



Optional Lab 1-1: Experimenting with JavaScript

This lab will provide an opportunity to experiment with the JavaScript-enabled file that you created in Lab 1-1.

- 1. Editor:** Open the **lab1-1.htm** file that you completed in this lesson. Save the file as **optionalLab1-1.htm**. Experiment with the `document.write()` statements by adding literal text to them. For example, you could add text that introduces the browser data that will appear in the window. The `document.write()` statement can receive text or any X/HTML (to format literal text) that you may want to use. At this point, there is no right or wrong action to take. Experiment with the document. It will help you become familiar with adding or deleting code inside a JavaScript block.
- 2. Editor:** After each change you make to the **optionalLab1-1.htm** file, save the file.
- 3. Browser:** Open the **optionalLab1-1.htm** file. View your changes. If an error occurs, return to the file in your editor and see whether you can determine why the error occurred.

This lab provided you with an opportunity to experiment with a `document.write()` statement. You added text and XHTML to the statements, and viewed the output using the browser of your choice.



Optional Lab 2-1: Performing calculations with the *Number* object

In this lab, you will see code that uses the `Number` object to perform simple mathematical calculations.

- Editor:** Open the file **optionalLab2-1.htm** from the Lesson 2 folder of the `Student_Files` directory.
- Editor:** review 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>Optional Lab 2-1: Retirement Calculator</title>
</head>

<body>
<h2 style="text-align: center; color: blue">CIW JavaScript Specialist</h2>
<h3 style="text-align: center; color: green">JavaScript Retirement
Calculator</h3>
<hr />

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

/*****
Adding two numbers ensuring they are numbers
*****/

var a = prompt("Enter your age: ", "");
var b = prompt("Enter your expected retirement age: ", "");

var a = Number(a);
var b = Number(b);
var c = b - a;
document.write(" You have " + c + " years until you retire. <br />");
var d = Number(10);
var e = Number(b) + Number(d);
document.write("If you wait until you are " + e + " your retirement savings
can grow much larger!");
</script>
</body>
</html>
```

- Browser:** Open the file **optionalLab2-1.htm** and test its functionality by entering some data.

Scripts that perform some mathematical calculation are commonly used on the Web and provide good practice for developers in techniques such as setting variables, using mathematical precedence, and obtaining specifications from a user.



Optional Lab 2-2: Performing calculations with the *Math* object

In your Web development career, you will sometimes be assigned tasks in which you may not understand the mathematics, but you still need to program the application and let the math expert (often your customer) verify that it works reliably. This process is common in the business world. Fortunately, you are not expected to be both a mathematician and a JavaScript developer. However, you should understand how to develop code that performs the calculations that your customer or math expert specifies.

In this lab, you will see code that uses the `Math` object to perform complex mathematical calculations. Do not focus on the mathematics in this script, but rather the scripting techniques, such as setting variables, using mathematical precedence, and obtaining specifications from a user.

1. **Editor:** Open the file **optionalLab2-2.htm** from the Lesson 2 folder of the `Student_Files` directory.
2. **Editor:** review 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>Optional Lab 2-2: Mortgage Calculator</title>
</head>

<body>
<script language="JavaScript" type="text/javascript">
// Calculates the payments and updates for fields to
// display the result
function calcpayment() {
    var amt = document.frm.amt.value;
    var ir = document.frm.ir.value / 1200;
    var term = document.frm.term.value * 12;
    var total=1;

    for (i=0; i<term; i++) {
        total = total * (1 + ir);
    }
    mp = amt * ir / (1 - (1/total));

    // Use Math object to chop all numbers after 2 digits
    document.frm.payment.value = Math.round(mp*100)/100;
    document.frm.total.value = Math.round(mp * term *100)/100 ;
}
</script>
<h1 style="text-align: center; color: blue">CIW JavaScript Specialist</h1>
<h2 style="text-align: center; color: green">JavaScript Mortgage
Calculator</h2>
<hr />
<center>
<form name="frm">
<table bgcolor="black" cellpadding="2"><tr><td>
<table bgcolor="wheat" cellspacing="10">
<tr>
<td>Mortgage amount:</td>
<td><input type="text" name="amt" /></td>
</tr>

<tr>
<td>Yearly interest rate:</td>
```

```

        <td><input type="text" name="i r" /></td>
    </tr>

    <tr>
        <td>Term (in years):</td>
        <td><input type="text" name="term" /></td>
    </tr>
    <tr >
        <td colspan="2" width="100%">
        <input type="button" value="Click to Calculate Monthly Payment"
            onclick="calcpayment();" />
        </td></tr>
    <tr><td colspan="2"><hr /></td></tr>
    <tr>
        <td>Monthly payment:</td>
        <td><input type="text" name="payment" /></td>
    </tr>

    <tr>
        <td>Total payments:</td>
        <td><input type="text" name="total" /></td>
    </tr>
</table>
</td></tr></table>
</form>
</center>
</script>
</body>
</html>

```

- 3. Browser:** Open the file **optionalLab2-2.htm** and test its functionality by entering some data. Try entering a mortgage amount using a comma (e.g., 200,000) and then try it again without a comma (e.g., 200000). How does the output change? You can add details to your form to help navigate users through programming details such as these.

Scripts such as a mortgage calculator are commonly used on the Web and provide good practice for developers in techniques such as setting variables, using mathematical precedence, and obtaining specifications from a user. The average Web developer may not understand the mathematics, but should understand the program itself. In a typical scenario, the customer asking for this scripted functionality will provide the mathematics information and details about how the program needs to work. The Web developer will take these formulas and specifications, and create a program based on the customer's needs.



Optional Lab 3-1: Calling functions and using arguments

In this optional lab, you will run a program that uses JavaScript functions and arguments. Functions in JavaScript can receive a varying number of arguments. Also, explicit placeholders need not be defined in the function signature to use these arguments.

1. **Editor:** Open the file **optionalLab3-1**. Examine the code for a form, 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>Optional Lab 3-1: JavaScript Functions</title>

<script type="text/javascript">
<!--
// return the value of the radio button that is checked
// return an empty string if none are checked, or
// there are no radio buttons
function getCheckedValue(radioObj) {
    if(!radioObj)
        return "";
    var radioLength = radioObj.length;
    if(radioLength == undefined)
        if(radioObj.checked)
            return radioObj.value;
    else
        return "";
    for(var i = 0; i < radioLength; i++) {
        if(radioObj[i].checked) {
            return radioObj[i].value;
        }
    }
    return "";
}

// set the radio button with the given value as being checked
// do nothing if there are no radio buttons
// if the given value does not exist, all the radio buttons
// are reset to unchecked
function setCheckedValue(radioObj, newValue) {
    if(!radioObj)
        return;
    var radioLength = radioObj.length;
    if(radioLength == undefined) {
        radioObj.checked = (radioObj.value == newValue.toString());
        return;
    }
    for(var i = 0; i < radioLength; i++) {
        radioObj[i].checked = false;
        if(radioObj[i].value == newValue.toString()) {
            radioObj[i].checked = true;
        }
    }
}
-->
</script>
<style type="text/css">
<!--
label { background-color: #FDFDF0; color: black; border: 2px outset #8B8378;
padding: 0.1em 2ex 0.1em 0.5ex; line-height: 1.6em; }
```

