**IS360**

**JavaScript Lab 1**

**Ekedahl**

In this lab, you will:

* Use the document.write and document.writeln methods to write data to the output stream.
* Call an intrinsic JavaScript function.
* Call the document.getElementById method to get a reference to input and other elements.
* Use the JavaScript Date object.
* Use the var statement to declare variables.
* Create a simple input validation form.
* Write decision-making statements.
* Create loops including pre-test, post-test, and for loops.

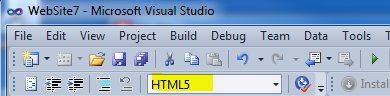
**AS YOU COMPLETE THIS LAB, REMEMBER THAT JAVASCRIPT IS CASE SENSITIVE**

**While it is possible to debug JavaScript in .NET, I have been much more successful using Chrome and its debugger. You access the Chrome debugging environment by pressing CTRL-SHIFT-J in the browser window. You might also try using the .NET Page Inspector but I have not had as much luck with it. It is not nearly as robust as the Chrome debugging tools. The IE and FireFox debugging tools are not as robust either.**

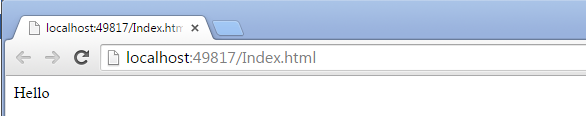
**EXERCISE 1 – CREATING A FIRST SCRIPT**

In this first exercise, you will write a first JavaScript in which you will use the document object and the Date object.

Remember that the document object is used to reference the HTML 5 document displayed in the browser.

1. In Visual Studio.NET, create an empty ASP.NET Web site as you have been doing.
2. In Visual Studio .NET, create a blank HTML 5 page. Name the file **index.htm**.  
     
   *Note that as you enter JavaScript statements, .NET will apply the same intellisense technology that you are used to. However, you must make sure that the enclosing <script> tags are correct. Otherwise, .NET will not know that you are editing JavaScript. Also, make sure that the .NET environment set to set to validate HTML 5 as follows:*
3. Create a script in the <body> section of the document. This script will execute as the document is rendered (the <body> section is processed). *Note that the* <script> *tag in HTML 5 requires no attribute or you can use “*text*/Javascript-version”. Note also that the document.writeln() method writes a carriage return at the end of the output stream. document.write() does not write a carriage return. Both methods accept one argument – the string to write to the output stream. Remember that strings are enclosed in quotation marks. In the following code, you are writing the string “hello” to the output stream. Here, you are using a literal value but you could also use a variable as an argument. Finally, note that all JavaScript statements are terminated by a semi-colon (;).*

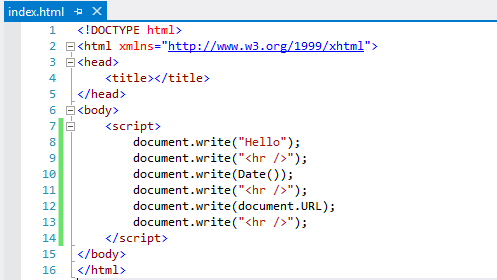
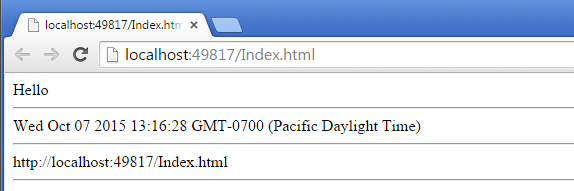
<body>  
 <script>  
 document.write(“Hello”);  
 </script>  
</body>

1. Make sure that the default browser is set to **Chrome** in Visual Studio.   
     
   Right-click on the page and select **View in Browser**. The page should run, and the string “Hello” should appear in the browser window as follows:  
     
   
2. End the program and close the browser window.
3. JavaScript supports intrinsic functions and objects much like any programming language. For example, there is an object to display the current date. The following statement, will display the current date to the output stream:  
     
   document.write(Date());

Add the above statement to the block that you just wrote. *The above statement executes by first calling the Date() function, which returns the current date. This value (the current date) is passed to the document.write method so the current date will be written to the output stream. In JavaScript and most other languages, expressions are evaluated from the innermost set of parenthesis to the outermost set of parenthesis. The above statement could have been written as follows using a variable:*

1. Run the page again to verify that the current date is written to the output stream.
2. The document object supports a property named URL. Again, this property is *case sensitive*. Objects and their properties are connected together with a period (.) just as they are in Visual Basic and most languages. The following statement displays the full URL of the Web page being rendered:  
     
   document.write(document.URL);

Enter the above statement in the same <script> block that you have been using. Run the program again to see that the URL is written to the output stream.

