COMP 484 Web Engineering I

Instructor: ~kaplan (adam.kaplan@csun.edu)

Lecture #4: 7/22/2020
JavaScript

JavaScript

- JavaScript
  - Scripting language which is used to enhance the functionality and appearance of web pages
  - Allows for *Dynamic HTML*
    - Script may change variables in web page's definition
      - After the page is loaded (i.e. while being viewed)
- Born in 1994 at Netscape
  - Intended as a lightweight scripting language complementing Java
  - Originally named LiveScript (Netscape 2.0)
    - Renamed to JavaScript in Netscape 2.0B3
    - Coincided with Java support being added to the browser

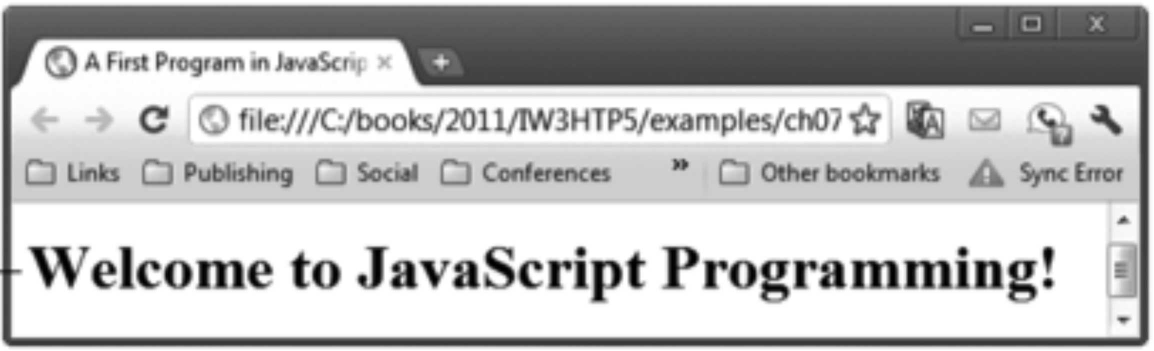
Displaying a Line of Text

```
<!DOCTYPE html>

<!-- Fig. 6.1: welcome.html -->
<!-- Displaying a line of text. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>A First Program in JavaScript</title>
      <script type = "text/javascript">

         document.writeln(
            "<h1>Welcome to JavaScript Programming!</h1>" );

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

Specifies that we are using Javascript scripting language

document.writeln calls writeln on this HTML document's object – adds a line

Script ends


A First Program in JavaScrip
file:///C:/books/2011/IW3HTP5/examples/ch07
Links
Publishing
Social
Conferences
Other bookmarks
Sync Error

Script result

Welcome to JavaScript Programming!

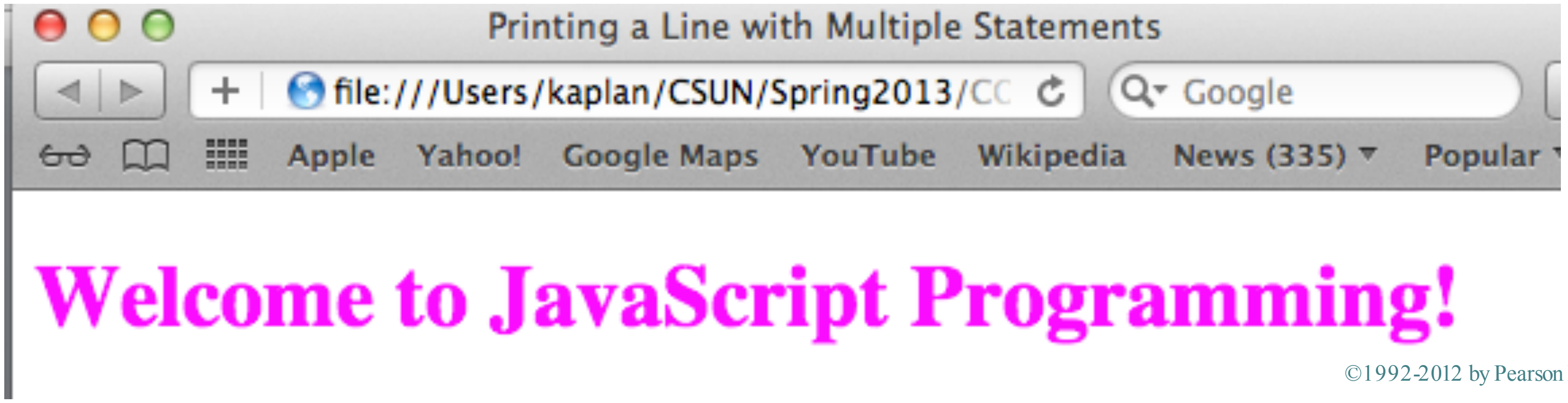
The Document Object

- Browser's `document` object represents the HTML5 document currently being displayed in the browser
  - Allows a you to specify HTML5 text to be displayed in the HTML5 document
- Browser contains a complete set of objects that allow scripts to access and manipulate all HTML5 elements
  - Uses the *Document Object Model (DOM)*
    - An object representation of an HTML document
- Objects (a refresher)
  - The term object normally implies that attributes (data) and behaviors (methods) are associated with the object
    - An object's methods use the attributes' data to perform useful actions for the client of the object
    - One such client is the script that calls object methods
    - E.g. previous example script called `document.writeln(…)`

```
<!DOCTYPE html>

<!-- Fig. 6.2: welcome2.html -->
<!-- Printing one line with multiple statements. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Printing a Line with Multiple Statements</title>
      <script type = "text/javascript">
         <!--
         document.write( "<h1 style = 'color: magenta'>" );
         document.write( "Welcome to JavaScript " +
            "Programming!</h1>" );
         // -->
      </script>
   </head><body></body>
</html>
```

+ string concat operator (just like Java)

`document.write` like **`document.writeln`** except does **not** append a newline \n at end

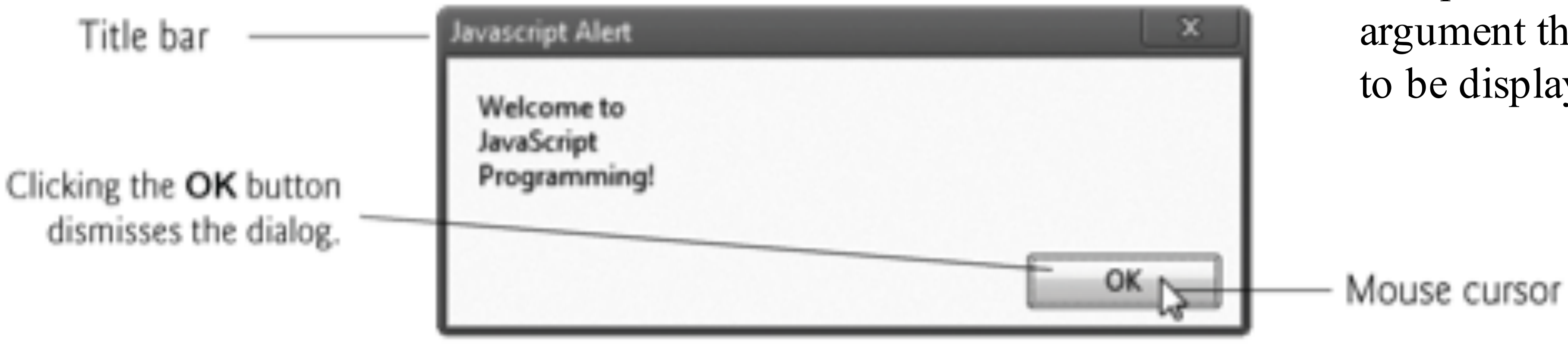

Printing a Line with Multiple Statements
file:///Users/kaplan/CSUN/Spring2013/CC
Google
Apple
Yahoo!
Google Maps
YouTube
Wikipedia
News (335)
Popular

Welcome to JavaScript Programming!

Displaying Text in Alert

```
<!DOCTYPE html>

<!-- Fig. 6.3: welcome3.html -->
<!-- Alert dialog displaying multiple lines. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Printing Multiple Lines in a Dialog Box</title>
      <script type = "text/javascript">
         <!--
         window.alert( "Welcome to\nJavaScript\nProgramming!" );
         // -->
      </script>
   </head>
   <body>
      <p>Click Refresh (or Reload) to run this script again.</p>
   </body>
</html>
```

- *Alert dialogs* "pop up" on the screen to grab the user's attention
  - Typically used to display important messages
- Browser's `window` object uses method `alert` to display an alert dialog
  - Requires as its argument the string to be displayed



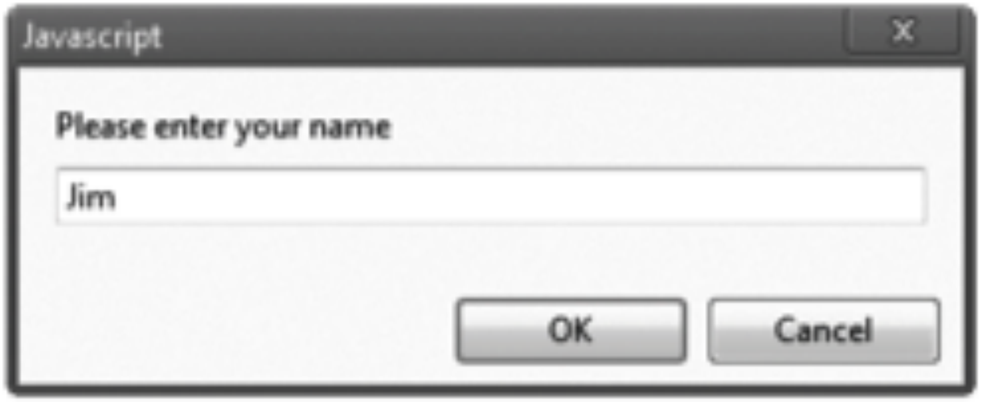
```
Welcome to
JavaScript
Programming!
```

Prompt Box

```
<!DOCTYPE html>

<!-- Fig. 6.5: welcome4.html -->
<!-- Prompt box used on a welcome screen -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Using Prompt and Alert Boxes</title>
      <script type = "text/javascript">
         <!--
         var name; // string entered by the user

