions of code that you actually use. The library file is a foundation for its scripts, and it provides your page with a syntax grammar so your page knows how to process the specific script(s) that you use. The specific script is attached to your X/HTML page in a separate file.

- 3. Editor:** In the `<head>` section, enter the following `<script>` tag shown in bold, then save the file. This line of code will link the jQuery library to the X/HTML page:

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<script src="jquery-1.4.4.min.js" type="text/javascript" charset="utf-8"></script>
<title>Lab 11-1</title>
</head>
```

- 4. Editor:** Enter the following code after the opening `<body>` tag, then save the file:

```
<body>
<h2>Resize your text with jQuery</h2>
<input type="button" value="Larger" id="large" />
<input type="button" value="Smaller" id="small" />
<p>You can enlarge or shrink the font size of text on your X/HTML page using
this script, as long as the text is within the paragraph tags.</p>
This is outside the paragraph tags and will not change!
```

- 5. Editor:** Create a file named **script.js** and place it in the same directory as your lab11-1.htm file. The script.js file is where you will place and save the jQuery commands that you want to use in your page. Having a script.js file is useful to help you store and organize the library code you use, and to reuse the code if you have the opportunity.
- 6. Editor:** Enter the following code into the script.js file, then save the file:

```
$(function(){
  $('input').click(function(){
    var ourText = $('p');
    var currFontSize = ourText.css('fontSize');
    var finalNum = parseFloat(currFontSize, 10);
    var stringEnding = currFontSize.slice(-2);
    if(this.id == 'large') {
      finalNum *= 1.2;
    }
    else if (this.id == 'small'){
      finalNum /=1.2;
    }
  });
});
```

```
        }
        ourText.css('fontSize', finalNum + stringEnding);
    });
});
```

This code comes from the jQuery library. Consider some of this code's notable functions:

- The `$(function())` keyword tells the interpreter that this is a jQuery script.
- The `var ourText = $('p');` statement tells the interpreter that this script will work only on the `<p>` tag.
- The program does the text resizing with this portion of the code, by increasing or decreasing the font size by 1.2 when the appropriate button is clicked (note the ID of the button):

```
if(this.id == 'large') {
    finalNum *= 1.2;
}
else if (this.id == 'small'){
    finalNum /=1.2;
}
```

Tech Note: The dollar sign (\$) is an alias for jQuery. Although you can use the \$ sign for other things in code, it is standard syntax for jQuery. Most jQuery scripts use \$ signs for the variables that they implement.

- 7. Editor:** Open the **lab11-1.htm** file. Add the following code (shown in bold) underneath the existing `<script></script>` section that references the jQuery directory, then save the file. This line places a reference to the script file you just created:

```
<script src="jquery-1.4.4.min.js" type="text/javascript" charset="utf-8">
</script>
<script src="script.js" type="text/javascript" charset="utf-8">
</script>
<title>Lab 10-1</title>
```

- 8. Browser:** Open the file **lab11-1.htm** to run the script. Click the **Larger** and **Smaller** buttons, and you will see the text that was formatted within the `<p></p>` tags grow and shrink on the fly. This is just the beginning of the power of libraries.



Lab 11-2: Using CSS and JavaScript to create a basic slideshow

In this lab, you will use jQuery to interact with both JavaScript and CSS to create a basic slideshow. You will load two external scripts. The first one is the jQuery library, and the second one is the script.js file that you will create. You will also reference a third file: a Cascading Style Sheets (CSS) file that will apply formatting to the page.

- 1. Editor:** Open the **lab11-2.htm** file from the Lesson_11/Lab11-2 folder of your student files.
- 2. Editor:** Create a blank file called **script.js**.
- 3. Editor:** Enter the following code as shown in bold in the <head></head> section, then save the file:

```
<link rel="stylesheet" href="css.css" type="text/css" media="screen"
charset="utf-8"/>
<script src="jquery-1.4.4.min.js" type="text/javascript" charset="utf-
8"></script>
<script src="script.js" type="text/javascript" charset="utf-8"></script>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

Reminder: This code creates the references to the jQuery library file and the script file you are creating, respectively. The reference to the jQuery library provides the basic syntax rules that are necessary to run a script from the jQuery library. The script file you create will use jQuery code.

- 4. Editor:** Open the file **css.css**. The code is as follows:

```
@charset "utf-8";
/* CSS Document */
#slideshow {
    position:relative;
    height:350px;
}
#slideshow IMG {
    position:absolute;
    top:0;
    left:0;
    z-index:8;
}

#slideshow IMG.active {
    z-index:10;
}

#slideshow IMG.last-active {
    z-index:9;
}
```

- 5. Editor:** Study this code. The portion that bears mention is the **z-index**, which is a CSS property that sets the stack order of specific elements. An element with greater stack order always appears on the Web page in front of another element with lower stack order. Basically, this sets up images on a Web page like the pages in a book, in which one is over the other, which is over the next, for as many as there are. In this manner, only one image will be seen at a time —the top image as determined and sequenced by the z-index.