1. *At this point, all of the output is being written to the same line. Next you will use document.write to write some HTML to the output stream.* Now, use the document.write() method to write a horizontal rule <hr /> tag to the output stream such that a line appears between each of the write statements that you have just created. Your final code should look like the following:  
     
   
2. Preview the Web page in a browser window. Again, I suggest that you use Chrome as the default browser.  
     
   Your output, when rendered in the browser window, should look like the following:  
     
   
3. Finally, in the same script block that you have been using, write your name to the output stream using document.write(). Write a paragraph with some content of your choosing. That is, write a <p> tag. Remember to include both the opening and closing tags.

*The point to be made is that you can write any valid HTML 5 and the browser will interpret it!*

**EXERCISE 2 – WORKING WITH DATES**

In this second exercise, you will work a bit more with the JavaScript Date object. The Date object supports properties and methods just like all objects. The following W3Schools page describes the members of the Date object: <http://www.w3schools.com/jsref/jsref_obj_date.asp> .  
  
In this exercise, you will use the following members:

* getTimezoneOffset – returns the number of minutes from Greenwich Mean Time.
* getMinutes – gets the elapsed minutes in the current hour.
* getHours – gets the current hour of the day.  
    
  *Note that there are many other date functions beyond the ones discussed in this exercise.*

1. Create a new blank HTML 5 page named **Dates.htm**.
2. Write the following HTML to display a header (<h1> tag) followed by a horizontal rule. Again, this code should appear in the <body> section of the document. *This is not JavaScript. It’s just ordinary HTML that will be rendered to the output stream.*

<h1>JavaScript Date Demo</h1>

<hr />

1. Following this code, create a <script> block as you did before. This <script> block should appear on the <body> section of the page immediately after the code that you just wrote. *Remember that JavaScript code (outside of a function) executes as it is seen by the browser.*
2. Write the following code to declare a variable to store the current date. The second of the following statements writes a message containing the number of minutes between the local time and UTC time. Enter the following statements into the script block that you just created:

<script>  
 var d = new Date();

document.write("You are " + d.getTimezoneOffset() +   
 " minutes west of GMT");  
 </script>

Note the following about the above statements that you just wrote:

* The new keyword is used to create objects, such as dates. This keyword is called the *constructor*. Without arguments, the Date() constructor creates a date object that stores the current date and time. The first of the preceding statements stores the current date in the variable named “d”. *Remember that variables are dynamic and can store any data type.*
* The plus sign (+) is an operator. In the above context, it is used to concatenate strings because some or all of the operands have a data type of string. Depending on the data type of the arguments, the operator will add numerical values or concatenate strings. The second of the preceding statements concatenates three strings. Note that you do not need to explicitly perform any type conversion.
* The fragment d.getTimezoneOffset() used the current date (stored in the variable d) to get the number of minutes west GMT from the current time zone.

1. Next, you can try it on your own. Write the code to display the following text. You can use the same script block that you just wrote. Use the getMinutes() method of the Date object(d.getMinutes). Put a horizontal rule between each of the output strings.  
     
   It is ***30*** minutes past the hour.
2. Write the code to display the following message using the getHours() method.  
     
   The hour of the day is *xx*.

**EXERCISE 3 – WORKING WITH THE BUTTON WIDGET**

In this third exercise, you will work with a button object (element), an input object (element), and create a first global function. When the button is clicked, it will display some information as the content of a <span> element. The content of the input element will be copied to the span element’s innerHTML property.

* The following Web page documents the new HTML 5 button object:  
  <http://www.w3schools.com/jsref/dom_obj_pushbutton.asp>.