         // read the name from the prompt box as a string
         name = window.prompt( "Please enter your name" );

         document.writeln( "<h1>Hello " + name +
            ", welcome to JavaScript programming!</h1>" );
         // -->
      </script>
   </head><body></body>
</html>
```

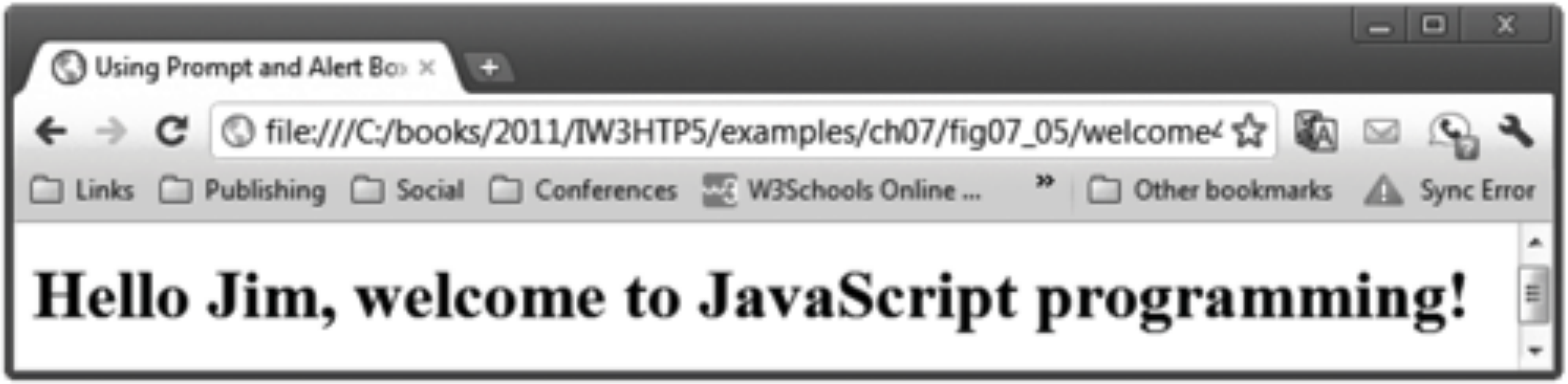
- Displays a prompt dialog to user
- Assigns the resulting string to name
- Name is printed in the welcome message

Prompt Box - Rendered



Please enter your name

Jim



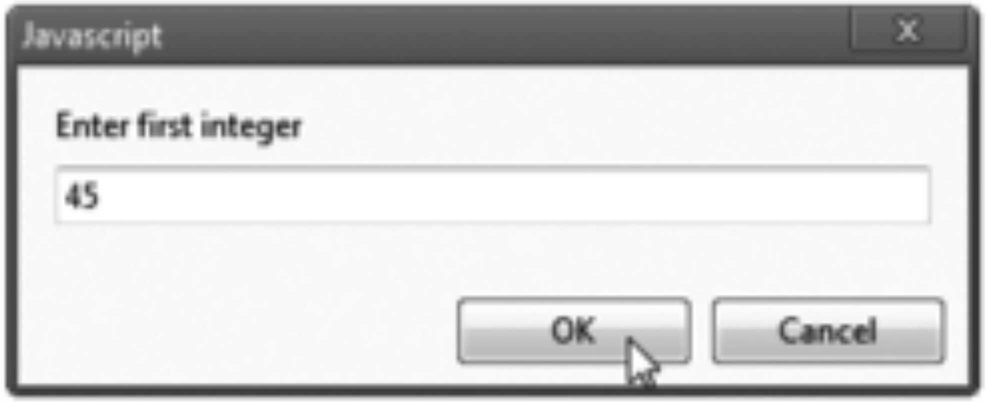
file:///C:/books/2011/IW3HTP5/examples/ch07/fig07_05/welcome

Hello Jim, welcome to JavaScript programming!

Adding Integers

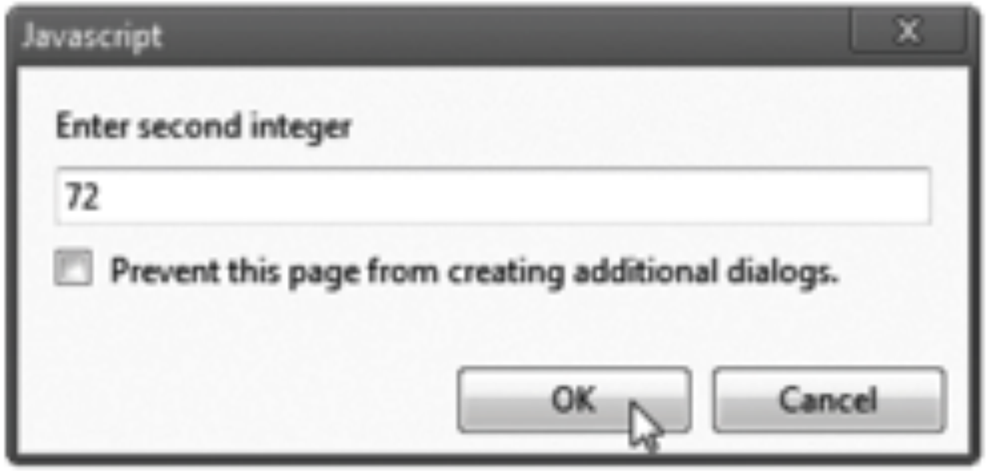
```
<!DOCTYPE html>

<!-- Fig. 6.7: addition.html -->
<!-- Addition script. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>An Addition Program</title>
      <script type = "text/javascript">
         <!--
         var firstNumber; // first string entered by user
         var secondNumber; // second string entered by user
         var number1; // first number to add
         var number2; // second number to add
         var sum; // sum of number1 and number2

         // read in first number from user as a string
         firstNumber = window.prompt( "Enter first integer" );

         // read in second number from user as a string
         secondNumber = window.prompt( "Enter second integer" );

         // convert numbers from strings to integers
         number1 = parseInt( firstNumber );
         number2 = parseInt( secondNumber );

         sum = number1 + number2; // add the numbers

         // display the results
         document.writeln( "<h1>The sum is " + sum + "</h1>" );
         // -->
      </script>
   </head><body></body>
</html>
```

Function `parseInt` converts its string argument to an integer



Enter first integer

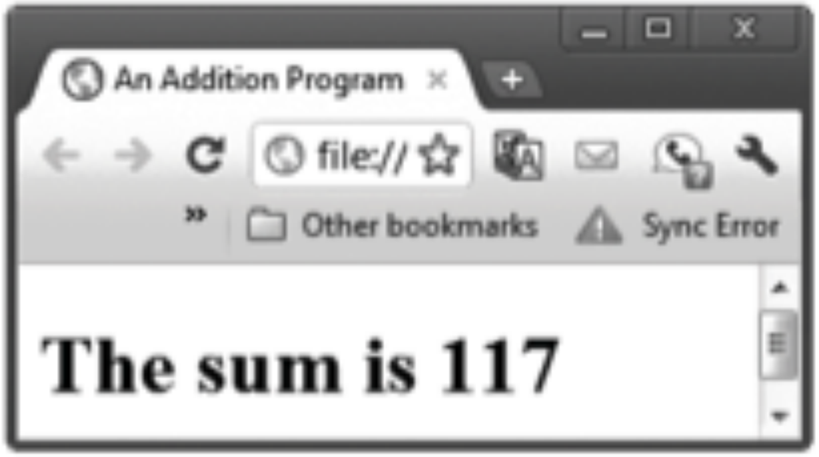
45



Enter second integer

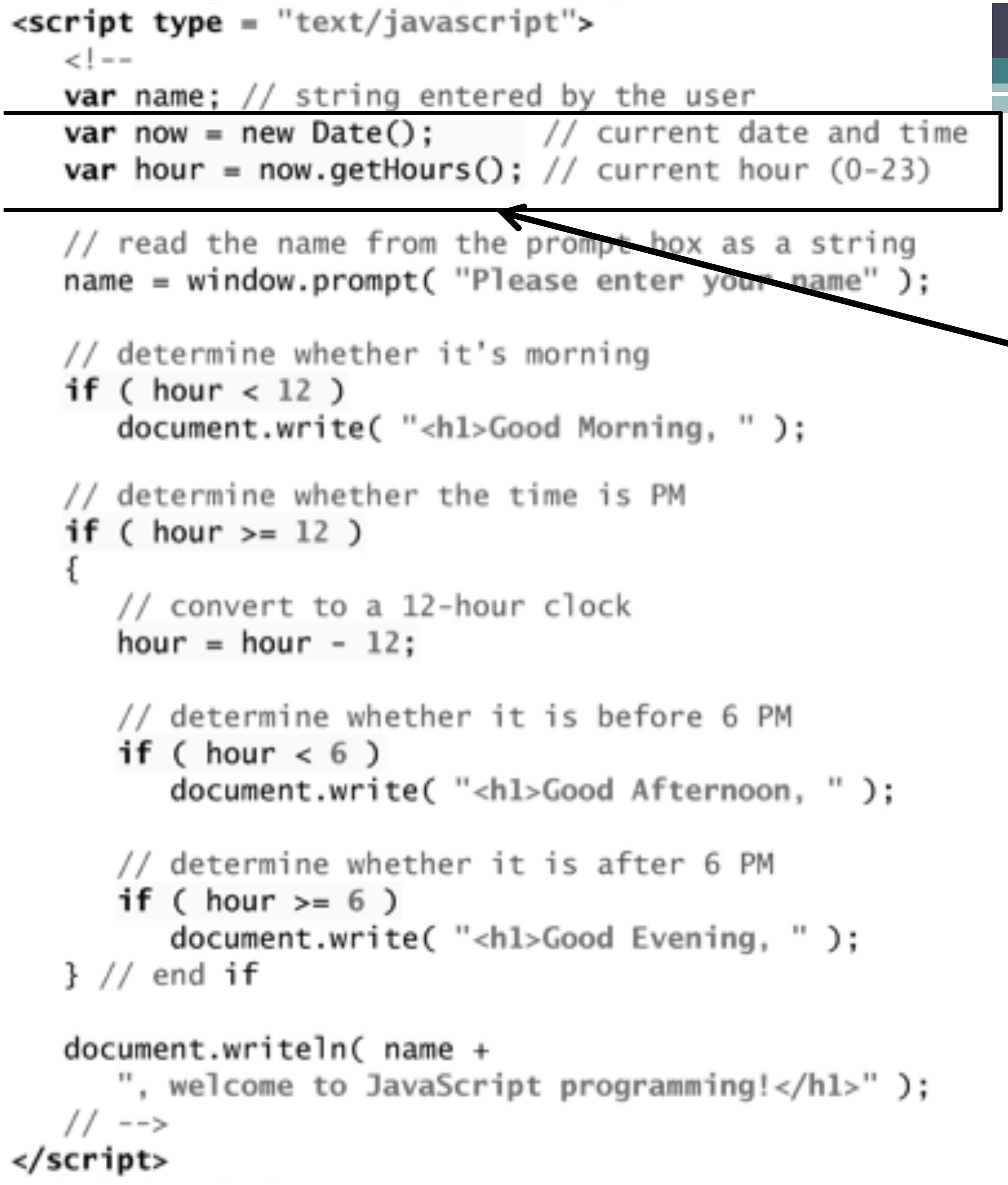
72

Prevent this page from creating additional dialogs.



The sum is 117

The Date Object



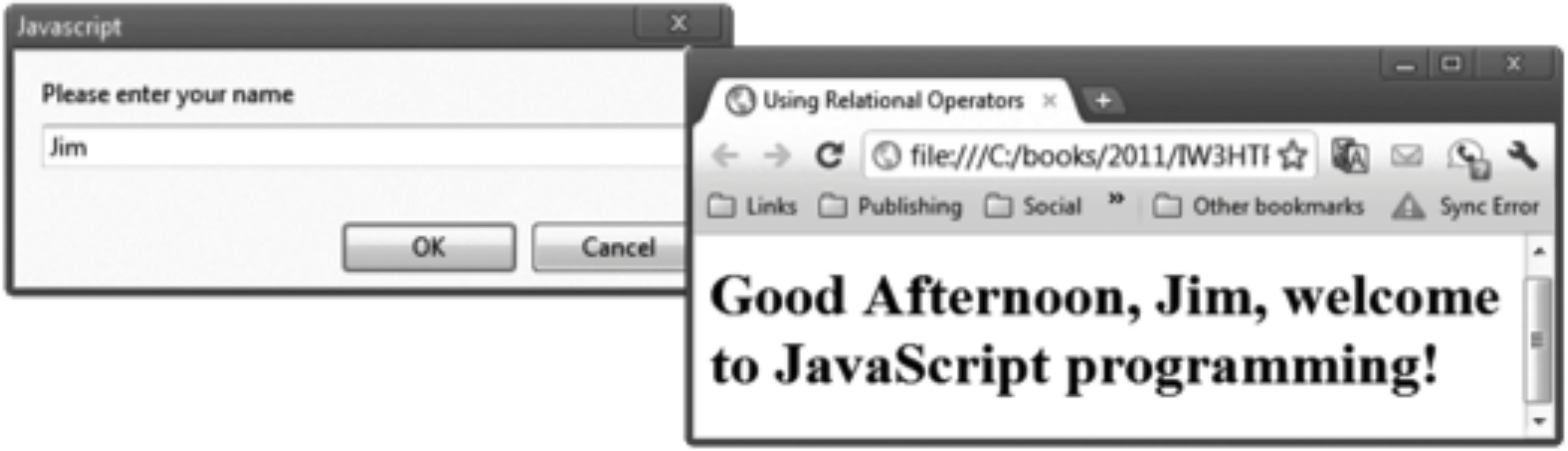
```
<script type = "text/javascript">
   <!--
   var name; // string entered by the user
   var now = new Date();      // current date and time
   var hour = now.getHours(); // current hour (0-23)