6. **Editor:** Enter the following code as shown in bold into the **script.js** file, then save the file:

```
function slideSwitch() {  
    var $active = $('#slideshow IMG.active');  
  
    if ( $active.length == 0 ) $active = $('#slideshow IMG:last');  
  
    var $next = $active.next().length ? $active.next()  
        : $('#slideshow IMG:first');  
  
    $active.addClass('last-active');  
  
    $next.css({opacity: 0.0})  
        .addClass('active')  
        .animate({opacity: 1.0}, 1000, function() {  
            $active.removeClass('active last-active');  
        });  
}  
  
$(function() {  
    setInterval( "slideSwitch()", 5000 );  
});
```

This script works by calling a method called `slideSwitch()` every 5 seconds. The `slideSwitch()` method takes the active image and rotates it to the bottom of the stack, then activates the next image at the top of the stack, allowing it to be viewed. This function will cycle for as long as it is in the browser.

7. **Browser:** Open the file **lab11-2.htm**. When the script runs, you will see the images rotate every 5 seconds.

In this lab, you used the jQuery library to help you create a basic slideshow that uses JavaScript and CSS. To create this program from scratch with JavaScript would have taken many hours of trial and error. But with jQuery's help, this can be done in less than an hour, with bug-free code.



Lab 11-3: Loading, testing and editing a library plug-in

In this lab, you will load, test and edit a jQuery plug-in that validates form fields.

- 1. Editor:** Open the file **lab11-3.htm**. Study the code, which appears as follows:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Lab 11-3</title>
<style type="text/css">
* { font-family: Verdana; font-size: 96%; }
label { width: 10em; float: left; }
label.error { float: none; color: red; padding-left: .5em; vertical-align:
top; }
p { clear: both; }
.submit { margin-left: 12em; }
em { font-weight: bold; padding-right: 1em; vertical-align: top; }
</style>
</head>

<body>
<form class="cmxform" id="commentForm" method="get" action="">
<fieldset>
<legend>A simple comment form with submit validation and default
messages</legend>
<p>
<label for="cfname">First Name</label>
<em>*</em><input id="cfname" name="fname" size="25" />
</p>
<p>
<label for="clname">Last Name</label>
<em>*</em><input id="clname" name="lname" size="25" />
</p>
<p>
<label for="cemail">E-Mail</label>
<em>*</em><input id="cemail" name="email" size="25" />
</p>
<p>
<label for="curl">URL of Your Web Site</label>
<em> </em><input id="curl" name="url" size="25" />
</p>
<p>
<label for="cnumber">Years Your Site Has Been in Business</label>
<em> </em><input id="cnumber" name="url" size="25" />
</p>
<p>
<label for="ccomment">Your Comment</label>
<em>*</em><textarea id="ccomment" name="comment" cols="22"></textarea>
</p>
<p>
<label for="ccheckbox">Please check this box to agree to our
terms</label>
<em>*</em><input type="checkbox" name="ccheckbox" />
</p>
<p>
<input class="submit" type="submit" value="Submit" />
</p>
</fieldset>
</form>
</body>
</html>
```

2. **Editor:** Enter the appropriate reference for the jQuery library and plug-in information in the <head></head> section of the **lab11-3.htm** file, as shown in bold, then save the file. Ensure that the plug-in is entered after the library:

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Lab 11-3</title>
<script src="jquery-1.4.4.min.js" type="text/javascript"></script>
<script src="jquery.validate.js" type="text/javascript"></script>
<style type="text/css">
* { font-family: Verdana; font-size: 96%; }
label { width: 10em; float: left; }
label.error { float: none; color: red; padding-left: .5em; vertical-align:
top; }
p { clear: both; }
.submit { margin-left: 12em; }
em { font-weight: bold; padding-right: 1em; vertical-align: top; }
</style>
</head>
```

3. **Browser:** Test your XHTML page. You may get warnings, depending on your browser, but the browser should display the page as shown in Figure 11-1.

Figure 11-1: Basic form page without validation

4. **Editor:** Now install the plug-in commands in the <head></head> section of the **lab11-3.htm** file (as shown in bold) to make the plug-in accessible to your page, then save the file:

```
</style>
<script>
$(document).ready(function(){
$("#commentForm").validate();
});
</script>
</head>
```

This step enables the plug-in to work. Now you can begin validating.

Tech Note: When you are developing code in any language, the best practice is "code a little, test a lot." In this case, you will ensure that the first validation works properly before you move on to the second validation, and so forth. This way, you will not be attempting to troubleshoot multiple errors at the same time.

- 5. Editor:** Enter your the code for the first field validation into the **lab11-3.htm** file. You will verify that the First Name field was submitted with text and is at least two characters long. Enter the following code, then save the file:

```
<legend>A simple comment form with submit validation and default
messages</legend>
<p>
  <label for="cfname">First Name</label>
  <em>*</em><input id="cfname" name="fname" size="25" class="required"
minlength="2" />
</p>
```

Notice that the label and the id match, that the class is required, and the minlength is 2. These are all documented in the script.

- 6. Browser:** Test the script. Open the Web page, and submit the form without entering anything into the First Name field. If the script ran correctly, you should see the message that appears in Figure 11-2.



Figure 11-2: First Name field is validated

- 7. Browser:** The message you see comes from the plug-in script evaluating the fields as directed by the script. Now try typing two letters into the First Name field and submitting the form again. The red warning will disappear.

Tech Note: Ensure that every ID is different and that you test the progress step-by-step. It is far easier to fix an error when you know exactly where it is, rather than testing large chunks of code with multiple actions all at once, and having an error that will take hours to track down.