1. Create a new blank HTML 5 page named **Button.htm**.
2. In the <body> section, write the following code to create an <input> element, a <span> tag, and a <button>:

<body>

<h1>Button Test</h1>

<label for="txtInput">Enter your name:</label>

<input type="text" id="txtInput" />

<br />

<button type="button" id="btnClickMe"

onclick="btnClickMe\_Click()" >Click me</button>

<br />

<span id="spnOutput"></span>

</body>

*Note the following about the code that you just wrote. The first line contains a simple header (<h1> tag). The second line contains a <label> tag. The text “Enter your name:” is displayed in the label. The label is for the control named txtInput. The next line contains an input text box named txtInput. It is configured as a text box because of the attribute type=”text”. The next line writes a line break. Next the <button> element is created. The following attribute registers the onclick event.* onclick="btnClickMe\_Click()"

1. Now create the following code in the <head> section of the page. This is the code for the button’s onclick event handler:

<head>

<title></title>

<script>

function btnClickMe\_Click()

{

var inputVar;

var spanVar;

inputVar = document.getElementById("txtInput");

spanVar = document.getElementById("spnOutput")

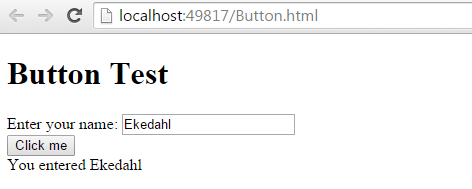
spanVar.innerHTML = "You entered " + inputVar.value;

}

</script>

</head>

*The keyword function declares a function. The function name (identifier) is btnClickMe\_Click. The () indicate that the function accepts no arguments. This empty argument list is required. Two variables are declared named inputVar and spanVar. The next statements get a reference to the <input> and <span> elements through the call to document.getElementById(). The final statement gets the value from the <input> element and stores that value in the innerHTML property of the <span> element.*

1. Run the page. Enter some text in the <input> element, and then click the button. The text should appear as the output in the <span> tag.  
     
   

**EXERCISE 4 – VALIDATING A FIRST INPUT ELEMENT**

In this exercise, you will create an input element and a button inside of a form.

1. Create a new page named **Validate.html** and enter the following code.

Take a look at the following code segment:

<body>

<form id="firstform">

<fieldset form="firstForm">

<legend>Validate Input</legend>

Enter Name: <input type="text" id="txtName" />

<br />

<button id="btnValidate" type="button"

onclick="btnValidate\_Click()">Validate</button>

<br />

Messages: <span id="spnMessages"></span>

</fieldset>

</form>

</body>

The <form> object is where you can tie everything together with forms and JavaScript. In this section, you will create a form, place HTML5 controls on that form, and then write JavaScript code to get the values from those input controls and validate them.

A form is typically posted to the server. A form contains a fieldset, which contains a legend. Inside of the fieldset, and after the legend, you typically create the various input controls with which the user will interact. In this lab, you use the following controls:

* The following link discusses the <input> tag: <http://www.w3schools.com/tags/tag_input.asp>. The following attributes are relevant in this exercise.  
  + The type attribute, when set to text, creates and editable text box.
  + As usual, the ID attribute contains a unique identifier.
  + The value attribute contains the value entered by the user.
* The following link discusses the <button> tag: <http://www.w3schools.com/tags/tag_button.asp>. The following attributes are relevant in this exercise.
  + The type attribute, when set to button, creates a button. The button is not a submit button and will not cause a page postback. You can also use the newer HTML 5 button element.
  + As usual, the ID attribute contains a unique identifier.
  + The onclick event fires when the user clicks the button. The attribute’s value contains the name of the function that will be called. Again, remember that all of these names are case sensitive.
* The following link discusses the <form> element: <http://www.w3schools.com/tags/tag_form.asp>. The following attributes are relevant in this exercise.
  + The form appears in the document body.
* The form contains an immediate fieldset child. The form attribute ties the fieldset back to the form.
* The legend appears as an immediate child of the fieldset.
* The input controls follow.

The next section discusses the JavaScript that you will write to validate that the text box contains a value. If it does, the text in the placeholder <span> tag will display the message “Name Valid”. If it does not, the contents will be “Name must exist”.

<head>

<title></title>

<script>

function btnValidate\_Click()

{

var tName;

var tErr;

tName = document.getElementById("txtName");

tErr = document.getElementById("spnMessages")

if (tName.value == "")

{

tErr.innerHTML = "Name must exist.";

}

else

{

tErr.innerHTML = "Name valid.";