```
//-->
</style>
</head>

<h2>CIW JavaScript Specialist</h2>
<h3>JavaScript Get or Set Checked Radio Value</h3>
<form name="radioExampleForm" method="get" action="" onsubmit="return false;">
<p>
<label for="number0">
<input type="radio" value="0" name="number" id="number0" /> Zero</label>
&nbsp;&nbsp;<label for="number1">
<input type="radio" value="1" name="number" id="number1" /> One</label>
&nbsp;&nbsp;<label for="number2">
<input type="radio" value="2" name="number" id="number2" /> Two</label>
&nbsp;&nbsp;<label for="number3">
<input type="radio" value="3" name="number" id="number3" /> Three</label>
&nbsp;&nbsp;<label for="number4">
<input type="radio" value="4" name="number" id="number4" /> Four</label>
</p>
<p>
<input type="button" onclick="alert('Checked value is:
'+getCheckedValue(document.forms['radioExampleForm'].elements['number'])); "
value="Show Checked Value" />
&nbsp;&nbsp;<input type="button"
onclick="setCheckedValue(document.forms['radioExampleForm'].elements['number'
, '2']);" value="Set Checked to Two" />
&nbsp;&nbsp;<input type="button"
onclick="setCheckedValue(document.forms['radioExampleForm'].elements['number'
, '4']);" value="Set Checked to Four" />
&nbsp;&nbsp;<input type="button"
onclick="setCheckedValue(document.forms['radioExampleForm'].elements['number'
, '']);" value="Uncheck All" />
</p>
</form>
</body>
</html>
```

2. **Browser:** Open the file **optionalLab3-1**. You will see the form shown in Figure OL3-1.

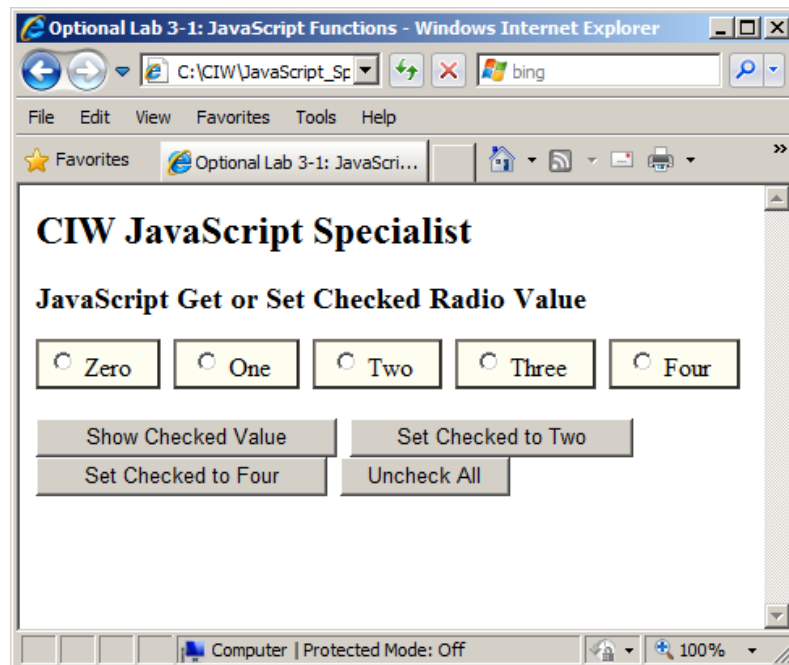


Figure OL3-1: JavaScript Functions form

3. **Browser:** Study the form. It contains five radio buttons and four buttons, each with a different purpose, which gets passed to the function. Depending on the radio button checked, it will return the value after it has been passed through the function. If there is no input (no radio buttons checked), then the form will gracefully pass a blank. This process is called validation: Every choice offered to a user is handled and returned through the program, even if the input is wrong. Gracefully handling validation is an important part of scripting.
4. **Browser:** Select one of the number values, then click the **Show Checked Value** button. You will see an alert that shows the value of the radio button you selected, as shown in Figure OL3-2. If you do not select a value radio button, the alert will show blank. The other buttons can select a value for the user, or clear the radio buttons.

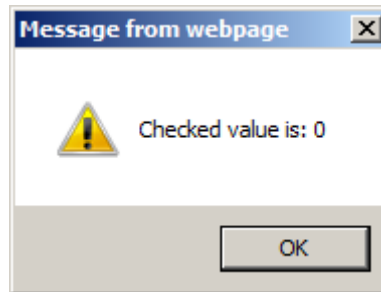


Figure OL3-2: Alert showing the checked value

In this lab, you ran a program that uses JavaScript functions and arguments. This example shows how to pass values through a function in a very simple, yet engaging manner. Functions in JavaScript can receive a varying number of arguments. Also, explicit placeholders need not be defined in the function signature to use these arguments.



Optional Lab 3-2: Using JavaScript event handlers

In this optional lab, you will run a program that uses JavaScript event handlers.

- 1. Editor:** Open the file **optionalLab3-2**. 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>Optional Lab 3-2: JavaScript Event Handlers</title>
<script type="text/javascript">
<!--
function vote()
{
    var t="Your favorite sport is: "
    for (i=0; i<document.myform.sports.length; i++)
    {
        if (document.myform.sports[i].checked==true)
        {
            t =t +document.myform.sports[i].value
        }
    }
    if (t=="Your favorite sport is: ")
    {
        document.getElementById("txt").value="Choose a sport"
    }
    else
    {
        document.getElementById("txt").value=t
        document.getElementById("txt1").value=t
        document.getElementById("txt2").value=t
    }
}
function upperCase()
{
    var x=document.getElementById("txt").value
    document.getElementById("txt").value=x.toUpperCase()
}
function SayHello()
{
    alert("Your mouse is over the text!");
}
function SayGoodbye()
{
    alert("Your mouse has left the text!");
}
-->
</script>

</head>
<body>
<h3>CIW JavaScript Specialist</h3>
<hr />
<h4>Event Handler Examples</h4>
Vote for your favorite sport:
<form name="myForm">
    <input type="radio" name="sports" value="Football" />Football<br />
    <input type="radio" name="sports" value="Hockey" />Hockey<br />
    <input type="radio" name="sports" value="Auto Racing" />Auto Racing<br />
    <input type="radio" name="sports" value="MMA" />MMA<br />
    <input type="radio" name="sports" value="Basketball" />Basketball<br />
    <input type="radio" name="sports" value="Tennis" />Tennis<br />
</form>
```



```
<input type="radio" name="sports" value="Golf" />Golf<br />
<p><input type="button" value="VOTE" onclick="vote()" />&nbsp;<input
type="reset" value="RESET" /></p>
<p><input type="text" id="txt" size="40" onblur="toUpperCase()" />
&nbsp; Click in then out! </p>
<p><input type="text" id="txt1" size="40" onmouseover="SayHello()" />
&nbsp; Hover over this field! </p>
<p><input type="text" id="txt2" size="40" onmouseout="SayGoodbye()" />
&nbsp; Hover this field then away! </p>
</form>
</body>
</html>
```

2. **Browser:** Open the file **optionalLab3-2**. Select a sport and submit the form. The event handler will pass a call to the function, which will populate the text boxes.
3. **Browser:** Click inside the top text box. Then click outside. You will see that the event handler (onblur) will change the text to uppercase for that text box.
4. **Browser:** Move your cursor over the bottom two text boxes. On the middle text box, the event handler triggers when you are over the textbox. On the bottom text box, the event handler triggers when you leave the text box.

