

VQS Study Guide ♦ Instructor Edition


Introduction

Welcome to the Instructor Edition of the *JavaScript for the World Wide Web, Fifth Edition: Visual QuickStart Guide, Student Edition*. This edition provides tools to help you guide and evaluate your students throughout your course on JavaScript.

Each chapter of the *JavaScript for the World Wide Web, Fifth Edition: Visual QuickStart Guide* has a corresponding study guide chapter in both the Instructor and Student Editions. (In the Student Edition the study guide section is found at the end of each chapter.) Each study guide chapter is divided into four main sections:

- **Learning Objectives** list the main points students should learn from the chapter.
- **Get Up and Running Exercises** are projects that help students synthesize and practice what they've learned. The exercises are based on techniques introduced in a chapter.
- **Class Discussion Questions** help you review a chapter with your class. You can also use this section to relate a chapter's points to workflow issues specific to the students in your class.
- **Review Questions** help you evaluate how well students have learned key details from a chapter. Each chapter contains a set of multiple choice and fill-in-the-blank questions. Some chapters have definition questions, and some include an exercise called "Find the Errors," where students are given some code and they are asked to find the errors in the code. You can incorporate the questions into your own tests.

The Instructor Edition includes additional material that doesn't appear in the Student Edition. You'll find instructor material in several places:

- Throughout the Instructor Edition, the instructor icon () marks instructor notes and answers to class discussion and definition questions. Paragraphs marked with this icon don't appear in the Student Edition.
- Answers to multiple-choice questions appear in boldface.
- Answers to fill-in-the-blank questions are filled in.

We hope you find the Instructor and Student Editions to be useful tools in your classes. If you have any suggestions that would make the Instructor and Student Editions more effective, please send them to studentedition@peachpit.com.

Chapter 1: Getting Acquainted with JavaScript ♦ Study Guide

Learning Objectives

- Review the history and uses of JavaScript.
- Understand the differences between JavaScript and Java.
 - ☞ Discuss when you would want to use each language and why. Why are they not interchangeable?
- Explore the differences between objects, properties, and methods.
 - ☞ Point out that properties are themselves objects.
- Understand the uses of values and variables.
 - ☞ Note the differences between (for example) `cat`, `"cat"`, `Cat`, and `"Cat"`. Strings and variables are different, and case sensitivity affects both.
- Understand assignments and comparisons.
 - ☞ Note the differences between single and double equals signs.

Get Up and Running Exercises

- Using dot syntax, how would you describe a text box inside a form on a Web page?
 - ☞ The goal isn't to get it exactly right, but to see if the students understand how dot syntax works in general. Correct answers would be along the lines of `document . form . textarea`.
- What tool would you choose to write your JavaScript with?
 - ☞ The goal here is to see if the students understand the difference between text editors, word processors, and WYSIWYG editors. There's no one correct answer, and it's particularly dependent on which platforms and tools you have available for student use. However, it's important that the tool creates plain text files, not a format such as a Microsoft Word document.

continues on next page

- Give an example of when you would use each of the event handlers from Table 1.1.
 - ☞ There are no single right answers here, but some examples are:
 - `onAbort`, `onError`: to put up an error message
 - `onMouseover`, `onMouseout`: to handle image rollovers
 - `onLoad`, `onUnload`: to open secondary windows
 - `onBlur`, `onFocus`: to know when a user has entered or left a form field
 - `onChange`: to respond when a user has changed a form field
 - `onClick`: to respond when a user clicks on a link or button
 - `onSelect`: to store a value prior to the user being able to change an entry
 - `onSubmit`: to process JavaScript form validation when the user has submitted a form

Class Discussion Questions

- What is the difference between `5 + 5`, `"5" + "5"`, `"5 + 5"`, `"5" + 5`, and `5 + "5"`?
 - ☞ Discuss how they came up with the answers. In order, the answers are the number 10, the string "55", the string "5 + 5", the string "55", and the string "55".
- When would you want to use `y = x + 1` vs. `y = x++`?
 - ☞ Make sure the students understand that these two are not equivalent. It's also worth discussing how `y = ++x` is different, too.
- What is the difference between the empty string (`"`) and `null`?
 - ☞ Using analogies helps here: for example, talk about the difference between an empty cup and having no cup at all.

Review Questions

Multiple choice

- Which of the following would be a poor tool for creating Web pages with JavaScript?
 - Microsoft FrontPage.

B. Microsoft Word.

☞ Word doesn't work, because it's a word processor, not a text editor.

 - Macromedia Dreamweaver.
 - Bare Bones BBEdit.
- If `a` has been set to 5 and `b` has been set to 6, which of the following is not true?
 - `a <= b`
 - `a != b`
 - `a < b`
 - `a == b`**
- Which of the following is a valid JavaScript variable name?
 - Date
 - 2day
 - today**
 - today'sDate

☞ Note that the problem with D is the apostrophe, not the use of "Date." A variable called `todaysDate` would be perfectly valid. Knowing that A is incorrect requires looking at Appendix B, as mentioned in the second Tip on page 10.
- Which of the following would be an appropriate use of JavaScript?
 - Validating form entries**
 - A Web site hit counter
 - An online guest book
 - Multi-player online games

☞ The incorrect answers all require the ability to read from and/or write to files, which JavaScript cannot do.
- Where would you find `<script>` tags?
 - In the `<head>` section of a Web page.
 - In the `<body>` section of a Web page.
 - Both.**
 - Neither.
- Variables contain:
 - Objects
 - Methods
 - Values**
 - Events

Fill-in-the-blank

- $a = 5$
 $b = a + 1$
After these lines:
What is the value of a ? 5
What is the value of b ? 6
- $a = 5$
 $b = a++$
After these lines:
What is the value of a ? 6
What is the value of b ? 5

☞ The ++ operator changes the value of a , but because it's after the variable, that increment occurs after the assignment.
- $a = 5$
 $b = ++a$
After these lines:
What is the value of a ? 6
What is the value of b ? 6

☞ Again, the ++ operator changes the value of a , and because it's before the variable, that increment occurs before the assignment, leaving both a and b with the same value.
- $thisVar = 7$
 $thisVar += 1$
After these lines:
What is the value of $thisVar$? 8
- $thatVar = 6$
 $thatvar *= 10$
After these lines:
What is the value of $thatVar$? 6

☞ Note that this is a trick question; the issue isn't the multiplication, but the fact that $thatVar$ and $thatvar$ are not the same variable.

Definitions

- What is LiveScript?
☞ LiveScript is what JavaScript was called when Netscape first released it.
- What is dot syntax?
☞ Dot syntax is the way JavaScript puts objects together, from the most generic object to the most specific.
- What is a property?
☞ A property is both a characteristic of an object and an object itself. For instance, `title` is a property of `document`, but `document.title` is an object in its own right that can then have properties and methods of its own.
- What is a method?
☞ A method is something that an object can do.
- What is an event handler?
☞ An event handler is a command JavaScript uses to deal with actions the user performs while visiting your Web page.

Chapter 2: Start Me Up! ♦ Study Guide

Learning Objectives

- Learn which browsers and situations require JavaScript code to be hidden by comments, and how, when, and why you'd want to use comments in your JavaScript code.
- Understand and identify how and when to use alerts, prompts, and confirmations, and when and why you'd want to use each.
- Learn how to control the flow of JavaScript code using `if/else` constructs.

Get Up and Running Exercises

- Create a Web page that uses JavaScript to display the user's browser.

```
<!DOCTYPE html PUBLIC "-//W3C//  
→ DTD XHTML 1.0 Transitional//EN"  
<html xmlns="http://www.w3.org/  
→ 1999/xhtml">  
<head>  
  <title>Exercise 2.1</title>  
</head>  
<body>  
  <scriptlanguage="Javascript"  
  → type="text/javascript" >  
  <!-- Hide scripts from old  
  → browsers  
    document.write  
    → (navigator.appName)  
  // End hiding scripts from old  
  → browsers -->  
  </script>  
</body>
```

```
</html>
```

- ☞ Note that (for this and all exercises in this book) the student's answer does not have to match this exactly. What's important is that they (1) show that they understand the material and (2) are able to create a Web page that accomplishes the desired task. It is perfectly acceptable, though, to give more credit for conciseness, simplicity, and clarity.

This exercise should be done by combining Scripts 2.1, 2.2, and 2.9.

continues on next page

- Create a Web page that uses JavaScript to ask a user their name, and then display that name on the Web page.

```
<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//EN"
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
  <title>Exercise 2.2</title>
  <script type="text/javascript"
  → language="Javascript">
  <!-- Hide scripts from old
  → browsers
    myName = prompt("What is your
    → name?", "")
  // End hiding scripts from old
  → browsers -->
  </script>
</head>
<body>
  <script type="text/javascript"
  → language="Javascript">
  <!-- Hide scripts from old
  → browsers
    document.write(myName)
  // End hiding scripts from old
  → browsers -->
  </script>
</body>
</html>
```

- ☞ This exercise should be done by combining Scripts 2.1, 2.2, and 2.6.

- Write the JavaScript code required to display a confirmation message when the user clicks on a link. Only proceed with the link if the user accepts the confirmation.

```
<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//EN"
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
  <title>Exercise 2.3</title>
</head>
<body>
  <a href="http://www.pixel.mu"
  → onclick="return confirm('Do
  → you really want to see his
  → page?')">This is my cat's
  → Web page</a>
</body>
</html>
```

- ☞ This exercise should be done by combining Scripts 2.5 and 2.8.

Class Discussion Questions

- Discuss the use of comments in your JavaScript code. When and why should you use them?
 - ☞ There's a list of browsers on page 16 that support JavaScript, so any browsers released prior to these versions will have problems if they come across uncommented JavaScript code. In addition, Web browsers for newer devices (especially consumer devices such as phones, PDAs, WebTV) may also have trouble if they come across uncommented JavaScript code, so it's never going to be safe to say, "No one uses those browsers anymore."

Regarding the use of comments:

```
// add one to myCount
myCount++
```

is redundant, but

```
// increment myCount so it is
in the range 1-10, not 0-9
myCount++
```

helps the next person who needs to modify the code understand why this code is there and what it is accomplishing. On the other hand,

```
/* Add one to myCount so it is
→ a number between 1 and 10
Do this by incrementing
→ myCount using the ++
→ operator
Not doing this will cause
→ errors to occur when we're
→ later
expecting it to be in the
→ correct range.
*/
myCount++
```

is just too much detail to describe one simple line of code. If you really need this much description, perhaps there was a simpler way to write the code in the first place.

- Discuss the differences between alerts, prompts, and confirmations, and when and why you'd want to use each.
 - ☞ Use alerts to inform the site visitor, prompts to get more information from the site visitor, and confirmations when you want the user to approve or cancel something that's about to happen.
- Discuss the value of adding a DOCTYPE declaration at the beginning of your HTML documents.

Review Questions

Multiple choice

- If you want to write a script that puts the current date and time on a Web page, the code would go into:
 - A header script.
 - A body script.**
 - A function.
 - An event handler.
- If you want to write a script that stores the user's browser name, the code would go into:
 - A header script.**
 - A body script.
 - A function.
 - An event handler.
- Which of the following sets `myVar` to 6?
 - ```
if (true) {
 myVar = 6
}
else {
 myVar = 8
}
```
  - `myVar = (true) ? 6 : 8`
  - `if (true) myVar = 6 else myVar = 8`
  - All of the above.**

### Fill-in-the-blank

Referring to this code, answer the following three questions:

```
ans = prompt("Please answer this
→ question: ", "")
```

- If the user enters a response and clicks the OK button, what is the value of `ans`? Whatever the user entered
- If the user enters a response and clicks the Cancel button, what is the value of `ans`? Null
- If the user clicks the OK button without entering a response, what is the value of `ans`? The empty string
  - Note that the answers to questions 2 and 3 are different. The students need to understand that these are not the same result.

### Definitions

- What is a conditional?
  - A conditional is a situation where the script poses a test, usually followed by an action based on the results of the test.
- What is redirecting?
  - Redirecting is automatically sending the user to a different Web page based on the results of a conditional test.
- What is browser detection?
  - Browser detection is the way you determine the kind of browser that the user has.

# Chapter 3: Language Essentials ♦ Study Guide

## Learning Objectives

- Learn how to control the flow of JavaScript code using for loops.
  - ☞ For loops are one of the basic building blocks of any language.
- Learn how to use do loops.
  - ☞ Virtually anything that can be done with a do loop can be done with a for loop, but it's still useful for beginning scripters to learn how to do both.
- Explore the use and purpose of JavaScript functions.
  - ☞ Functions are another basic building block. Students should learn that functions should be in header (not body) scripts, and that they're preferable to ever having to do the same thing twice.
- Learn to combine two or more smaller JavaScript fragments into one larger piece.
  - ☞ For example, students should never need to have two script tags inside the <head> of their Web page. Instead, the code should be combined and put into one script tag.
- Learn to use switch/case to handle conditionals with more than two choices.
  - ☞ The last of the basic ways of handling flow control within JavaScript.

## Get Up and Running Exercises

- Write the JavaScript code to ask a user how many times they want to loop, and then write the code to loop that many times, each time displaying the message "I've gone around x time(s) out of y", where x is the iteration count and y is the number of times that the user entered.

```
<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Exercise 3.1</title>
 <script type="text/javascript"
 → language="Javascript">
 <!-- Hide scripts from old
 → browsers
 function loopAlert
 → (thisIteration) {
 // Increment thisIteration so
 → it displays starting at 1,
 → not 0
 thisIteration++
 alert("I've gone around " +
 → thisIteration + " time(s)
 → out of " + loopCount)
 }
 // End hiding scripts from old
 → browsers -->
</script>
</head>
<body>
 <script type="text/javascript"
 → language="Javascript">
 <!-- Hide scripts from old
 → browsers
```

*continues on next page*

```

loopCount = prompt("How many
→ times do you want to loop?",
→ "")
for (i=0;i<loopCount;i++) {
 loopAlert(i)
}
// End hiding scripts from old
→ browsers -->
</script>
</body>
</html>

```

- ☞ This exercise can be done by combining Scripts 2.4, 2.6, 3.2, and 3.7.

Note that this example will do nothing at all (no error message or loop alert) if the user gave a non-numeric response to the prompt. As checking for data validity hasn't yet been covered (that's in Chapter 8), consider giving extra credit if a student figures out how to do it.

- Script 3.12 creates a scrolling status bar message that scrolls from right to left. Revise it to scroll your own message in the opposite direction.

```

<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>My JavaScript page</
 → title>
 <script language="Javascript"
 → type="text/javascript">
 <!-- Hide script from old
 → browsers

```

```

myMsg = "Student's message
→ here "
i = myMsg.length

function scrollMsg() {
 window.status = myMsg.
 → substrng(i,myMsg.length)
 → + myMsg.substrng(0,i)
 if (i > 0) {
 i--
 }
 else {
 i = myMsg.length
 }
 setTimeout("scrollMsg()",
 → 50)
}

```

```

// End hiding script from old
→ browsers -->
</script>
</head>
<body bgcolor="#FFFFFF"
→ onload="scrollMsg()">
<h2>I'm a kewl JavaScript dood
with a scrolling status bar!</h2>
</body>
</html>

```

- ☞ What's being tested here is not the ability to create scrolling status bars (they're not that common these days), but the ability to modify and debug code, along with an understanding of string manipulation. Students should realize that they don't need to change what `window.status` is set to—they only need to change those lines involving setting and checking the value of `i`.

- Re-write Script 3.10 (Handling Errors) to use multi-level conditionals instead.

```

<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Square Root Calculator</
→ title>
 <script language="Javascript"
→ type="text/javascript">
 <!-- Hide script from old
→ browsers

 ans = prompt("Enter a
→ number", "")

 switch(true) {
 case (!ans):
 alert("You didn't enter
→ anything")
 break
 case (isNaN(ans)):
 alert("That isn't a
→ number")
 break
 default:
 alert("The square
→ root of " + ans + " is "
→ + Math.sqrt(ans))
 }

 // End hiding script from old
→ browsers -->
 </script>
</head>
<body bgcolor="#FFFFFF">
<noscript>
 <h2>This page requires
→ JavaScript.</h2>

```

```

</noscript>
</body>
</html>

```

- ☞ This uses an alternate version of `switch/case` than the one presented in the book, so make sure that you cover this approach in class. This version is used when there isn't a single value that can be checked against, so instead, `switch` is passed `true`, and each individual case does its own check. Of course, there are many different ways to approach this problem, so students can come up with their own unique answer, too, so long as they end up with the same result.

## Class Discussion Questions

- What are the reasons for and against using the plug-in detection method described in the book?
  - ☞ Pro: it's simple, and it works with JavaScript. Con: it doesn't work with Internet Explorer for Windows, the most popular browser in use.
- What are the pros and cons of using a zero-based system like JavaScript uses?
  - ☞ This is covered in the sidebar on page 33, but it's important to make sure your students understand how this works. Point out that while `indexOf` is zero-based, `length` shows the one-based number that you might or might not expect.
- When is it right to use body scripts and when is it right to use header scripts?
  - ☞ Header scripts are for functions and those variables that must be set when the page first loads. Body scripts are used for code that actually modifies the appearance of the page itself. Whenever possible, body scripts should call functions that are in the head.
- What are the benefits of using functions, instead of just repeating code as needed?
  - ☞ You can call a function many times during a script, saving much typing. Changing the function changes the way it works throughout the script, which helps simplify debugging.

## Review Questions

### Multiple choice

1. If you want a loop to go around ten times, which of the following is the correct JavaScript syntax?
  - A. `for (i=1;i<10;i++)`
  - B. `for (i=1;i=10;i++)`
  - C. `for (i=0;i<10;i++)`**
  - D. `for (i=0;i<=10;i++)`
2. A JavaScript function begins with the word `function` followed by:
  - A. A parenthesis
  - B. The function name**
  - C. A semicolon
  - D. Nothing
3. Giving information to a function is called:
  - A. Sending
  - B. Teleporting
  - C. Passing**
  - D. Giving
4. NaN means:
  - A. Sodium Nitrate
  - B. Nary a Number
  - C. Non Activated Number
  - D. Not a Number**
5. You can put how many scripts on a single Web page?
  - A. Only one
  - B. Two; one head script and one body script
  - C. As many as you want**

6. The `setTimeout` command does what?

A. Tells the script to end

**B. Pauses the script**

C. Tells the script to go sit in the corner

D. Tells the script to execute faster

### Fill-in-the-blank

Referring to this code, answer the following three questions:

```
myString = "Hello World!"
worldFound = myString.indexOf("world")
WorldFound = myString.indexOf("World")
```

1. What is the value of `worldFound`? -1
2. What is the value of `WorldFound`? 6
3. What is the value of `myString.length`? 12

### Find the Errors

This code should print out the numbers 1-10, but it doesn't. Find all the errors. Fill in as many as you can find.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
→ XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Display Integers</title>
</head>
<body>
 <script type="text/javascript"
 → language="Javascript">
 <!-- Hide scripts from old
browsers
 for (i=0;i++;i<10) {
 i++
 documentWrite(i)
 // End hiding scripts from old
 → browsers -->
```

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

1. The method `documentWrite(i)` should be written as `document.write(i)`.
2. The code within the `for` loop is in the wrong order: it should be the initialization step, then the limitation step, and then the increment step. The code as written will loop forever.
3. Incrementing `i` within the loop causes it to be incremented twice: once by the `++`, and once by the loop itself. Either use a temporary variable, or make the loop go from 1 to 10 instead of 0 to 9.
4. The closing brace that ends the `for` loop is missing.
5. One extra line is left blank just in case the student decides to fix the bugs in some other manner (which is perfectly valid). There are any number of ways to write any given example, and the students should know that there's no one right way (as mentioned on page 21).

# Chapter 4: Image Basics ♦ Study Guide

## Learning Objectives

- Learn the basics of adding rollovers to a Web page.
  - ☞ Rollovers are the most common usage of JavaScript, and students should have a solid grasp of how to do them by the end of this chapter.
- Learn how to handle more complex rollovers, including multiple rollovers, how to trigger rollovers from a link, and how to use links to change both single and multiple rollovers.
  - ☞ This may not cover all permutations of rollovers that exist, so discuss with students other variations that they've seen.
- Understand how to use functions to ease coding multiple rollovers and rollovers with multiple images.
  - ☞ Any time code is repeated, it's a candidate for turning into a function, so that's been done here to remind students of this principle.

## Get Up and Running Exercises

- The tip on page 68 describes a travel site, with three text links (Scotland, Tahiti, and Cleveland) each changing a single image to the desired destination when rolled over. Create this page with your own three destinations, using Script 4.4 as a starting point.