- 8. Editor:** Enter the code for each validation line by line, testing each line as you go and saving the file each time. When you are finished with all the validations, the code should read as follows:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Lab 11-3</title>
<script src="jquery-1.4.4.min.js"></script>
<script type="text/javascript" src="jquery.validate.js"></script>
<style type="text/css">
* { font-family: Verdana; font-size: 96%; }
label { width: 10em; float: left; }
label.error { float: none; color: red; padding-left: .5em; vertical-align:
top; }
p { clear: both; }
.submit { margin-left: 12em; }
em { font-weight: bold; padding-right: 1em; vertical-align: top; }
</style>
<script>
$(document).ready(function(){
$("#commentForm").validate();
```

```

});
</script>
</head>

<body>
  <form class="cmxform" id="commentForm" method="get" action="">
    <fieldset>
      <legend>A simple comment form with submit validation and default
messages</legend>
      <p>
        <label for="cfname">First Name</label>
        <em>*</em><input id="cfname" name="fname" size="25" class="required"
minlength="2" />
      </p>
      <p>
        <label for="clname">Last Name</label>
        <em>*</em><input id="clname" name="lname" size="25" class="required"
minlength="2" />
      </p>
      <p>
        <label for="cemail">E-Mail</label>
        <em>*</em><input id="cemail" name="email" size="25" class="required
email" />
      </p>
      <p>
        <label for="curl">URL of Your Web Site</label>
        <em> </em><input id="curl" name="url" size="25" class="url" value="" />
      </p>
      <p>
        <label for="cnumber">Years Your Site Has Been in Business</label>
        <em> </em><input id="cnumber" name="url" size="25" class="digits"
value="" />
      </p>
      <p>
        <label for="ccomment">Your Comment</label>
        <em>*</em><textarea id="ccomment" name="comment" cols="22"
class="required"></textarea>
      </p>
      <p>
        <label for="ccheckbox">Please check this box to agree to our
terms</label>
        <em>*</em><input type="checkbox" name="ccheckbox" class="required"/>
      </p>
      <p>
        <input class="submit" type="submit" value="Submit" />
      </p>
    </fieldset>
  </form>
</body>
</html>

```

- 9. Browser:** When you test the script in your browser, the page will display messages directing you to revise your input as appropriate (based on whether you entered invalid data or no data in required fields), as shown in Figure 11-3.

A simple comment form with submit validation and default messages

First Name	*	<input type="text" value="O"/>	Please enter at least 2 characters.
Last Name	*	<input type="text"/>	This field is required.
E-Mail	*	<input type="text" value="@ozzy.com"/>	Please enter a valid e-mail address.
URL of Your Web Site		<input type="text" value="ozzy.com"/>	Please enter a valid URL.
Years Your Site Has Been in Business		<input type="text" value="many"/>	Please enter only digits
Your Comment	*	<input type="text"/>	This field is required.
Please check this box to agree to our terms	*	<input type="checkbox"/>	This field is required.

Figure 11-3: Completed form validation script result

When testing your script, the best practice is to enter one line at a time, test to see if it works correctly as expected under all circumstances, and then move on to the next line. If your project has a separate site designer who outlined the X/HTML page, then you would send your script back to that person for further testing, and to ensure that the specifications you were given match the results you have returned.



Lab 12-1: Using AJAX to dynamically edit the DOM with button clicks

In this lab, you will use AJAX to dynamically edit the DOM and allow the user to change the page's appearance by clicking buttons.

- Editor:** From the Lesson_12/Lab12-1 folder of your student files, open the file **lab12-1.htm**. Study the code, which is as follows:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<title>Lab 12-1</title>
<script language="JavaScript" src="ajax.js"></script>

<script type="text/javascript">
function ChangeData(str)
{
    document.getElementById('MyID').innerHTML = str;
}
</script>
</head>

<body>
<h3>CIW JavaScript Specialist</h3>
<hr />
    <h2>AJAX Button Clicker</h2>
    <p>This script shows how to use multiple external scripts that load
dynamically when you click a button.</p>
    <input type="button" value="Button 1" onclick="loadScript('Data1.js')" />
    <input type="button" value="Button 2" onclick="loadScript('Data2.js')" />
    <input type="button" value="Button 3" onclick="loadScript('Data3.js')" />
    <p>
    <div id="MyID">&nbsp;</div>
    </p>
    <hr />
</body>
</html>
```

The most important portion of this script to notice is the `onclick` function: Upon the click of each button, the `loadScript` method will display the appropriate text for the button clicked.

- Editor:** Now open the **ajax.js** file and study the code. This file is attached to the XHTML file and contains all the actual code. Included in this code is the following function:

```
<script type="text/javascript">
function ChangeData(str)
{
    document.getElementById('MyID').innerHTML = str;
}
</script>
```

- Editor:** Continue to study the **ajax.js** file. Following is the code for the `loadScript` and `loadData` functions:

```
function loadScript(scriptURL)
{
    var newScript = document.createElement("script");
    newScript.src = scriptURL;
```

```
        document.body.appendChild(newScript);
    }

    function loadData(URL)
    {
    // Create the XML request
        xmlReq = null;
        if(navigator.appName == "Microsoft Internet Explorer") xmlReq = new
ActiveXObject("Microsoft.XMLHTTP");
        else xmlReq = new XMLHttpRequest();

        if(xmlReq==null) return; // Failed to create the request

        xmlReq.open("GET", URL, true);

    // Anonymous function to handle changed request states
        xmlReq.onreadystatechange = function()
        {
            switch(xmlReq.readyState)
            {
            case 0: // Uninitialized
                break;
            case 1: // Loading
                break;
            case 2: // Loaded
                break;
            case 3: // Interactive
                break;
            case 4: // Done!
                // Retrieve the data between the <quote> tags
                ChangeData(xmlReq.responseText);
                break;
            default:
                break;
            }
        }

    // Make the request
        xmlReq.send (null);
    }
}
```