In this lab, you saw some of the functionality that event handlers can bring to a Web page. Consider ways that you can use event handlers for various purposes on your pages.



Optional Lab 3-3: Using a JavaScript conversion function

This optional lab will provide an opportunity to use the JavaScript `parseFloat()` conversion function.

1. **Editor:** Open **optionalLab3-3.htm** from the Lesson 3 folder of the Student_Files directory.

2. **Editor:** In the first `<script>` block, examine the `addNumbers()` function that has been defined for you.

```
function addNumbers (arg1, arg2) {  
    var result = arg1 + arg2;  
    return result;  
}
```

3. **Editor:** This simple function receives two values, adds them together and returns the result. In the second `<script>` block, examine the following code:

```
alert("Please enter a numerical value in each of the  
following prompt dialog boxes. \nThe two numbers will  
be added together.");  
  
var num1 = prompt("Please enter a numerical value.", "");  
var num2 = prompt("Please enter a numerical value.", "");  
  
document.write("<p>The result of the addNumbers()  
function is: <strong>" + addNumbers(num1, num2) + "</strong></p>");
```

4. **Editor:** Recall that the return value of a `prompt()` method is a string. Determine the various places in the code to use the `parseFloat()` function to convert the user's input to a numerical value. The `parseFloat()` function can be used in several places to achieve the desired result.
5. **Editor:** Save **optionalLab3-3.htm**.
6. **Browser:** Open **optionalLab3-3.htm**. You will see an alert dialog box informing the user of the purpose of the page. You will then see two prompt dialog boxes asking for user input. After you supply the input, your screen should resemble Figure OL3-3, depending on the data input.

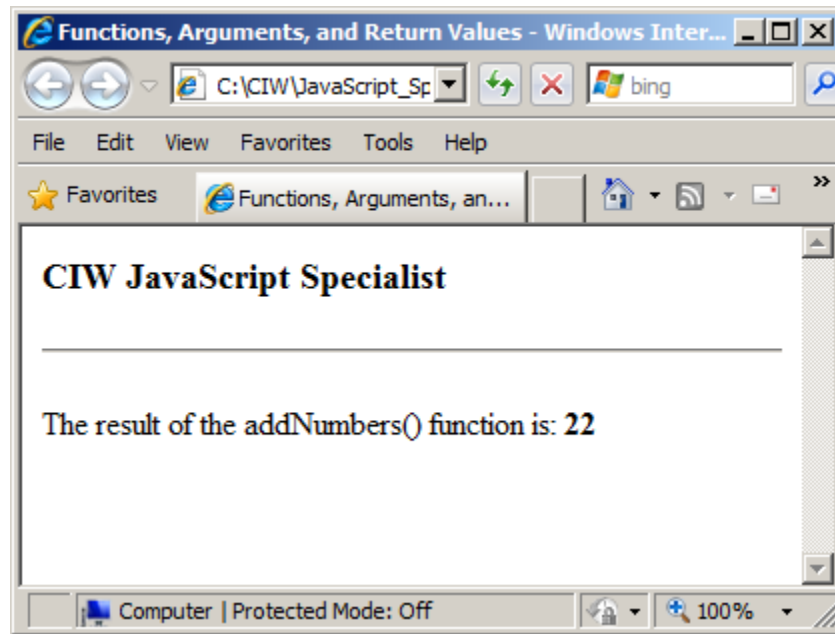


Figure OL3-3: Resulting page optionalLab3-1.htm after supplying input

7. **Browser:** Test the page to ensure that the `addNumbers()` function returns the proper result.

This lab provided an opportunity to use a JavaScript conversion function. You saw that the `parseFloat()` function is used to convert string values to numerical values.



Optional Lab 4-1: Using a *switch* statement instead of nested *if* statements

In this optional lab, you will learn to break down a complex series of nested *if* statements and convert it into a simple *switch* statement. You will find that in a scripting and programming career, you will spend a lot of time rewriting code that already works fine but is messy, either because the developer did not know a more elegant way to write the script, or because the program has been changed so many times that it has become almost unreadable. As part of standard practice, developers in many organizations perform periodic code checks to find this type of coding and standardize it.

1. Editor: Open **optionalLab4-1.htm** from the Lesson 4 folder of the Student_Files directory.

2. Editor: 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>Optional Lab 4-1</title>

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

/*****
You will be creating a switch statement from complex, multiple if
statements
*****/

function tooManyIfStatements(){
    OptionsSelect = document.MyForm.cities.value;
    if(OptionsSelect == "1"){
        alert("Albany");
    }else{
        if(OptionsSelect == "2"){
            alert("Boston");
        }else{
            if(OptionsSelect == "3"){
                alert("Chicago");
            }else{
                if(OptionsSelect == "4"){
                    alert("Denver");
                }else{
                    if(OptionsSelect == "5"){
                        alert("Elmsville");
                    }else{
                        if(OptionsSelect == "6"){
                            alert("Franklin");
                        }else{
                            if(OptionsSelect == "7"){
                                alert("Anaheim");
                            }else{
                                if(OptionsSelect == "8"){
                                    alert("Hartland");
                                }else{
                                    if(OptionsSelect == "9"){
                                        alert("Iona");
                                    }else{
                                        if(OptionsSelect == "10"){
                                            alert("Jacksonville");
                                        }else{
```


inside which and whether all brackets are nested correctly. This style of coding lacks elegance and is considered an amateur approach. Further, editing this code will take a lot of work.

4. **Browser:** Run this script and verify that it works.
5. **Editor:** Change the nested if statements into a switch statement. As 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>Optional Lab 4-1</title>

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

/*****
You are creating a switch statement from complex, multiple if
statements
*****/

function tooManyIfStatements(){
    OptionsSelect = document.MyForm.cities.value;
    switch (OptionsSelect) {
    case "1":
    alert("Albany")
    break
    case "2":
    alert("Boston")
    break
    case "3":
    alert("Chicago")
    break
    case "4":
    alert("Denver")
    break
    case "5":
    alert("Elmsville")
    break
    case "6":
    alert("Franklin")
    break
    case "7":
    alert("Anaheim")
    break
    case "8":
    alert("Hartland")
    break
    case "9":
    alert("Iona")
    break
    case "10":
    alert("Jacksonville")
    break
    default:
    alert("Please Select an option")
    }
}

//-->
</script>
</head>
<body>
<h3>CIW JavaScript Specialist</h3>
<hr />
<form name="MyForm">
```

```
<select name="cities">
  <option value="0">Select A City</option>
  <option value="1">Albany</option>
  <option value="2">Boston</option>
  <option value="3">Chicago</option>
  <option value="4">Denver</option>
  <option value="5">Elmsville</option>
  <option value="6">Franklin</option>
  <option value="7">Anaheim</option>
  <option value="8">Hartland</option>
  <option value="9">Iona</option>
  <option value="10">Jacksonville</option>
</select>
<br />
<input type="button" name="buttonClick" value="Select City"
onclick="toomanyifstatements();" />
</form>
</body>
</html>
```