```
<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
<title>Exercise 4.1</title>
<script language="Javascript"
→ type="text/javascript">
<!-- Hide script from old
→ browsers
 if (document.images) {
 scotland = new Image
 tahiti = new Image
 cleveland = new Image
 backgrnd = new Image

 scotland.src = "images/
→ scotland1.jpg"
 tahiti.src = "images/
→ tahiti1.jpg"
 cleveland.src = "images/
→ cleveland1.jpg"
 backgrnd.src = "images/
→ spacer.gif"
 }

function chgImg(imgField,
→ newImg) {
 if (document.images) {
```

*continues on next page*

```

 document[imgField].src=
 → eval(newImg + ".src")
 }
}

// End hiding script from old
→ browsers -->
</script>
</head>
<body bgcolor="#FFFFFF">
<a href="scotland.html"
→ onmouseover="chgImg('picField',
→ 'scotland')" onmouseout="chgImg
→ ('picField','backgrnd')">
→ >Scotland

<a href="tahiti.html"
→ onmouseover="chgImg('picField',
→ 'tahiti')" onmouseout="chgImg
→ ('picField','backgrnd')">Tahiti
→

<a href="cleveland.html"
→ onmouseover="chgImg('picField',
→ 'cleveland')" onmouseout=
→ "chgImg('picField','backgrnd')">
→ >Cleveland

</body>
</html>

```

- ☞ There are a number of ways to do this one; this example combines the rollovers from Scripts 4.4 and 4.5 with the function from Script 4.7.

- Modify the previous exercise to change multiple images (at least three) when one of the destination links is rolled over.

```

<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
<title>Exercise 4.2</title>
<script language="Javascript"
→ type="text/javascript">
<!-- Hide script from old
→ browsers
 if (document.images) {
 scotland1 = new Image
 tahiti1 = new Image
 cleveland1 = new Image

 scotland2 = new Image
 tahiti2 = new Image
 cleveland2 = new Image

 scotland3 = new Image
 tahiti3 = new Image
 cleveland3 = new Image

 scotland1.src = "images/
→ scotland1.jpg"
 tahiti1.src = "images/
→ tahiti1.jpg"
 cleveland1.src = "images/
→ cleveland1.jpg"

 scotland2.src = "images/
→ scotland2.jpg"
 tahiti2.src = "images/
→ tahiti2.jpg"

```

*continues on next page*



```

cleveland2.src = "images/
→ cleveland2.jpg"

scotland3.src = "images/
→ scotland3.jpg"
tahiti3.src = "images/
→ tahiti3.jpg"
cleveland3.src = "images/
→ cleveland3.jpg"

backgrnd = new Image
backgrnd.src = "images/
→ spacer.gif"
}

function chgImg
→ (imgField,newImg) {
 if (document.images) {
 document[imgField].src=
 → eval(newImg + ".src")
 }
}

// End hiding script from old
→ browsers -->
</script>
</head>
<body bgcolor="#FFFFFF">
<a href="scotland.html" onmouse
→ over="chgImg('picField1',
→ 'scotland1');chgImg('picField2',
→ 'scotland2');chgImg('picField3',
→ 'scotland3');" onmouseout=
→ "chgImg('picField1','backgrnd');
→ chgImg('picField2','backgrnd');
→ chgImg('picField3','backgrnd');"
→ >Scotland


```

```

<a href="tahiti.html"
→ onmouseover="chgImg('picField1',
→ 'tahiti1');chgImg('picField2',
→ 'tahiti2');chgImg('picField3',
→ 'tahiti3');" onmouseout="chgImg
→ ('picField1','backgrnd');chgImg
→ ('picField2','backgrnd');chgImg
→ ('picField3','backgrnd');"
→ >Tahiti

<a href="cleveland.html"
→ onmouseover="chgImg
→ ('picField1','cleveland1');
→ chgImg('picField2',
→ 'cleveland2');chgImg
→ ('picField3','cleveland3');"
→ onmouseout="chgImg('picField1',
→ 'backgrnd');chgImg('picField2',
→ 'backgrnd');chgImg('picField3',
→ 'backgrnd');">Cleveland
→

</body>
</html>

```

- ☞ If the previous exercise was done correctly, all that should have been added here was two more images and two more calls to `chgImg()` for each `onmouseover` and `onmouseout`.

*continues on next page*

- Modify the previous exercise to use images instead of text links.

```

<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
<title>Exercise 4.3</title>
<script language="Javascript"
→ type="text/javascript">
<!-- Hide script from old
→ browsers
if (document.images) {
 scotland1 = new Image
 tahiti1 = new Image
 cleveland1 = new Image

 scotland2 = new Image
 tahiti2 = new Image
 cleveland2 = new Image

 scotland3 = new Image
 tahiti3 = new Image
 cleveland3 = new Image

 scotland1.src = "images/
→ scotland1.jpg"
 tahiti1.src = "images/
→ tahiti1.jpg"
 cleveland1.src = "images/
→ cleveland1.jpg"

 scotland2.src = "images/
→ scotland2.jpg"
 tahiti2.src = "images/
→ tahiti2.jpg"
 cleveland2.src = "images/
→ cleveland2.jpg"

 scotland3.src = "images/
→ scotland3.jpg"

```

```

 tahiti3.src = "images/
→ tahiti3.jpg"
 cleveland3.src = "images/
→ cleveland3.jpg"

 backgrnd = new Image
 backgrnd.src = "images/
→ spacer.gif"
}

function chgImg
→ (imgField,newImg) {
 if (document.images) {
 document[imgField].src=
→ eval(newImg + ".src")
 }
}

// End hiding script from old
→ browsers -->
</script>
</head>
<body bgcolor="#FFFFFF">
<a href="scotland.html" onmouse
→ over="chgImg('picField1',
→ 'scotland1');chgImg
→ ('picField2','scotland2');
→ chgImg('picField3',
→ 'scotland3');" onmouseout="
→ chgImg('picField1','backgrnd');
→ chgImg('picField2',
→ 'backgrnd');chgImg
→ ('picField3','backgrnd')">
→


```

*continues on next page*

```

<a href="tahiti.html"
→ onmouseover="chgImg
→ ('picField1','tahiti1');
→ chgImg('picField2','tahiti2')
→ ;chgImg('picField3','tahiti3')"
→ onmouseout="chgImg('picField1',
→ 'backgrnd');chgImg('picField2',
→ 'backgrnd');chgImg('picField3',
→ 'backgrnd')">

<a href="cleveland.html"
→ onmouseover="chgImg
→ ('picField1','cleveland1');
→ chgImg('picField2',
→ 'cleveland2');chgImg
→ ('picField3','cleveland3')"
→ onmouseout="chgImg('picField1',
→ 'backgrnd');chgImg('picField2',
→ 'backgrnd');chgImg('picField3',
→ 'backgrnd')">

</body>
</html>

```

- ☞ If the previous examples were done correctly, all that should have been added here was changing the three destinations from text to image tags.

## Class Discussion Questions

- What are the pros and cons of using image rollovers as part of your user interface?
  - ☞ Pro: User interface design on the Web has turned rollovers into a standard. People expect that buttons will change color when you mouse over them, to the point where a button that doesn't change won't even be thought of as a possible link (despite the cursor change). Con: Some Web sites depend on having JavaScript enabled to navigate a site. Good UI design should always keep in mind that some people might be visiting with JavaScript disabled.
- Why is the image rollover placed in the <a> tag, and not in the <img> tag?
  - ☞ This is the case primarily for historical reasons, as Netscape started off doing it this way in Navigator 2.0. Some browsers allow `onmouseover` and `onmouseout` in the <img> tag, and students should be encouraged to try them in as many browsers as possible to learn which ones do and don't support this approach. In general, they'll find that that approach works in IE 4+ and Netscape 6+. Discuss when and why it's feasible to ignore browsers other than these.
- What are the benefits and drawbacks of using functions to handle rollovers?
  - ☞ Pro: Less code, no need to work around Netscape 2 issues.
 

Con: Slightly slower, as the function always (1) checks to see if `document.images` exists, and (2) does an `eval()` (always a particularly slow call) on the new image name.

*continues on next page*

- If you're willing to limit the flexibility of what your images and objects are named, how much JavaScript can be cut out of these examples?
  - ☞ Many other places that teach how to do image rollovers use approaches that constrain what the images can be called. If you set absolute naming conventions, you should be able to minimize the code drastically.

## Review Questions

### Multiple choice

1. Which of the following could happen if you use the same `name` attribute for multiple images?
    - A. All of the images change when one is moused over.
    - B. A random image changes when one is moused over.
    - C. Both A and B.**
    - D. No images change when one is moused over.
  2. Match the following image rollover errors with their associated cause:
    - A. Nothing changes when you put the cursor over the image. **(3)**
    - B. Part of the original image displays behind the rolled-over image ("ghosting"). **(1)**
    - C. There's a long pause before the roll-over version of the image displays. **(4)**
    - D. When using Netscape, the rolled-over version of the image displays at the wrong size, or when using Internet Explorer, rolling over the image causes the page layout to reflow. **(2)**
    1. One or more of the images is a transparent GIF.
    2. The on and off versions of the image have different dimensions.
    3. The `name` attribute was put on the `<a>` tag, not the `<img>` tag, or the `onmouseover` and `onmouseout` event handlers were put on the `<img>` tag, not the `<a>` tag.
    4. The `onmouseover` version of the image is not being pre-loaded.
-

3. Which of the following is the correct syntax for creating a new image object:
  - A. `myPic = new Image`
  - B. `myPic = new image`
  - C. `myPic = new images`
  - D. `myPic = new Images`
4. Which of the following will set the `src` property of `myPic` to the `src` property of `redButton`:
  - A. `myPic.src = redButton`
  - B. `myPic.src = redButton.src`
  - C. `myPic = redButton`
  - D. `myPic.src = eval(redButton + ".src")`
5. The `onmouseover` and `onmouseout` event handlers go:
  - A. Inside the script tag
  - B. In the `img` tag
  - C. **In the `a` tag**
  - D. You can put them anywhere in the script
6. The `name` attribute goes in:
  - A. **In the `img` tag**
  - B. In the `a` tag
  - C. In the header script
  - D. You can put it anywhere in the script
3. Each `image` object and `img` tag must have a unique name.
4. You can call multiple functions in the same line by putting a semicolon between commands.
5. The original and replacement images for a rollover must have identical dimensions.
6. GIF images used in rollovers should not be transparent.
7. To check to see if a rollover script is running in a modern browser, you need to use the code `if (document.images)`.

### Find the Errors

This code should work identically to Script 4.3, but it doesn't. Find all the errors:

```
<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Image Errors</title>
<script language="Javascript"
→ type="text/javascript">
<!-- Hide script from old browsers
 if (document.images) {
 button1Red = new image
 button1Blue = new image
 button2Red = new image
 button2Blue = new image
 button1Red.src = "images/
→ redButton1.gif"
 button1Blue.src = "images/
→ blueButton1.gif"
 button2Red.src = "images/
→ redButton2.gif"
 button2Blue.src = "images/
blueButton2.gif"
 }
```

### Fill-in-the-Blank

1. Making images change when the user mouses over the image is called a rollover.
2. You use the onmouseover event handler to make the rollover happen.

*continues on next page*



# Chapter 5: More Fun with Images ♦ Study Guide

## Learning Objectives

- Add both auto-running and user-driven slideshows and banners to a Web page.
  - ☞ Note that auto-running slideshows and cycling banners are the same thing.
- Add individual and multiple random images to a Web page.
  - ☞ This can be used to make a page look different each time it's loaded, but the random images all need to have the same dimensions.
- Combine image rollovers with image maps to create menu effects.
  - ☞ This can be a simple way to add menus to a page without having to get into the hard-core DHTML that they'll learn later in the book.

## Get Up and Running Exercises

- Modify Script 5.5 so that the slideshow only goes forward and warns the user when they've reached the end of the slideshow.

```
<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Exercise 5.1</title>
<script language="Javascript"
→ type="text/javascript">
<!-- Hide script from old
→ browsers
 myPix = new Array("images/
→ pathfinder.gif", "images/
→ surveyor.gif", "images/
→ surveyor98.gif")
thisPic = 0
imgCt = myPix.length - 1
function processNext() {
 if (document.images) {
 if (thisPic == imgCt) {
 alert("You've reached the
→ end of the slideshow")
 }
 else {
 thisPic++
 document.myPicture.src=
→ myPix[thisPic]
 }
 }
}
// End hiding script from old
→ browsers -->
</script>
</head>
```

*continues on next page*

```

<body bgcolor="white">
<h1 align="center">US Missions to
→Mars

 <a
→href="javascript:
processNext()">Next >>
</h1>
</body>
</html>

```

As always, there are many possible ways for the students to successfully complete this task. This example combines Script 5.5 with an alert message (from Chapter 2).

- Write an auto-running slideshow (one that runs without input from the user) that can run either forwards or backwards if/when the user chooses to switch direction.

```

<!DOCTYPE html PUBLIC "-//W3C//
→DTD XHTML 1.0 Transitional//
→EN">
<html xmlns="http://www.w3.org/
→1999/xhtml">
<head>
<title>Exercise 5.2</title>
<script language="Javascript"
→type="text/javascript">
<!-- Hide script from old
→browsers
 myPix = new Array("images/
→callisto.jpg","images/
→europa.jpg","images/io.jpg",
→"images/ganymede.jpg")
thisPic = 0
imgCt = myPix.length - 1
direction = 1

```

```

function chgSlide() {
if (document.images) {
 thisPic = thisPic + direction
 if (thisPic > imgCt) {
 thisPic = 0
 }
 if (thisPic < 0) {
 thisPic = imgCt
 }
 document.myPicture.src=
→myPix[thisPic]
 setTimeout("chgSlide()",
→3 * 1000)
}
}
// End hiding script from old
→browsers -->
</script>
</head>
<body bgcolor="white" onload=
→"chgSlide()">
<h1 align="center">Jupiter's
→Moons

<a href="javascript:direction=
→-1"><< Backwards</
→a> <a href=
→"javascript:direction=
→1">Forwards >></h1>
</body>
</html>

```

This example combines Scripts 5.1 and 5.6. The key to this for students is that they need to understand that the user can change the value of `direction` at any time, and because `direction` has a numeric value, the script can run either forwards or backwards just by being based on its value. This is covered in the description of Script 5.6.



- Using Script 5.10 as a model, design a Web page with at least two menus, each of which contains at least four items. For example, you could design a site for your Web design business, with the first menu displaying services you offer and the second displaying examples from your portfolio.

```

<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
 <head>
 <title>Example 5.3</title>
 <script language="javascript"
→ type="text/javascript">
 <!-- Hide script from old
→ browsers

 if (document.images) {
 img1 = new Image
 img2 = new Image
 img3 = new Image
 img4 = new Image
 img5 = new Image
 img6 = new Image
 img7 = new Image
 img8 = new Image
 img9 = new Image
 img10 = new Image
 img11 = new Image
 img12 = new Image

 img1.src =
→ "images/nav_01_01.gif"
 img2.src =
→ "images/nav_01_02.gif"
 img3.src =
→ "images/nav_01_03.gif"
 img4.src = "images/nav_01_
→ 04.gif"

```

```

 img5.src =
→ "images/nav_01_on.gif"
 img6.src =
→ "images/nav_01_off.gif"
 img7.src =
→ "images/nav_02_01.gif"
 img8.src =
→ "images/nav_02_02.gif"
 img9.src =
→ "images/nav_02_03.gif"
 img10.src =
→ "images/nav_02_04.gif"
 img11.src =
→ "images/nav_02_on.gif"
 img12.src =
→ "images/nav_02_off.gif"
 }

 function chgImg
→ (imgField,newImg) {
 if (document.images) {
 document[imgField].src =
→ eval(newImg + ".src")
 }
 }

 // End hiding script from old
→ browsers -->
 </script>
 </head>
 <body bgcolor="#FFFFFF">
 <map name="menu01" id="menu01">
 <area shape="rect" coords=
→ "14,3,105,15" href=
→ "company/index.html"
 <onmouseover="chgImg
→ ('menu1','img5')">
 <alt="company" />

```

*continues on next page*