   // read the name from the prompt box as a string
   name = window.prompt( "Please enter your name" );

   // determine whether it's morning
   if ( hour < 12 )
      document.write( "<h1>Good Morning, " );

   // determine whether the time is PM
   if ( hour >= 12 )
   {
      // convert to a 12-hour clock
      hour = hour - 12;

      // determine whether it is before 6 PM
      if ( hour < 6 )
         document.write( "<h1>Good Afternoon, " );

      // determine whether it is after 6 PM
      if ( hour >= 6 )
         document.write( "<h1>Good Evening, " );
   } // end if

   document.writeln( name +
      ", welcome to JavaScript programming!</h1>" );
   // -->
</script>
```

Look, a Date object!

`new Date()` creates a Date object with current local time

`getHours()` returns the hour of day

The Date Object

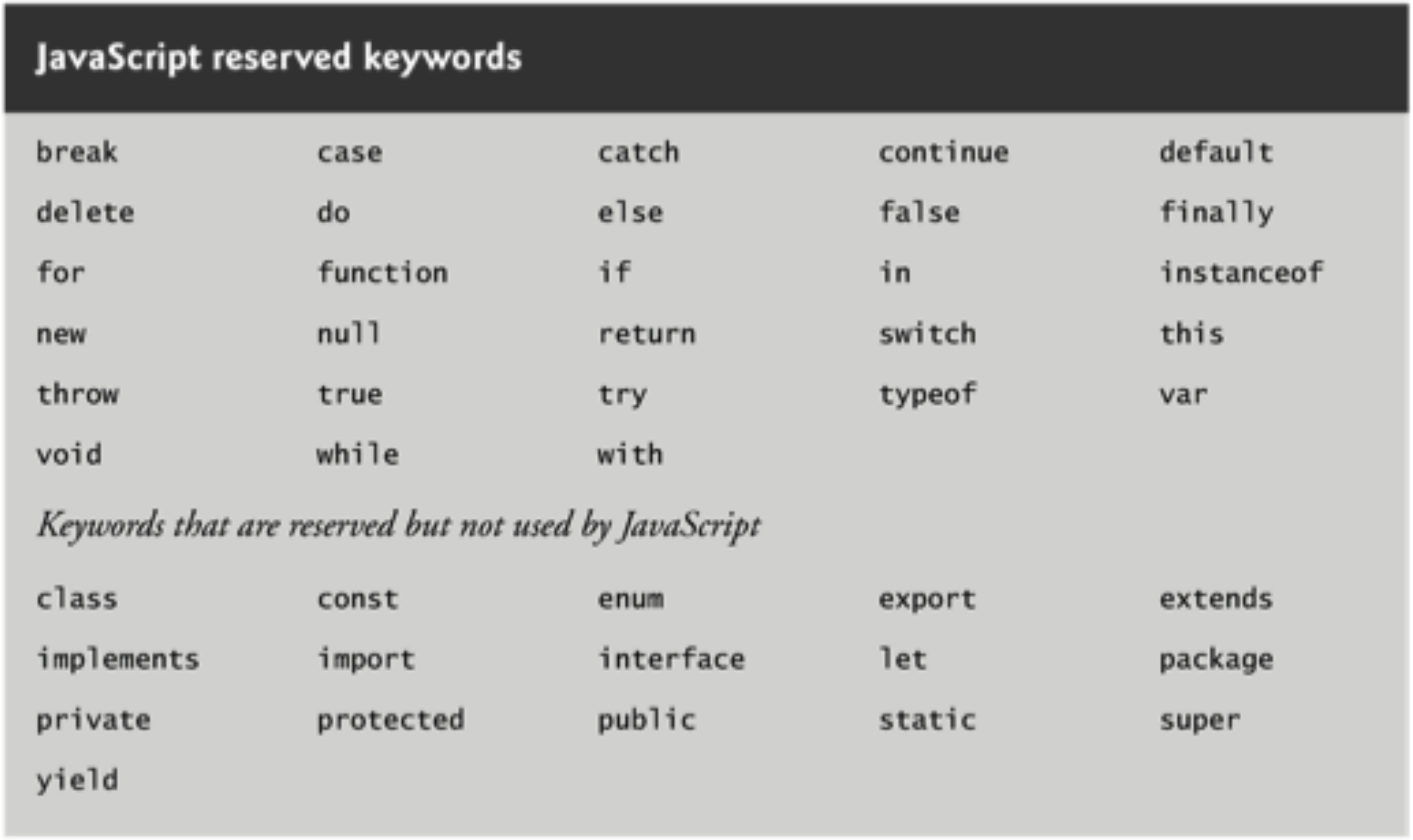


Please enter your name

Jim

Good Afternoon, Jim, welcome to JavaScript programming!

JavaScript Keywords



JavaScript reserved keywords

`break`	`case`	`catch`	`continue`	`default`
`delete`	`do`	`else`	`false`	`finally`
`for`	`function`	`if`	`in`	`instanceof`
`new`	`null`	`return`	`switch`	`this`
`throw`	`true`	`try`	`typeof`	`var`
`void`	`while`	`with`		

Keywords that are reserved but not used by JavaScript

`class`	`const`	`enum`	`export`	`extends`
`implements`	`import`	`interface`	`let`	`package`
`private`	`protected`	`public`	`static`	`super`
`yield`				

Arithmetic Operators

Assignment operator	Initial value of variable	Sample expression	Explanation	Assigns
`+=`	`c = 3`	`c += 7`	`c = c + 7`	10 to c
`-=`	`d = 5`	`d -= 4`	`d = d - 4`	1 to d
`*=`	`e = 4`	`e *= 5`	`e = e * 5`	20 to e
`/=`	`f = 6`	`f /= 3`	`f = f / 3`	2 to f
`%=`	`g = 12`	`g %= 9`	`g = g % 9`	3 to g

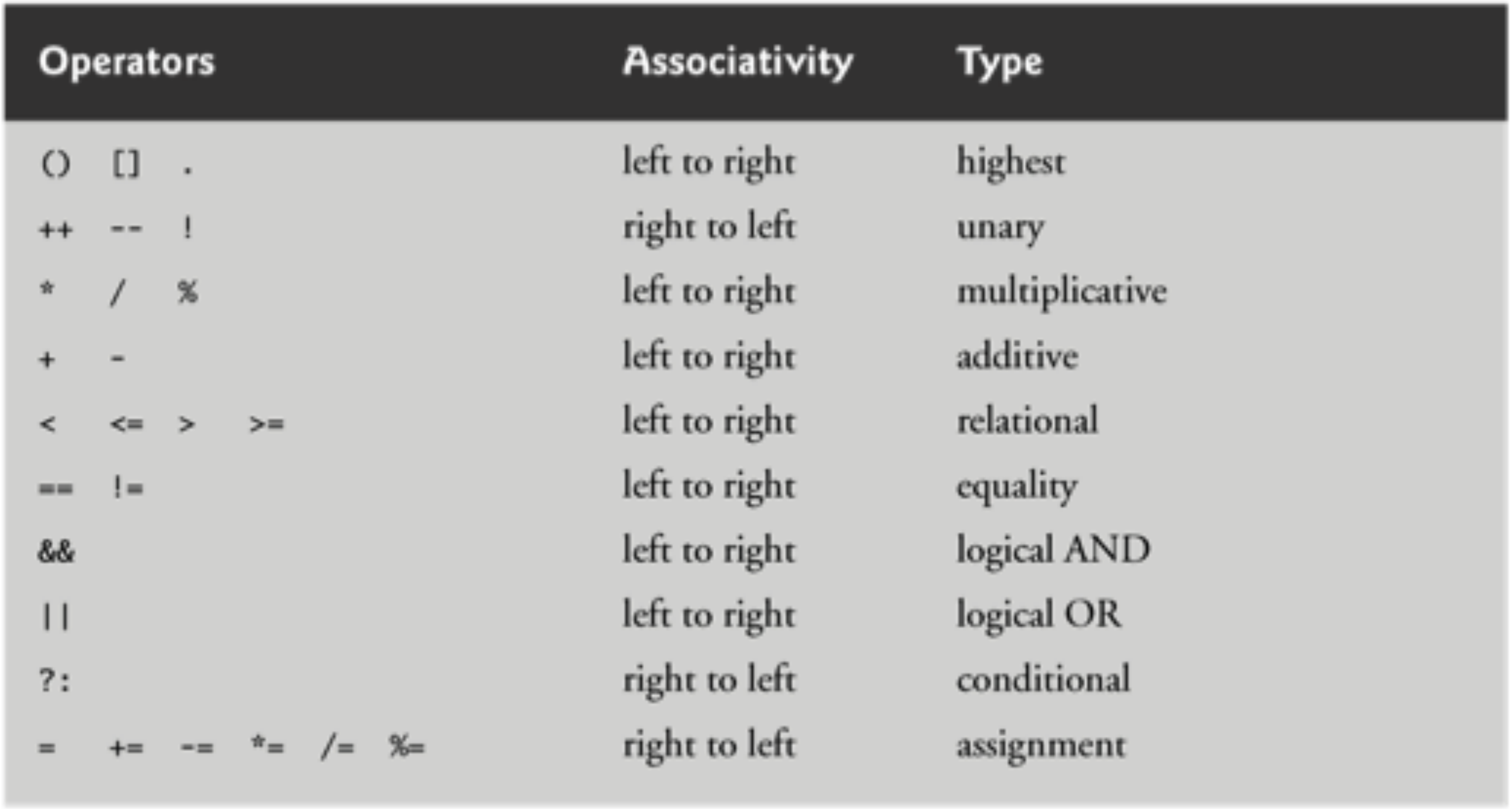
Operator	Example	Called	Explanation
`++`	`++a`	preincrement	Increment a by 1, then use the new value of a in the expression in which a resides.
`++`	`a++`	postincrement	Use the current value of a in the expression in which a resides, then increment a by 1.
`--`	`--b`	predecrement	Decrement b by 1, then use the new value of b in the expression in which b resides.
`--`	`b--`	postdecrement	Use the current value of b in the expression in which b resides, then decrement b by 1.

Logic Operators

- **The || (logical OR) operator**
  - True if either or both of two conditions are true
- **The && (logical AND) operator**
  - True if both of two conditions are true
  - Has a higher precedence than the || operator
- An expression containing && or || operators is evaluated only until truth or falsity is known
  - This is called *short-circuit evaluation*

- **The ! (logical negation) operator**
  - Reverses the meaning of a condition
  - (i.e., a true value becomes false, and a false value becomes true)

Operator Precedence



Operators	Associativity	Type
() [] .	left to right	highest
++ -- !	right to left	unary
* / %	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relational
== !=	left to right	equality
&&	left to right	logical AND
||	left to right	logical OR
?:	right to left	conditional
= += -= *= /= %=	right to left	assignment

Control Structures

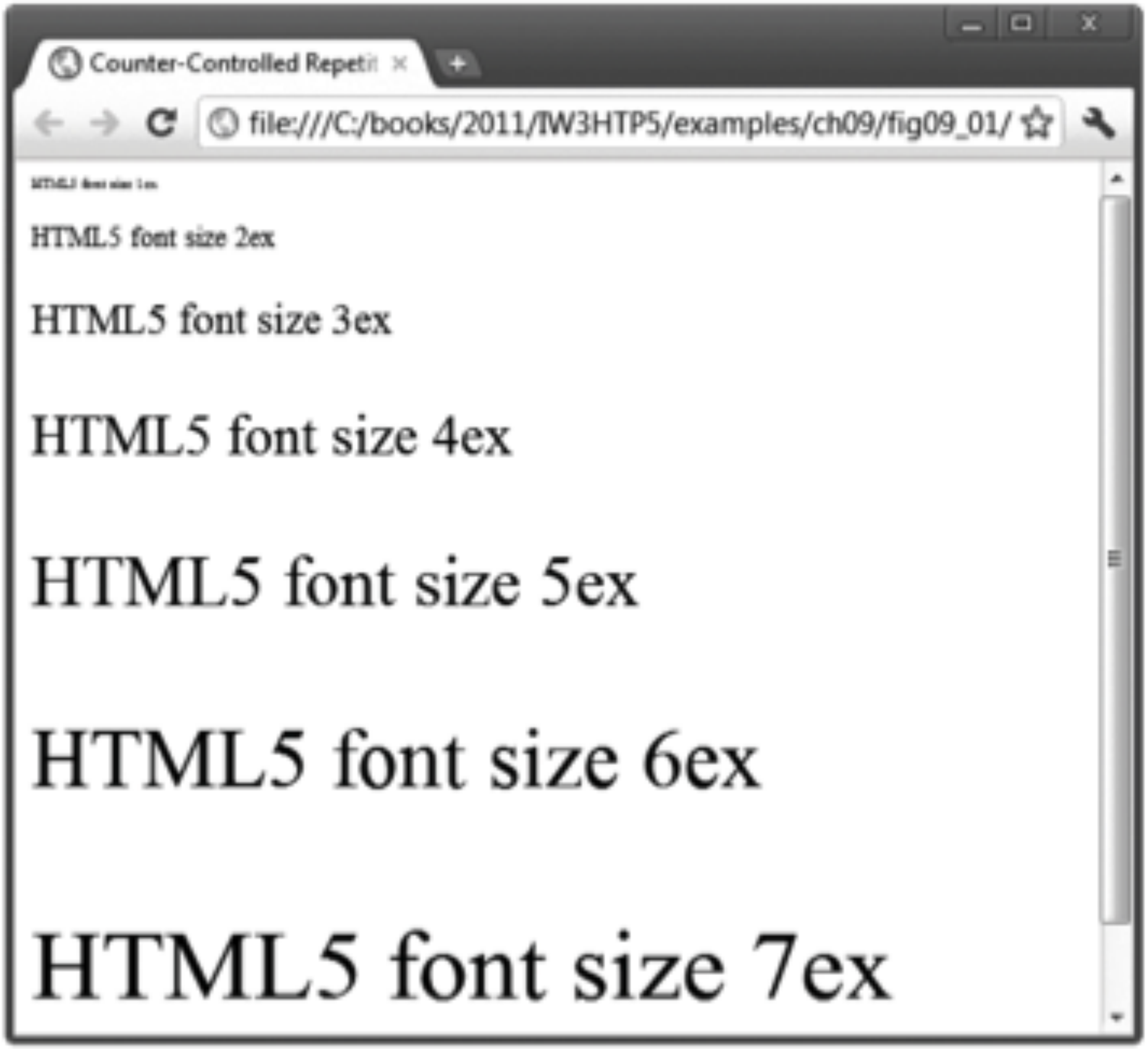
- All your favorites are present…
  - Conditionals
    - if, if/else
    - switch
  - Loops
    - while
    - do/while
    - for
- These look syntactically much like they do in Java
  - You will see them throughout the examples

While Loop Example