4. **Editor:** Most of the methods you see here have been shown previously. In the following section of code, `document.createElement` creates a new element called "script", then assigns it the variable name "newScript". It is then used with the method `appendChild(newScript)` to enter the results of the clicked button. The code appears as follows:

```
function loadScript(scriptURL)
{
    var newScript = document.createElement("script");
    newScript.src = scriptURL;
    document.body.appendChild(newScript);
}
}
```

5. **Editor:** The next portion of code creates the AJAX `XMLHttpRequest`, and also performs the browser check to see which browser is used (Internet Explorer or another) in order to maintain browser compatibility. The last line begins the communication with the XML data:

```
        xmlReq = null;
        if(navigator.appName == "Microsoft Internet Explorer") xmlReq = new
ActiveXObject("Microsoft.XMLHTTP");
        else xmlReq = new XMLHttpRequest();
```



```
        if(xmlReq==null) return; // Failed to create the request
xmlReq.open("GET", URL, true);
```

- 6. Editor:** The next portion of code shows where the `onreadystatechange` property is instantiated. It will only work if it is case 4, which means that it is in the fully loaded state:

```
xmlReq.onreadystatechange = function()
{
    switch(xmlReq.readyState)
    {
        case 0: // Uninitialized
            break;
        case 1: // Loading
            break;
        case 2: // Loaded
            break;
        case 3: // Interactive
            break;
        case 4: // Done!
```

- 7. Editor:** Open the file **Data1.js** and study the code. This is the JavaScript code that displays the data that your AJAX application retrieved, which can be XML or text data. In this case, each of the Data.js files calls a text file (instead of XML), whose name corresponds to the associated JavaScript file name:

```
loadData("Data1.txt");
```

- 8. Browser:** Open the file **lab12-1.htm** and test the script. You will see that every time you click a button, the text changes without refreshing the page, based on the loading of the various scripts ('Data1.js', 'Data2.js', 'Data3.js'). Test the script in several different browsers, and it should run the same. AJAX can render quite differently in different browsers, so be sure to test all your AJAX scripts thoroughly.



Lab 12-2: Using AJAX and libraries to create tooltips

In this lab, you will use AJAX, JavaScript, libraries and plug-ins to create aesthetically pleasing tooltips with your code.

- Editor:** From the Lesson_12/Lab_12-2 folder in your student files, open the file **lab12-2.htm** and study the code, which appears as follows:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
  <title>Lab 12-2</title>

  <link rel="stylesheet" href="jquery.cluetip.css" type="text/css" />
  <link rel="stylesheet" href="demo.css" type="text/css" />
</head>
<body>

  <h1 id="top">AJAX with Plug-ins and Add-ons</h1>
  <p><a class="basic" href="test1.html" rel="test1.html">Test 1</a></p>
  <p><a id="sticky" href="test2.html" rel="test2.html">Test 2</a></p>
  <p><a class="custom-width" href="test3.html" rel="test3.html">Test 3</a></p>
  <h4 title="Fancy Title!" id="test4.html">Test 4</h4>
  <p><a class="basic" href="http://www.CIWcertified.com/" title="about this
link:" rel="test5.html">CIW Certification is well worth the effort!</a></p>
</body>
</html>
```

- Browser:** Open the file **lab12-2.htm** and run the script. You will see that the links do work, but no effects are added to the XHTML. The page directs to another XHTML page without any other events.
- Editor:** Add JavaScript and AJAX external scripts to this page by adding the following code to the **lab12-2.htm** file, as shown in bold. Then save the file:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
  <title>Lab 12-2</title>

  <link rel="stylesheet" href="jquery.cluetip.css" type="text/css" />
  <link rel="stylesheet" href="demo.css" type="text/css" />

  <script type="text/javascript" src="ajax_jquery.js"></script>
</head>
<body onload="loadData()">

  <h1 id="top">AJAX with Plug-ins and Add-ons</h1>
  <p><a class="basic" href="test1.html" rel="test1.html">Test 1</a></p>
  <p><a id="sticky" href="test2.html" rel="test2.html">Test 2</a></p>
  <p><a class="custom-width" href="test3.html" rel="test3.html">Test 3</a></p>
  <h4 title="Fancy Title!" id="test4.html">Test 4</h4>
  <p><a class="basic" href="http://www.CIWcertified.com/" title="about this
link:" rel="test5.html">CIW Certification is well worth the effort!</a></p>

</body>
</html>
```

4. **Browser:** Reload the page and run the script again. You will see that each link has a different effect. AJAX is dynamically pulling in jQuery files, which are stored in the *ajax.jquery.js* file instead of referenced in separate `<script>` tags in the XHTML (the DOM is manipulated to create the `<script>` tags on the fly). The XML data file is also stored in the JavaScript file. These files reload the selected elements without refreshing the entire page, so the tooltips appear to pop up.