Clean code is code that another developer can look at and understand in a few minutes. Developers need to know as much as possible about the language in order to write — and read — code. There is code that works, and there is well-written code, and they are not always one and the same.



Optional Lab 5-1: Opening pop-up windows with JavaScript

In the following optional lab, you will see how JavaScript can be used to launch a series of pop-up windows, which also demonstrates why pop-up blockers are so popular. Remember to be responsible with your use of scripts on your Web sites. Otherwise, your visitors are unlikely to return.

1. Editor: Open the file **optionalLab5-1.htm** from the Lesson 5 folder of the Student_Files directory.

2. Editor: 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>Optional Lab 5-1: Pop-Ups and Pup-Ups</title>
<script language="javascript" type="text/javascript">

<!-- Hide script from old browsers
function nextwindow() {
    two = window.open("optlab5-1newwindow.htm", "two", "toolbar=no, width=190,
        height=257, left=200, top=0")
    three = window.open("optlab5-1newwindow1.htm", "three", "toolbar=no, width=190,
        height=257, left=400, top=0")
    four = window.open("optlab5-1newwindow2.htm", "four", "toolbar=no, width=190,
        height=257, left=600, top=0")
    five = window.open("optlab5-1newwindow3.htm", "five", "toolbar=no, width=190,
        height=257, left=0, top=286")
    six = window.open("optlab5-1newwindow4.htm", "six", "toolbar=no, width=190,
        height=257, left=200, top=286")
    seven = window.open("optlab5-1newwindow5.htm", "seven", "toolbar=no, width=190,
        height=257, left=400, top=286")
    eight = window.open("optlab5-1newwindow6.htm", "eight", "toolbar=no, width=190,
        height=257, left=600, top=286")
}
function closeall() {
    eight.close()
    seven.close()
    six.close()
    five.close()
    four.close()
    three.close()
    two.close()
    window.close()
}
// End hiding script from old browsers -->

</script>
</head>
<body onload="nextwindow()">
<h3>Not a CIW JavaScript Specialist</h3>
<hr />
<form>
<input type="button" value="New Window!" onclick="closeall()" />
</form>
</body>
</html>
```

3. Browser: Open the file **optionalLab5-1.htm**. Notice that several small pop-up windows open, one after another. This demonstrates the power that JavaScript gives

you to manipulate the DOM. Pop-ups can be very useful, but applications such as this can be annoying or even detrimental, and should not be used without some compelling reason.

These types of applications are part of the Web environment, so you should be familiar with them. However, it is important that you understand the negative consequences of using any script or programming irresponsibly. You may simply annoy users and make them avoid your site. You may be shut down by your server host. Your Web site owner may receive complaints that lead to repercussions. And employers are sure to be unimpressed.

You are encouraged to spend your time and energy developing Web applications that reflect well upon your character as well as your skills.





Optional Lab 5-2: Determining browser type and version

In this optional lab, you will determine browser type, version and other pertinent information using the navigator object. This optional lab extends Lab 5-6 to include the Chrome, Opera and Safari browsers.

1. **Editor:** Open the file **optionalLab5-2.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>Optional Lab 5-2: Browser Names and Versions</title>
</head>

<body>
<script language="JavaScript" type="text/javascript">
var nVer = navigator.appVersion;
var nAgt = navigator.userAgent;
var browserName = navigator.appName;
var fullVersion = '' + parseFloat(navigator.appVersion);
var majorVersion = parseInt(navigator.appVersion, 10);
var nameOffset, verOffset, ix;

// In MSIE, the true version is after "MSIE" in userAgent
if ((verOffset=nAgt.indexOf("MSIE"))!=-1) {
    browserName = "Microsoft Internet Explorer";
    fullVersion = nAgt.substring(verOffset+5);
}
// In Opera, the true version is after "Opera"
else if ((verOffset=nAgt.indexOf("Opera"))!=-1) {
    browserName = "Opera";
    fullVersion = nAgt.substring(verOffset+6);
}
// In Chrome, the true version is after "Chrome"
else if ((verOffset=nAgt.indexOf("Chrome"))!=-1) {
    browserName = "Chrome";
    fullVersion = nAgt.substring(verOffset+7);
}
// In Safari, the true version is after "Safari"
else if ((verOffset=nAgt.indexOf("Safari"))!=-1) {
    browserName = "Safari";
    fullVersion = nAgt.substring(verOffset+7);
}
// In Firefox, the true version is after "Firefox"
else if ((verOffset=nAgt.indexOf("Firefox"))!=-1) {
    browserName = "Firefox";
    fullVersion = nAgt.substring(verOffset+8);
}
// In most other browsers, "name/version" is at the end of userAgent
else if ( (nameOffset=nAgt.lastIndexOf(' ') + 1) <
(verOffset=nAgt.lastIndexOf('/') ) )
{
    browserName = nAgt.substring(nameOffset, verOffset);
    fullVersion = nAgt.substring(verOffset+1);
    if (browserName.toLowerCase() == browserName.toUpperCase()) {
        browserName = navigator.appName;
    }
}
// trim the fullVersion string at semicolon/space if present
if ((ix=fullVersion.indexOf(";"))!=-1)
fullVersion=fullVersion.substring(0, ix);
if ((ix=fullVersion.indexOf(" "))!=-1)
fullVersion=fullVersion.substring(0, ix);
```

```
majorVersion = parseInt(''+fullVersion, 10);
if (isNaN(majorVersion)) {
    fullVersion = '' + parseFloat(navigator.appVersion);
    majorVersion = parseInt(navigator.appVersion, 10);
}

document.write(' Browser name   = ' + browserName + '<br />');
document.write(' Full version   = ' + fullVersion + '<br />');
document.write(' Major version = ' + majorVersion + '<br />');
document.write(' navigator.appName = ' + navigator.appName + '<br />');
document.write(' navigator.userAgent = ' + navigator.userAgent + '<br />');
</script>
</body>
</html>
```

2. **Browser:** Open the file **optionallab5-2.htm** in your usual browser and notice the information it provides.
3. **Browser:** Now open the file **optionallab5-2.htm** in some other browsers. Be sure to try it in Google Chrome and Safari.
4. Review the code. Notice that the `indexOf()` method returns the position of the first occurrence of a specified value in a string. This allows the developer to write script that will detect Opera and Firefox.