```
<!DOCTYPE html>

<!-- Fig. 8.1: WhileCounter.html -->
<!-- Counter-controlled repetition. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Counter-Controlled Repetition</title>
      <script>

         var counter = 1; // initialization

         while ( counter <= 7 ) // repetition condition
         {
            document.writeln( "<p style = 'font-size: " +
               counter + "ex'>HTML5 font size " + counter + "ex</p>" );
            ++counter; // increment
         } //end while

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

What does this code do?

While Loop Example


Counter-Controlled Repetit
file:///C:/books/2011/IW3HTP5/examples/ch09/fig09_01/

```
HTML5 font size 1ex
HTML5 font size 2ex
HTML5 font size 3ex
HTML5 font size 4ex
HTML5 font size 5ex
HTML5 font size 6ex
HTML5 font size 7ex
```

Program Modules

- Modules in JavaScript
  - Functions
    - You may write these yourself
  - Methods
    - (Belong to an object)
    - You may write something close to these using function constructors (advanced JavaScript)

- JavaScript includes many useful predefined methods
  - Can combine with your own programmer-defined functions to make most programs

RECAP: Pass-by-value vs Pass-by-reference

- Pass-by-value (PBV)
  - A copy is made of argument and is passed to the called function
- Pass-by-reference (PBR)
  - Caller gives called function direct access to data in place
  - Caller may modify data
  - Better performance (no copy overhead)
  - Can introduce bugs (unexpected variable modifications)

- JavaScript does not allow programmer to choose PBV or PBR
  - Scalars (numbers, strings, and booleans) are PBV
  - Objects and arrays are PBR

Returned Values

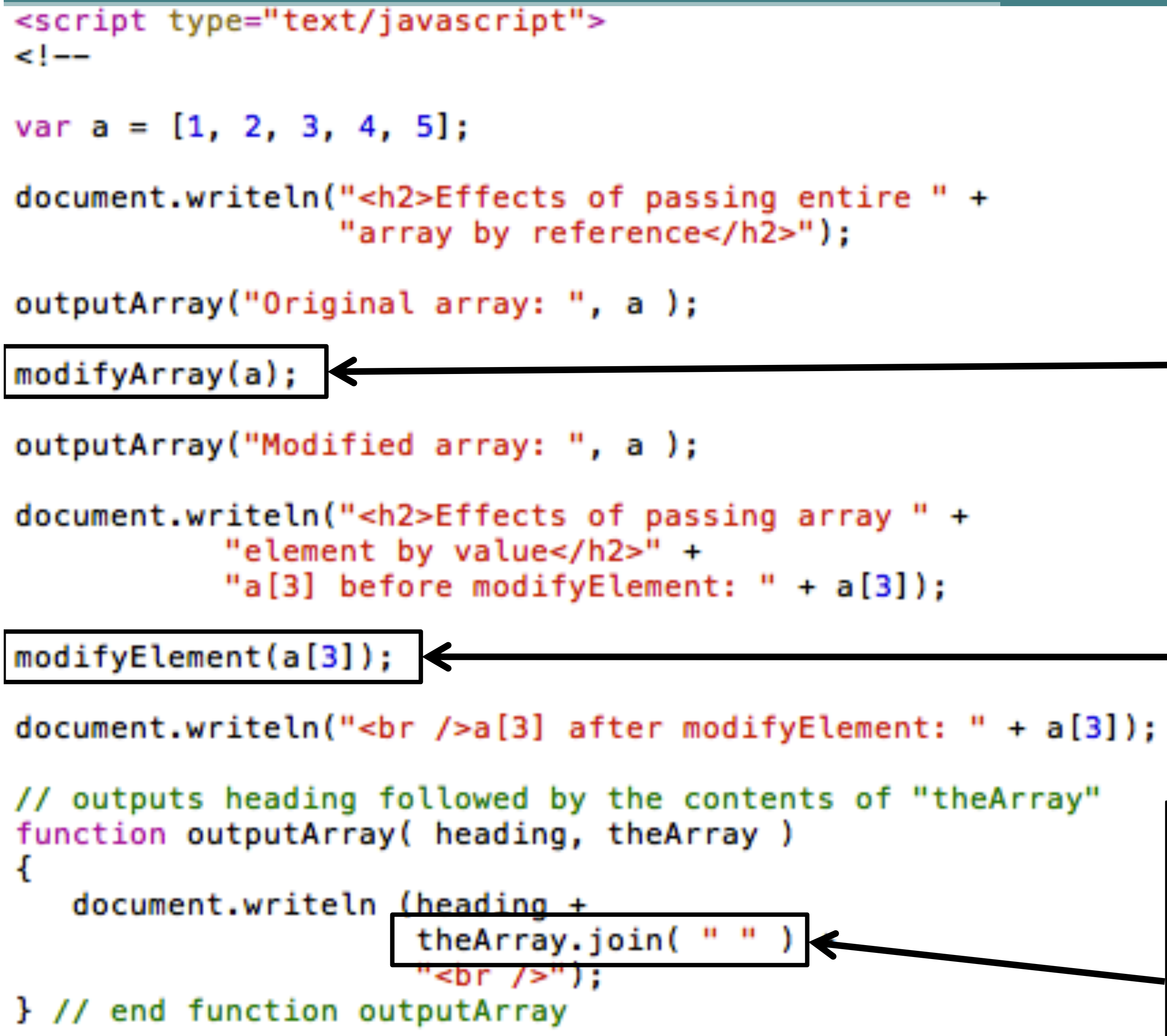
- Scalars
  - Returned by value
- Objects
  - Returned by reference
  - No need to use a return statement if passed into function, as objects are passed-by-reference