*Tech Note: Using Firebug to view the source code in the *ajax.jquery.js* file, you can see the DOM as it truly is after JavaScript has fired. You can see that the browser has created all the script tags and placed them in the code for you. You cannot see this in View Source because they are not typed into the initial code; they are created dynamically when the script is run.*

5. **Browser:** Try running this script in some different browsers. Is anything different? It should run similarly in various browsers, with some small differences. However, determining this is the important goal of testing your script in multiple browsers.
6. **Editor:** Open the **jquery.cluetip.js** file. This code is a jQuery plug-in that creates the tooltips.
7. **Browser:** Open the **demo.js** file in another instance of your text editor. This is the file that you (the developer) would create to reference the effects that you choose to use in your Web page. Study the code, which is as follows:

```
$(document).ready(function() {
//default theme
$('a.title').cluetip({splitTitle: '|'});
$('a.basic').cluetip();
$('a.custom-width').cluetip({width: '200px', showTitle: false});
$('h4').cluetip({attribute: 'id', hoverClass: 'highlight'});
$('#sticky').cluetip({sticky: true, closePosition: 'title', arrows: true });
$('#examples a:eq(5)').cluetip({
  hoverClass: 'highlight',
  sticky: true,
  closePosition: 'bottom',
  closeText: '',
  truncate: 60
});
$('a.load-local').cluetip({local:true, hideLocal: true, sticky: true,
arrows: true, cursor: 'pointer'});
$('#clickme').cluetip({activation: 'click', sticky: true, width: 650});
$('ol:first a:last').cluetip({tracking: true});

// jTip theme
$('a.jt:eq(0)').cluetip({
  cluetipClass: 'jtip',
  arrows: true,
  dropShadow: false,
  sticky: true,
  mouseOutClose: true,
  closePosition: 'title',
  closeText: ''
});
$('a.jt:eq(1)').cluetip({cluetipClass: 'jtip', arrows: true, dropShadow:
false, hoverIntent: false});
$('span[title]').css({borderBottom: '1px solid #900'}).cluetip({splitTitle:
'|', arrows: true, dropShadow: false, cluetipClass: 'jtip'});

$('a.jt:eq(2)').cluetip({
  cluetipClass: 'jtip',
  arrows: true,
  dropShadow: false,
  height: '150px',
```

```

        sticky: true,
        positionBy: 'bottomTop'
    });

    $('a.jt:eq(3)').cluetip({local: true, hideLocal: false});

    $('a.jt:eq(4)').cluetip({
        cluetipClass: 'jtip', arrows: true,
        dropShadow: false,
        onActivate: function(e) {
            var cb = $('#cb')[0];
            return !cb || cb.checked;
        }
    });

    // Rounded Corner theme
    $('ol.rounded a:eq(0)').cluetip({sticky: true, splitTitle: '|', dropShadow:
false, cluetipClass: 'rounded', showTitle: false});
    $('ol.rounded a:eq(1)').cluetip({cluetipClass: 'rounded', dropShadow: false,
showTitle: false, positionBy: 'mouse'});
    $('ol.rounded a:eq(2)').cluetip({cluetipClass: 'rounded', dropShadow: false,
showTitle: false, positionBy: 'bottomTop', topOffset: 70});
    $('ol.rounded a:eq(3)').cluetip({cluetipClass: 'rounded', dropShadow: false,
sticky: true, ajaxCache: false, arrows: true});
    $('ol.rounded a:eq(4)').cluetip({cluetipClass: 'rounded', dropShadow:
false});
});

//Unrelated to clueTip -- just for the demo page...

$(document).ready(function() {
    $('div.html, div.jquery').next().css('display',
'none').end().click(function() {
        $(this).next().toggle('fast');
    });

    $('a.false').click(function() {
        return false;
    });
});

// inserting jQuery UI Themeswitcher tool

$('#themeswitcher').themeswitcher({loadTheme: 'UI Lightness'});

```

Tech Note: The file `jquery.cluetip.js` is the base plug-in. The file `demo.js` is the file that you (the developer) create to reference the effects that you choose to apply to your Web page. This separation of duties makes the script extremely lightweight, because the Web page will only reference the needed functions. Then you can place your scripts in a repository where you can easily reuse the effects you create.

- 8. Editor:** The effects for this plug-in are defined by the defaults in the `jquery.cluetip.js` file. You will find the defaults toward the bottom of the script (lines 538 to 573).
- 9. Editor:** In the `demo.js` file, find `'a.basic'`. You will see that it is the default tooltip with none of the special effects. Everything is set as default by the file `jquery.tooltip.js`. This means that the special functions of `jquery.tooltip.js` are not used.
- 10. Editor:** In the `demo.js` file, find the following line:

```
$('#sticky').cluetip({sticky: true, closePosition: 'title', arrows: true });
```

Each property within this line means something in relation to the defaults:

- **sticky: true** — indicates that the tooltip will remain on until closed.
- **closePosition: 'title'** — means the X to close the box will be at the title (top).
- **arrows:** — the value true will place an arrow pointing to the text.

You can see that by understanding the plug-in *jquery.js*, the commands within the *jquery.cluetip.js* are relatively simple to interpret, understand and deploy.

Warning: Thorough research is required before selecting a library or plug-in. Many plug-ins have security holes or bugs, or are not cross-browser compatible. The time you spend researching your libraries and plug-ins will pay off with better, cleaner and safer code.