The `fullVersion` variable is different for each browser. For example, study the following line:

```
fullVersion = nAgt.substring(verOffset+8);
```

This line would return Firefox. The `substring` method returns the relevant part of an entire string, which allows your script to pinpoint the specific portion of the `userAgent` property for each browser, as described in the code comments.

The reason for this more complex version detection manner is that many current browsers (such as Firefox and Chrome) will return Netscape as the application name. This script helps you dig much deeper to find the exact browser type and version.



Optional Lab 6-1: Using arrays in complex math

In this optional lab, you will see arrays used to determine the day of the week that a particular date falls upon. You will also see the use of form validation to control the user's input into the form. This script answers the question: What day of the week were you born?

1. **Editor:** Open the file **optionalLab6-1.htm** from the Lesson 6 folder of the Student_Files directory.
2. **Editor:** Study the code. You may or may not understand the math in this script, but you will be able to see how JavaScript is used to perform validation and to process complex mathematics. The code appears as follows:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Optional Lab 6-1</title>
</head>
<body>

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

//General Array Function
function MakeArray(n) {
    this.length = n;
    for (var i = 1; i <=n; i++) {
        this[i] = 0;
    }
}

//Initialize Days of Week Array
days = new MakeArray(7);
days[0] = "Saturday"
days[1] = "Sunday"
days[2] = "Monday"
days[3] = "Tuesday"
days[4] = "Wednesday"
days[5] = "Thursday"
days[6] = "Friday"

//Initialize Months Array
months = new MakeArray(12);
months[1] = "January"
months[2] = "February"
months[3] = "March"
months[4] = "April"
months[5] = "May"
months[6] = "June"
months[7] = "July"
months[8] = "August"
months[9] = "September"
months[10] = "October"
months[11] = "November"
months[12] = "December"

//Day of Week Function
function compute(form) {
    var val1 = parseInt(form.day.value, 10)
    if ((val1 < 0) || (val1 > 31)) {
        alert("Day is out of range")
    }
}
```

```

var val 2 = parseInt(form.month.value, 10)
if ((val 2 < 0) || (val 2 > 12)) {
    alert("Month is out of range")
}
var val 2x = parseInt(form.month.value, 10)
var val 3 = parseInt(form.year.value, 10)
if (val 3 < 1900) {
    alert("You're over 110 years old!")
}
if (val 2 == 1) {
    val 2x = 13;
    val 3 = val 3-1
}
if (val 2 == 2) {
    val 2x = 14;
    val 3 = val 3-1
}
var val 4 = parseInt(((val 2x+1)*3)/5, 10)
var val 5 = parseInt(val 3/4, 10)
var val 6 = parseInt(val 3/100, 10)
var val 7 = parseInt(val 3/400, 10)
var val 8 = val 1+(val 2x*2)+val 4+val 3+val 5-val 6+val 7+2
var val 9 = parseInt(val 8/7, 10)
var val 0 = val 8-(val 9*7)
form.result1.value = months[val 2]+" "+form.day.value +", "+form.year.value
form.result2.value = days[val 0]
}


// end script -->
</script>

<h3><em>On what day of the week were you born?</em></h3>
<p><strong>Enter your birthday</strong> then click the Update button:</p>
<form>
<p>
Numeric Month (1-12): <input type="text" name="month" size="2" />
</p>
<p>
Numeric Day of Month (1-31): <input type="text" name="day" size="2" />
</p>
<p>
Four-Digit Year (e.g., 1960): <input type="text" name="year" size="4" />
</p>
<p>
<input type="button" value="Update" onclick="compute(this.form)" />
<input type="reset" value="Clear" />
</p>
<p>
Date of Birth: <input type="text" name="result1" size="18" />
</p>
<p>
Day of Week: <input type="text" name="result2" size="18" />
</p>
</form>
</body>
</html>

```

3. **Browser:** Open the file **optionalLab6-1.htm**. Enter your answers in the fields, then submit the form. Notice that the script will stop with a graceful error if you enter letters or numbers that are out of the month, day or year ranges given. Be sure that you understand the error validation in this script, as well as how the arrays are set up.

Remember that throughout your scripting and programming career, you will often be asked to write scripts that use the formulas provided to you by accountants, analysts and mathematicians. You do not necessarily need to know how the math is done, but you do need to know how to enter it into your JavaScript code properly.





Optional Lab 7-1: Assessing form validation

In this lab, you will use various form validation techniques, and you will evaluate the effectiveness and appropriateness of the validations.

1. **Editor:** Open the file **optionalLab7-1.htm** from the Lesson_7 folder of the Student_Files directory. Study the code, which appears as follows. What do you notice about the validation?

```
<!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>Optional Lab 7-1</title>

<script language="JavaScript" 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 = form.phone_number.value.substring(0, 2);
    alert(first2Chars);
}

function annoying(form) {
    alert("Hello");
    if(((form.phone_number.value.length == 0) || (form.fax_number.value.length
== 0) || (form.name.value.length == 0) || (form.email_address.value.length ==
0) ){
        alert("One of your fields is blank")
    }
    if(((form.phone_number.value.length == 12) || (form.fax_number.value.length
== 12) || (form.name.value.length == 12) || (form.email_address.value.length
== 12) ){
        alert("Your Phone or Fax Number is entered incorrectly")
    }
    if (form.email_address.value.indexOf("@", 0) < 0) {
        alert("This is not a valid e-mail address!");
    }
    if(((form.phone_number.value.length > 0) && (form.fax_number.value.length >
0) || (form.name.value.length > 12) || (form.email_address.value.length == 0)
){
        alert("Your Name is incorrect")
    }
    if(((form.phone_number.value.length > 0) && (form.fax_number.value.length
== 0) || (form.name.value.length < 15) || (form.email_address.value.length ==
0) ){
```

```

        alert("Check your Name for spelling")
    }
}

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

<body>
<h3>CIW JavaScript Specialist</h3>
<h4>When the form is filled out correctly, press "Submit"</h4>
<hr />