Passing Arrays

- Arrays are Objects
  - Whole arrays are passed by reference
  - To pass an array, the parameter is the name without square brackets

- The name of an array is actually a reference to an object
  - Contains the array elements
  - Contains the `length` variable, which indicates the number of elements in the array

Passing Arrays



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

var a = [1, 2, 3, 4, 5];

document.writeln("<h2>Effects of passing entire " +
                 "array by reference</h2>");

outputArray("Original array: ", a );

modifyArray(a);

outputArray("Modified array: ", a );

document.writeln("<h2>Effects of passing array " +
           "element by value</h2>" +
           "a[3] before modifyElement: " + a[3]);

modifyElement(a[3]);

document.writeln("<br />a[3] after modifyElement: " + a[3]);

// outputs heading followed by the contents of "theArray"
function outputArray( heading, theArray )
{
   document.writeln (heading +
                  theArray.join( " " ) +
                  "<br />");
} // end function outputArray
```

Passes array a to modifyArray by reference

Passes array element a[3] to modifyElement by value

Creates a string containing all the elements in theArray separated by " "

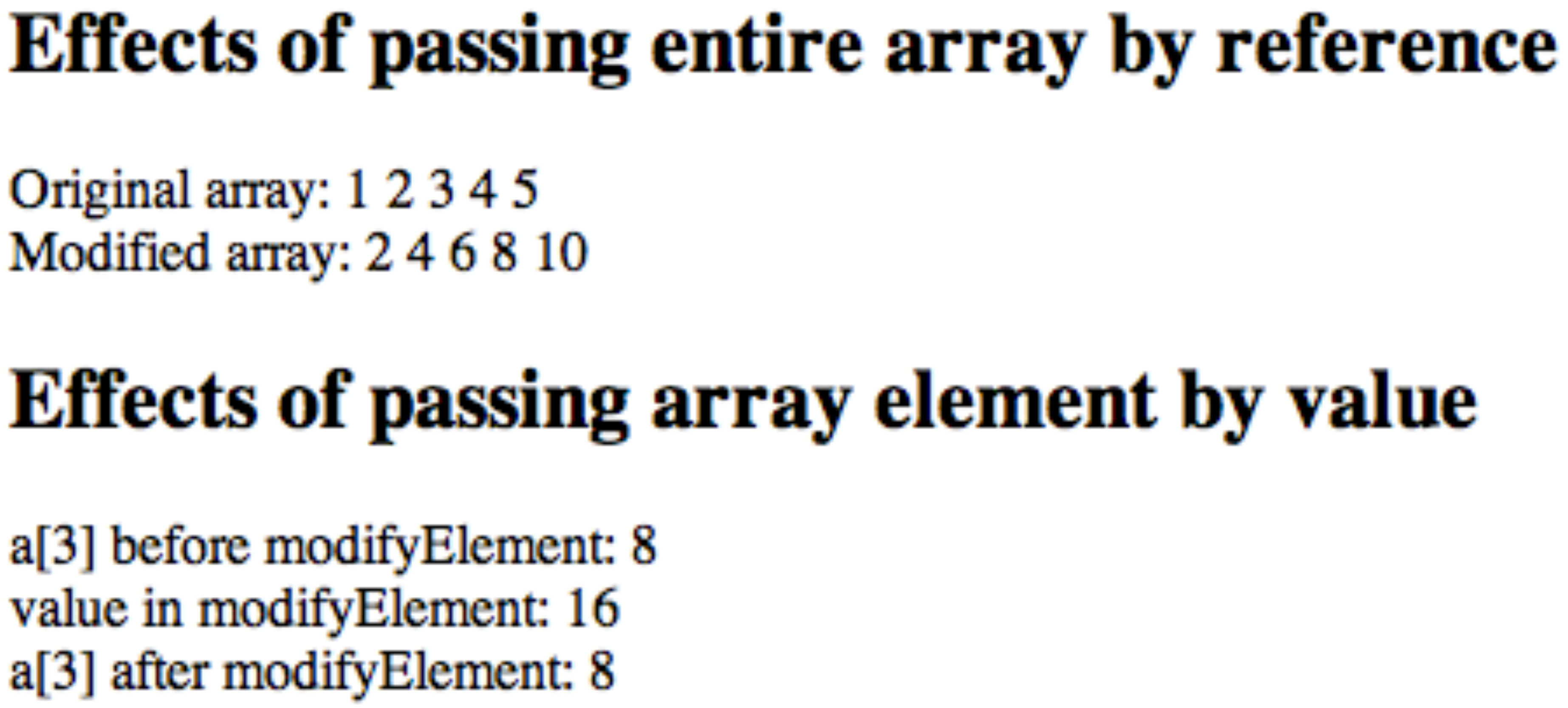
Passing Arrays (cont)

```
// function that modifies the elements of an array
function modifyArray( theArray )
{
   for ( var j in theArray )
   {
      theArray[ j ] *= 2;
   } // end for
} // end function modifyArray

// function that modifies the value passed
function modifyElement( e )
{
   e *= 2; // scales element e only for the duration of the function
   document.writeln ("<br />value in modifyElement: " + e);
} // end function modifyElement

-->
</script>
```

Multiplies each element in the array by 2, which persists after function has finished

Multiplies the passed-in element by 2, but this change is only for the duration of the function

Passing Arrays: Output



```
Effects of passing entire array by reference

Original array: 1 2 3 4 5
Modified array: 2 4 6 8 10

Effects of passing array element by value

a[3] before modifyElement: 8
value in modifyElement: 16
a[3] after modifyElement: 8
```

More on Functions

- JavaScript does not check number/types of arguments passed to function
  - It is possible to pass any number of values to a function
  - JavaScript will attempt type conversion as necessary
- Functions in JavaScript are considered to be data
  - Functions can be stored in variables/arrays
  - Functions can be passed to functions as arguments

JavaScript: ==, ===, != and !==

- The strict equality operators: === and !==
  - These behave as you might expect from strongly typed languages
  - If the type matches and they have the same value, === returns true and !== returns false
    - And vice-versa

- The regular equality operators: == and !=
  - These operate on value only
  - If the types don't match, will perform conversions to test value
    - 5 == "5" is true, 5 != "5" is false
    - 5 === "5" is false, 5 !== "5" is true

JavaScript: *null vs undef*

- `null`
  - A special value meaning “no value”
  - `typeof null` returns `object`
- `Undefined`
  - Variable has either not been declared or never been assigned a value
  - `typeof undefined` returns `undefined`
  - The following code displays “undefined” both times

```
// i is not declared anywhere in code
alert(typeof i);
var i;
alert(typeof i);
```

JavaScript: *null vs undef* (cont)

- `null == undefined` (true)

but…

- `null === undefined` (false)

User Interaction & Event Handling

- Basic User I/O
  - Until now, many of the example user interactions with scripts have been through ***annoying*** pop-ups…
    - Prompt dialog
    - Alert dialog
  - Dialogs are valid ways to receive input & display messages, but limited
    - Prompt dialog can obtain only one value at a time
    - Message dialog can display only one message
- Typical User I/O
  - Inputs are typically received via HTML form
  - Outputs are typically displayed in the web page (as HTML)

```
<head>
   <title>Online quiz</title>
   <script type="text/javascript">
   <!--

   function checkAnswers()
   {
      var myQuiz = document.getElementById("myQuiz");
      // determine if answer is correct
      if (myQuiz.elements[7].checked)
         alert("Congratulations, your answer is correct!");
      else
         alert("Your answer is incorrect. Please try again.");

   } // end function checkAnswers

   -->
   </script>
</head>
<body>
   <form id = "myQuiz" onsubmit="checkAnswers()" action="">
       <p>2013's Academy Award for Best Picture went to...<br />
          <input type="radio" name="radiobutton" value="AmerHustle">
          <label>American Hustle</label><br />

          <input type="radio" name="radiobutton" value="CaptainPhil">
          <label>Captain Phillips</label><br />

         <!-- .... -->

          <input type="radio" name="radiobutton" value="12Years">
          <label>12 Years a Slave</label><br />

          <input type="radio" name="radiobutton" value="Wolf">
          <label>The Wolf of Wall Street</label><br />