Lab 13-1: Installing and debugging with the Mozilla Firebug add-on

In this lab, you will install the Firebug add-on to Mozilla Firefox, and use it to locate and view errors in a script.

Note: Use the Mozilla Firefox browser for this lab. If you do not already have Firefox installed on your system, install it from the www.firefox.com site before beginning the lab.

1. **Firefox browser:** Enter the URL **www.firefox.com**, which will take you to the Mozilla Firefox home page.
2. **Firefox:** At the top of the screen you will see menu choices. Hover your cursor over the **Add-ons** menu, and a drop-down list will appear, as shown in Figure 13-6.



Figure 13-6: Mozilla Firefox home page — Add-ons menu

3. **Firefox:** Click **Firefox Add-ons**, which is the top choice in the menu. On the Add-ons page, in the left column under Categories, click the **Web Development** link. Scroll down to the Top Downloads list. You will see Firebug at or near the top of this list. Click the **Firebug** link.

Warning: Mozilla offers thousands of add-ons for the Firefox browser. Many of these are written by third parties, so use caution. You should thoroughly investigate any interesting add-on prior to installing it on your system.

4. **Firefox:** You will see the Firebug Add-ons page shown in Figure 13-7. Read the information about Firebug, then click the **Add To Firefox** button.

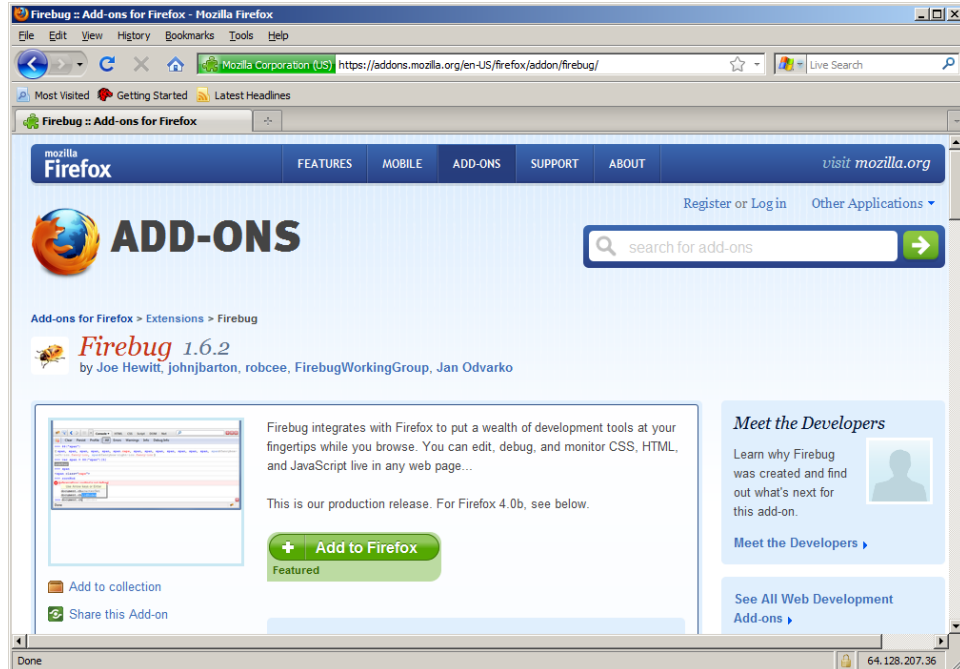


Figure 13-7: Firebug Add-ons home page

5. **Firefox:** Notice that after you click the Add To Firefox button, you will see the security alert shown in Figure 13-8. Your browser is reminding you to use caution when choosing to install add-ons, even from its own site.

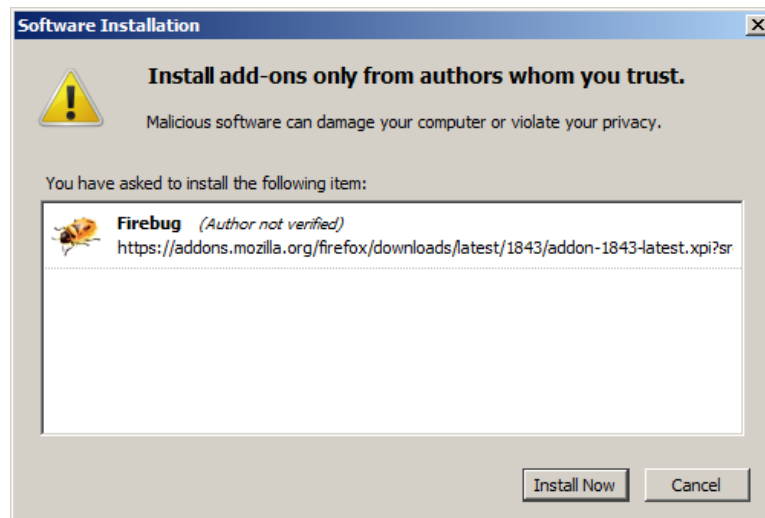


Figure 13-8: Firefox security alert warning about add-on installation

6. **Firefox:** Click the **Install Now** button. The site will install the add-on and prompt you to restart Firefox. Restart the browser now. The Firebug debugger is now installed.
7. **Firefox:** After restarting, you will notice no change in Firefox at first. From your student files, open the **lab13-1.htm** file in Firefox. In the **Tools** drop-down, select **Firebug**, then select **Open Firebug In New Window**, as shown in Figure 13-9.