```

<area shape="rect" coords=
→ "13,18,78,27" href=
→ "company/background.html"
→ onmouseover="chgImg
 → ('menu1','img1')"
 → alt="background" />
<area shape="rect" coords=
→ "13,28,45,36" href=
→ "company/press.html"
→ onmouseover="chgImg
→ ('menu1','img2')"
→ alt="press" />
<area shape="rect" coords=
→ "13,37,40,45" href=
→ "company/staff.html"
→ onmouseover="chgImg
→ ('menu1','img3')"
→ alt="staff" />
<area shape="rect" coords=
→ "13,46,38,55" href=
→ "company/jobs.html"
→ onmouseover="chgImg
→ ('menu1','img4')"
→ alt="jobs" />
</map>
<map name="menu02" id="menu02">
 <area shape="rect" coords=
 → "14,3,105,15" href=
 → "portfolio/index.html"
 → onmouseover="chgImg
 → ('menu2','img11')"
 → alt="portfolio" />
 <area shape="rect" coords=
 → "13,18,78,27" href=
 → "portfolio/clients.html"
 → onmouseover="chgImg
 → ('menu2','img7')"
 → alt="clients" />
 </map>
 <area shape="rect" coords=
 → "13,28,45,36" href=
 → "portfolio/projects.html"
 → onmouseover="chgImg
 → ('menu2','img8')"
 → alt="projects" />
 <area shape="rect" coords=
 → "13,37,40,45" href=
 → "portfolio/awards.html"
 → onmouseover="chgImg
 → ('menu2','img9')"
 → alt="awards" />
 <area shape="rect" coords=
 → "13,46,38,55" href=
 → "portfolio/pro_bono.html"
 → onmouseover="chgImg
 → ('menu2','img10')"
 → alt="pro bono" />
</map>
<a href="company/index.html"
→ onmouseout="chgImg('menu1',
→ 'img6')">
<a href="portfolio/index.html"
→ onmouseout="chgImg('menu2',
→ 'img12')">
</body>
</html>

```

☞ If you're providing graphics for the students to work with, design them so that the "off" versions of the images are just empty white space except for the top-most rollover area. This approach will give the impression of a pull-down menu. Also, note that for each item on a given menu, you'll need that many image files plus one (the latter for the "off" version). The above example has two menus, each of which have five choices, so there's two times six or twelve total images.

## Class Discussion Questions

■ The effect of Script 5.10, "Combining a Rollover with an Image Map," can be duplicated by using multiple images that touch each other, and then changing each image individually when one is rolled over. What are the pros and cons of using image map rollovers vs. using multiple rollover images placed precisely with a table?

☞ Pro: With image maps, you know that your images are going to always be precisely where you want them. Forcing multiple images to butt up against each other by placing them within a table is prone to errors unless you test with every possible browser.

Con: The entire image has to be redisplayed every time one part of the image map is rolled over. If any part of the image needs to change, all of the images in the image map may need to be recreated.

■ What are the benefits of using JavaScript to create animated banners, rather than using an animated GIF?

☞ With a JavaScript banner, you can use higher-quality images, such as JPEG or PNG.

*continues on next page*

- You could use a CGI to create a slideshow, instead of JavaScript. What are the pros and cons of the JavaScript approach?

☞ Pro: A new page doesn't have to be loaded for every image when you use JavaScript (as it would if CGI was used). The JavaScript approach also puts less stress on the server, as it doesn't require the server to generate an entirely new page every time the user wants to see a new slide. Lastly, using JavaScript the Web developer doesn't need to have access to the server or an account with CGI privileges in order to write a slideshow.

Con: The person viewing the Web page needs to have JavaScript enabled.

## Review Questions

### Multiple choice

1. Which of the following lines of code will cause the `rotate()` function to be called every three seconds:
  - A. `setTimeout("rotate()", 3)`
  - B. `setTimeout(3, "rotate()")`
  - C. `setTimeout(3000, "rotate()")`
  - D. `setTimeout("rotate()", 3000)`**
2. Which of the following is a valid line of JavaScript?
  - A. `bigPix = new Array("planes.jpg", "trains.jpg", "automobiles.jpg")`**
  - B. `bigPix = new Array("planes.jpg", "trains.jpg", "automobiles.jpg")`
  - C. `bigPix = new Array(planes.jpg, trains.jpg, "automobiles.jpg")`
  - D. `bigPix = new Array("planes.jpg"; "trains.jpg"; "automobiles.jpg")`
  - E. `Array bigPix = new("planes.jpg", "trains.jpg", "automobiles.jpg")`
3. You can use an array to:
  - A. Contain the names of image files in a cycling banner.
  - B. List multiple variables for later use.
  - C. Contain the caption text for each image in a slideshow.
  - D. All of the above.**

4. If your page is XHTML, you should make sure that your JavaScript is marked as:
- A. PCDATA
  - B. MSHTML
  - C. CDATA**
  - D. XML Text
5. If there are six images in the `bannerImages` array, what is the value of `bannerImages.length`?
- A. 0
  - B. 5
  - C. 6**
  - D. 7

### Fill-in-the-blank

Referring to this code, answer the following set of questions:

```
myImages = new Array("image1.gif",
 → "image2.gif", "image3.gif")
imgCt = myImages.length
randomNum = Math.floor(Math.random() *
 → imgCt)
document.myPicture.src =
 → myImages[randomNum]
```

1. What is the value of `myImages[1]`?  
"image2.gif"
2. What type of object is `myImages[randomNum]`? A text string
3. What is the value of `imgCt`? 3
4. What is the lowest possible value of `randomNum`? 0
5. What is the highest possible value of `randomNum`? 2

### Find the Errors

This attempt to answer the first “Get Up and Running” exercise, above, has a number of errors. Find them all.

```
<!DOCTYPE html PUBLIC "-//W3C//
 → DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/
 → 1999/xhtml">
<head>
 <title>Exercise 5.1</title>
<script language="Javascript"
 → type="text/javascript">
<!-- Hide script from old browsers
 myPix = new Array(pathfinder.gif,
 → surveyor.gif, surveyor98.gif)
 thisPic = 0
 imgCt = myPix.length
 function processNext() {
 if (document.images) {
 if (thisPic = imgCt) {
 alert("You've reached the end of
 → the slideshow")
 }
 else {
 thisPic++
 document.myPicture.src=
 → myPix[thisPic]
 }
 }
 }
 // End hiding script from old
 → browsers -->
</script>
</head>
<body bgcolor="white">
<h1 align="center">US Missions to
 → Mars

```

*continues on next page*

```
<a href=
→ "javascript:processNext()">Next
→ >>
</h1>
</body>
</html>
```

1. The variable `imgCt` is set to `myPix.length`, where it should be `myPix.length-1`.
2. The names of the images stored in the `myPix` array should each be enclosed in quotes.
3. The variable `thisPic` is being set to `imgCt` when it should be compared to it instead.
4. The `name` attribute of the slideshow image has been left off.
5. One extra line is here in case the student finds different errors.

# Chapter 6: Frames, Frames, and More Frames ♦ Study Guide

## Learning Objectives

- Learn how to use JavaScript to control how your pages appear in framesets.
  - ☞ Framed sites are less common than they once were, but some sites still use frames, and other sites still try to put others inside their own frames. Consequently, it's worth learning how JavaScript can control framed sites.
- Use JavaScript to share functions and store information in frames.
  - ☞ For example, you could use a combination of JavaScript and frames to create an online quiz that would store information about how the student has done on the quiz so far in one of the frames.
- Learn how to work with multiple frames.
  - ☞ HTML can't change multiple frames with a single click, so JavaScript is required to do this.

## Get Up and Running Exercises

- Find one or more sites on the Web that use frames. Analyze why the site's designer chose to use frames, and whether this was the best approach to designing the site. If the site uses JavaScript to handle its frames, describe what the scripts do. Come to class prepared to report on your research.
- Create a site with three frames: one navigation bar and two content areas. Write a script that uses links in the navigation bar to change the content in one of the two content areas, using Scripts 6.14–6.16 as a starting point.

```
<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Frameset//EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Exercise 6.2</title>
</head>
<frameset cols="30%,70%">
 <frame src="navbar.html"
 → name="left" />
 <frameset rows="50%,50%">
 <frame src="content1.html"
 → name="content1" />
 <frame src="content2.html"
 → name="content2" />
 </frameset>
</frameset>
</html>
<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
```

*continues on next page*

```

<head>
 <title>Nav Bar</title>
 <script language="Javascript"
 → type="text/javascript">
 <!-- Hide script from old
 → browsers

 function writeContent
 → (frameNo,thisPage) {
 thisFrame = eval("parent.
 → content" + frameNo)
 thisFrame.document.write
 → ("<html><head>
 → <\/head><body bgcolor=
 → '#FFFFFF'><h1>")
 thisFrame.document.
 → write("You are now
 → looking at page "
 → +thisPage+ " in frame
 → "+frameNo+ ".")
 thisFrame.document.write
 → ("<\/h1><\/body>
 → <\/html>")
 thisFrame.document.close()
 }

 // End hiding script from old
 → browsers -->
 </script>
</head>
<body bgcolor="#FFFFFF">
<h1>Navigation Bar</h1>
<h2>
 <a href="javascript:
 → writeContent(1,1)">Frame 1,
 → Page 1

 <a href="javascript:
 → writeContent(1,2)">Frame 1,
 → Page 2

 <a href="javascript:
 → writeContent(1,3)">Frame 1,
 → Page 3


```

```

 <a href="javascript:
 → writeContent(2,1)">Frame 2,
 → Page 1

 <a href="javascript:
 → writeContent(2,2)">Frame 2,
 → Page 2

 <a href="javascript:
 → writeContent(2,3)">Frame 2,
 → Page 3
</h2>
</body>
</html>
<!DOCTYPE html PUBLIC "-//W3C//
 → DTD XHTML 1.0 Transitional//
 → EN">
<html xmlns="http://www.w3.org/
 → 1999/xhtml">
<head>
 <title>Content frame 1</title>
</head>
<body bgcolor="#FFFFFF">
</body>
</html>
<!DOCTYPE html PUBLIC "-//W3C//
 → DTD XHTML 1.0 Transitional//
 → EN">
<html xmlns="http://www.w3.org/
 → 1999/xhtml">
<head>
 <title>Content frame 2</title>
</head>
<body bgcolor="#FFFFFF">
</body>
</html>

```

- ☞ Note that there are four pages here (one frameset, one navigation page, and two content pages); by this point, the students should know how to figure out which is which.



- Modify the previous exercise to have each link in the navigation bar update both content areas simultaneously.

```

<!DOCTYPE html PUBLIC "-//W3C//
 → DTD XHTML 1.0 Transitional//
 → EN">
<html xmlns="http://www.w3.org/
 → 1999/xhtml">
<head>
 <title>Nav Bar</title>
 <script language="Javascript"
 → type="text/javascript">
 <!-- Hide script from old
 → browsers

 function writeContent
 → (frameNo,thisPage) {
 thisFrame = eval("parent.
 → content" + frameNo)
 thisFrame.document.write
 → ("<html><head>
 → <\head><body
 bgcolor='#FFFFFF'><h1>")
 thisFrame.document.write
 → ("You are now looking at
 → page "+thisPage+" in
 → frame "+frameNo+".")
 thisFrame.document.write
 → ("<\h1><\body>
 → <\html>")
 thisFrame.document.close()
 }

 function writePages(pageNo) {
 writeContent(1,pageNo)
 writeContent(2,pageNo)
 }

 // End hiding script from old
 → browsers -->
 </script>
</head>

```

```

<body bgcolor="#FFFFFF">
<h1>Navigation Bar</h1>
<h2>
 <a href="javascript:
 → writePages(1)">Page 1
 →

 <a href="javascript:
 → writePages(2)">Page 2
 →

 <a href="javascript:
 → writePages(3)">Page 3
</h2>
</body>
</html>

```

- ☞ This exercise lists only the navigation bar frame, as it's the only one that should have to change. Keep in mind, though, that there are many ways that students can do this exercise, so any approach that updates both content frames with one click should be considered acceptable.

## Class Discussion Questions

- Why would you want to keep a page out of a frame?
  - ☞ If your site doesn't have frames, and someone else is putting your site inside their site (making it appear as if your content is theirs), you'll need to break your page out of their frame.
- Why would you want to force a page into a frame?
  - ☞ If your site does have frames, you'll still want it to be search engine friendly. That means that people will come into a random page on the site, and the page will have to be reloaded inside its proper frameset, which puts the page into its proper context.
- When would you want to store information in a frame? Are there better ways to store information?
  - ☞ There are many ways to store information about a user's travels through a site, but if you don't have access to server-side utilities, your only client-side options are cookies and frames.
- Are there situations where you should not use frames?
  - ☞ Frames are not as popular as they once were, so you'll want to discuss how styles of sites go in and out of fashion.

## Review Questions

### Multiple choice

1. Which of the following is not an attribute of the frameset tag?
  - A. border
  - B. columns**
  - C. rows
  - D. frameborder
2. What is the minimum number of HTML pages you need for a framed site?
  - A. One
  - B. Two**
  - C. Three
  - D. Four
3. What is the usual number of HTML pages in a frameset?
  - A. One
  - B. Two
  - C. Three**
  - D. Four
4. To escape an HTML ending tag that you are writing with JavaScript:
  - A. Enclose the forward slash character in single quote marks.
  - B. Enclose the forward slash character in double quote marks.
  - C. Precede the forward slash character with a backslash.**
  - D. Do nothing; JavaScript will know what you want to do.

5. The URL object:
- A. Is supported in every JavaScript-capable browser.
  - B. Is supported only in pre-JavaScript 1.1 browsers.
- C. Is supported in JavaScript 1.1 and later.**
- D. Both A and C.
6. The topmost frameset page can be referred to (by itself and other pages) as:
- A. `top`
  - B. `parent`
  - C. `self`
- D. All of the above.**
- E. None of the above.

### Fill-in-the-blank

1. When someone hijacks a page from your web site, they are displaying it as a child frame to their parent window.
2. In the JavaScript hierarchy, the parent window is at the top.
3. When you have a frameset with a navigation bar and one or more content frames, it's more efficient to put shared functions into the page with the navigation bar.
4. To change the content of a frame, you only need a single link with a target attribute.
5. An iframe is an inline frame.

### Definitions

1. What is a frame?
  - ☞ A frame is a Web page within a frameset.
2. What is a frameset?
  - ☞ A frameset is a Web page that defines how subsections of a browser window will be split up into frames.
3. What is an iframe?
  - ☞ An iframe is an inline frame, which is to say, a frame that has the external appearance of a regular frame but (1) doesn't require a frameset page (it can just be defined within a normal HTML page) and (2) can be positioned anywhere inside a browser window.

# Chapter 7: Working with Browser Windows ♦ Study Guide

## Learning Objectives

- Open and close windows with JavaScript.
- Put varying contents into a window based on the link a user clicked.
- Use clicked images to open a new window.
- Open multiple windows at the same time.
- Open and scroll a window to a desired spot.
  - 📖 As noted in the text, this only works in certain browsers, so use it with care.
- Create a new window and then write information into it with a script.
- Use a window as a control panel to change the contents of another window.
- Position a window on the screen under script control.

## Get Up and Running Exercises

- Write four scripts similar to Script 7.1. Have each script create a window with one of the following four parameters showing: toolbar, status bar, scrollbars, and location.

```
<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Exercise 7.1</title>
 <script language="Javascript"
→ type="text/javascript">
 <!-- Hide script from old
→ browsers

 function newWindow() {
 catWindow =
window.open("images/pixel.jpg",
→ "catWin", "width=330,
→ height=250,toolbar=yes")
 }

 // End hiding script from old
→ browsers -->
</script>
</head>
<body bgcolor="#FFFFFF">
 <h1>The Master of the House
→ </h1>
 <h2>Click on His name to behold
→ He Who Must Be Adored
→

 <a href="javascript:
→ newWindow()">Pixel</h2>
</body>
</html>
```

*continues on next page*

```

<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Exercise 7.2</title>
 <script language="Javascript"
→ type="text/javascript">
 <!-- Hide script from old
→ browsers

 function newWindow() {
 catWindow = window.open
→ ("images/pixel.jpg",
→ "catWin", "width=330,
→ height=250,status=yes")
 }

 // End hiding script from old
→ browsers -->
 </script>
</head>
<body bgcolor="#FFFFFF">
 <h1>The Master of the House
→ </h1>
 <h2>Click on His name to behold
→ He Who Must Be Adored
→

 <a href="javascript:
→ newWindow()">Pixel</h2>
</body>
</html>
<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Exercise 7.3</title>
 <script language="Javascript"
→ type="text/javascript">

```

```

 <!-- Hide script from old
→ browsers

 function newWindow() {
 catWindow = window.open
→ ("images/pixel.jpg",
→ "catWin", "width=330,
→ height=250,scrollbars=yes")
 }

 // End hiding script from old
→ browsers -->
 </script>
</head>
<body bgcolor="#FFFFFF">
 <h1>The Master of the House
→ </h1>
 <h2>Click on His name to behold
→ He Who Must Be Adored
→

 <a href="javascript:
→ newWindow()">Pixel</h2>
</body>
</html>
<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Exercise 7.4</title>
 <script language="Javascript"
→ type="text/javascript">
 <!-- Hide script from old
→ browsers

 function newWindow() {
 catWindow = window.open
→ ("images/pixel.jpg",
→ "catWin", "width=330,
→ height=250,location=yes")

```

*continues on next page*

```

}

// End hiding script from old
→ browsers -->
</script>
</head>
<body bgcolor="#FFFFFF">
 <h1>The Master of the House
 → </h1>
 <h2>Click on His name to behold
 → He Who Must Be Adored
 →

 <a href="javascript:
 → newWindow()">Pixel</h2>
</body>
</html>

```

☞ Something the students should note by the time they've completed this exercise: you never need to set any element to no. Just leaving it off will have the same effect.

- Modify Script 7.3 to use your own images for a travel agency site, with thumbnails of destinations opening up larger pictures of that destination. Use five images instead of the three used in the script. Use different variable names than those in the book.

```

<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Exercise 7.5</title>
 <script language="Javascript"
 → type="text/javascript">
 <!-- Hide script from old
 → browsers

```

```
function newWindow(destjpg) {
```

```

 destWindow = window.open
 → (destjpg, "destWin",
 → "width=500,height=650")
 destWindow.focus()
}

// End hiding script from old
→ browsers -->
</script>
</head>
<body bgcolor="#FFFFFF">
<h1>Best Travel</h1>
<h3>Click on a location to see
→ the full-size image</h3>
<table cellpadding="10">
 <tr>
 <td>
 <a href="javascript:
 → newWindow('images/
 → scotland-big.jpg')">
 →

 Scotland
 </td>
 <td>
 <a href="javascript:
 → newWindow('images/
 → tahiti-big.jpg')">
 →

 Tahiti
 </td>
 </tr>
</table>

```

*continues on next page*

```

<a href="javascript:
→ newWindow('images/
→ cleveland-big.jpg')">
→

Cleveland
</td>
<td>
<a href="javascript:
→ newWindow('images/
→ alaska-big.jpg')">
→

Alaska
</td>
<td>
<a href="javascript:
→ newWindow('images/
→ hawaii-big.jpg')">
→

Hawaii
</td>
</tr>
</table>
</body>
</html>

```

- ☞ Make sure that when students change the variable names, they change them to be something meaningful, and that they don't, for instance, change book-Win to be named one thing on one line and something else on a different line.

- The second tip on Page 154 refers to using alert boxes as debugging aids. Create a page with a loop that increments a counter, and use an alert box to let you know the value of the loop variable every time it increments.