          <input type="submit" name="submit" value="Submit">
          <input type="reset" name="reset" value="Reset">
       </p>
   </form>
</body>
```

Accesses document element by id

Checks to see if 8th radio button is selected

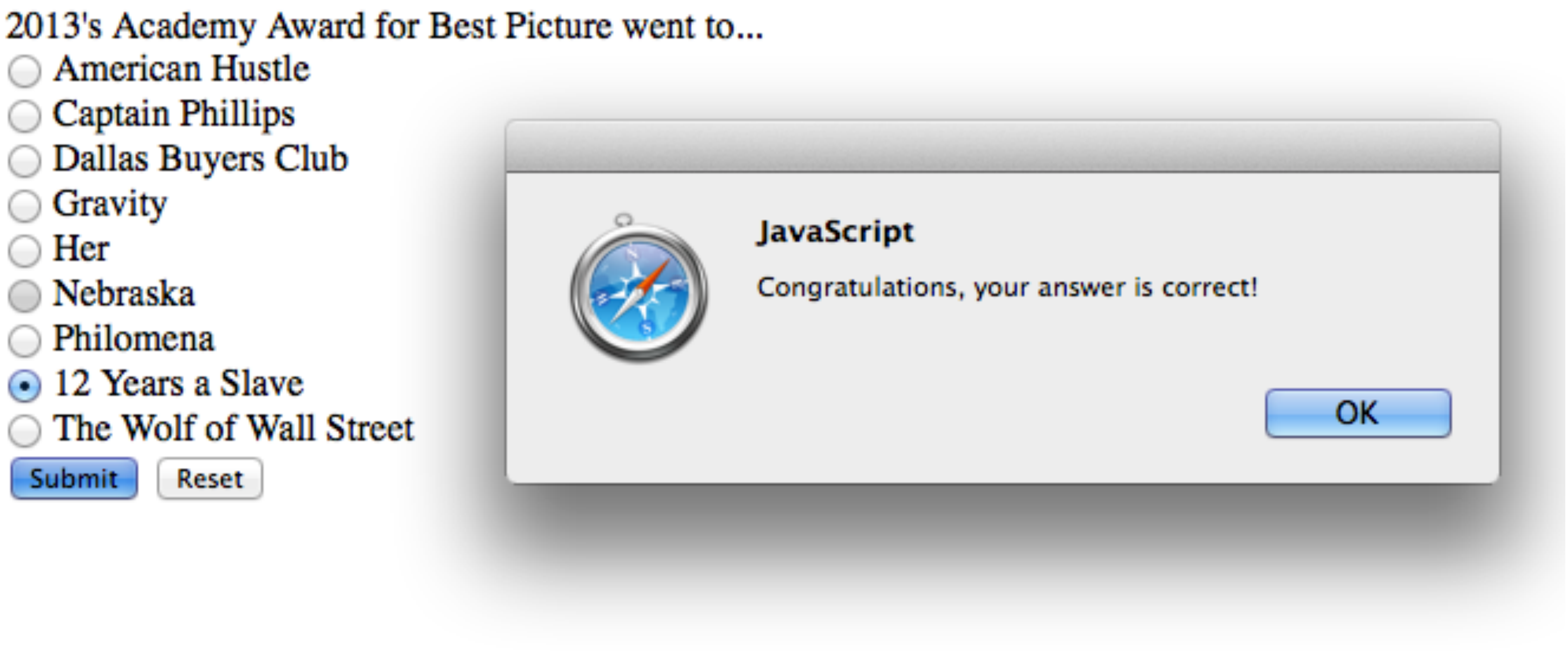
myQuiz.elements[0]

myQuiz.elements[1]

⋮

myQuiz.elements[7]

myQuiz.elements[8]



2013's Academy Award for Best Picture went to...

- ◯ American Hustle
- ◯ Captain Phillips
- ◯ Dallas Buyers Club
- ◯ Gravity
- ◯ Her
- ◯ Nebraska
- ◯ Philomena
- ● 12 Years a Slave
- ◯ The Wolf of Wall Street

JavaScript

Congratulations, your answer is correct!

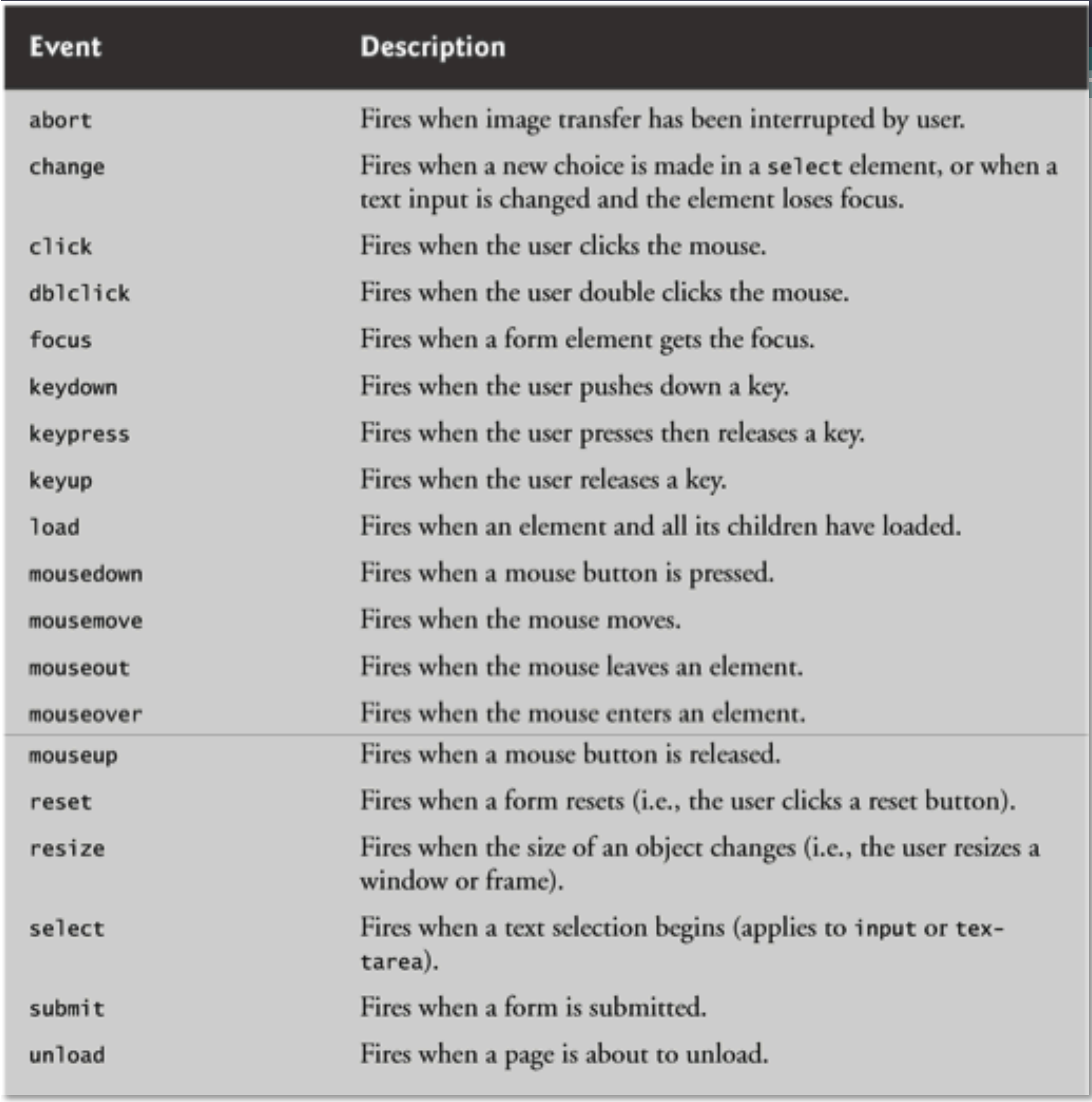
Event-Driven Programming

1. User interacts with an element in the web page
   - User's interaction with GUI "drives" the program
   - Example event: a button click

2. Script is notified of the event

3. Script processes the event
   - ***Event Handler***: the function called when an event occurs
     - When a GUI event occurs in a form, the browser calls the specified event-handling function
     - Before any event can be processed, each element must know which event-handling function will be called when a particular event occurs

Common Events



Event	Description
`abort`	Fires when image transfer has been interrupted by user.
`change`	Fires when a new choice is made in a `select` element, or when a text input is changed and the element loses focus.
`click`	Fires when the user clicks the mouse.
`dblclick`	Fires when the user double clicks the mouse.
`focus`	Fires when a form element gets the focus.
`keydown`	Fires when the user pushes down a key.
`keypress`	Fires when the user presses then releases a key.
`keyup`	Fires when the user releases a key.
`load`	Fires when an element and all its children have loaded.
`mousedown`	Fires when a mouse button is pressed.
`mousemove`	Fires when the mouse moves.
`mouseout`	Fires when the mouse leaves an element.
`mouseover`	Fires when the mouse enters an element.
`mouseup`	Fires when a mouse button is released.
`reset`	Fires when a form resets (i.e., the user clicks a reset button).
`resize`	Fires when the size of an object changes (i.e., the user resizes a window or frame).
`select`	Fires when a text selection begins (applies to `input` or `textarea`).
`submit`	Fires when a form is submitted.
`unload`	Fires when a page is about to unload.

More on Event Listeners

- Method `addEventListener` can be called multiple times on an element to register more than one event-handling method for an event
- It's also possible to remove an event listener by calling **`removeEventListener`** with the same arguments that you passed to `addEventListener` to register the event handler

Also out there in the wild…

- Two other models for registering event handlers
  - Inline model (original)
    - Treats events as attributes of HTML elements
      - <body onload="start()">
        - Still in widespread use
      - <form action="#" onsubmit="formSubmitted()">
      - <input type="button" onclick="buttonClicked()">
  - Traditional model
    - Same concept as inline model, but assigned to element properties in JavaScript code
    - document.onload="start()";
- The inline model places calls to JavaScript functions directly in HTML code
  - Many feel this is clunky and that HTML code need not contain JavaScript references
  - May cause maintenance issues
    - Event and its handler defined in different places

JavaScript Scoping Rules

- The part of a script in which a variable's name can be referenced is known as the variable's *scope*

- **Global variables** or **script-level variables** are accessible in any part of a script
  - Said to have **global scope**

- Identifiers declared inside a function have **function** (or **local**) **scope**
  - Can be used only in that function
  - Scope begins with the opening left brace (`{`) of the function and ends at the terminating right brace (`}`)
  - If local variable in a function has the same name as a global variable, the global variable is "hidden" from the body of the function

Scoping Example