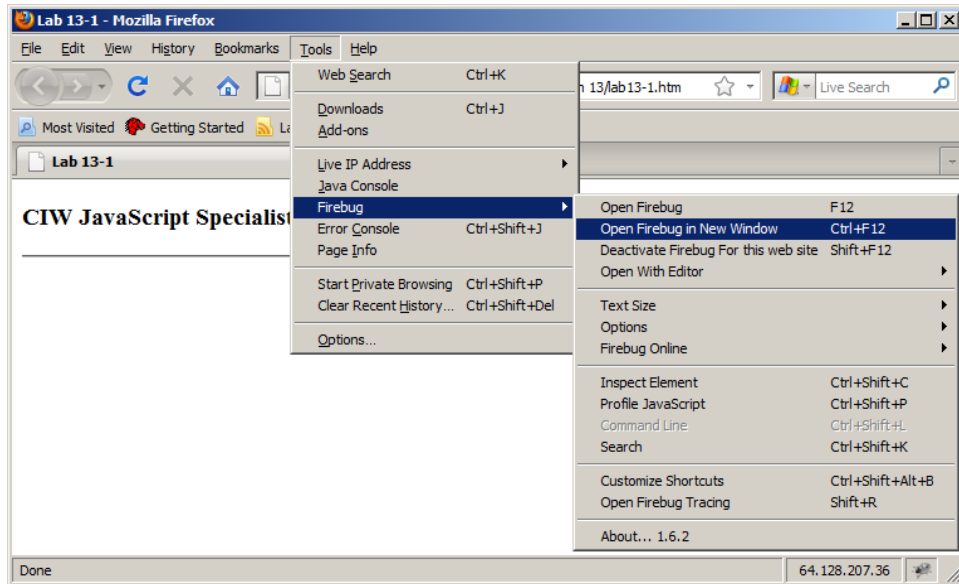


Figure 13-9: Opening Firebug debugger in Firefox

8. **Firefox:** A new window will appear, as shown in Figure 13-10. Firebug tells you there is an error, specifies the nature of the error ("hello is not defined"), and also specifies which line in the code you should review to find it (line 12).

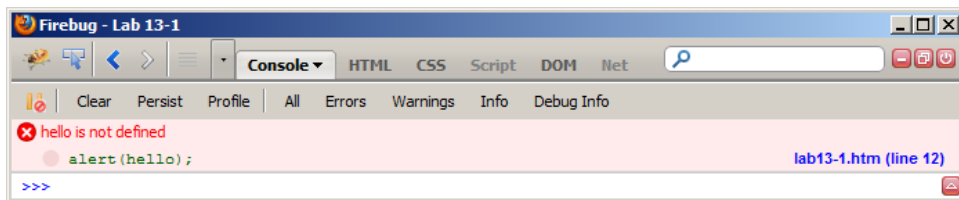


Figure 13-10: Firebug error alert box

9. **Editor:** Open the **lab13-1.htm** file in your editor, and review the code, which is as follows:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Lab 13-1</title>
</head>

<body>
<h3>CIW JavaScript Specialist</h3>
<hr />
<script language="JavaScript" type="text/javascript">
alert(hello);
</script>
</body>
</html>
```

Notice Line 12, where the `alert()` method is used. Do you see the error? There are no quotes around `hello`.

- 10. Editor:** Place the quotes around `hello`, then save the file as **error-debugged.htm**. Run this file in Firefox again. You will see that the error is gone. The pop-up from the script will appear as expected (as shown in Figure 13-11), and Firebug will return no errors.

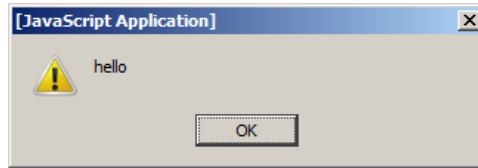


Figure 13-11: Debugged JavaScript rendered as expected

- 11. Browsers:** Test the repaired **error-debugged.htm** file in Internet Explorer. You will get the same results. Now test the same file in Google Chrome, and you will again get the same results. You have just repaired your first bug in JavaScript using a debugger.

Firefox does not show errors on a page by default. You must open Firebug to show errors. However, if you want to disable Firebug from debugging a page, you can press the SHIFT+F12 keys.



Lab 13-2: Troubleshooting a logic error in JavaScript

In this lab, you will learn how to debug logic errors in JavaScript code that do not return error alerts, as well as repair the broken script.

- Editor:** Open the **lab13-2.htm** file and study the code, which is as follows:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Lab 13-2</title>
</head>

<body>
<h3>CIW JavaScript Specialist</h3>
<hr />

<script type="text/javascript">
<!--

/*****
 Mistakes in code cost time and money. Watch your
 scripts!
*****/

var a =  prompt("Enter a number", "");
var b =  prompt("Enter a second number", "");
var c =  prompt("Enter a third number", "");
var d =  prompt("Enter a fourth number", "");
var e =  prompt("Enter a fifth number", "");

var solution = a*b+c+d*e;

document.write("The total you are looking for is a*b+c+d*e = " +
solution);/*Using all 2s for input, the answer should be 10*/
</script>
</body>
</html>
```