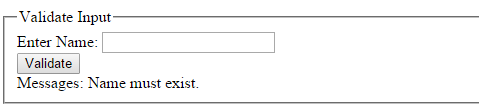
}

}

</script>

</head>

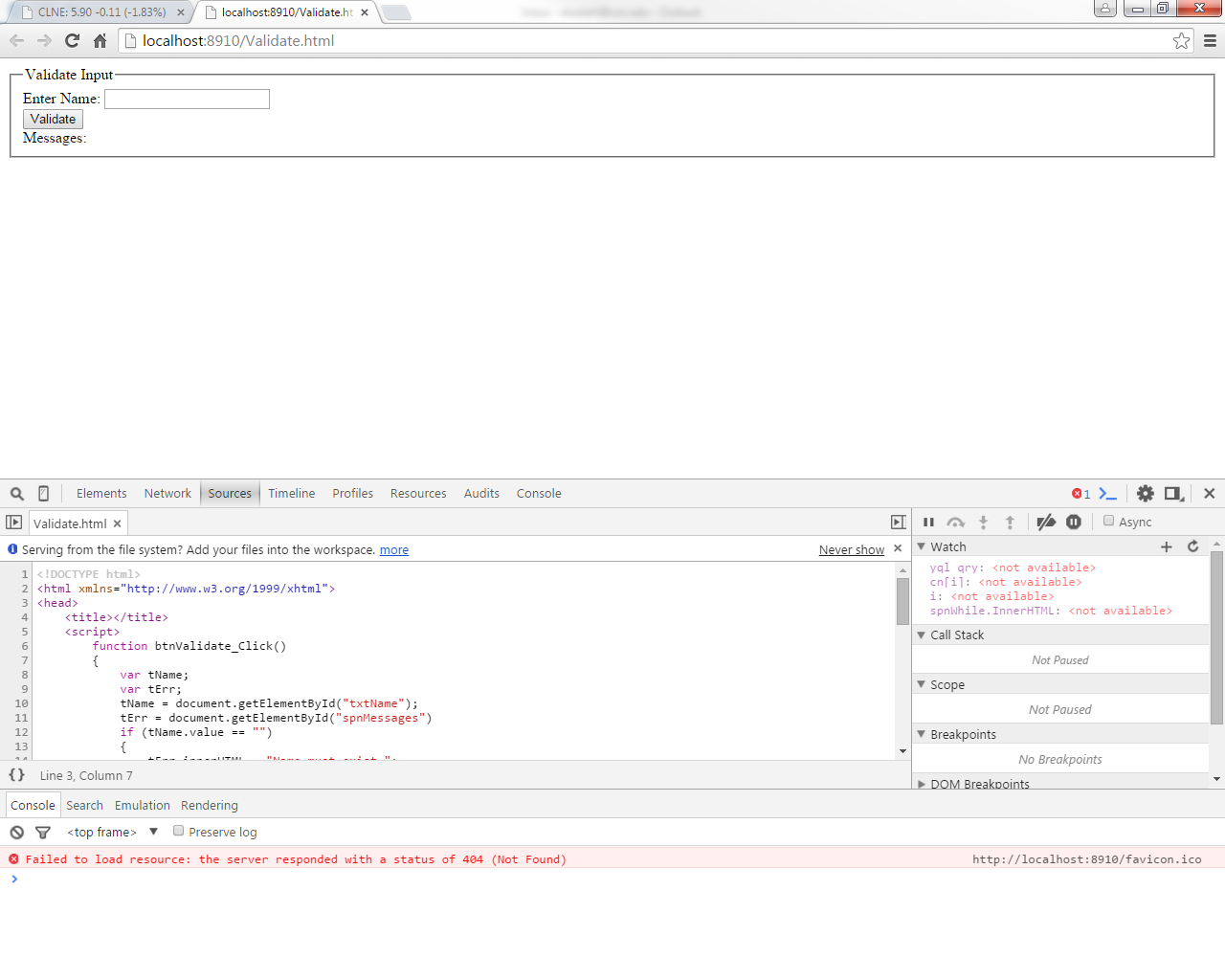
* The document.getElementById method gets the document node having the corresponding ID value. In this case *txtName*. A reference to that object is stored in the variable tName. The second call to document.getElementById does the same thing. It gets a reference to the <span> tag.
* The if statement tests whether the value is an empty string. If it is, the Messages placeholder (<div> tag) is updated to display a message. Again, remember when writing an if statement or other blocks, The {} characters surround the block.