```

<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Exercise 7.6</title>
 <script language="Javascript"
→ type="text/javascript">
 <!-- Hide script from old
→ browsers

 for (i=0;i<10;i++) {
 alert(i)
 }

 // End hiding script from old
→ browsers -->
</script>
</head>
<body bgcolor="#FFFFFF">
 <h1>A drab web page</h1>
</body>
</html>

```

- ☞ Stress to the students that while this particular script doesn't do a whole lot, this approach can be incredibly helpful when debugging future problems. Encourage discussion of where this approach would have been helpful to solve problems that they've already encountered.

## Class Discussion Questions

- Discuss the advantages and disadvantages of opening windows for the user with JavaScript. What are some examples of sites that use such pop-up windows for good or for evil?
  - ☞ Good: Captive portals like T-Mobile, where you're required to have a child window open while you're connected using their wireless service. Bad: Advertising windows.
- When would you want to open multiple windows?
  - ☞ You might want to open multiple windows when browsing an artist's or photographer's portfolio site.
- What are `focus()` and `blur()` used for?
  - ☞ Bringing a primary or secondary window to the front or back. `focus()` brings the window to the front, and `blur()` sends it to the back.
- What are the benefits and disadvantages of using a separate window as a control panel, instead of putting the information on every page?
  - ☞ Overall, both control panels and frames are less popular now than they used to be. The advantages are primarily on the designer's side: only one page to update vs. an entire site if changes need to be made, for instance. On the user's side, they've frequently got their main window open large enough that control panels get hidden. Or worse, they've got pop-up blockers enabled so that your control panel never appears in the first place.
- Discuss how some but not all browsers allow users to change which elements are displayed, whether or not JavaScript set those elements to be shown when the window was first opened.
  - ☞ For instance, if you set a child window to open with its location hidden, some browsers allow the user to change the window to display that information. Consequently, you shouldn't expect to dependably use this approach to hide an address from a visitor.



## Review Questions

### Multiple choice

- Which method is used to bring a window to the front?
  - `front()`
  - `focus()`**
  - `blur()`
  - `windowfront()`
- When opening a window, how many spaces must exist after each comma in the parameters?
  - One
  - Two
  - Zero**
  - It doesn't matter
- When opening a new window, the `width` and `height` elements are:
  - Required
  - Optional**
  - Forbidden
  - Irrelevant
- Which of the following is a valid window element?
  - `scrollbar`
  - `statusbar`
  - `toolbar`**
  - All of the above
  - None of the above
- Which of these is how a child window should refer to its parent?
  - opener**
  - parent
  - top
  - None of the above
- Attempting to open a window when there's already a window open with that name will cause:
  - The browser to crash
  - It depends on the browser
  - Two windows to open with the same name
  - The window to redraw**
  - The window to come to the front

☞ You may consider giving partial credit for answer B. D is what browsers are supposed to do, but not all browsers always do exactly what they're supposed to.

### Fill-in-the-blank

- The window is the most important interface element in a Web browser.
- The opener property references the parent window.
- The opposite of the `focus()` method is `blur()`.
- You must declare the name of a window in JavaScript.
- Your script can take action when a window comes to the front by using the `onfocus` event handler.

6. To open a window or an alert dialog when the page is loaded, you use the `onload` event handler.

## Definitions

1. What are the elements of a window?
  - ☞ toolbar, status, menubar, location, scrollbars, resizable
2. Explain the difference between a parent window and a child window.
  - ☞ A child window is opened by its parent window.
3. Explain how you would position a new window at the top left of the screen.
  - ☞ The top-left corner of the screen is position 0,0—so set both `left` and `top` to zero in the parameter list.
4. What is a modal window?
  - ☞ A modal window is a window that forces the user to complete an action before it can be closed or blurred.

# Chapter 8: Form Handling ♦ Study Guide

## Learning Objectives

- Understand how to verify a password with JavaScript.
  - ☞ Checking that users entered the same password twice is one of the most common and useful applications of JavaScript. Be sure that students understand that this just verifies the string against itself—it doesn't actually check against or write to a database.
- Learn how to use an image as a Submit button.
  - ☞ By using images, rather than the browser's native interface widgets, the Web designer has much better control over the look of the site.
- Create select-and-go menus.
  - ☞ Using Go buttons for menus requires a needless step for the site's users. Select-and-go menus solve this problem.
- Understand how to change menus dynamically.
  - ☞ Dynamic menus are often used on sites to improve the user experience. For example, a site can have a menu with names of U.S. states, and when the user chooses a state, another menu is automatically populated with the names of the counties in that state.
- Learn how to validate data entered in forms.
  - ☞ Data validation is a vital application of JavaScript.

## Get Up and Running Exercises

- Modify Script 8.3. Add a Cancel button with a rollover to the page.

```
<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Exercise 8.1</title>
 <script language="Javascript"
→ type="text/javascript">
 <!-- Hide script from older
→ browsers

 if (document.images) {
 submitOn = new Image
 submitOff = new Image
 cancelOn = new Image
 cancelOff = new Image

 submitOn.src =
→ "images/submitOn.jpg"
 submitOff.src =
→ "images/submitOff.jpg"
 cancelOn.src =
→ "images/cancelOn.jpg"
 cancelOff.src =
→ "images/cancelOff.jpg"
 }

 function chgImg
→ (imgField,newImg) {
 if (document.images) {
 document[imgField].src=
→ eval(newImg + ".src")
 }
 }
```

*continues on next page*

```

}

function validForm(passForm) {
 if (passForm.passwd1.value
 → == "") {
 alert("You must enter a
 → password")
 passForm.passwd1.focus()
 return false
 }
 if (passForm.passwd1.value !=
 → passForm.passwd2.value) {
 alert("Entered passwords
 → did not match")
 passForm.passwd1.focus()
 passForm.passwd1.select()
 return false
 }
 return true
}

function subForm() {
 if (validForm(document.
 → myForm)) {
 document.myForm.submit()
 }
}

// End hiding script -->
</script>
</head>
<body bgcolor="#FFFFFF">
<form onsubmit="return
→ validForm(this)"
action="someAction.cgi"
→ name="myForm">
 Your name: <input type="text"
 → size="30" />
 <p>Choose a password: <input
 → type="password" name=
 → "passwd1" /></p>

```

```

<p>Verify password: <input
→ type="password" name=
→ "passwd2" /></p>
<p><a href="javascript:
→ document.myForm.reset()"
→ onmouseover="chgImg
→ (canButton, cancel0n)"
→ onmouseout="chgImg
→ (canButton, cancel0ff)">
→ <a href="javascript:
→ subForm()" onmouseover=
→ "chgImg(subButton, submit0n)"
→ onmouseout="chgImg(subButton,
→ submit0ff)"></p>
</form>
</body>
</html>

```

- ☞ The students may need some guidance to know what they need to link the Cancel button to, but they should be able to find the answer (the `reset()` method) in Appendix A.

- 2004 is a leap year. Change Script 8.5 to account for the extra day in February.

```

<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Exercise 8.2</title>
 <script language="Javascript"
→ type="text/javascript">
 <!-- Hide script from older
 → browsers

 monthDays = new Array(31,28,
→ 31,30,31,30,31,31,30,31,30,
→ 31)
 now = new Date
 calcYr = now.getFullYear()
 if (now.getMonth() > 1) {
 calcYr++
 }

 if (calcYr % 4 == 0 &&
→ (calcYr % 100 > 0 ||
→ calcYr % 400 == 0)) {
 monthDays[1] = 29
 }

 function populateDays
→ (monthChosen) {
 monthStr = monthChosen.
→ options[monthChosen.
→ selectedIndex].value
 if (monthStr != "") {
 theMonth=parseInt
→ (monthStr)

 document.myForm.days.
→ options.length = 0
 for(i=0;i<monthDays
→ [theMonth];i++) {

```

```

 document.myForm.days.
→ options[i] =
→ new Option(i+1)
 }
 }
 }
}

// End hiding script from
→ older browsers -->
</script>
</head>
<body bgcolor="#FFFFFF" onload="
→ document.myForm.months.
→ selectedIndex=0">
 <form action="#" name="myForm">
 <select name="months" onchange=
→ "populateDays(this)">
 <option value="">Month</option>
 <option value=
→ "0">January</option>
 <option value=
→ "1">February</option>
 <option value=
→ "2">March</option>
 <option value=
→ "3">April</option>
 <option value=
→ "4">May</option>
 <option value=
→ "5">June</option>
 <option value=
→ "6">July</option>
 <option value=
→ "7">August</option>
 <option value=
→ "8">September</option>
 <option value=
→ "9">October</option>
 <option value=
→ "10">November</option>

```

*continues on next page*

```

 <option value=
 → "11">December</option>
 </select>

```

```

<select name="days">
 <option>Day</option>
</select>
</form>

```

```

</body>
</html>

```

- ☞ There's any number of ways to do this; this one actually uses some coding concepts that aren't introduced until Chapter 10. The minimum the student should be required to do is reset `monthDays[1]` based on a calculation—they shouldn't just change the value within the array. In addition, they shouldn't have modified any code inside `populateDays()`. Give extra credit:
  - ▲ If they figure out how to use the Date methods (as shown above).
  - ▲ If they know the correct rule for deciding whether or not a year has a leap day (only if it's divisible by 4, and it's not divisible by 100, unless it's also divisible by 400—in other words, 2000 was a leap year, but 1900 wasn't and 2100 won't be).
  - ▲ If they figure out that the script should be checking for the upcoming February, not one that's already past. So, November 2003 should show the upcoming February with a leap day, but November 2004 should not.
  - ▲ If they can remember modulo (%) from chapter 1 and understand that it's the simplest way to do an every-four-years check.

- Modify Script 8.9 to also include a check that the Zip Code is exactly five numbers in length. We'll ignore the existence of 9-digit Zip Codes for now.

```

function validZip(inZip) {
 if (inZip == "") {
 return true
 }
 if (inZip.length ==
 → 5 && isNum(inZip)) {
 return true
 }
 return false
}

```

- ☞ Rather than include the entire script, we've only included the single function that needs to be changed. What's important here is to make sure that the `length` check is not added after the return from `isNum()`, as that wouldn't catch some invalid entries (such as "1234"). While we've combined the two checks here, it's acceptable for students to do it as two separate steps (so long as they're in the correct order).

## Class Discussion Questions

- Why would you want to use images as Submit buttons, rather than a browser's native button?
  - ☞ There are two reasons: to be able to use rollovers, and to be able to create web pages that look the same (or mostly the same) cross-browser and cross-platform.
- What are the benefits of using select-and-go menus? ANY drawbacks?
  - ☞ They're simple ways of managing what could otherwise take up a good deal of screen real estate. In addition, if they're coded as shown in the book, it's simple for someone with just a WYSIWYG editor to keep them updated versus having to hire a JavaScript expert if you're using complex DHTML menus.
- Why should you use JavaScript, rather than a CGI, to validate data in forms? When is JavaScript not the best choice for validation?
  - ☞ You should use JavaScript to validate user input because it is faster and more efficient to do the checks on the client side, rather than the server side. JavaScript is not a good choice when you need to, for example, check a user entry against a database of some kind. To check against a database, you need to use a CGI.
- Discuss how you can use parts of Script 8.11 in other scripts that you will write in the future. For example, you can reuse the part of the script that validates email addresses in other forms.
  - ☞ It's a good idea at this point to discuss reusability in general. Many scripters maintain libraries of code fragments that they've tested thoroughly that they combine and reuse on future projects.

## Review Questions

### Multiple choice

- Validating user input with JavaScript, rather than using a CGI, is:
  - Slower
  - Faster
  - More efficient
  - Both B and C**
- The term CGI stands for:
  - Computer Gateway Interface
  - Commonly Generated Instructions
  - Common Gateway Interface**
  - Computer Generated Instructions
- The comparison `x == ""` checks to see if:
  - The value of `x` is null.
  - The value of `x` is empty.**
  - The value of `x` is equal to zero.
  - The value of `x` is not equal to zero.
- The `email.indexOf("@", 1)` expression checks for the `@` sign, starting at which character?
  - The first.
  - The last.
  - The second.**
  - It does not check at all.
- A Select-and-Go menu should display a “Go” button:
  - Always
  - Never
  - Only when the user has JavaScript disabled**
  - Only when the user has a non-Web standards compliant browser.
- JavaScript can validate passwords against a database:
  - Always
  - Never**
  - Only if the database is written in JavaScript
  - Only for users with Internet Explorer for Windows.

### Fill-in-the-blank

- The JavaScript keyword `this` allows a script to pass a value to a function based on the context in which `this` is used.
- Checking a user’s entries in a form with JavaScript is called form validation.
- The onsubmit event handler is called when the user clicks on a submit button.
- A function called via the submit handler tells the browser that the form data is valid by returning a value of true.
- Dynamic menus populate the choices in one menu based on the choice the user makes in another.
- Checking email addresses with JavaScript validates them, but does not verify them.



## Find the Error

This code has a number of errors.  
Find them all.

```

<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Find the errors</title>
 <script language="Javascript"
→ type="text/javascript">
 <!-- Hide script from older
→ browsers

 function validEmail(email) {
 invalidChars = " /:;";

 for (i=0; i<invalidChars.length;
→ i++) {
 badChar = invalidChars.
→ charAt(0)
 if (email.indexOf(badChar,0)
→ > -1) {
 return false
 }
 }
 charPos = email.indexOf("@",1)
 if (charPos == -1) {
 return false
 }
 if (email.indexOf("@",
→ charPos +1) != -1) {
 return false
 }
 charPos = email.indexOf(".",
→ charPos)
 if (charPos == -1) {
 return false
 }
 if (charPos+3 > email.length) {
 return false

```

```

 }
 return true
 }

 function submitIt(theForm) {
 if (!validEmail(theForm.
→ emailAddr)) {
 alert("Invalid email
→ address")
 theForm.emailAddr.focus()
 theForm.emailAddr.select()
 return false
 }
 }
}

```

```

 // End hiding script -->
 </script>
</head>
<body bgcolor="#FFFFFF">
<form onsubmit="submitIt()"
→ action="someAction.cgi">
 Email Address: <input
→ name="emailAddr" type="text"
→ size="30" />

 <input type="submit" value="Submit"
→ /> <input type="reset" />
</form>
</body>
</html>

```

1. The line `if (!validEmail(theForm.emailAddr)) {` should be `if (!validEmail(theForm.emailAddr.value)) {`
2. The line `<form onsubmit="submitIt()" action="someAction.cgi">` should be `<form onsubmit="return submitIt()" action="someAction.cgi">`.
3. There's no `return true` at the end of the `submitIt()` function, and it's required in order for the validation to work properly.

*continues on next page*

4. The line `badChar = invalidChars.charAt(0)` should be `badChar = invalidChars.charAt(i)`.
5. One extra line here in case the student decides to fix the errors in some other fashion. There are several places, though, where the above code differs from that in the book, so be careful they don't just do a strict comparison.

# Chapter 9: Forms and Regular Expressions ♦ Study Guide

---

## Learning Objectives

- Understand what a regular expression is.
  - ☞ As mentioned in the sidebar on page 198, many people happily write JavaScript for years without ever going near regular expressions. Nothing in the later chapters will refer back to this one, so if there's one chapter in this book that you need to skip, it may as well be this one. On the other hand, REs are a powerful way to express complex concepts in a minimum of lines of code, so learning about them will add a powerful tool to a student's toolbox.
- Learn how to validate an email address using regular expressions.
  - ☞ O'Reilly and Associates has a book that covers nothing but regular expressions, and that book has serious coverage of using REs to validate email addresses. And in the end, their conclusion is that there's no way for REs to validate every possible valid email address, so don't worry if there's some odd, rarely used combination (did you know that it's valid to have comments in email addresses?) that your student's code doesn't properly catch. And even given that, there's no way to use JavaScript to verify that it's a working email address.
- Validate file names using regular expressions.
  - ☞ Regular expressions can be used to validate any kind of data; file names are just a single example.
- Use regular expressions to extract, format, and sort strings.
  - ☞ String manipulation is one of the main uses of regular expressions.

## Get Up and Running Exercises

- Extend Script 9.2 to also handle FTP URLs, and PNG graphic files (.png).

```
re = /^(file|http|ftp):\\\/\\
→ S+\\\/S+\\. (gif|jpg|png)$/i
```

☞ This is the only line of code that students should have to change. Deduct points or point out to them if they change any other lines, or if they remove any existing functionality (allowing file and http URLs, and .gif and .jpg files).

- Modify Script 9.3 to handle middle names, as suggested in the first Tip on page 205.

```
re = /((\S+\s+)(\S+))/
lastName = new Array
newNameField = ""

for (i=0;i<nameList.length;
→ i++) {
 lastName[i] = nameList[i].
 → replace(re, "$3, $1")
}
```

☞ With regular expressions, even more than most JavaScript code, there are a multitude of ways to write any given exercise. The goal here is to see if the student is using their new knowledge of REs to make the change. In this variant, `re` has been changed to grab everything before the last string and store it in `$1`. `$2` will now contain whatever happens to be in the second to last string, and `$3` will contain the last string. Consequently, when `lastName[i]` is being initialized, all that matters is `$1` and `$3`.

- Modify Script 9.6 to format and validate Social Security numbers, in the form ###-##-####.

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

```
<head>
<title>Exercise 9.3</title>
<script language="Javascript"
→ type="text/javascript">
<!-- Hide script from older
→ browsers
re = /\d{3}\[\-]?(\d{2})
→ \[\-]?(\d{4})$/
```

```
function submitIt(myForm) {
 validSSN = re.exec
 → (myForm.SSN.value)
 if (validSSN) {
 myForm.SSN.value =
 → validSSN[1] + "-" +
 → validSSN[2] + "-" +
 → validSSN[3]
 }
 else {
 alert(myForm.SSN.value +
 → " isn't a valid SSN
 → number")
 myForm.SSN.focus()
 myForm.SSN.select()
 }
 return false
}
```

```
// End hiding script -->
</script>
</head>
<body bgcolor="#FFFFFF">
```

*continues on next page*

```

<h2 align="center">Validate a SSN
→ number</h2>
<form onsubmit="return
→ submitIt(this)"
action="someAction.cgi">
 <table border="0"
 → cellspacing="8"
 → cellpadding="8">
 <tr>
 <td align="right"
 → valign="top">
 Enter your SSN number:
 </td>
 <td>
 <input name="SSN"
 → type="text" size="20" />
 <p><input type="reset"
 → /> <input type=
 → "submit" value=
 → "Submit" /></p>
 </td>
 </tr>
 </table>
</form>
</body>
</html>

```


- ☞ Students should understand that the only line of code that needs real changes is the initialization of `re`. Be careful that they've removed the code that allows parentheses, spaces, and periods, and that it now looks for the correct 3 digits-2 digits-4 digits format. If the user enters anything other than 9 digits, it shouldn't be allowed, and neither should any dashes anywhere other than the two approved locations.

## Class Discussion Questions

- What are the benefits and disadvantages of using regular expressions in your scripts?
  - ☞ The benefit is being able to do a maximum of processing in a minimal amount of code. The drawbacks are that it can be difficult for one coder to understand another coder's writing style, and that some programmers don't like to use REs because they're difficult to understand and remember.
- Can you do things using regular expressions that would be too difficult or impossible using other methods? If so, what are some examples?
  - ☞ Anything you can do with REs you can do without them—it's just that it could (for example) take hundreds of lines of RE-less code to do what a student could do with one RE line.
- Use of the `RegExp` object is essential in several of this chapter's examples. Discuss the characteristics and uses of the object, and its properties and methods.
  - ☞ Along with this, touch on the use of the `RegExp.exec()` method to use the object as a string. It's also worth pointing out that `RegExp` has the properties `RegExp.$1` through `RegExp.$9`, and because REs are one-based, not zero-based as with many JavaScript numbers, there's no `RegExp.$0`.

## Review Questions

### Multiple choice

- The variable `re` is required when you use regular expressions:
  - Always.
  - Never.**
  - Only when used as part of a comparison.
  - Whenever you need to reuse a regular expression variable.  
 Some variable is required, but any other valid JavaScript variable can be used where the book uses the `re` variable.
- You can use the result of a regular expression as:
  - A variable.
  - An object.
  - A property of the `RegExp` object.
  - All of the above.**
  - None of the above.
- Special characters in regular expressions are case sensitive.
  - True.**
  - False.
  - Only if the rest of the script is lowercase.
  - Only in a header script.
- JavaScript's `RegExp` object:
  - Contains the results of a regular expression method.
  - Contains the pattern described by a regular expression.**
  - All of the above.
  - None of the above.
- Which of these is not a property of the `RegExp` object?
  - `$3`
  - `lastMatch`
  - `exec(re)`**
  - `$&`
- If you wanted to search for a return character, you would use which special character?
  - `/r`
  - `\r`**
  - `\R`
  - `^c`
- If you wanted to search for zero or one instance of the letter B, you would use which construction?
  - `[B]?`**
  - `(b)?`
  - `[^b]`
  - `[b?]`

**Fill-in-the-blank**

1. Typing a backslash `\` before a special character escapes that character.
  2. To denote the beginning of a string, you use the caret character.
  3. To denote the end of a string, you use the dollar sign character.
  4. In a regular expression, a period means any character except newline.
  5. The `exec()` method is a method of the RegExp object.
  6. You read regular expressions from left to right.
  7. In the regular expression `/^\s)(\S+)\s(\S)(\S+)$/` the `(\S+)` means any non-white space character.
  8. To search for either of the strings “chocolate” or “candy” you would write it as `(chocolate|candy)`.
4. What’s the difference between `\D` and `[0-9]`?
    - ☞ They’re opposites—the former is any non-digit character, and the latter is any digit.
  5. What’s the difference between `a*` and `a{0,}`? Give a string this check won’t match.
    - ☞ They’re equivalent, in that they both search for zero or more instances of the character “a” in a string. All strings have zero or more a’s, so any string will match this criterion.

**Definitions**

1. What is the difference between `s` and `S` when using regular expressions?
  - ☞ The lower-case `s` is any single white space character, and the upper-case `S` is any single character that isn’t a white space character.
2. What is a white space character?
  - ☞ It’s a form feed, carriage return, newline, tab, or vertical tab character.
3. What characters are being searched for in `[aeiou]`?
  - ☞ Any single vowel will match.

# Chapter 10: Making Your Pages Dynamic ♦ Study Guide

## Learning Objectives

- Understand how to write and format dates on Web pages.
  - ☞ Some of the zero-based vs. one-based methods may be a little confusing, so make sure that students know to refer to the chart on pages 233-234.
- Learn to work with dates and adjust them in relation to time zones.
  - ☞ Also note their limitations: anything regarding time and dates can only work correctly if the end user has the correct date, time, and time zone set on their computer.
- Understand how to create date and time-based countdowns.
  - ☞ The exercises below include two examples of this; one on a static page and one on a dynamic page.
- Understand how to use JavaScript to write text into Web pages in response to user actions.
  - ☞ At this point in the book, writing text that changes while the page is up requires the new information to be written into form fields.

## Get Up and Running Exercises

- Modify Script 10.1 to include the current year in the Web page the script writes.