- Browser:** Open the file **lab13-2.htm** and run the script. At each prompt, enter the number **2**. If the equation were solved from left to right as you expected, the script should end with an answer of 16. Instead, it is returning 424. Why? No error alerts appear in the browser, but something is wrong. This is called a logic error. Now the troubleshooting begins.
- Editor:** The first step is to ensure that the variables you think are running in the script are actually running. Enter the following code as shown in bold, then save the file. This alert is called a watchpoint, and it allows you to monitor this point in the script to verify whether it is functioning as expected:

```
var solution = a*b+c+d*e;

alert(a + " " + b + " " + c + " " + d + " " + e)

document.write("The total you are looking for is a*b+c+d*e = " + solution);
```

- Browser:** Run the script again, entering **2** in each prompt. After the variables are declared, you will be able to see the variable conditions where you expect to see them (i.e., an alert [watchpoint] will show you the numbers you entered). In the alert, you

should see a series of 2s, as shown in Figure 13-12. This verifies that the script did receive the same input you thought you entered. But the result is still 424. Something is still wrong.



Figure 13-12: Watchpoint showing user input

5. **Editor:** Next, you will divide the mathematical expression into parts and examine it. This may help determine where the math is not working. Enter the following bolded code just above the alert you placed previously, then save the file. If the script runs correctly, then the answer for *f* should be 10:

```
var solution = a * b + c + d * e;
alert(a + " " + b + " " + c + " " + d + " " + e)
```

```
var firstpart = a * b + c;
alert("a * b + c = " + firstpart);
var f = (a*b+c+d*e);
alert("f = a*b+c+d*e =" + f);
```

```
document.write("The total you are looking for is a*b+c+d*e = " + solution);
```

6. **Browser:** Run the script again, entering **2** each time. You now see three alerts. The first alert re-confirms that you entered the correct input: a series of 2s. The second alert (shown in Figure 13-13) shows the first part of the equation and reveals the problem: $a*b+c=42$. The script is concatenating, instead of multiplying and then adding. It multiplies $2*2$ to get 4, then concatenates 2 with the + operator for a result of 42. The third alert (also in Figure 13-13) confirms the concatenation: $f = a*b+c+d*e = 424$. So your watchpoints revealed that both + operators are performing concatenation, instead of addition as intended. This is a logic error. You can fix this.



Figure 13-13: Two watchpoints revealing logic error — concatenation instead of addition

7. **Editor:** Find the following code shown in ~~strike through~~ and replace it with the code shown in **bold**, then save the file:

```
var solution = a * b + c + d * e;
alert(a + " " + b + " " + c + " " + d + " " + e)
var firstpart = a * b + c;
alert("a * b + c = " + firstpart);
var f = (a*b+c+d*e);
alert("f = a*b+c+d*e =" + f);
```

```
var f = (Number(a) * Number(b) + Number(c));
var solution = (Number(a)*Number(b)+Number(c)) + d*e;
alert(a + " " + b + " " + c + " " + d + " " + e)
```

```
alert("a times b plus c is " + f);  
var dtimese = d*e;  
alert("d times e = " + dtimese);
```

- 8. Browser:** Run the script again. When you input all **2**s, you should again see the watchpoint alerts, but this time they are performing the math correctly. The first alert re-confirms your input. The second alert (shown in Figure 13-14) shows the first part of the equation with the correct result of 6. The third alert (also in Figure 13-14) shows the second part of the equation with the correct result of 4. The final page shows the complete equation with the correct result of 10 (the two parts of the equation added instead of concatenated).

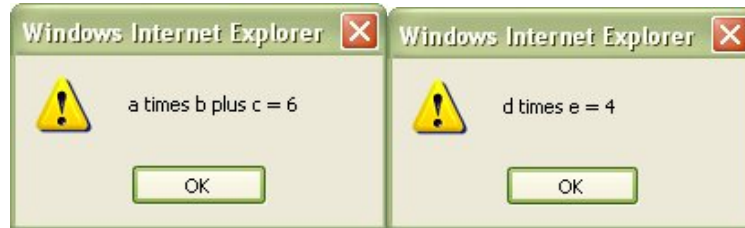


Figure 13-14: Watchpoints showing logic error corrected

Notice that if you performed this equation yourself working from left to right, you would expect a result of 16 (because $2*2=4... +2=6... +2=8... *2=16$). This result would be incorrect, but if it was what you expected, then the unexpected result of 10 is another type of logic error and must be addressed. The fact is that computer programs give mathematical precedence to certain operators (unless otherwise directed); for example, multiplication is performed before addition. Dividing the equation into two parts showed this: The first multiplication operation resulted in 6, the second resulted in 4, and the addition of those two operations (performed last) resulted in 10. Therefore, your watchpoints that divided the equation helped to show you the correct mathematical precedence that took place. These watchpoints helped you troubleshoot the second logic error, which was the unexpected result. You now know why the result was 10 instead of 16, and you realize it is correct based on math precedence.

- 9.** But how was the concatenation error solved? In the original equation, the script did not recognize all the variables (a, b, c, d and e) as numbers, so it concatenated some of them left to right. To resolve this, you declared the numbers explicitly as numbers using the `Number()` method. Specifying or changing data types in this way is called casting; in this case, it forced the script to treat all the variables like numbers. The script then performed addition instead of concatenation for the `+` operator, and solved the equation using proper mathematical precedence.

In this lab, you encountered two logic errors. You performed troubleshooting, and by running alerts to check your progress and declaring the numbers explicitly as numbers, you solved both logic errors.