1. Enter the code in the preceding two code segments into the form. Make sure that the content precisely matches the above code. ***Be very careful of case sensitivity.***
2. Test the application. If all is correct, the message should be updated. If not, check your code and take a close look at the next section discussing the Chrome debugger. Most problems will be caused by improper casing.
3. Run the page and display it in the browser. Click the button with and without content in the text box. The message should be updated as shown in the following figure:  
     
   

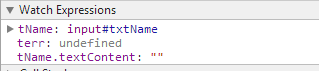
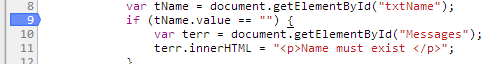
**EXERCISE 5 – USING THE CHROME DEBUGGING TOOLS**

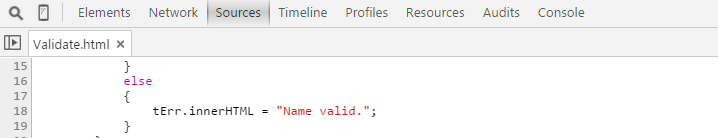
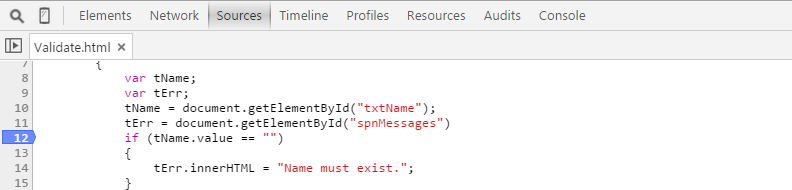
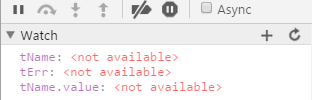
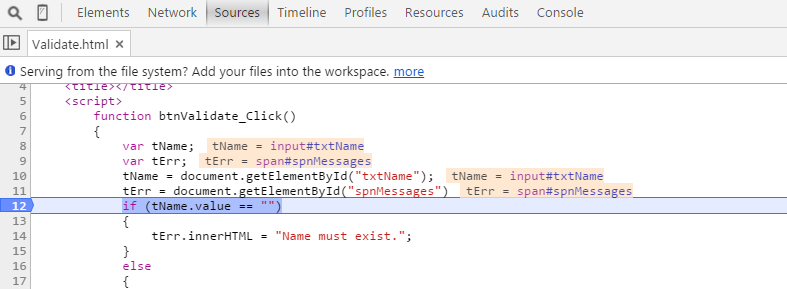
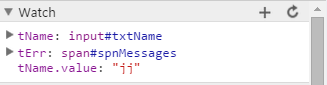
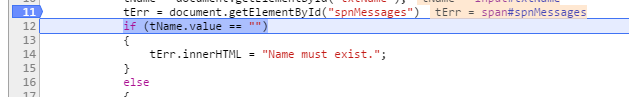
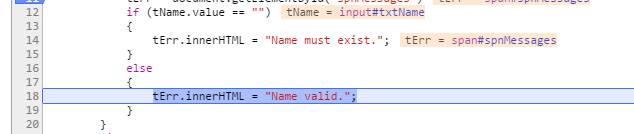
The following link describes how to use the CHROME debugging tools. <https://developers.google.com/chrome-developer-tools/docs/javascript-debugging#sources-panel>. I strongly encourage you to read this page! It’s very complete and an easy read. We will not cover all of the tools in this lab.

**CLICKING SHIFT – CTRL – J ACTIVATES THE CHROME DEBUGGER. THE FOLLOWING FIGURE SHOWS THE JAVASCRIPT APPEARING IN THE CHROME DEBUGGER.**



Note the following of the above window.

* The Web page appears in the top window.
* The **Sources** tab is active and shows the JavaScript available for debugging.
* On the right side of the window appears the debugging control windows.
  + The **Watch** window allows you to interrogate the value of JavaScript variables and objects.  
      
      
      
    In the above figure, the variable tName appears (Watch expression is defined). Other Watch expressions to look at the object tName.txtContent and tErr. The plus (+) sign allows you to create Watch expressions.   
      
    **You create a Watch Expression by right-clicking in the Watch window and selecting Add Watch Expressions from the content menu. A blank line appears in the editor. Enter the expression or variable that you want to watch.**
  + The **Call Stack** window shows the procedures that have been called and the order in which those procedures are called.
  + Breakpoints allow you to temporarily suspend execution of a JavaScript program and appear in the **Breakpoints** section. To set a breakpoint, you click in the Sources window in the left margin containing the line numbers. When set, the line appears highlighted as follows:  
      
    
  + The remaining functions are not discussed in this lab. The DOM breakpoints section will be discussed later when we talk more about the DOM.