```
document.write("<h1>Today is
→ " + dayName[now.getDay()] +
→ ", " + monName[now.getMonth()]
→ + " " + now.getDate() + ", " +
→ now.getFullYear() + ".</h1>")
```

- ☞ Make sure that students use `getFullYear()` and understand why it's important that they do so. You can best get this across by showing them what the results of `getYear()` look like in a number of browsers—during 2004 (for example), some will show it as 2004, while others will display it as 104. Using `getFullYear()` works around this issue. Additionally, make sure that they add the comma and the space before the year so that it's displayed appropriately.
- Write a page to produce a countdown that includes days, hours, minutes, and seconds before an event in the form dd:hh:mm:ss. You can do this by rewriting Script 10.6. This countdown is static, meaning that it will not update continually.

```
<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Exercise 10.2</title>
 <script language="Javascript"
 → type="text/javascript">
```

*continues on next page*





- Combine Scripts 10.5 and Exercise 10.2 (above) to create a continually updating countdown, using a form field that can be updated once a second instead of a static page.

```

<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Exercise 10.3</title>
 <script language="Javascript"
→ type="text/javascript">
 <!-- Hide script from old
→ browsers

 now = new Date
 xmasEve = new Date
 → (now.getFullYear(),11,24)
 if (xmasEve.getTime() <
→ now.getTime()) {
 xmasEve.setYear
 → (now.getFullYear()+1)
 }

 function showTheTime() {
 document.theForm.showTime.
 → value = timeTill(xmasEve)
 setTimeout("showTheTime()",
 → 1000)
 }

 function timeTill(inDate) {
 now = new Date
 daysTill = Math.ceil
 → (dayToDays(inDate) -
 → dayToDays(now))
 hoursTill = 23 -
 → now.getHours()

```

```

 minTill = 59 -
 → now.getMinutes()
 secTill = 59 -
 → now.getSeconds()
 return (daysTill +
 → showZeroFilled
 → (hoursTill) +
 → showZeroFilled(minTill) +
 → showZeroFilled(secTill))
 }

 function dayToDays(inTime) {
 return (inTime.getTime() /
 → (1000 * 60 * 60 * 24))
 }

 function showZeroFilled
 → (inValue) {
 if (inValue > 9) {
 return ":" + inValue
 }
 return ":0" + inValue
 }

 // End hiding script from old
 → browsers -->
</script>
</head>
<body bgcolor="#FFFFFF"
→ onload="showTheTime()">
<form name="theForm" action="#">
 <h1>It's only
 <input type="text"
 → name="showTime" size="10" />
 until Christmas so you'd better
 → start your shopping now!</h1>
</form>
</body>
</html>

```

*continues on next page*

☞ The only place where students are likely to run into trouble in modifying their Exercise 10.2 script is having to update `now` to be the current time every iteration of `timeTill()`. If the student puts that code into `showTheTime()` instead, that's fine too, but if it's left off entirely the script won't update.

- Create a mock e-commerce page that uses the `document.referrer` object to display a special welcoming page for visitors coming from a domain that you choose, as suggested in the Tip on page 229. You can use Script 10.7 as a beginning, but you will need to add to it.

```
<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Exercise 10.4</title>
<script language="Javascript"
→ type="text/javascript">
 <!-- Hide script from old
 → browsers

switch(document.referrer) {
 case "http://www.amazon.com":
 referCode = "amazon"
 break
 case "http://www.
→ peachpit.com":
 referCode = "peachpit"
 break
 case "http://www.
→ javascriptworld.com":
 referCode =
 → "javascriptworld"
 break
 default:
 referCode = ""
```

```
}

// End hiding script from old
→ browsers -->
</script>

</head>
<body bgcolor="#FFFFFF">
<h1>Welcome to Bob's Discount
→ House of Widgets!
<script language="Javascript"
→ type="text/javascript">
 <!-- Hide script from old
 → browsers

 if (referCode != "") {
 document.write("
Thanks
 → for visiting! Here's your
 → special discount code: " +
 → referCode + ". Just enter
 → it at checkout for your
 → fabulous savings.")
 }

 // End hiding script from old
 → browsers -->
</script>

Here are today's specials:
→ </h1>
</body>
</html>
```

☞ The way to approach this exercise is to use the `switch/case` statement (also called multi-level conditionals) introduced in Chapter 3. While it's not a requirement for students to use it here, it's a good idea for them to get used to using it. What they do need to be sure to do is have different results occurring based on what site the user arrived from, including handling what will happen if the user arrives from somewhere else entirely.

- Rewrite Script 10.8. Instead of the calendar, use a list of at least ten text links of some items (things, people, bands, you make the choice). In the text box, add detailed descriptions of each item. Combine this with concepts you learned in Chapter 4 to make text links rollovers. If you click the links, rather than roll over them, the description should “stick” in the text box.

```

<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Exercise 10.5</title>
 <script language="Javascript"
→ type="text/javascript">
 <!-- Hide script from old
→ browsers

 clicked = false
 infoList = new Array
 infoList[0] = ""
 infoList[1] = "JavaScript
→ for the World Wide Web:
→ Visual QuickStart Guide,
→ 5th Edition, by Tom Negrino
→ and Dori Smith, is an
→ introductory text for
→ learning JavaScript the
→ fast and easy way."
 infoList[2] = "Java 2 for the
→ World Wide Web: Visual
→ QuickStart Guide, by Dori
→ Smith"
 infoList[3] = "Keynote for
→ Mac OS X: Visual QuickStart
→ Guide, by Tom Negrino"

```

```

 infoList[4] = "Macromedia
→ Contribute 2 for Windows
→ and Macintosh: Visual
→ QuickStart Guide, by Tom
→ Negrino"
 infoList[5] = "Quicken 2003
→ for Macintosh: Visual
→ QuickStart Guide, by Tom
→ Negrino"

```

```

function showInfo(thisItem) {
 document.infoForm.infoBox.
 → value = infoList
 → [thisItem]
}

```

```

// End hiding script from old
→ browsers -->
</script>

```

```

</head>
<body bgcolor="#FFFFFF">
<p><a href="javascript:showIn
→ fo(1);clicked=true"
→ onmouseover="showInfo(1);
→ clicked=false" onmouseout="if
→ (!clicked)showInfo(0)">
→ JavaScript: Visual QuickStart
→ Guide</p>
<p><a href="javascript:showIn
→ fo(2);clicked=true"
→ onmouseover="showInfo(2);
→ clicked=false" onmouseout="if
→ (!clicked)showInfo(0)">Java 2:
→ Visual QuickStart Guide</p>
<p><a href="javascript:showIn
→ fo(3);clicked=true"
→ onmouseover="showInfo(3);
→ clicked=false" onmouseout="if
→ (!clicked)showInfo(0)">Keynote:
→ Visual QuickStart Guide</p>

```

*continues on next page*

```

<p><a href="javascript:showIn
→ fo(4);clicked=true"
→ onmouseover="showInfo(4);
→ clicked=false" onmouseout="if
→ (!clicked)showInfo(0)">
→ Contribute 2: Visual QuickStart
→ Guide</p>
<p><a href="javascript:showIn
→ fo(5);clicked=true"
→ onmouseover="showInfo(5);
→ clicked=false" onmouseout="if
→ (!clicked)showInfo(0)">Quicken:
→ Visual QuickStart Guide</p>
<form name="infoForm" action="#">
 <textarea rows="7" cols="30"
 → name="infoBox" readonly=
 → "readonly">Click or rollover
 → any link for more
 → information</textarea>
</form>
</body>
</html>

```

- ☞ This example only has five links for brevity's sake, but the student's should (as mentioned above) have at least ten. The information should display in the text box both when the link is clicked on and when it's rolled over, but the information should be cleared `onmouseout` only if the user has not clicked on the link. That requires that some sort of flag be set; give students less credit if their code doesn't reflect this.

- Sites abound that display personalized pages that are written dynamically for each user. Search the Web to find examples of at least three sites that do this, and be prepared to talk about them in class.
  - ☞ Some examples include Amazon.com, most travel sites, such as Travelocity and Expedia, and features of sites such as the My Yahoo! Pages.

## Class Discussion Questions

- Discuss the reasons why it's useful to use JavaScript to make dynamic events occur in the user's browser.
  - ☞ If a site looks the same every time a user visits it, why should a visitor ever come back? JavaScript is the simplest way to add dynamic elements to a previously static Web page.
- Dynamic elements can be provided with JavaScript, but some users have JavaScript turned off, for a variety of reasons. Discuss how you should deal with users who have JavaScript turned off in their browsers.
  - ☞ Stress to students that they should always test their pages without JavaScript and verify that, while the visitors may not get all the bells and whistles, they should always be able to navigate through the site and access all the content.
- As mentioned in the sidebars on pages 217 and 228, JavaScript handles time in sometimes-unexpected ways. Discuss how numbering of hours, days, dates, and months are done in JavaScript.
  - ☞ The easiest way to keep track of what works how and where is to keep a cheat sheet handy, which is why there's one provided on pages 233-234. Students should learn to keep this handy when working with date and time values in JavaScript.
- You can use JavaScript to timestamp a page that the user sees. What are the pros and cons of using JavaScript for this, rather than a CGI? Are there situations where you would not want to use JavaScript?
  - ☞ The positive side of this is that the Web page developer doesn't need to have access to server-side functionality. The downside, unfortunately, is that this means that the time is dependent on the user setting the time and location on their PC correctly. As this is undependable, students won't want to use it in a financial transaction where the correct date and time are legally important.

## Review Questions

### Multiple choice

- The JavaScript method `getDay()`:
  - Gets the integer value of the day of the week.**
  - Gets the date, including the year.
  - Gets the day's Julian date, starting from 0. For example, January 30 would have the value 29.
  - Gets today's numeric date, e.g., 08/07/04.
- JavaScript dates and times are:
  - Synchronized automatically to UTC (Coordinated Universal Time).
  - Identical to GMT (Greenwich Mean Time).
  - Dependent on correct time and time zone settings of the user's computer.**
  - All of the above.
- Military time is:
  - 12-hour format.
  - 24-hour format.**
  - Correct, sir!
  - Unnecessary; give peace a chance.
- The `getTime()` method returns the number of milliseconds since:
  - January 1, 1968
  - January 1, 1970**
  - January 1, 2000
  - January 1, 1904
  - January 1, 1900
- JavaScript's Date methods are methods of the:
  - document object.
  - date property.
  - date object.**
  - `document.referrer` object.

### Fill-in-the-blank

- Pages with scripts that write the page's contents on the fly are called dynamic pages.
- The page the user clicked through to visit the current page is called the referrer page.
- To JavaScript, the numeric value of the month November is 10.
- The expression `new Date(thisYr,9,27)` refers to what date? October 27th of the current year.

For the next two questions, refer to the following code fragment:

```
now = new Date
dayVal = (now.getTime() / (1000 *
→ 60 * 60 * 24))
```

- What is the value of `now.getTime()`? The number of milliseconds since January 1, 1970.
- What is the value of `dayVal`? The number of days since January 1, 1970.

## Find the Errors

The following code contains a number of errors. Find them all.

```
<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Exercise 10.5</title>
</head>
<body bgcolor="#FFFFFF">
<h1>Today is:
 <script language="Javascript"
 → type="text/javascript">
 <!-- Hide script from old browsers

 now = new Date
 document.write(now.getMonth() +
 → "/" + now.getDay() + "/" +
 → now.getYear() + " " +
 → (now.getHours() % 12) + ":" +
 → now.setMinutes() + ":" +
 → now.getSeconds())

 // End hiding script from old
 → browsers -->
 </script>
</h1>
</body>
</html>
```

1. The value of `now.getMonth()` goes from zero to 11, and so, needs to be incremented in order for it to actually display the correct month.
2. The value of `now.getDay()` should not be used here; it's actually the day of the week, not the day of the month. The correct object would have been `now.getDate()`.
3. The value of `now.getYear()` varies from browser to browser post 2000. For dependable results, use `now.getFullYear()` instead.
4. For the minutes, the above uses `now.setMinutes()`, but `now.getMinutes()` should have been used instead.
5. One extra line here in case students come up with alternate ideas for how to fix the above code.



# Chapter 11: Handling Events ♦ Study Guide

---

## Learning Objectives

- Understand event handling.
  - ☞ Make sure that students understand the basic concepts behind how JavaScript handles events, as this is a key to most of the JavaScript they'll ever write.
- Understand JavaScript's window event handlers.
  - ☞ Make sure that the students are familiar with all of the different window event handlers and how they are used.
- Learn how to use JavaScript to handle the user's mouse events.
- Use form event handling to both help validate forms and manage the user's interaction with forms.
- Use key event handling to capture user key events.

## Get Up and Running Exercises

- Read the discussion of the `onunload` event handler on page 237. Write a description of how you could use this handler for good, rather than evil, i.e., figure out a useful way to implement this handler that won't annoy the user.
  - ☞ Virtually all uses of the `onunload` event handler are nasty and unpleasant. An example of one that's only minimally intrusive would be a school site, where some links go to school-run sites and some don't, and an `onunload` handler is triggered when the latter are clicked, warning that user that they're going to content that isn't controlled by the school.
- Read the first Tip on page 241, then write and open two windows that each contains Script 11.4 (or a similar script that you write). Describe the result, including which browser version you used, and explain how you solved the problem (if any). Then repeat the test using at least one different browser and report on the result. Draw comparisons to how different browsers handle coding errors such as these.
  - ☞ Depending on the browsers, chances are good that they'll crash, and hard. There are two goals for this exercise: (1) learn how different browsers can give widely varying results for the same code, and (2) learn why they should be very careful using `blur` and `focus` commands, so that they don't cause problems for users.

*continues on next page*

- Modify Script 11.10 so that it displays the value of a pressed key, as described in the Tip on page 254. Run the script in as many browsers as possible, and report back what the values are for up arrow, down arrow, left arrow, and right arrow.

```

<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Exercise 11.3</title>
 <script language="Javascript"
→ type="text/javascript">
 <!-- Hide script from old
→ browsers

 document.onkeydown = keyHit
 if (document.layers) {
 document.captureEvents
→ (Event.KEYDOWN)
 }

 function keyHit(evt) {
 if (evt) {
 thisKey = evt.which
 }
 else {
 thisKey = window.event.
→ keyCode
 }
 alert(thisKey)
 }
 // End hiding script from old
 → browsers -->
</script>
</head>
<body bgcolor="#FFFFFF">
</body>
</html>

```

- ☞ As just a few examples, Internet Explorer and more recent versions of Netscape 6 showed left arrow as 37, up arrow as 38, right arrow as 39, and down arrow as 40. Netscape 4 showed left arrow as 28, right arrow as 29, and up arrow as 30. Safari for Mac OS X doesn't understand the onkeydown handler at all as of this writing.

- Based on Exercise 11.3 (above), enhance Script 11.10 so that pressing the up arrow starts the slideshow from the beginning. Use your own graphics (at least 6).