<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. Please submit in uppercase<br />
E-Mail Address: <br />
<input type="text" size="30" name="email_address" />
<input type="checkbox" onClick="(this.checked) ? emailTest(this.form) : '';" />
Test 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>
<p>
<input type="button" value="Submit" onClick="annoying(this.form);" />
</p>
</form>
</body>
</html>

```

2. **Browser:** Open the file **optionalLab7-1.htm**. Try entering some information in the various fields. Notice that it is impossible to get past the validation.
3. Recall the form validation tips presented in this lesson. How many of those tips did this form follow? How many did it violate? Recall the tip that said, "If you require something in your form, be sure to test for it in the validation." Notice that this script did not check for upper/lower case, even though the form asked for it.

Form validation is very important, but it must be used properly to serve its purpose without annoying users. This example shows what can happen when too much validation or incorrect validation is used. Incorrect validation is a waste of your scripting time and can even pollute the data you collect. And too much validation can discourage customers from using your Web site.



Optional Lab 8-1: Redirecting the browser using a cookie

In this lab, you will redirect to another site using a cookie

- Editor:** Open the file **optionalLab8-1.htm** from the Lesson_8 folder of the Student_Files directory.
- Editor:** Examine the following 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>Optional Lab 8-1: Cookie Redirect</title>

<script>
<!--
go_to = "http://www.CIWcertified.com";

// Number of days cookie lives for
num_days = 60;
function ged(noDays){
    var today = new Date();
    var expr = new Date(today.getTime() + noDays*24*60*60*1000);
    return expr.toGMTString();
}

function readCookie(MyCookie){
    var start = document.cookie.indexOf(MyCookie);
    if (start == -1){
        document.cookie = "RedirectCookie=yes; expires=" + ged(num_days);
    } else {
        window.location = go_to;
    }
}

readCookie("RedirectCookie");
// -->
</script>

</head>

<body onload=readCookie(MyCookie);>
<h3>CIW JavaScript Specialist</h3>
<hr />
Cookie not active yet!
</body>
</html>
```

- Browser:** Open the file **optionalLab8-1.htm**. The first time you load this file, you will see the CIW JavaScript Specialist lab page that simply tells you the cookie is not yet active.
- Browser:** Now try refreshing the page. Instead of seeing the simple lab page, you should be redirected to the CIW Web site. Close the browser.
- Browser:** Open the file **optionalLab8-1.htm** again in a new browser window. You should be redirected again. In fact, for the next 60 days (or until you delete the cookie), this page (optionalLab8-1.htm) will redirect automatically to the CIW Web page.

You can learn several important points from this:

- A browser can easily be redirected to another Web site, either friendly or malicious.
 - The user has little control over his or her browser once JavaScript takes over.
 - JavaScript is not an inherently safe language. It should be used responsibly and with caution.
 - Automatic redirects are a part of the Web experience. However, a responsible developer should warn users before redirecting them. Sending users off to another location with no warning may annoy them. And performing this functionality with a cookie that will repeat it again and again can be considered both a nuisance and a security problem, because it hinders functionality. If you want users to return to your sites, you should avoid practices such as these.
-



Optional Lab 9-1: Creating a ratings page with custom objects

In this lab, you will see how to create a basic rating system using custom JavaScript objects and properties. More and more often, this type of processing will be performed directly from a database or an automatically generated text file. Creating custom objects helps you use XML feeds and other types of data objects.

1. Editor: Open the file **optionalLab9-1.htm** from the Lesson_9 folder of the Student_Files directory.

2. Editor: 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>Optional Lab 9-1: Ratings Page</title>

<script language="JavaScript" type="text/javascript">
<!--
function WebLanguages (myName, MyBestPart, MyEaseToLearn)
{
    this.name = myName;
    this.BestPart = MyBestPart;
    this.EaseToLearn = MyEaseToLearn;
}

function show (LanguageShow)
{
    document.write ("Language Name: " + LanguageShow.name + "<br />");
    document.write ("Best Part: " + LanguageShow.BestPart + "<br />");
    document.write ("Ease of Learning: " + LanguageShow.EaseToLearn + "<br />");
    document.write ("<hr />");
}

JS = new WebLanguages ("JavaScript", "Makes cool effects", "Moderate");
JV = new WebLanguages ("Java", "Makes platform-independent applications",
"Hard");
HT = new WebLanguages ("X/HTML", "Makes pretty pages", "Easy");

function start ()
{
    document.write("<h2>CIW JavaScript Specialist</h2>");
    document.write("<h3>Rating Web Languages</h3>");
    document.write ("<hr />");
    show (HT);
    show (JS);
    show (JV);
}
// -->
</script>
</head>

<body onload="start()">
<h2>CIW JavaScript Specialist</h2>
<h3>Rating Web Languages</h3>
<hr />
</body>
</html>
```

3. Editor: Notice that there are three properties in the custom object:

```
this.name = myName;
```

```
this.BestPart = MyBestPart;
this.EaseToLearn = MyEaseToLearn;
```

4. **Editor:** You can add additional properties to your object easily. Make the following changes to your script, as shown in bold, then save the file:

```
<script language="JavaScript" type="text/javascript">
<!--
function WebLanguages (myName, MyBestPart, MyEaseToLearn, MyRating)
{
    this.name = myName;
    this.BestPart = MyBestPart;
    this.EaseToLearn = MyEaseToLearn;
    this.Rating = MyRating;
}

function show (LanguageShow)
{
    document.write ("Language Name: " + LanguageShow.name + "<br />");
    document.write ("Best Part: " + LanguageShow.BestPart + "<br />");
    document.write ("Ease of Learning: " + LanguageShow.EaseToLearn + "<br />");
    document.write ("Rating 1 to 10: " + LanguageShow.Rating + "<br />");
    document.write ("<hr />");
}

JS = new WebLanguages ("JavaScript", "Makes cool effects", "Moderate", 10);
JV = new WebLanguages ("Java", "Makes platform-independent applications",
"Hard", 8);
HT = new WebLanguages ("X/HTML", "Makes pretty pages", "Easy", 9);

function start ()
{
    document.write("<h2>CIW JavaScript Specialist</h2>");
    document.write("<h3>Rating Programming Languages</h3>");
    document.write ("<hr />");
    show (HT);
    show (JS);
    show (JV);
}
// -->
</script>
```

5. **Browser:** Open the file **optionalLab9-1.htm**. Notice that without changing the structure of the script, you have added a new property. In a real-world application, the data will come from a database, XML or an RSS feed. You can see when a change is made in the data, and you can change the structure very easily.

In this lab, you learned to add a property to an existing custom object.



Optional Lab 10-1: Overwriting the DOM with *document.write*

In this lab, you will see how the `document.write` method overwrites the DOM and when it happens.

1. **Editor:** Open the file **optionalLab10-1.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>Optional Lab 10-1</title>

<script type="text/javascript">
function noOverwrite() {
document.write("<p>This text will stay, but there is another document.write
coming ... </p>");
}
</script>
</head>

<body>
<h3>CIW JavaScript Specialist</h3>
<hr />
<script type="text/javascript">
noOverwrite();
document.write("This second document.write did not overwrite the page because
the page was still rendering.");
</script>
</body>
</html>
```

2. **Browser:** Open the file **optionalLab10-1.htm** to run the script. Notice that the `document.write` method did not overwrite the CIW JavaScript Specialist heading. Why? Because the page was still rendering when the `document.write` was executed.
3. **Editor:** Open the file **optionalLab10-1a.htm** and 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>Optional Lab 10-1a</title>
</head>

<body>
<h3>CIW JavaScript Specialist</h3>
<hr />
<script type="text/javascript">
function overwritePage() {
document.write("I will overwrite your text...");
}
window.onload = overwritePage;
</script>
</body>
</html>
```

- 4. Editor:** Notice the following line of code:

```
wi ndow. onl oad = overwri tePage;
```

This code will reload the window object and reset the DOM. Therefore, the CIW JavaScript Specialist heading will be overwritten by this code.

- 5. Browser:** Open the file **optionalLab10-1a.htm** to run the script. Notice that the document.write method did overwrite the heading this time because it was executed after the page had rendered with the window.onload statement.

In this lab, you learned the importance of using the document.write method carefully and properly, because it can overwrite your entire page at a time that you do not intend.



Optional Lab 11-1: Creating a simple menu with jQuery

In this lab, you will create a simple drop-down menu using code from the jQuery library. Notice the separation of programming (JavaScript) and presentation (X/HTML page and CSS) into separate files in this lab. Recall from the lesson that this separation is standard practice in Web development.

1. **Editor:** From your Lesson 11 folder, open the file **optionalLab11-1.htm**. 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>
<head>
  <title>Optional Lab 11-1</title>
  <meta name="keywords" content="" />
  <meta name="description" content="" />
  <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"></script>
  <script src="script.js" type="text/javascript" charset="utf-8"></script>
</head>
<body>
<div id="all">
<h3>CIW JavaScript Specialist</h3>
<hr />
<p>Simple Drop-Down Menu</p>
<ul id="menu">
  <li><a href="#">Level 1</a>
    <ul>
      <li><a href="#">Level 1-1</a></li>
      <li><a href="#">Level 1-2</a></li>
      <li><a href="#">Level 1-3</a></li>
    </ul>
  </li>
  <li><a href="#">Level 2</a>
    <ul>
      <li><a href="#">Level 2-1</a></li>
      <li><a href="#">Level 2-2</a></li>
      <li><a href="#">Level 3-3</a></li>
    </ul>
  </li>
  <li><a href="#">Level 3</a>
    <ul>
      <li><a href="#">Level 3-1</a></li>
      <li><a href="#">Level 3-2</a></li>
      <li><a href="#">Level 3-3</a></li>
    </ul>
  </li>
</ul>
<br /><br />
<p>
This is a very simple drop-down menu. It is easy to use, and all the code is
from the jQuery library.
</p>
</div>
</div>
</body>
</html>
```

This code creates a simple drop-down menu from the unordered (bullet) lists. This X/HTML page is the presentation layer that the client (browser) will display and the

user will see. Notice the main menu portion (the top) and the nested tags that display the menu.

2. Editor: Now open the file **css.css**. Study the code, which appears as follows:

```
@charset "utf-8";
/* CSS Document */

/* common page styles */
body
{
    background: #6595A3;
    padding: 0 20px}

.clear
{
    clear: both;
    overflow: hidden;
    height: 0}

#all
{
    width: 80%;
    min-width: 650px;
    margin: 40px auto 0 auto;
    background: #FCFFED;
    padding: 20px 35px}

h1
{
    font: 26px tahoma, arial;
    color: #324143}

p
{
    font: 12px tahoma, arial;
    color: #171F26;
    margin-bottom: 25px}

a
{
    color: #324143}

/* menu styles */
#menu
{
    margin: 0;
    padding: 0}

#menu li
{
    float: left;
    list-style: none;
    font: 12px Tahoma, Arial}

#menu li a
{
    display: block;
    background: #324143;
    padding: 5px 12px;
    text-decoration: none;
    border-right: 1px solid white;
    width: 70px;
    color: #EAFED;
    white-space: nowrap}

#menu li a: hover
{
    background: #24313C}

#menu li ul
{
    margin: 0;
    padding: 0;
    position: absolute;
    visibility: hidden;
    border-top: 1px solid white}
```



```
#menu li ul li
{
    float: none;
    display: inline}

#menu li ul li a
{
    width: auto;
    background: #A9C251;
    color: #24313C}

#menu li ul li a: hover
{
    background: #8EA344}
```

This file contains standard CSS (Cascading Style Sheets) code. Notice that the CSS is layered (#menu li ul li a), so the drop-downs are different styles and colors from the main menu, as well as from the links. This CSS specifies exactly what it is intended to do, instead of just using the tag for the entirety (which would give you just one color and style for everything).

3. **Editor:** Now open the file **script.js**. Study the code, which appears as follows:

```
var timeout          = 500;
var closetimer       = 0;
var ddmenuitem       = 0;

function menu_open()
{
    menu_cancel timer();
    menu_close();
    ddmenuitem = $(this).find('ul').eq(0).css('visibility', 'visible');}

function menu_close()
{
    if(ddmenuitem) ddmenuitem.css('visibility', 'hidden');}

function menu_timer()
{
    closetimer = window.setTimeout(menu_close, timeout);}

function menu_cancel timer()
{
    if(closetimer)
    {
        window.clearTimeout(closetimer);
        closetimer = null;}}

$(document).ready(function()
{
    $('#menu > li').bind('mouseover', menu_open);
    $('#menu > li').bind('mouseout', menu_timer);});

document.onclick = menu_close;
```

4. **Editor:** Notice that the opening and closing action of the menu is performed by the following commands:

```
$('#menu > li').bind('mouseover', menu_open);
```

When the mouse hovers (mouseover), it triggers the event menu.open, and the appropriate menu is opened using the menu_open function.

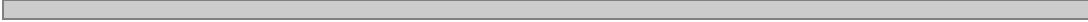
When the menu is clicked, it is closed, as follows:

```
document.onclick = menu_close;
```

The menu also requires a click to link to another site (or another page within the same site).

5. **Browser:** Open the file **optLab11-1.htm** and test the menu program. Try all the drop-downs to see how they work. Think of ways that you could use a menu such as this in your own or your company's Web pages.

In this basic menu, the drop-downs act immediately and close immediately. It is an excellent starter program for making menus. Consider again how the code for this program is neatly separated into three files (X/HTML, JavaScript and CSS), making maintenance and editing of this Web page easier. Remember this separation practice throughout your development career; it will help you avoid confusion and errors in your code.





Optional Lab 12-1: Using an AJAX image rotator

In this lab, you will study an image-rotating program that uses AJAX to pull the image information from an XML document.