1. View the page named **Validate.html** in Chrome.
2. Click **Shift – Ctrl – J** to activate the Chrome debugger.
3. Click on the **Sources** tab to view the JavaScript source debugger. It’s possible that you might need to explicitly load the source file. If you do not see the source code displayed, type **Ctrl+P** and select the file named **Validate.html** from the popup menu.
4. The source code will appear on the left side of the window. The Debugging windows will appear on the right side of the window.   
     
   
5. Set a breakpoint on the statement shown in line 12. Again, to set a breakpoint, click along the left margin (column containing the line numbers). Your line numbers will likely differ. Note that you can delete a breakpoint by clicking the line again. The blue marker will disappear. In addition, you can set as many JavaScript breakpoints as you see fit.  
     
   
6. By default, some watch expressions are set for you. Right-click in the Watch window and click **Remove All Watch Expressions.** All of the Watch expressions should disappear from the Watch window.
7. Set Watch expressions for the objects / variables named tName, tErr, and tName.value. Again, to add a Watch expression, you can click the + sign. Or you can right-click in the window and select Add Watch Expression. Enter the desired Watch Expression in the edit box. Note that for each Watch Expression that you create at this point, the value <not available> appears because the JavaScript is not running.  
     
   
8. Enter the text “JJ” in the text box and click the **Validate** button.
9. The program will begin executing until the breakpoint is hit. The statement containing the breakpoint will be highlighted as shown in the following figure:  
     
     
   When the breakpoint is “hit”, execution is suspended and the following message appears in the browser indicating that the program is presently paused:  
     
   
10. The Watch window displays the current value of the Watched variables. You can see that the variable tName contains a reference to the input text box named txtName. tName.value contains the current value of the text box, which is “jj”.  
      
    
11. Press **F11** again. You should see the following statement highlighted. This is the next statement that will execute (the if statement).   
      
    
12. Press F11 again. This step is interesting. Note that the following line is highlight. You are now seeing the execution path of your program. Because there is content in the text box, the test tName.value == “” is false. Thus, the statement in the else part of the if statement executes.  
      
    
13. Press **F5** to continue running the program. You will see the message displayed.

**EXERCISE 6 – EXTENDING THE FORM ON YOUR OWN**

In this section of the lab, you will extend the above program to accomplish the following:

1. Change the prompt and element name accordingly so that the name is txtFirstName instead of tName. Change the JavaScript accordingly.
2. Add a second input control named txtLastName so that it appears just below the first name.
3. Enhance the above JavaScript so that it validates the last name and displays a similar message.

**EXERCISE 7 – CREATING A LOOP**

In the following exercise, you will again use JavaScript to create loops. You will create pre-test loops, post-test loops, and a for loop.

1. In the same Web site, create a page named **Loops.html**.
2. In the <body> of the document, create three buttons. I suggest that you name the first button btnWhileLoop, the second button btnDoWhileLoop, and the third button btnForLoop.
3. Create and wire the onclick event handlers as necessary. Use the same procedure that you did in the previous steps to create buttons and their corresponding event handlers.
4. Finally, create a <div> tag after the buttons and name it divPlaceholder.
5. Create the code for the button named btnWhileLoop as follows:

function btnWhileLoop\_Click()

{

var counter = 1;

ph = document.getElementById("divPlaceHolder");

ph.innerHTML = "Pretest loop: ";

while (counter <= 10)

{

ph.innerHTML += counter.toString() + " ";

counter++;

}

}

1. Create the code for the button named btnDoWhileLoop as follows:

function btnDoWhileLoop\_Click()

{

var counter = 1;

ph = document.getElementById("divPlaceHolder");

ph.innerHTML = "Posttest loop: ";

do

{

ph.innerHTML += counter.toString() + " ";

counter++;

}

while (counter <=10)

}

1. Create the code for the button named btnForLoop as follows:

function btnForLoop\_Click()

{

var counter = 1;

ph = document.getElementById("divPlaceHolder");

ph.innerHTML = "for loop: ";

for (counter = 1; counter <= 10; counter++)

{

ph.innerHTML += counter.toString() + " ";

}

}

1. Run the program and make sure that each of the loops is displaying the counting numbers from 1 to 10.
2. Next, you can try this on your own. Modify the above loops so that instead of counting from one to 10, they count down from 10 to 1.