```

<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Exercise 11.4</title>
 <script language="Javascript"
→ type="text/javascript">
 <!-- Hide script from old
→ browsers

 myPix = new Array("images/
→ callisto.jpg", "images/
→ europa.jpg", "images/
→ io.jpg", "images/
→ ganymede.jpg")
 thisPic = 0
 imgCt = myPix.length - 1

 document.onkeydown = keyHit
 if (document.layers) {
 document.captureEvents
→ (Event.KEYDOWN)
 ltArrow = 28
 rtArrow = 29
 upArrow = 30

```

*continues on next page*

```

}
else {
 ltArrow = 37
 rtArrow = 39
 upArrow = 38
}

function keyHit(evt) {
 if (evt) {
 thisKey = evt.which
 }
 else {
 thisKey = window.event.
 → keyCode
 }

 switch (thisKey) {
 case (ltArrow):
 chgSlide(-1)
 break
 case (rtArrow):
 chgSlide(1)
 break
 case (upArrow):
 chgSlide(-thisPic)
 break
 default:
 }
}

function chgSlide(direction) {
 if (document.images) {
 thisPic = thisPic +
 → direction
 if (thisPic > imgCt) {
 thisPic = 0
 }
 if (thisPic < 0) {
 thisPic = imgCt
 }
 document.myPicture.src=myPi
x[thisPic]
 }
}

```

```

}
// End hiding script from old
→ browsers -->

</script>
</head>
<body bgcolor="#FFFFFF">
<h3 align="center">

 Use the right and left arrows
 → on your keyboard to view the
 → slideshow, or up arrow to
 → start at the beginning.</h3>
</body>
</html>

```

👉 Students should be able to figure out how to reset direction with a minimal amount of code, as in the example above. What's most important is that the script handles the largest possible number of browsers.

- Use the object table in Appendix A to review the event handlers associated with the JavaScript objects mentioned in this chapter.

## Class Discussion Questions

- Discuss why event handling is important.
  - ☞ Much of the entire point of JavaScript is making pages that are responsive to user actions. Handling events is how that's done; consequently, event handling is at the core of most real world usage of JavaScript.
- Discuss the different window event handlers listed in the book. Talk about the different uses for each handler in pages you would create.
  - ☞ It's absolutely the case that some of the handlers are more common/more useful than others, so don't expect students to give an equal weight to each.
- One of the values of the mouse event handlers is to make Web pages act more like operating systems such as Windows or Mac OS X. What are some specific examples of how you can use mouse event handlers to provide users with a more familiar user experience?
  - ☞ One possible example for you to discuss is how `onclick` is an extremely common way of working with applications and documents, but it is rarely if ever used on the Web.
- How are the `onfocus` and `onselect` event handlers similar and different? Where would you want to use each one?
  - ☞ The `onfocus` handler is useful for many things other than just the ability to keep the user out of a field (the example given in the book). It's triggered any time that user clicks or tabs into a field. The `onselect` handler, on the other hand, is triggered when the user selects the contents of a text box or text area. So, if the user both clicks into and selects the contents of a text area, both the `onfocus` and the `onselect` event handlers will be triggered.
- Does key event handling differ depending on the platform of the user? Should you add special cases to your code if the user is browsing from, say, a Mac or a Windows machine?
  - ☞ Platform matters (as mentioned above, for instance, Safari doesn't handle `onkey` events), but not as much as browser and version.

## Review Questions

### Multiple choice

- Clicking twice on an object will trigger which event handler:
  - onmousedown
  - onmouseup
  - onclick
  - ondblclick
  - All of the above.**
- If you want to capture an event when the user hits a particular key, you would use which event handler?
  - onclick
  - onpress
  - onkeydown**
  - onkeypress
  - All of the above.
- Using the onunload event handler:
  - Is tasteless.
  - Annoys the heck out of users.
  - Makes your site look like a porn site.
  - All of the above.**
- Hiding your JavaScript code from visitors:
  - Is easy.
  - Is totally secure.
  - Is rarely possible.**
  - Is the default in most browsers.
- The onclick handler triggers when the user:
  - Clicks and releases the mouse button.**
  - Clicks and holds the mouse button.
  - Moves the mouse.
  - All of the above.
- The onabort handler:
  - Triggers when the user cancels an image loading on the Web page.
  - Is unsupported in some browsers.
  - Is rarely used.
  - All of the above.**

### Fill-in-the-blank

- Actions the user performs while visiting your Web page are called events.
- The onresize event triggers when the user changes the size of the Web page.
- You would use the onunload event to take action when a window is closed.
- To take more than one action when a window opens, the onload handler should call a function that contains all the actions.
- To make windows always come to the front, you can use the onfocus event handler.
- The onmouseout event triggers when the user moves the mouse out of a defined area.
- To refill a form's default values when the user clicks the Cancel button, you would use the onreset event handler.
- You would use the onkeydown event to take action when a key is hit.

## Definitions

1. What is an event handler?
    - ☞ An event handler is the way that JavaScript handles user-controlled actions on a Web page.
  2. What is a window event? Give an example of a window event.
    - ☞ Window events are those JavaScript events that involve the window: `onload`, `onunload`, `onresize`, `onmove`, `onabort`, `onerror`, `onfocus`, and `onblur`.
  3. What is a mouse event? Give an example of a mouse event.
    - ☞ Mouse events are those JavaScript events that involve mouse movement: `onmousedown`, `onmouseup`, `onmousemove`, `onmouseover`, `onmouseout`, `ondblclick`, and `onclick`.
  4. What is a key event? Give an example of a key event.
    - ☞ Key events are those JavaScript events that involve keyboard actions: `onkeydown`, `onkeyup`, and `onkeypress`.
  5. What is a form event? Give an example of a form event.
    - ☞ Form events are those JavaScript events that involve interactions with Web page forms: `onsubmit`, `onreset`, `onchange`, `onselect`, `onclick`, `onblur`, and `onfocus`.
-

# Chapter 12: JavaScript and Cookies ♦ Study Guide

## Learning Objectives

- Create a cookie and write it to the user's machine.
- Read and display cookie contents.
  - ☞ Just writing cookies doesn't do much if you can't read them back in again later.
- Use cookies as counters.
  - ☞ Make sure that students understand the limitations of this. By no means can this be used as a count of how many people have visited a site, just how many times this particular user at this particular machine using this particular browser has visited.
- Learn how to delete cookies.
  - ☞ Deletion is the same as addition, but with a date in the past.
- Learn how to read and write more than a single cookie.
  - ☞ Multiple cookies are slightly more complex, but significantly more powerful. For example, if you have a site that offers user registration, you might want to set one cookie with a username, another one with a password, and others that keep track of which pages on the site the user has visited.

## Get Up and Running Exercises

- Find the cookie file on your computer, and open it up in a text editor. Try to understand the different kinds of information that is being stored, and give specific examples of how three different sites are using cookies.
  - ☞ Based on the browser, version, and platform, the file can be named and stored in any one of a number of places—there's no standard. If students have trouble finding it, one method for detecting the file is searching the hard drive for the word "cookie."
- Combine Scripts 12.3 and 12.5 to display the name and value of each cookie before it's deleted.

```
<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Exercise 12.2</title>
</head>
<body bgcolor="#FFFFFF">
 <script language="Javascript"
 → type="text/javascript">
 <!-- Hide script from older
 → browsers
```

```
if (document.cookie != "") {
 if (confirm("Do you want to
 → delete the cookies?")) {
 thisCookie = document.
 → cookie.split("; ")
 expireDate = new Date
```

*continues on next page*

```

expireDate.setDate
→ (expireDate.getDate()-1)

for (i=0; i<thisCookie.
→ length; i++) {
cookieName = thisCookie
→ [i].split("=")[0]
document.write("Cookie
→ name was '"+cookieName)
document.write("'", and
→ the value was '"+
→ thisCookie[i].split
→ ("=")[1]+' '
")
document.cookie =
→ cookieName +
→ " =; expires=" +
→ expireDate.toGMTString()
}
document.write("Number
→ of cookies deleted: " +
→ thisCookie.length)
}
}
else {
document.write("No cookies
→ were found")
}
// End hiding script -->
</script>
</body>
</html>

```

- Make sure that the students are doing the display and the deletion within the same loop. If they're looping more than once (separately for the display and the deletion, for instance), they're making their code more complex than it needs to be.

- Combine Scripts 12.6 and 12.7 to read and write the date to a cookie, then put up an alert message if it has been more than a month since the user last visited the site.

```

<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
<title>Exercise 12.3</title>
<script language="Javascript"
→ type="text/javascript">
<!-- Hide script from older
→ browsers

```

```

now = new Date
expireDate = new Date
expireDate.setMonth(expire
→ Date.getMonth()+6)
lastVisit = new Date(cookie
→ Val("pageVisit"))
tooOld = new Date
tooOld.setMonth(tooOld.
→ getMonth()-1)
document.cookie =
→ "pageVisit="+now+
→ "; expires=" +
→ expireDate.toGMTString()

```

```

if (tooOld.getTime() >
→ lastVisit.getTime()) {
alert("It's been more
→ than a month since you
→ were last here")

```

*continues on next page*



```

}

function cookieVal
→ (cookieName) {
 thisCookie = document.
 → cookie.split("; ")
 for (i=0; i<thisCookie.
 → length; i++) {
 if (cookieName ==
 → thisCookie[i].split
 → ("=")[0]) {
 return thisCookie[i].
 → split("=")[1]
 }
 }
}
return "1 January 1970"
}

// End hiding script -->
</script>
</head>
<body bgcolor="#FFFFFF">
 <h2>Welcome to my Web site</h2>
</body>
</html>

```

## Class Discussion Questions

- What are the security implications of cookies? How much concern should the user have about sites that set cookies?
  - ☞ While there are occasional security warnings about cookies, most of the bugs concerning them were tracked down and fixed years ago. At this point, cookies should simply be considered a way of saving a user's preferences on the user's own machine.
- Why can't you use JavaScript to create page counters for a Web site?
  - ☞ Students need to remember that JavaScript can neither read from nor write to files on the server. So JavaScript can't write information about user visits to a counter on the server. Page counters are a job for a CGI.
- Why might a single site need to write multiple cookies to your machine?
  - ☞ Different parts of a site might need to keep track of different information. For instance, one area might need to keep track of the visitor's display preferences, and another might need to keep track of a visitor's subscription status.
- Script 12.7 uses cookies to keep track of a user's last visit, and then creates "New to You" messages as an image next to items on the page. How else could a site designer let users know about new content? Discuss the different interface choices, and give examples of least intrusive and most intrusive methods.
  - ☞ Least: A single text message somewhere on the page—a user can ignore this if they choose to. Most: A modal alert—a user has no choice but to interact with it.

## Review Questions

### Multiple choice

1. Cookies can get the following information from the user's machine:
  - A. Their mailing address
  - B. Their email address**
  - C. The computer's serial number
  - D. All of the above
  - E. None of the above
2. The `split()` method:
  - A. Divides a text string in half.
  - B. Terminates the script's execution.
  - C. Divides a text string into an array.**
  - D. Scans the cookie object for information.
3. Which of these cookie fields is not optional?
  - A. `expires`
  - B. `value`**
  - C. `path`
  - D. `domain`
4. Cookies are:
  - A. Plain text information.**
  - B. Executable code.
  - C. Binary file information.
  - D. MIME information.
5. Cookies on your machine can be read by:
  - A. Anyone who sends you email.
  - B. The server that originally wrote the cookie.**
  - C. Only your browser.
  - D. The Cookie Monster.

6. After `document.cookie.split(";")`:
  - A. The contents of the cookie will be stored in the Web browser's database.
  - B. The contents of the cookie will be displayed in your browser.
  - C. The contents of the cookie will be stored in your attic.
  - D. The contents of the cookie will be stored in an array.**

### Fill-in-the-blank

1. The cookie file is a plain text file on the user's hard disk.
2. Cookies always include the address of the server that sent it.
3. One way to delete a cookie is to set its expiration date to the past.
4. If there is only one cookie, after `cookieValue = document.cookie.  
→ split("=")[0]`, `cookieValue` contains the cookie's name.
5. Cookie fields are separated by a semicolon.

## Find the Errors

The following code contains several errors. Find them all.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
→ 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Cookie Check</title>
</head>
<body bgcolor="#FFFFFF">
<h2>
 <script language="Javascript"
 → type="text/javascript">
 <!-- Hide script from older
 → browsers

 if (document.cookies == "") {
 document.write("There are no
 → cookies here")
 }
 else {
 thisCookie = document.cookie.
 → split("=")

 for (i=0; i<document.cookie.
 → length; i++) {
 document.write("Cookie name
 → is '"+thisCookie[i].split
 → ("=")[1])
 document.write("'", and the
 → value is '"+thisCookie[i].
 → split("=")[2]+'
")
 }
 }

 // End hiding script -->
 </script>
</h2>
</body>
</html>

```

1. The `thisCookie` array indices should be 0 and 1, not 1 and 2.
2. The variable `thisCookie` is set incorrectly; it should be split based on `" ; "`, not `"="`.
3. The initial check for `if (document.cookies == "")` is incorrect—it's looking at cookies, not a cookie.
4. The `for` loop is checking against `document.cookie.length`, but it should be checking `thisCookie.length` instead.
5. One additional blank in case the students find a different way to fix the problems.

# Chapter 13: Introducing CSS ♦ Study Guide

---

## Learning Objectives

- Understand CSS rules.
  - ☞ This chapter is just the briefest of introductions to CSS, as you'll need this information for the following DHTML chapters. Ideally, the students will already have covered this topic in a previous HTML class; if so, this chapter can be skipped or just used as a simple refresher.
- Write CSS rules.
  - ☞ Students should be able to work with CSS, including writing rules and figuring out which rules take precedence.
- Understand how to set font styles with CSS.
- Understand how to write CSS rules using tag, class, and id selectors.
- Create style rules for links.

## Get Up and Running Exercises

- Write a style rule that creates a class called `headlinetext` that makes any text that it is applied to 24 points and boldface.

```
.headlinetext {font-size: 24pt;
→ font-weight: bold}
```

  - ☞ This is all that's required here.
- Write a style rule that creates an ID selector called `headlinetext` that makes any text that it is applied to 24 points and boldface.

```
#headlinetext {font-size: 24pt;
→ font-weight: bold}
```

  - ☞ All that should be changed from the previous exercise is to substitute a `#` for the period. This changes the rule from a class selector to an ID selector.
- Write a style rule that eliminates the underlining from text links, and turns the link red when you put the cursor over it.

```
.redLink {text-decoration: none}
.redLink:hover {color: #FF0000}
```
- Modify Script 13.8 to position four items on a page. Use your own images and ID names.

```
<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Exercise 13.4</title>
 <style type="text/css">
 #him {position: absolute;
 → top: 120px; left: 200px}
 #her {position: absolute;
 → top: 70px; left: 350px}
```

*continues on next page*

```

 #table {position: absolute;
 → top: 75px; left: 250px}
 #stage {position: absolute;
 → top: 100px; left: 100px}
 </style>
</head>
<body bgcolor="#FFFFFF">
<div id="stage">

</div>
<div id="her">

</div>
<div id="him">

</div>
<div id="table">

</div>
</body>
</html>

```

- ☞ Obviously, each student's example will differ from this one, but they should include enough to make it clear that they understand the concepts behind positioning.

## Class Discussion Questions

- What are the benefits of using CSS with your Web sites?
  - ☞ Entire books have been written on this topic. At a minimum, make sure students understand how CSS allows changes made in one place to affect an entire site, thus providing huge time and (often) cost savings when a site needs to be updated.
- When would you want to use ID selectors on your pages?
  - ☞ Make sure that students understand that the big difference between IDs and classes is IDs signify something unique. This is an important foundation concept for Chapter 14, in which JavaScript will need to be able to uniquely identify specific elements on a page so that they can then be modified.
- What are the differences between the `<div>` and `<span>` tags? Give examples of when you would want to use each one.
  - ☞ A `div` is a text block; a `span` is for inline text. One example of how each would be used is if this text was changed into a Web page: the code in exercises would be inside a `<div class="code">`, whereas the words themselves in this paragraph that are in code style would be inside `<span class="code">`.
- What is the difference between relative sizes and absolute sizes? Is it preferable to use one or the other?
  - ☞ This is a matter of some controversy, mainly because older browsers don't properly support some of the units of measurement. There's more detail in the sidebar on page 281. As noted there, the W3C strongly recommends the use of relative sizes on Web pages.

## Review Questions

### Multiple choice

1. The <style> tag requires which attribute?

A. text/css

**B. type**

C. css

D. text

2. Which is a style rule for a class?

**A. .widget {color: #0000FF;  
→ background: #00FF00;}**

B. div:widget {color: #0000FF;  
→ background: #00FF00;}

C. #widget {color: #0000FF;  
→ background: #00FF00;}

D. p widget {color: #0000FF;  
→ background: #00FF00;}

3. Which would make an h1 red?

**A. h1 {color:#FF0000}**

B. h1 {style="color:#FF0000"}

C. .h1 {color:#FF0000}

D. h1 [style:#FF0000]

4. You can apply which selectors simultaneously to a page element?

A. Tag selectors.

B. Class selectors.

C. ID selectors.

**D. All of the above.**

E. B and C.

### Fill-in-the-blank

1. A CSS selector indicates which page elements will be formatted.

2. If an element has both a class and an ID selector applied to it, the ID takes precedence.

3. The <div> tag is a container tag.

4. Class names must be preceded by a period.

5. An id selector must be preceded by a #.

6. The <span> tag applies a style to part of a text block.

7. Absolute positioning begins at the upper-left corner of the page.

8. The <div> tag produces a line break before and after the tag.

### Definitions

1. What does CSS stand for?

☞ CSS stands for Cascading Style Sheets.

2. What are “cascading” styles?

☞ The same rule can be defined differently in different areas of your Web site and Web page. The “cascade” effect is how the browser decides which of the conflicting rules to apply.

3. What is an em?

☞ An em is a measurement of relative size, equivalent to the width of the letter M in the chosen font.

4. What is the difference between CSS1, CSS2, and CSS-P?

☞ CSS1 (from December 1996) was the original specification for applying styles to Web pages. CSS-P (from January 1997) added positioning elements. CSS2 (from May 1998) combined the two into one specification and added more features.

# Chapter 14: Working with DHTML ♦ Study Guide

## Learning Objectives

- Understand what DHTML is.
  - ☞ While the book says that DHTML consists of JavaScript, CSS, and HTML, it's also correct to include the DOM in that list.
- Learn how the DOMs (Document Object Models) differ in various browsers.
  - ☞ Different DOMs implemented in different browsers are one of the largest drawbacks to implementing DHTML solutions, because each DOM has its own way of doing things.
- Learn to manipulate objects in various browsers.
  - ☞ While the book shows how to move both text and images, these examples are used to demonstrate how the DOM handles each, not because moving text and images is a recommended idea.
- Understand how to manipulate text effects in Internet Explorer for Windows.
  - ☞ Stress to students that, while Internet Explorer for Windows is the most popular browser, creating a site that only works in a single browser is a bad idea.

## Get Up and Running Exercises

- Visit the Web Standards Project Website at <http://www.webstandards.org>. Browse the site, especially the “Learn” section. Come to class prepared to discuss the benefits of creating sites that support Web standards.
  - ☞ This site (especially Learn) should be added to students' bookmarks for their future use, as it's frequently updated with new information.
- Modify Script 14.4 to move two objects around the screen. Use your own images.

```
<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Exercise 14.2</title>
 <script type="text/javascript"
 → language="Javascript">
 <!-- Hide script from older
 → browsers

 firstTime = true
 if (document.getElementById)
 {
 stdBrowser = true
 }
 else {
 stdBrowser = false
 }

 function moveIt() {
 if (firstTime) {
 if (stdBrowser) {
```

*continues on next page*

```

 mover1Obj = document.
 → getElementById
 → ("mover1").style
 mover2Obj = document.
 → getElementById
 → ("mover2").style

 mover1Obj.top = "5px"
 mover1Obj.left = "5px"

 mover2Obj.top = "50px"
 mover2Obj.left = "50px"
}

if (document.all) {
 maxHeight = document.
 → body.clientHeight-40
 maxWidth = document.
 → body.clientWidth-40
}
else {
 maxHeight = window.
 → innerHeight-40
 maxWidth = window.
 → innerWidth-40
}
firstTime = false
}

moveTheObject("mover1")
moveTheObject("mover2")
setTimeout("moveIt()",20)
}

function moveTheObject
→ (thisObj) {
 if (stdBrowser) {
 moverObj = document.
 → getElementById
 → (thisObj).style
 topPos = parseInt
 → (moverObj.top)

 leftPos = parseInt
 → (moverObj.left)
 }
 else {
 document.mover = eval
 → ("document."+thisObj)
 topPos = document.
 → mover.top
 leftPos = document.
 → mover.left
 }

 chgXBy = Math.floor
 → (Math.random() * 10)
 if ((halfChance() || topPos
 → >= maxHeight) && topPos
 → > 5) {
 topPos -= chgXBy
 }
 else {
 topPos += chgXBy
 }

 chgYBy = Math.floor
 → (Math.random() * 10)
 if ((halfChance() ||
 → leftPos >= maxWidth) &&
 → leftPos > 5) {
 leftPos -= chgYBy
 }
 else {
 leftPos += chgYBy
 }

 if (stdBrowser) {
 moverObj.top = topPos
 → + "px"
 moverObj.left = leftPos
 → + "px"
 }
 else {

```

*continues on next page*



```

 document.mover.top =
 → topPos
 document.mover.left =
 → leftPos
 }
}

function halfChance() {
 if (Math.random() < .5) {
 return true
 }
 return false
}

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

#mover1 {position: absolute;
→ left: 5px; top: 5px;}
#mover2 {position: absolute;
→ left: 50px; top: 50px;}

-->
</style>
</head>
<body bgcolor="#FFFFFF" onload=
→ "moveIt()">
<div id="mover1">

</div>
<div id="mover2">

</div>
</body>
</html>

```

☞ Students should only have a minimal amount of duplicated code—as in the above sample answer, they should be able to figure out how to use a function instead. You also might consider requiring them to prove that it works in multiple browsers, including (at least) some version of Netscape 4 and some current standards-compliant browser.