- 1. Editor:** From the OptionalLab12-1 folder in your student files, open the file **optionalLab12-1.htm**. 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>Optional Lab 12-1</title>
<link rel="stylesheet" href="demo.css" type="text/css" />
<script type="text/javascript" src="ajax_rotator.js"></script>
</head>

<body onload="loadData(0)">
<h1 id="top">Image Rotator: Loading Pictures with AJAX</h1>
<p>
<a href="#" onclick="" id="prev" name="prev">Previous Image</a>
</p>
<p>
<a href="#" onclick="" id="next" name="next">Next Image</a>
</p>
</body>
</html>
```

Notice that the XHTML file references the JavaScript file *ajax_rotator.js* in a `<script>` tag, as well as a CCS file *demo.css* (in a `<link>` tag) for page formatting. Next you will study the JavaScript file.

- 2. Editor:** Open the file **ajax_rotator.js** and study the code, which appears as follows:

```
function loadScript(XMLscripts, x)
{
    var newScript = document.createElement("img");
    var prev = document.getElementById("prev");
    var next = document.getElementsByName("next")[0];
    next.onclick = function () { loadData(x+1); return
false; }
    prev.onclick = function () { loadData(x-1); return
false; }
    newScript.src =
XMLscripts[x].getElementsByTagName("FILE")[0].firstChild.nodeValue;
    newScript.setAttribute('height',
XMLscripts[x].getElementsByTagName("HEIGHT")[0].firstChild.nodeValue);
    newScript.setAttribute('width',
XMLscripts[x].getElementsByTagName("WIDTH")[0].firstChild.nodeValue);
    newScript.setAttribute('alt',
XMLscripts[x].getElementsByTagName("ALT")[0].firstChild.nodeValue);
    newScript.setAttribute('id', 'toggle');
    if ( document.getElementById("toggle")){
        document.body.removeChild(document.getElementById("toggle"));
    }
    if (x == 0){
        prev.setAttribute('onclick', '');
    }
    if (x == XMLscripts.length-1){
        next.setAttribute('onclick', '');
    }
}
```

```

        document.body.appendChild(newScript);
    }

    function loadData(num)
    {
        // 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", "picture.xml", true);

        // Anonymous function to handle changed request states
        xmlReq.setRequestHeader("Content-Type", "text/xml");

        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 <script> tags
                    if(navigator.appName == "Microsoft Internet Explorer"){
                        xmlPar=new ActiveXObject("Microsoft.XMLDOM");
                        xmlPar.async="false";
                        xmlPar.loadXML(xmlReq.responseText);
                        loadScript(xmlPar.getElementsByTagName("PICTURE"), num);
                    }
                    else{

                        loadScript(xmlReq.responseXML.getElementsByTagName("PICTURE"), num);
                    }
                    break;
                default:
                    break;
            }
        }

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

    // JavaScript Document

```

You can see this script requesting the XML file (*picture.xml*) in the bold section of code. Does this script work in both Internet Explorer and Firefox? The answer is yes. Notice that some of the bold code detects the browser and makes the necessary alterations. If you do not include these statements, then this program may not run in some browsers or versions. From there, it makes the necessary adjustments for cross-browser compatibility. You can see why it is important to always test your code in multiple browsers.

Tech Note: Many AJAX scripts perform browser detection first. For this reason, AJAX was slow to catch on for many years. The latest versions of Firefox, Chrome and Opera have made it much easier to perform validation, thus AJAX's recent exponential growth.

- 3. Editor:** Now open the XML file **picture.xml** and study the code, which appears as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ROTATOR>
<PICTURE>
  <FILE>1.jpg</FILE>
  <HEIGHT>300px</HEIGHT>
  <WIDTH>400px</WIDTH>
  <ALT>First image - it will change when you click the Next Image link</ALT>
</PICTURE>
<PICTURE>
  <FILE>2.jpg</FILE>
  <HEIGHT>300px</HEIGHT>
  <WIDTH>400px</WIDTH>
  <ALT>Image #2 - it will change when you click either link</ALT>
</PICTURE>
<PICTURE>
  <FILE>3.jpg</FILE>
  <HEIGHT>300px</HEIGHT>
  <WIDTH>400px</WIDTH>
  <ALT>Image #3 - it will change when you click either link</ALT>
</PICTURE>
<PICTURE>
  <FILE>4.jpg</FILE>
  <HEIGHT>300px</HEIGHT>
  <WIDTH>400px</WIDTH>
  <ALT>Image #4 - it will change when you click either link</ALT>
</PICTURE>
<PICTURE>
  <FILE>5.jpg</FILE>
  <HEIGHT>300px</HEIGHT>
  <WIDTH>400px</WIDTH>
  <ALT>Last image - it will change when you click the Previous Image
  link</ALT>
</PICTURE>
</ROTATOR>
```

Notice the tree structure of the XML file, including the root element and sub-elements. Consider how you might write the DTD for this file. Notice too that some of the elements look like HTML elements, whereas some do not; remember that the creator of the associated XML DTD determined the elements names and functions. Also notice that the element names in this file use ALL CAPs for letter case. Remember that although XML is case-sensitive, it does not require lowercase; it requires that the same case be used consistently in all element references. These element names could be written with different case only if you changed all references to each particular element to match each other, and only if the accompanying DTD specified the same case.

- 4. Browser:** Open the file **optionalLab12-1.htm** and test the script. Notice how smoothly the images change, without disturbing the rest of the page. The image files are supplied separately, but the XML file provides the instructions for displaying them. For example, when you hover your cursor over an image, you can see the alternative (<ALT>) text supplied by the XML file. The JavaScript file requested and accessed the XML file, and pulls the elements from the XML file into the XHTML display. The CSS file also provides some page formatting. Notice also that the code disables the Previous Image and Next Image links for the first and last images,

respectively, so if you go all the way to either end of the rotation, you will not receive an error when the browser sees there is no previous or next image.

5. **Browser:** Remember to test this script in multiple browsers and versions. Do you see any differences in the action of the script?

In this optional lab, you saw how AJAX can be used to display an image rotator, which is a popular use for AJAX because it allows the images to be refreshed without disturbing the rest of the page. A similar program written only in JavaScript would run slower, and on older software the user might notice a flicker.





Optional Lab 13-1: Troubleshooting a run-time error

In this lab, you will troubleshoot a run-time error.

- 1. Browser:** Open the file **optionalLab13-1.htm**. You will see an error reported by the debugger.
- 2. Editor:** Open the file **optional_lab13-1.htm** 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> Optional Lab 13-1</title>
</head>

<body>

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

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

var a = 2;
var b = 2;
var c = 2;

var solution = a*B+c;

document.write("The total you are looking for is a*b+c = " +
solution.toFixed(2));
/*Using all 2s for input, the answer should be 6*/
</script>
<h3>CIW JavaScript Specialist</h3>
<hr />
</body>
</html>
```

The browser you are running should have detected that a variable B is not defined. When you take a look, you will see that JavaScript's case sensitivity is the cause of the issue (the variable b was declared lowercase, but the equation uses a capital B).

- 3. Editor:** In the line where you declare the solution variable (shown in bold), correct the code so that the capital B is lowercase and matches the variable b, as follows, then save the file:

```
var a = 2;
var b = 2;
var c = 2;

var solution = a*b+c;

document.write("The total you are looking for is a*b+c = " +
solution.toFixed(2));
```

- 4. Browser:** Reload your corrected file **optionalLab13-1.htm**. You should no longer see an error. The script assumes a value of 2 for each variable in the equation, so it should return the correct answer of 6.

This lab demonstrates why debugging tools make your troubleshooting quick and easy. If you know where to look, you can usually find your errors and begin the work of correcting them. Be sure to practice using a variety of debuggers to see the differences between them.