```
<script>
   var output;
   var x = 1; // global variable

   function start()
   {
      var x = 5; // variable local to function start

      output = "<p>local x in start is " + x + "</p>";

      functionA(); // functionA has local x
      functionB(); // functionB uses global variable x
      functionA(); // functionA reinitializes local x
      functionB(); // global variable x retains its value

      output += "<p class='space'>local x in start is " + x +
         "</p>";
      document.getElementById( "results" ).innerHTML = output;
   } // end function start

   function functionA()
   {
      var x = 25; // initialized each time functionA is called

      output += "<p class='space'>local x in functionA is " + x +
         " after entering functionA</p>";
      ++x;
      output += "<p>local x in functionA is " + x +
         " before exiting functionA</p>";
   } // end functionA

   function functionB()
   {
      output += "<p class='space'>global variable x is " + x +
         " on entering functionB";
      x *= 10;
      output += "<p>global variable x is " + x +
         " on exiting functionB</p>";
   } // end functionB

   window.addEventListener( "load", start, false );
</script>
```

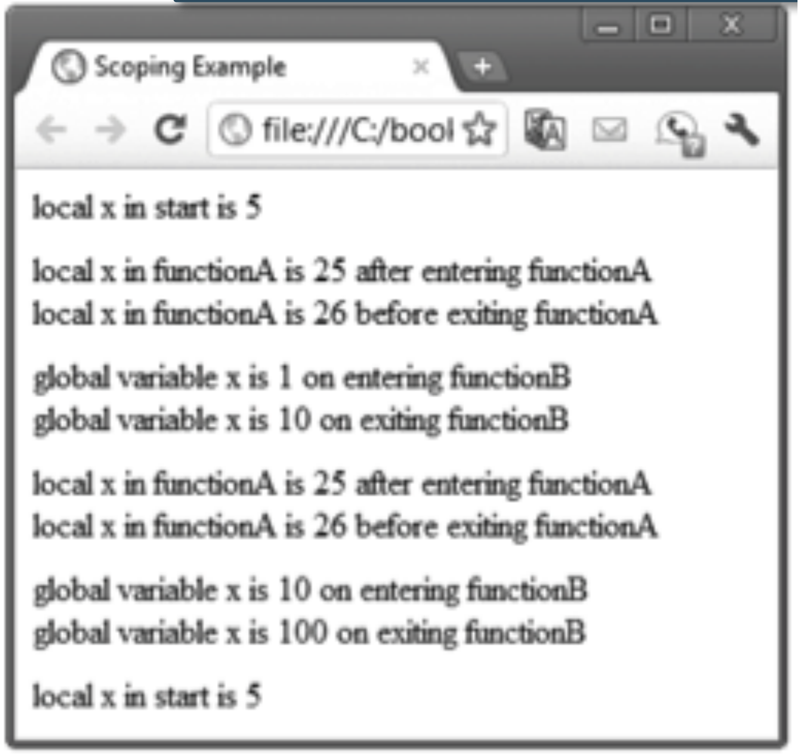
Calls function `start` when the body of the document has loaded into the browser window

```
<body>
   <div id = "results"></div>
</body>
```

Scoping Example

```
<script>
   var output;
   var x = 1; // global variable

   function start()
   {
      var x = 5; // variable local to function start

      output = "<p>local x in start is " + x + "</p>";

      functionA(); // functionA has local x
      functionB(); // functionB uses global variable x
      functionA(); // functionA reinitializes local x
      functionB(); // global variable x retains its value

      output += "<p class='space'>local x in start is " + x +
         "</p>";
      document.getElementById( "results" ).innerHTML = output;
   } // end function start

   function functionA()
   {
      var x = 25; // initialized each time functionA is called

      output += "<p class='space'>local x in functionA is " + x
         " after entering functionA</p>";
      ++x;
      output += "<p>local x in functionA is " + x +
         " before exiting functionA</p>";
   } // end functionA

   function functionB()
   {
      output += "<p class='space'>global variable x is " + x +
         " on entering functionB";
      x *= 10;
      output += "<p>global variable x is " + x +
         " on exiting functionB</p>";
   } // end functionB

   window.addEventListener( "load", start, false );
</script>
```

Calls function `start` when the body of the document has loaded into the browser window



```
local x in start is 5

local x in functionA is 25 after entering functionA
local x in functionA is 26 before exiting functionA

global variable x is 1 on entering functionB
global variable x is 10 on exiting functionB

local x in functionA is 25 after entering functionA
local x in functionA is 26 before exiting functionA

global variable x is 10 on entering functionB
global variable x is 100 on exiting functionB

local x in start is 5
```

Reviewing the `load` Event

- `window` object’s `load` event fires when the window finishes loading successfully
  - all its children are loaded
  - all external files referenced by the page are loaded
- *Every* DOM element has a `load` event
  - most commonly handled for the `window` object

String Object

- Characters are the building blocks of JavaScript programs (…and CSS rules, and HTML…)
  - Every program is composed of a sequence of characters

- A string is a series of characters treated as a single unit
  - May include letters, digits and various **special characters**, such as +, -, *, /, and $
  - JavaScript supports **Unicode**, which represents a large portion of the world's languages
  - **String literals** or **string constants** are written as a sequence of characters in double or single quotation marks

String example

```
// CharacterProcessing.js
function start()
{
    var s = "ZEBRA";
    var s2 = "AbCdEfG";
    var result = "";

    result = "<p>Character at index 0 in '" + s + "' is " +
       s.charAt( 0 ) + "</p>";
    result += "<p>Character code at index 0 in '" + s + "' is " +
       s.charCodeAt( 0 ) + "</p>";

    result += "<p>'" + String.fromCharCode( 87, 79, 82, 68 ) +
       "' contains character codes 87, 79, 82 and 68</p>";

    result += "<p>'" + s2 + "' in lowercase is '" +
       s2.toLowerCase() + "'</p>";
    result += "<p>'" + s2 + "' in uppercase is '" +
       s2.toUpperCase() + "'</p>";

    document.getElementById( "results" ).innerHTML = result;
} // end function start

window.addEventListener( "load", start, false );
```

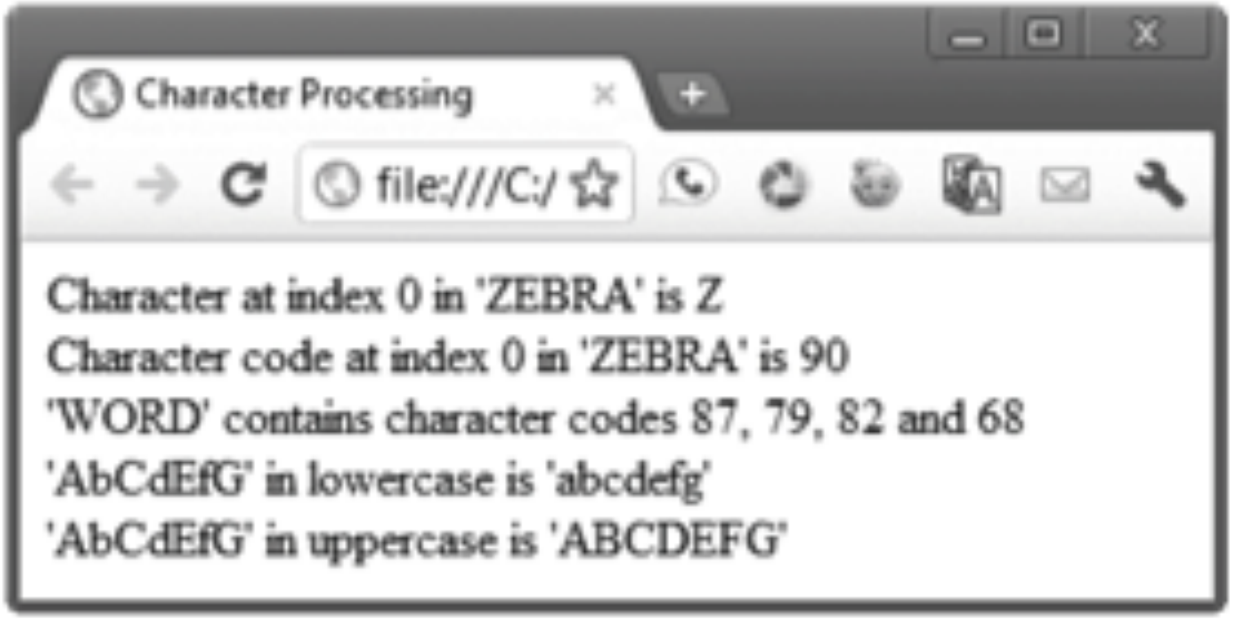
Returns char at index 0 of string s (a 'Z')

Returns the Unicode value of the character at index 0 of string s

Creates a string from the characters with the Unicode values 87, 79, 82 and 68

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset = "utf-8">
      <title>Character Processing</title>
      <link rel = "stylesheet" type = "text/css" href = "style.css">
      <script src = "CharacterProcessing.js"></script>
   </head>
   <body>
      <div id = "results"></div>
   </body>
</html>
```

String example (cont)



```
Character at index 0 in 'ZEBRA' is Z
Character code at index 0 in 'ZEBRA' is 90
'WORD' contains character codes 87, 79, 82 and 68
'AbCdEfG' in lowercase is 'abcdefg'
'AbCdEfG' in uppercase is 'ABCDEFG'
```

String Tokenization

- Breaking a string into tokens is called *tokenization*
  - Tokens are separated from one another by *delimiters*
    - Typically white-space characters (blank, tab, newline and carriage return)
    - Other characters may also be used as delimiters

- `String` method `split` breaks a string into its component tokens
  - Argument: the delimiter string
  - Returns: an array of strings containing the tokens

- `String` method `substring`
  - Returns the substring from the starting index (its first argument) up to but not including the ending index (its second argument)
    - If ending index is greater than the length of the string, return from the starting index to the end of the original string

```
// SplitAndSubString.js
function splitButtonPressed()
{
   var inputString = document.getElementById( "inputField" ).value;
   var tokens = inputString.split( " " );

   var results = document.getElementById( "results" );
   results.innerHTML = "<p>The sentence split into words is: </p>" +
      "<p class = 'indent'>" +
      tokens.join( "</p><p class = 'indent'>" ) + "</p>" +
      "<p>The first 10 characters of the input string are: </p>" +
      "<p class = 'indent'>'" + inputString.substring( 0, 10 ) + "'</p>";
} // end function splitButtonPressed

// register click event handler for searchButton
function start()
{
   var splitButton = document.getElementById( "splitButton" );
   splitButton.addEventListener( "click", splitButtonPressed, false );
} // end function start

window.addEventListener( "load", start, false );
```