- Modify Script 14.7 to move around two objects, where one of the objects is always displayed behind the other (referred to in the text as three dimensions) when the two overlap. Use your own images.

```

#mover1 {position: absolute;
→ left: 5px; top: 5px;
→ z-index: 1}
#mover2 {position: absolute;
→ left: 50px; top: 50px;
→ z-index: 2}

```

☞ The only lines that should have to change from the previous example are ones that set the z-index for the two objects.

- If you are using Internet Explorer for Windows, experiment with different filters, as listed in Table 14.1. You can modify Script 14.10, 14.11, or 14.12.

☞ Designers who want their effects to be seen by all site visitors may want to shun the use of these filters, but if you are targeting Internet Explorer for Windows users, the filters can provide some interesting effects. There's no code here because students should be given the option of which filters they want to play with.

## Class Discussion Questions

- Why are Web standards important? Which standards are most valuable for the Web site creator to know? Are there standards that can be safely ignored?
  - ☞ If your students are likely to work on government or education-related sites in the future, this would be a good time to talk about Section 508 compatibility, which requires that federal agencies' electronic and information technology is accessible to people with disabilities. You'll find more information at <http://www.section508.gov>.
- If you want to include animation on your site, you can use DHTML or Macromedia Flash. What are some of the pros and cons of each approach?
  - ☞ Using DHTML doesn't require costly tools. With Flash you have more control over how objects appear. Both DHTML and Flash work well in most current browsers.
- Discuss the differences between the older Document Object Models (DOMs) and the W3C DOM. What changes should you have to make to your pages to take advantage of the W3C DOM?
  - ☞ If you like, point students to online resources where they can learn more about the W3C DOM, such as <http://www.webstandards.org/learn/resources/dom/index.html> or [http://www.dmoz.org/Computers/Programming/Internet/W3C\\_DOM/](http://www.dmoz.org/Computers/Programming/Internet/W3C_DOM/)

## Review Questions

### Multiple choice

1. In different DOMs, the objects `document.all.airplane.style.pixelTop` and `document.airplane.top` are:
  - A. Properties of each other.
  - B. Mirror images.
  - C. Equivalent.**
  - D. W3C standards-compliant.
2. Which of the following units should not be used in Web development?
  - A. Points**
  - B. Pixels
  - C. Ems
  - D. All of the above.
  - E. A and B.
3. In the W3C DOM, you use which method to select objects?
  - A. `document.getElementById()`**
  - B. `document.all`
  - C. `document.id`
  - D. `document.element`
4. If more than one page element is placed in the same position, how do you specify which element will be on top?
  - A. `setVerticalPos=top`
  - B. `align: top`
  - C. z-index**
  - D. None of the above.

5. In the W3C DOM, you can specify an object ID with:
- A. <div>
  - B. <span>
  - C. <p>
- D. Any of the above.**
6. The test `if (document.getElementById())` can be used to:
- A. Check that the document is formatted correctly.
- B. Check if the browser is standards-compliant.**
- C. Check that the document has all required elements.
  - D. None of the above.

### Fill-in-the-blank

1. DHTML is a combination of HTML, Cascading Style Sheets, and JavaScript.
  2. The standards body responsible for Web standards is the W3C or World Wide Web Consortium.
  3. The best way to discover if a particular browser supports a particular measurement unit is to test that unit with the browser.
  4. The Web would be a better place if people would stop using Netscape 4.x.
  5. In standards-compliant browsers, you should use the `getElementById()` method to select objects.
  6. You can use Internet Explorer 6+'s `document.compatMode` object to see if the browser is set to be standards-compliant.
7. The filters in Internet Explorer are not HTML or standard JavaScript.
    - ☞ 'Proprietary' or 'non-standards-compliant' would also be acceptable here.
  8. When using `getElementById()`, the ID goes inside the parenthesis.
    - ☞ 'Inside quotes inside the parenthesis' would also be acceptable here.

### Definitions

1. What is DHTML?
  - ☞ DHTML is the combination of HTML, CSS, JavaScript, and the DOM.
2. Describe how you use `getElementById`.
  - ☞ There are two ways to use `getElement` → `ById`: (1) as an object, to check to see if the browser can handle standards-compliant code, and (2) as a method, so that JavaScript can find and manipulate objects on a page.
3. Define `z-index`.
  - ☞ The `z-index` of an object is how JavaScript decides which of two objects will display when they overlap on a Web page. The object with the higher numbered `z-index` will display over the one with the lower number.
4. What is an Internet Explorer filter?
  - ☞ Internet Explorer filters are a non-standard but powerful way to bring graphics capabilities to the Web browser.

# Chapter 15: User Interface Design with JavaScript ♦ Study Guide

## Learning Objectives

- Learn how you can use JavaScript to improve the user experience.
  - ☞ A good user experience is all about interacting with the user, and JavaScript is the only way to make Web sites interactive.
- Understand how to simulate pull-down menus on your pages.
  - ☞ One drawback of creating pull-down menus in this way is that it, like the menus used by operating systems, are tied to the top of the window. Make sure students understand that the way the script works is to move the unseen menu parts offscreen.
- Learn to implement sliding menus.
  - ☞ Students should understand how this one script is handling two sets of users (those with current browsers versus those with older browsers) at the same time.
- Explore how to add pop-up messages (“tool tips”) over regions of your Web page.
  - ☞ Note that these are not the type of pop-up windows that pop-up blockers block.
- Learn how to make form fields on Web pages act more like their equivalents in Windows or Mac OS.

## Get Up and Running Exercises

- Write a short essay discussing the differences between the ways users interact with operating systems and applications as compared to the way they interact with Web pages.
  - ☞ The content of this is up to the student, but one point they should include is that Web pages are cross-platform, and therefore, must straddle the differences between how different operating systems accomplish the same tasks.
- Modify Script 15.1 to add a third pull-down menu. Let the menu you add have at least four items on it.

```

<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Exercise 15.2</title>
 <script type="text/javascript"
→ language="Javascript">
 <!-- Hide from older browsers

 if (document.getElementById) {
 stdBrowser = true
 }
 else {
 stdBrowser = false
 }

 function toggleMenu(currElem,
→ nextPos) {
 if (stdBrowser) {

```

*continues on next page*

```

 menuObj = document.
 → getElementById
 → (currElem).style
 }
 else {
 menuObj = eval
 → ("document." + currElem)
 }
 if (toggleMenu.arguments.
 → length == 1) {
 if (parseInt(menuObj.top)
 → == -5) {
 nextPos = -90
 }
 else {
 nextPos = -5
 }
 }
 if (stdBrowser) {
 menuObj.top = nextPos
 → + "px"
 }
 else {
 menuObj.top = nextPos
 }
}

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

.menu {position:absolute;
→ font:12px arial, helvetica,
→ sans-serif;
→ background-color:
#CCCCCC; layer-
→ background-color:#CCCCCC;
→ top:-90px}
#fileMenu {left:10px;
→ width:70px}
#searchMenu {left:85px;
→ width:100px}

#blogMenu {left:190px;
→ width:80px}
A {text-decoration:none;
→ color:#000000}
A:hover {background-color:
→ #000099; color:#FFFFFF}

-->
</style>
</head>
<body bgcolor="white">
<div id="fileMenu" class="menu"
→ onmouseover="toggleMenu
→ ('fileMenu',-5)" onmouseout=
→ "toggleMenu('fileMenu',
→ -90)">

 <a href="javascript:
 → window.open()">Open

 <a href="javascript:
 → window.print()">Print
 →

 <a href="javascript:
 → history.back()">Back
 →

 <a href="javascript:history.
 → forward()">Forward

 <a href="javascript:
 → window.close()">Close
 →

 <a href="javascript:toggleMenu
 → ('fileMenu')">File
</div>
<div id="searchMenu" class="menu"
→ onmouseover="toggleMenu
→ ('searchMenu',-5)" onmouseout=
→ "toggleMenu('searchMenu',
→ -90)">

 <a href="http://www.google.
 → com">Google

 <a href="http://www.ask.
 → com">Ask Jeeves


```

*continues on next page*

```

<a href="http://www.alltheweb.
→ com">All The Web

<a href="http://www.av.
→ com">AltaVista

<a href="http://www.dmoz.
→ com">Open Directory<hr />
<a href="javascript:toggleMenu
→ ('searchMenu')">Search
</div>
<div id="blogMenu" class="menu"
→ onmouseover="toggleMenu
→ ('blogMenu',-5)" onmouseout=
→ "toggleMenu('blogMenu',
→ -90)">

 <a href="http://www.backupbrain.
 → com">Backup Brain

 <a href="http://www.webstandards.
 → org">WaSP Buzz

 <a href="http://www.boingboing.
 → net">BoingBoing

 <a href="http://www.TeeVee.
 → org">TeeVee

 <a href="http://www.dailykos.
 → com">Daily Kos<hr />
 <a href="javascript:toggleMenu
 → ('blogMenu')">Weblogs
</div>
</body>
</html>

```

- ☞ Students should be able to figure out that they don't actually have to add any JavaScript to make this work—in fact, all they should have to add is a single style rule, and a `<div>` block at the end containing the chosen links.

- Modify Script 15.2 to make all of the names of the pages clickable links.
  - ☞ And as with Exercise 15.2 (above), they shouldn't have to add any JavaScript here, either. All they should do is add `<a>` tags around the lines within the `<span>` tags.
- Create a Web page implementing one of the interface concepts developed during class discussion.
  - ☞ Students should have come up with cool interface ideas during the class discussion. Here's their chance to see if they can make their ideas reality.

## Class Discussion Questions

- Discuss the differences in the user experience between computer operating systems and the Web. What was lost in the browser environment, compared to working with other applications on your computer?
  - ☞ One of the many differences that could be mentioned here is that OS vendors, such as Apple, create human interface guidelines for how software developers should create applications with standard widgets. The Web doesn't have anything similar; consequently, any given Web site can have a totally different user interface than any other.
- Discuss what, if anything, the browser environment brought to the user experience that the previous world of OS/applications did not include.
  - ☞ Students may come up with anything here, but the main theme that should be stressed here is that Web pages work on all operating systems, making them available to everyone, not just those with a particular type of computer.
- Discuss specific things you can do with JavaScript to make the user's visit to your Web page easier.
  - ☞ Instructors should stress clear navigation and accessibility and discuss how to make sites that work in the largest possible number of browsers.

## Review Questions

### Multiple choice

1. It is important to provide user feedback on your site:
  - A. Only in site navigation.
  - B. When the visitor is using a standards-compliant browser.
  - C. When the user requests it.
  - D. Wherever you can.**
2. In Script 15.1, the largest part of each of the menus is initially:
  - A. Contained in the `toggleMenu` object.
  - B. Defined as the `.menu` CSS class.
  - C. Positioned above the top of the screen.**
  - D. Hidden behind the active window.
3. Which is valid JavaScript code?
  - A. `<a href="javascript;toggleMenu → ('fileMenu')">File</a>`
  - B. `<a href="javascript:toggleMenu → ('fileMenu')">File</a>`**
  - C. `<a href="javascript:toggleMenu → ("fileMenu")">File</a>`
  - D. `<a href="toggleMenu → ['fileMenu']">File</a>`
4. The technique used in Script 15.3 uses JavaScript to create the tool tips for:
  - A. Any map of the moon.
  - B. An image map.**
  - C. The `moon` object.
  - D. The `popup` function.

5. Click-anywhere form fields:
  - A. Allow the user to click anywhere in the form.
  - B. Allow the form to accept mouse clicks in all fields.
- C. Extend the clickable area of checkboxes and radio buttons.**
- D. Allow the user to edit anywhere inside scrollable text boxes.

### Fill-in-the-blank

1. When doing pull-down menus, you must take into account that older browsers cannot do rollovers on `<div>` tags.
2. Script 15.2, the sliding menus example, takes advantage of the fact that modern browsers can add page elements via JavaScript and automatically reflow text.
3. You can use tool tips to give users pop-up instructions when filling out forms.
4. Click-anywhere form fields allow the user to click on the text label of a checkbox or radio button to select it.
5. The values of a group of radio buttons are stored in an array.

### Find the Errors

The following code contains a number of errors. Find them all.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
→ XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Shakespeare's Plays</title>
 <script type="text/javascript"
→ language="Javascript">
 <!-- Hide script from older
→ browsers

 function toggleMenu(currMenu) {
 if (document.getElementById) {
 thisMenu = document.
→ getElementById(currMenu).
→ style
 if (thisMenu.display ==
→ "block") {
 thisMenu.display = "none"
 }
 else {
 thisMenu.display = "block"
 }
 }
 }

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

 .menu {display:none; margin-left:
→ 20px}

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

*continues on next page*



```
<body bgcolor="#FFFFFF">
<h1>Shakespeare's Plays</h1>
<a href="page1.html" onclick="toggleMenu
→ ('menu1')">Comedies

 All's Well That Ends Well

 As You Like It

 Love's Labour's Lost

 The Comedy of Errors

</body>
</html>
```

1. The `toggleMenu()` function doesn't return any value (it should return `true` or `false`).
2. The `onclick` handler in the `<a>` doesn't look for a return value, causing `page1.html` to always load even when it shouldn't.
3. The value (`menu1`) being passed to `togglemenu()` doesn't match the `id` of the `<span>` (`menu`), and it has to in order to be changed.
4. The `<span>` doesn't have the `class` attribute, and it's needed in order for the correct style rule to apply.
5. One extra number here in case the student comes up with an alternate way to fix the code.

# Chapter 16: Applied JavaScript ♦ Study Guide

---

## Learning Objectives

- Learn how to put your JavaScript code into external .js files.
  - ☞ In the real world, most code is put into external files. It's time for students to learn how to do this.
- Learn how to put your style sheets into external files.
  - ☞ Much of the power of CSS (i.e., being able to change an entire site's look by changing a single rule) is possible only when you put your style sheets into external files.
- Understand how to create custom JavaScript objects.
  - ☞ This is an immensely powerful tool, and the book touches on it only briefly. Students should be encouraged to play around with its possibilities.
- Understand how to use JavaScript to change a page's style sheet.
  - ☞ This handy widget is getting to be popular on more and more Web sites, as it lets visitors change the look and feel of sites to suit their own needs.

## Get Up and Running Exercises

- Take one of the exercises that you did earlier in the book, and move its JavaScript into an external .js file.
  - ☞ The resulting file should be checked to make sure that students removed any “hide from older browsers” comments as well as any <script> tags. Additionally, the <script> tags in the HTML file should only contain “hide from older browsers” comments if there actually is code to be hidden.
- Combine three scripts from earlier in this book into a new page, pulling the JavaScript code (when reasonable) into an external file. Make the page do something reasonably useful. Make sure to identify the scripts you used as inspiration.
  - ☞ This is the student's choice, so you just need to make sure that there's a minimum of duplicated JavaScript and that full advantage is taken of the external JavaScript file.
- Script 16.13 creates a custom JavaScript object. Write an entirely different script that creates one or more custom objects.
  - ☞ Once again, this is the student's choice; the resulting page(s) should be checked to make sure that students have come up with something both meaningful and valid.
- Modify the Style Sheet Switcher example (see page 361) to use your own custom style sheets and to allow the user to switch between three style sheet possibilities.

## Class Discussion Questions

- Discuss when you would want to place code into an external `.js` file. Are there situations on sites where you would not want to use an external file?
  - ☞ Students may come up with a number of reasons, but here are two examples: (1) if there's a small amount of JavaScript code that's only used on one page, leaving it on the page may make more sense, and (2) if one page of a site does something slightly different from how the rest of the site does it, you might want to keep the code for that one page internal so that the code can't incorrectly be changed to match the rest of the site.
- The Style Sheet Switcher example shows one way you can use JavaScript to make your sites more user friendly. Can you come up with other examples of how sites can use style sheet switching to cater to different user needs?
  - ☞ As one example, sites might want to offer different style sheets based on the needs of users with disabilities (e.g., impaired vision or color-blindness).
- Discuss when and how you would want to create custom JavaScript objects.
  - ☞ Students should take a look at the book example, in which custom objects are being set to other custom objects, creating new properties for the former on the fly. This is a handy technique for future use.

## Review Questions

### Multiple choice

1. You can insert references to how many external `.js` files in one script?
  - A. Zero or one
  - B. Only one
  - C. As many as you want**
  - D. Any number between 1 and 256
2. You should put references to an external JavaScript file inside the:
  - A. <head> tag.**
  - B. <title> tag.
  - C. <body> tag.
  - D. DOCTYPE declaration.
3. External `.js` files are:
  - A. Encapsulated in an HTML file.
  - B. Required by the W3C.
  - C. Never more than 8K of text.
  - D. None of the above.**
4. The `charCodeAt()` method:
  - A. Encodes a character for later secure encryption.
  - B. Returns the ASCII value for a character.**
  - C. Returns the octal value for a character.
  - D. None of the above.

5. You can create a new custom object in JavaScript:
  - A. Whenever you want.
  - B. Only in an external .js file.
  - C. Using new Object().
  - D. Both A and C.**
6. To change shared scripts:
  - A. Change them in each Web page.
  - B. Change the external .js file.**
  - C. Use the defineScriptSource() method.
  - D. Any of the above.

### Fill-in-the-blank

1. To share scripts between different Web pages, you should place the scripts in a(n) external .js file.
2. You can share style sheets between different Web pages by placing them into (an) external .css file.
3. The expression charCodeAt(5) looks at the sixth character in the string.
4. In the two Bar Graph examples, lilRed.gif is one pixel in size.
5. In Script 16.10, arrayArray is an array that contains other arrays.

### Find the Errors

The following code contains several errors. Find them all.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
→ XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Exercise 16.1</title>
 <style src="myStyles.css" type=
→ "text/stylesheet" />
 <script href="external.js"
→ type="text/javascript"
language="Javascript" />
</head>
<body>
 Body text goes here
</body>
</html>
```

1. The correct syntax for the <script> tag is to use a src attribute, not an href attribute.
2. The <style> tag cannot refer to an external style sheet; you need to use the <link> tag for that.
3. The correct attribute for the <link> tag is href, not src.
4. If the <style> tag was correct, the correct value for the type attribute would be text/css, not text/stylesheet. The <link> tag doesn't have a type attribute at all.
5. One extra space is left here for students to come up with their own answer, but be aware that closing the <script> tag with /> is valid—it doesn't need a </script> tag.

# Chapter 17: Manipulating Nodes ♦ Study Guide

## Learning Objectives

- Learn what a node is and how it fits into the DOM's tree structure.
  - ☞ The concepts of nodes and the DOM's tree structure are important parts of understanding how the browser handles the underlying content of Web pages. In order to change that content, you need to understand the concepts.
- Learn why you'd want to manipulate nodes.
  - ☞ As node manipulation is fairly new technology, students may wonder what it can be used for. See the first discussion question, below, for some real-world usage.
- Learn how nodes can be added, deleted, and replaced.
  - ☞ This chapter's examples only cover text nodes, but students should be encouraged to extrapolate how other types of nodes would be manipulated.

## Get Up and Running Exercises

- Modify Script 17.1 to add images to a page instead of adding paragraphs of text.

```
<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN"\>
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Exercise 17.1</title>
 <script type="text/javascript"
→ language="Javascript">
 <!-- Hide script from older
→ browsers

 function addNode(inImg) {
 newImg = document.
→ createElement("img")
 newImg.src = inImg

 docBody = document.
→ getElementsByTagName
→ ("body").item(0)
 docBody.appendChild(newImg)
 docBody.appendChild
→ (document.createElement
→ ("br"))

 return false
 }

 // End hiding script from
→ older browsers -->
 </script>
</head>
<body>
<form action="#" onsubmit=
→ "return addNode(this.textField.
→ value)">
```

*continues on next page*

```



```

- ☞ There's some slightly tricky stuff here—students will need to understand that adding a paragraph means adding two nodes (one for the tag, one for the text inside the tag, as shown in the book's text) whereas adding an image is only one node. Consequently, where the book calls `appendChild()` twice to add the paragraph, a student's example should only have to call `appendChild()` once to add the image. An additional `appendChild()` has also been added above to put a `<br />` between images; this is just to make the resulting page look nicer.

- Modify Script 17.5 to handle images instead of paragraphs.

```

<!DOCTYPE html PUBLIC "-//W3C//
→ DTD XHTML 1.0 Transitional//
→ EN"\>
<html xmlns="http://www.w3.org/
→ 1999/xhtml">
<head>
 <title>Exercise 17.2</title>
 <script type="text/javascript"
→ language="Javascript">
 <!-- Hide script from older
→ browsers

function addNode(inImg) {
 newNode = document.
→ createElement("img")

```

```

newNode.src = inImg

docBody = document.
→ getElementsByTagName
→ ("body").item(0)
docBody.appendChild
→ (newNode)
docBody.appendChild
→ (document.createElement
→ ("br"))
}

```

```

function delNode(delChoice) {
 allImgs = document.
→ getElementsByTagName
→ ("img")
 killImg = allImgs.item
→ (delChoice)

 docBody = document.
→ getElementsByTagName
→ ("body").item(0)
 removed = docBody.
→ removeChild(killImg)
}

```

```

function insertNode
→ (inChoice,inImg) {
 newNode = document.
→ createElement("img")
 newNode.src = inImg

 allImgs = document.
→ getElementsByTagName
→ ("img")
 oldImgs =
allImgs.item(inChoice)

 docBody = document.
→ getElementsByTagName
→ ("body").item(0)

```

*continues on next page*

```

 docBody.insertBefore
 → (newNode,oldImgs)
 docBody.insertBefore
 → (document.createElement
 → ("br"),oldImgs)
}

function replaceNode
→ (inChoice,inImg) {
 newNode = document.
 → createElement("img")
 newNode.src = inImg

 allImgs = document.
 → getElementsByTagName
 → ("img")
 oldImgs = allImgs.item
 → (inChoice)

 docBody = document.
 → getElementsByTagName
 → ("body").item(0)
 docBody.replaceChild
 → (newNode,oldImgs)
}

function nodeChanger() {
 actionType = -1
 for (i=0;i<nodeForm.
 → nodeAction.length;i++) {
 if (nodeForm.nodeAction
 → [i].checked) {
 actionType = i
 }
 }
}

switch(actionType) {
 case -1:
 alert("No action was
 → chosen")
 break
 case 0:
 addNode(nodeForm.
 → textField.value)
 break
 case 1:
 delNode(nodeForm.
 → imgCount.selected
 → Index)
 break
 case 2:
 insertNode(nodeForm.
 → imgCount.selected
 → Index,nodeForm.
 → textField.value)
 break
 case 3:
 replaceNode(nodeForm.
 → imgCount.selected
 → Index,nodeForm.
 → textField.value)
 break
 default:
}
document.nodeForm.
→ imgCount.options.
→ length = 0
for(i=0;i<document.
→ getElementsByTagName
→ ("img").length;i++) {
 document.nodeForm.
 → imgCount.options[i] =
 → new Option(i+1)
}
return false
}

// End hiding script from
→ older browsers -->
</script>
</head>
<body>
<form action="#" name="nodeForm"
→ onsubmit="return nodeChanger()">

```

*continues on next page*

```

<input type="text"
 → name="textField" size="30"
 → />

<input type="radio" name=
 → "nodeAction" />Add image
<input type="radio" name=
 → "nodeAction" />Delete image
<input type="radio" name=
 → "nodeAction" />Insert before
 → image
<input type="radio" name=
 → "nodeAction" />Replace
 → image

Image #: <select name=
 → "imgCount">
</select>
<input type="submit" name=
 → "submit" value="Submit" />
</form>
</body>
</html>

```

- ☞ This exercise should be similar to the first exercise, but students should get more of a feel for how image nodes are similar to and differ from paragraph nodes.
- Combine Script 17.5 and Exercise 17.2 to let the user add or delete either text or images on the page.
  - ☞ This should just be a combination of the two scripts. Give students more credit for reusing the most amount of code (i.e., the less duplicate code on the page, the better).

- Visit the W3C site mentioned on page 372, and come to class prepared to discuss what DOM 2 is. For lots of extra credit, translate the specification from the mystic language of geeks into plain English.

☞ This isn't a bad place in the book to expose students to the full force of a W3C spec. It is useful to see how the real-world pages that we write are derived from such formalistic, academic specifications.



## Class Discussion Questions

- Give some examples of how node manipulation can be used in real-world Web sites.
  - ☞ Let students come up with their own ideas, but some that we've heard of include being able to rewrite a page on the fly based on user feedback (e.g., sorting on various fields) and form fields that are added to the page based on answers to previous questions.
- When would you want to use node manipulation versus server-side technology?
  - ☞ Node manipulation means that you don't have to reload the entire page from the server, but server-side CGIs let you (for example) read from databases.
- Discuss the tree structure of nodes, as shown on page 373.
  - ☞ Note how the HTML container tags correspond to element nodes.

## Review Questions

### Multiple choice

1. The DOM tree structure contains:
  - A. Nodes.
  - B. HTML tags.
  - C. Text on the page.
  - D. All of the above.**
2. An element node contains:
  - A. The DOM tree structure.
  - B. An HTML tag.**
  - C. Binary code.
  - D. Only JavaScript.
3. You can add a node using which method?
  - A. `makeElement()`
  - B. `createNode()`
  - C. `createElement()`**
  - D. `buildNode()`
4. To get all of the paragraph tags on a page, you would use which method?
  - A. `getElementsById("p")`
  - B. `getElementsByTagName("p")`**
  - C. `getParagraphElements()`
  - D. `getParagraphs()`
5. Which of the following is a valid method?
  - A. `insertNodeAfter()`
  - B. `insertAfter()`
  - C. `appendAfter()`
  - D. `appendChild()`**

**Fill-in-the-blank**

1. The DOM Level 2 specification is produced by the W3C.
2. You can use JavaScript to manipulate any aspect of the DOM tree structure.
3. To delete a node, use the removeChild() method.
4. Text nodes are contained by element nodes.
5. You can exchange one node with another by using the replaceChild() method.

**Definitions**

1. What is the tree structure?
    - ☞ The tree structure is how the browser internally represents the structure of a Web page. The tree structure contains nodes, each of which represents a tag or text element on the page. See Figure 17.1 on page 373 for a representation of a simple page's tree structure.
  2. What is a node?
    - ☞ A node is any aspect of a Web page, whether tags or text. JavaScript is able to manipulate nodes.
  3. What is an element node?
    - ☞ An element node is how any and every tag on a Web page is represented in a tree structure view of a Web page.
  4. What is a text node?
    - ☞ A text node is how the text on a Web page is represented in a tree structure view of a Web page.
-

# Chapter 18: Bookmarklets ♦ Study Guide

---

## Learning Objectives

- Learn what bookmarklets are and what they're useful for.
  - ☞ Bookmarklets have been called the most powerful tool that Web designers have that they don't use. We recommend spending some time talking about how darn useful and powerful they are.
- Learn what the limitations and differences are of bookmarklets in some browsers.
  - ☞ For instance, Netscape 4 bookmarklets must be 255 characters or less, and Internet Explorer for Windows uses a different syntax for getting a selection off a Web page than that used by both Internet Explorer for Mac and Netscape.
- Learn how to add bookmarklets to your browser.
  - ☞ Every browser has a different way to add bookmarklets, so how students add them depends on their chosen browser.

## Get Up and Running Exercises

- Create a bookmarklet that opens a new window that calls the W3C HTML validator on the opening window.

```
javascript:void(window.
→ open('http://validator.w3.org/
→ check?uri='+window.location.
→ href, '_blank', ''))
```

  - ☞ This is one of those incredibly useful bookmarklets that every Web designer and developer should have installed on their main browser.
- Create a bookmarklet that resizes the current window to be 700 pixels wide, as high as the screen is tall, and flush against the left side of the screen.

```
javascript:moveTo(0,0);resizeTo
→ (700,screen.availHeight)
```

  - ☞ Note that this (and in fact, most bookmarklets) may vary slightly due to differences between browsers. Students should be graded on how well their bookmarklets work in their browser, not just on matching this example.

*continues on next page*

- Modify the previous exercise to resize the window to the same dimensions, but flush against the right side of the screen.

```
javascript:moveTo(screen.
→ availWidth-700,0);resizeTo
→ (700,screen.availHeight)
```

- ☞ This isn't as easy as students might think at first, as they'll need to figure out where to put the top-left corner of the browser. Given that we know the ending width of the window, the student will have to calculate where the window should move to based on the width of the screen.
- Visit <http://www.bookmarklets.com> and <http://www.favelets.com>, and report back to the class on at least one cool and/or useful bookmarklet that you found there.
- ☞ There's considerably more that can be done with bookmarklets than can be covered in this book, so students should be familiar with these great resources.

## Class Discussion Questions

- Discuss why bookmarklets are useful.
  - ☞ They are useful because you can perform a wide variety of handy, practical work with them, without having to load a Web page.
- Can you dream up bookmarklets that you would like to have in your browser?
  - ☞ This could be an interesting discussion, depending on what students are interested in. For example, students who are strongly interested in Web design and coding might be interested in bookmarklets that help them code, whereas students who do a lot of writing may be more interested in things like dictionary lookups.

## Review Questions

### Multiple choice

- What is the maximum number of lines a bookmarklet can be?
  - One**
  - 255
  - No limit
  - None of the above.
- A bookmarklet must:
  - End with a semicolon.
  - Return a value.**
  - Use single character variable names.
  - All of the above.
  - None of the above.
- Which of the following statements can refer to the text selected in a browser window?
  - `window.getSelection()`
  - `document.getSelection()`
  - `document.getSelection().createRange().text`
  - All of the above.**
  - None of the above.

☞ The answer to A, `window.getSelection()`, is the syntax used in Safari. This isn't mentioned in the text (it hadn't been documented at that point), so it should be brought up in class prior to students having to answer this question. C works in Internet Explorer for Windows, B works in Netscape (all platforms) and Internet Explorer for Mac.

- Bookmarklets use single character variable names because:
  - It's a requirement.
  - It's a requirement, but only for Netscape 4.
  - It's recommended for Netscape 4.**
  - None of the above.

### Fill-in-the-blank

- Bookmarklets call the browser's internal JavaScript interpreter.
- You separate JavaScript commands in a bookmarklet with a semicolon.
- The book examples create unit converters and calculators in a bookmarklet using JavaScript's built in Math object.
- Bookmarklets must be written in a single line.
- Bookmarklets that work in Netscape 4 cannot be longer than 255 characters.

### Definitions

- What is a bookmarklet?
  - A bookmark or favorite that contains a call to the browser's JavaScript interpreter.
- Explain what `javascript:void()` does and why it's useful.
  - Absolutely nothing, but it's useful for bookmarklets because in order to work correctly bookmarklets must return a value, which `void()` does (although it's a null value).

# Chapter 19: Working with Visual Tools ♦ Study Guide

---

## Learning Objectives

- Understand the advantages and disadvantages of visual tools in comparison to hand coding JavaScript.
  - ☞ Using a visual tool is certainly easier for the casual designer, plus it allows the designer to use the free extensions that are available for many tools. On the other hand, it's more expensive to use a visual tool instead of a text editor. And the code is often clunky, bloated, and difficult or impossible for humans to read, understand, and change.
- Use a visual tool to accomplish a task using JavaScript, and compare the results to hand coding.
  - ☞ Remind the students that if they don't have any visual tools, trial versions of Macromedia Dreamweaver, Adobe GoLive, Microsoft FrontPage, Macromedia Fireworks, and Adobe ImageReady (as part of Photoshop) may be downloaded from the companies' respective Web sites.

## Get Up and Running Exercises

- Many of the visual tools on the market are extensible, meaning that programmers can write additional features for them. Go to the Macromedia Exchange at <http://www.macromedia.com/exchange/> and browse through the extensions for Dreamweaver. Check out the range of extensions, and identify some that would be useful to you in your work.
  - ☞ The Macromedia Exchange is the biggest of the extension repositories, though they also exist for Adobe and Microsoft products. Students should be able to come up with several examples of extensions that would help them.
- Using the visual tool of your choice, recreate one of the book's more involved examples found in Chapters 5, 6, and 7. View the source code of the result, and compare it to the code found in the book. Come to class prepared to discuss the differences.
  - ☞ The code created by the tool will almost always be longer and much harder to read and understand than the hand-coded examples.

*continues on next page*

- One of the interesting uses of JavaScript is that many applications now use it as an internal scripting language. These applications use JavaScript to manipulate their own documents, and as a result, they need to have their own DOM (Document Object Model). To see an example of this, check out the specification for the Dreamweaver DOM, at <http://livedocs.macromedia.com/dreamweaver/mx2004/extending/index.htm>. See how Dreamweaver's DOM allows the program to be extended in many different ways.
  - ☞ Besides Dreamweaver, other Macromedia products, such as Fireworks and Flash, have their own DOMs, as do Adobe products such as Acrobat, Photoshop, and GoLive.

## Class Discussion Questions

- If a visual tool will write your JavaScript for you, why should you bother to learn the language yourself?
  - ☞ You'll be able to write JavaScript that can do things that the tool cannot. For example, out of the box, Dreamweaver can't do any but the simplest form validation. If you want to, say, check that a user entered the same password twice in a form, you'll have to either write your own code or try to find an extension that someone else has already written.
- Are there examples where the benefit of using a visual tool outweighs the loss of control that you get with hand coding?
  - ☞ Some things are quite difficult to hand code, such as the pop-up menus example that uses Fireworks (see page 420). In cases like those, even the coder who is comfortable with JavaScript might want to turn to machine-written code. Another reason might be that you're under a heavy deadline, and you simply don't have the time to write your own code.
- Discuss how you might want to customize a program like Dreamweaver.
  - ☞ One possibility would be to deploy to clients a limited version of Dreamweaver that they can use to make some changes to their sites. Or you could provide your Web developers with a custom version that includes a set of extensions and commands that they need to handle specific features of your Web site.

## Review Questions

### Multiple choice

1. Which program cannot write JavaScript for you?
  - A. Macromedia Dreamweaver.
  - B. Adobe GoLive.
  - C. Microsoft FrontPage.
  - D. Microsoft Word.**
2. Dreamweaver calls JavaScript it writes:
  - A. Behaviors.**
  - B. Actions.
  - C. JavaScript.
  - D. Events.
3. In the “Flying Objects in Adobe GoLive” example, the images must be:
  - A. GIFs.
  - B. Transparent GIFs.**
  - C. TIFFs.
  - D. JPEGs.
4. You can customize Dreamweaver using:
  - A. JavaScript.
  - B. HTML.
  - C. XML.
  - D. All of the above.**
5. GoLive calls JavaScript it writes:
  - A. Behaviors.
  - B. Actions.**
  - C. JavaScript.
  - D. Events.

### Fill-in-the-blank

1. The “Flying Objects in Adobe GoLive” example animates objects using CSS-P.
2. To create a pop-up menu in Fireworks, you need to start from a button.
3. Fireworks needs a hotspot to trigger a JavaScript behavior.
4. The JavaScript created by Fireworks will be in a file named menu.js.
5. To customize Dreamweaver’s menus, you must change the menus.xml file.

### Definitions

1. What are the differences between hand-written and machine-written JavaScript?
  - ☞ Hand-written code will be easier to understand, can be much shorter, and yields the same result.
2. What is the DHTML Timeline Editor in GoLive?
  - ☞ It allows the user to schedule events in animations, and to play them without previewing the page in a browser. Dreamweaver has a similar editor, the Timelines window.
3. What is a hotspot in Fireworks?
  - ☞ A hotspot in Fireworks is the overlay over a button or region that Fireworks uses to trigger a JavaScript behavior.



## Chapter 20: Debugging Common Errors ♦ Study Guide

---

### Learning Objectives

- Improve the debugging skills learned as you've worked your way through this book.
  - 📖 Please note that the Study Guide for this chapter does not include Exercises and Review Questions, because students should, by this point, be able to provide plenty of examples of buggy code. The authors feel that students will benefit more from debugging their own mistakes than from any examples we might invent.

### Class Discussion Questions

- What are the most common bugs that you've run into?
  - 📖 Different students will identify different common bugs, but sharing their experiences with others will help them all know what to keep an eye out for. Some common problems might involve case sensitivity, or unbalanced quotes, braces, and parentheses.
- What debugging techniques have worked best for you?
  - 📖 Again, the goal here is for students to share the knowledge they've gained. One approach that seems to work without fail is asking someone else to look at your code—a second set of eyes will almost always find something you've overlooked. Other suggestions are listed in the chapter.
- What browsers have you found best for debugging?
  - 📖 Some browsers are better than others for debugging, as some give more information and some give less. For example, Safari gives no JavaScript debugging information at all, whereas Netscape's JavaScript Console gives some good hints as to where code problems might lie.