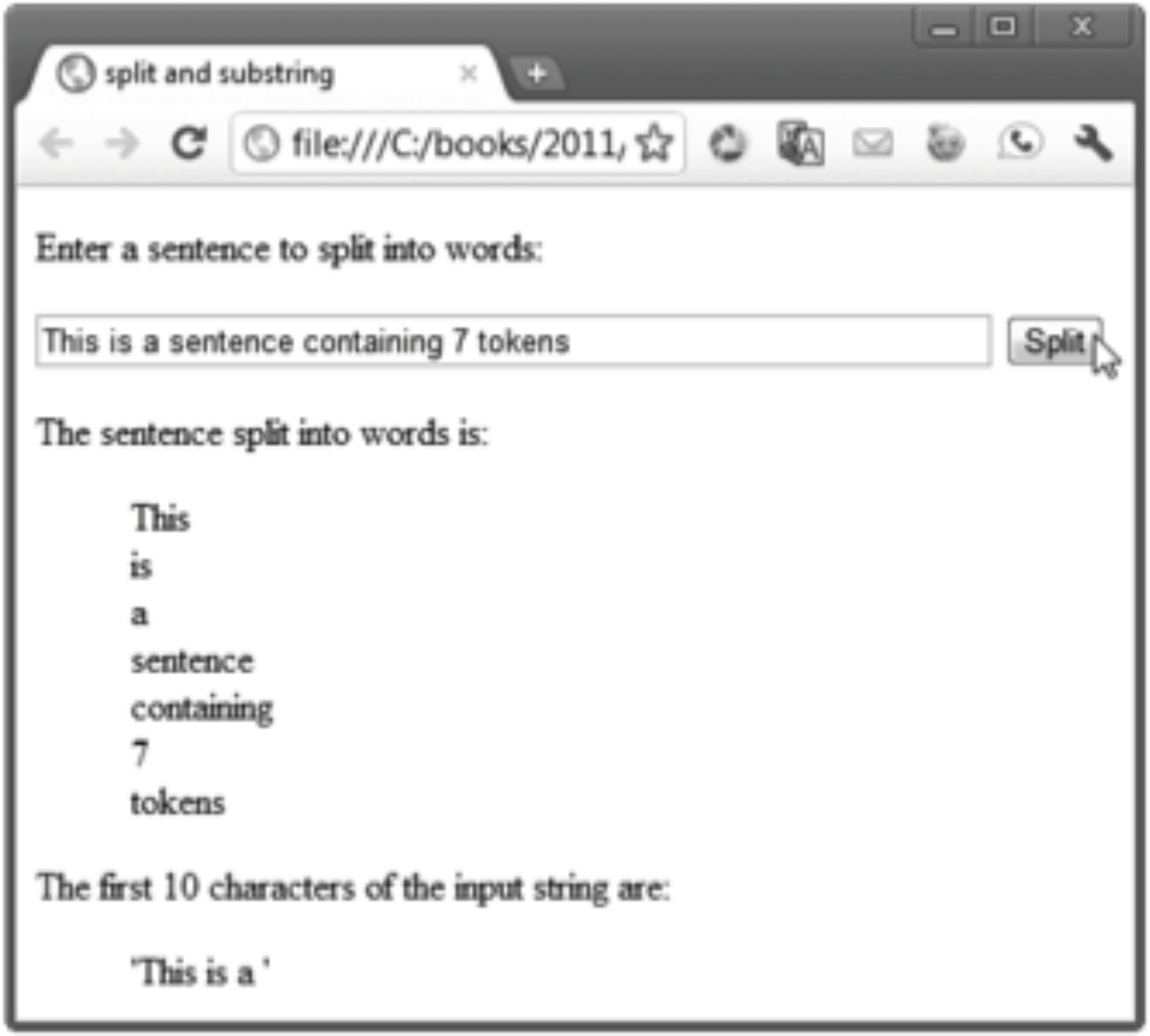
Splits inputString into new strings at each space and stores them in array tokens

Creates a string from elements of tokens, adding a new indented paragraph for each one

First 10 characters of inputString

```
<body>
   <form action = "#">
      <p>Enter a sentence to split into words:</p>
      <p><input id = "inputField" type = "text">
         <input id = "splitButton" type = "button" value = "Split"></p>
      <div id = "results"></p>
   </form>
</body>
```

String Tokenization - Output



```
Enter a sentence to split into words:

This is a sentence containing 7 tokens

The sentence split into words is:

        This
        is
        a
        sentence
        containing
        7
        tokens

The first 10 characters of the input string are:

        'This is a '
```

String Searching Methods

- method `indexOf`
  - Determines the location of the first occurrence of its argument in the string used to call the method
  - If the substring is found, the index at which the first occurrence of the substring begins is returned; otherwise, `-1` is returned
- method `lastIndexOf`
  - Determines the location of the last occurrence of its argument in the string used to call the method
  - If the substring is found, the index at which the last occurrence of the substring begins is returned; otherwise, `-1` is returned

- Both methods take an optional second argument specifying the index from which to begin the search

```
<body>
   <form id = "searchForm" action = "#">
      <h1>The string to search is:
         abcdefghijklmnopqrstuvwxyzabcdefghijklm</h1>
      <p>Enter the substring to search for
      <input id = "inputField" type = "search">
      <input id = "searchButton" type = "button" value = "Search"></p>
      <div id = "results"></div>
   </form>
</body>
```

```
var letters = "abcdefghijklmnopqrstuvwxyzabcdefghijklm";

function buttonPressed()
{
   var inputField = document.getElementById( "inputField" );

   document.getElementById( "results" ).innerHTML =
      "<p>First occurrence is located at index " +
         letters.indexOf( inputField.value ) + "</p>" +
      "<p>Last occurrence is located at index " +
         letters.lastIndexOf( inputField.value ) + "</p>" +
      "<p>First occurrence from index 12 is located at index " +
         letters.indexOf( inputField.value, 12 ) + "</p>" +
      "<p>Last occurrence from index 12 is located at index " +
         letters.lastIndexOf( inputField.value, 12 ) + "</p>";
} // end function buttonPressed

// register click event handler for searchButton
function start()
{
   var searchButton = document.getElementById( "searchButton" );
   searchButton.addEventListener( "click", buttonPressed, false );
} // end function start

window.addEventListener( "load", start, false );
```

Executes when searchButton is pressed

```
<body>
   <form id = "searchForm" action = "#">
      <h1>The string to search is:
         abcdefghijklmnopqrstuvwxyzabcdefghijklm</h1>
      <p>Enter the substring to search for
      <input id = "inputField" type = "search">
      <input id = "searchButton" type = "button" value = "Search"></p>
      <div id = "results"></div>
```

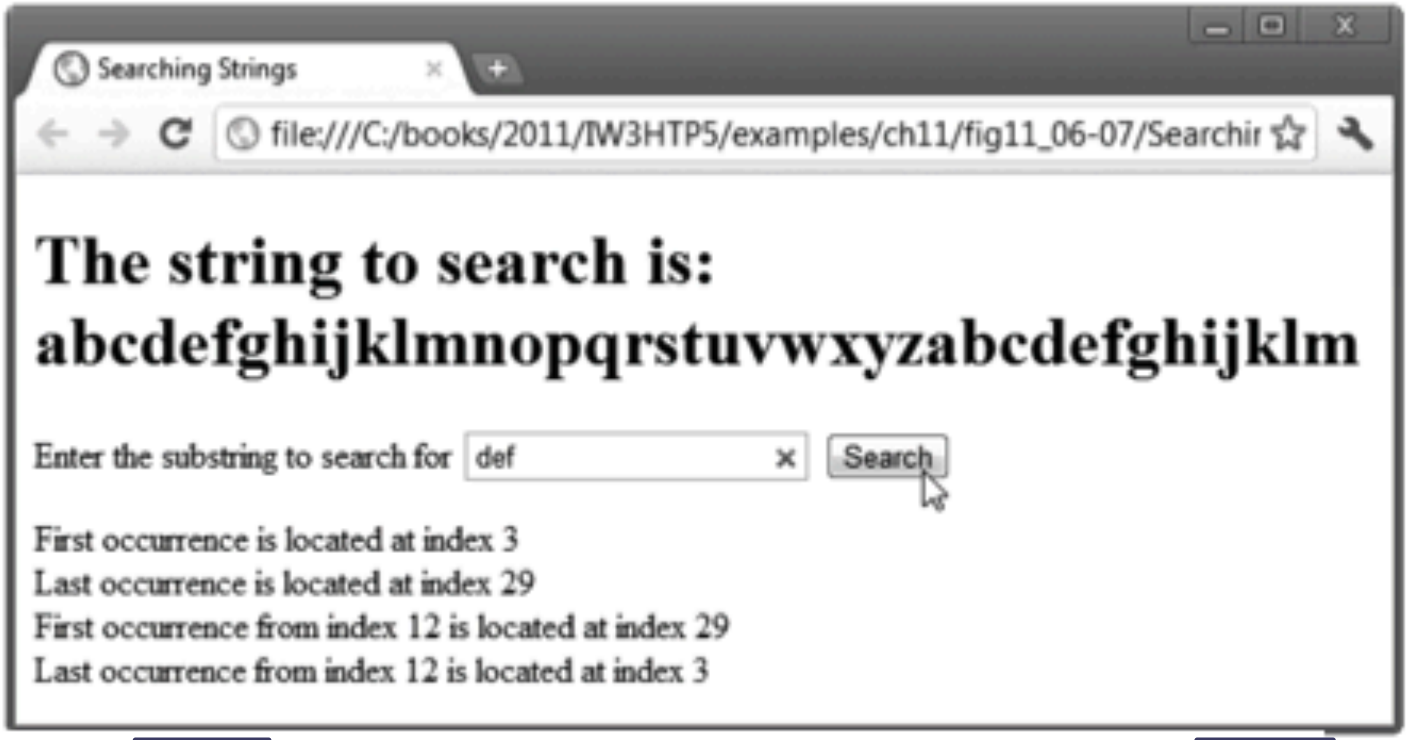
Searches `letters` for the first occurrence of the text in `inputField`, and returns its index

Searches `letters` for the ***LAST*** occurrence of the text in `inputField`, and returns its index

Searches `letters` for the first occurrence of the text in `inputField`, starting at char 12 (13th char)

Searches `letters` for the last occurrence of the text in `inputField`, backwards from char 12 (13th char)

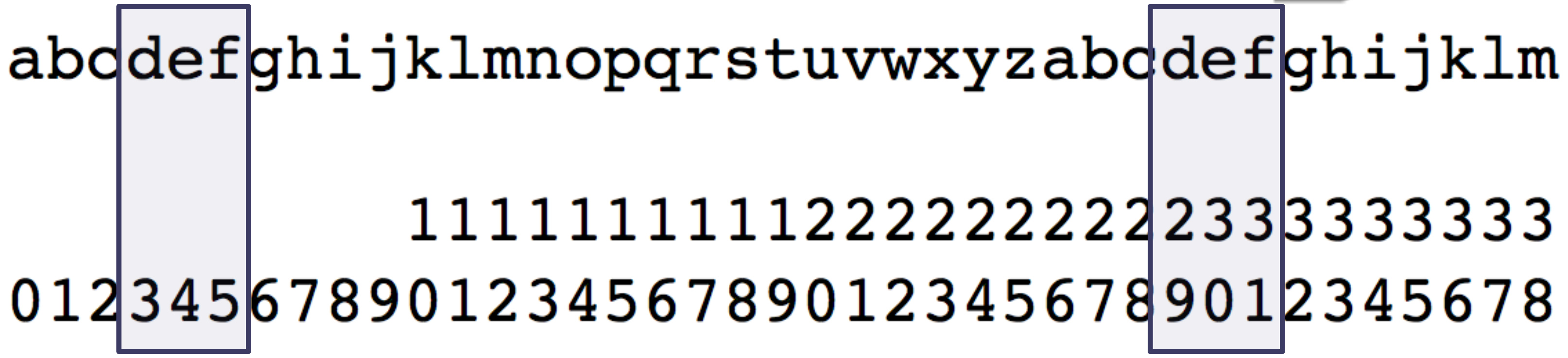
```
var letters = "abcdefghijklmnopqrstuvwxyzabcdefghijklm";

function buttonPressed()
{
   var inputField = document.getElementById( "inputField" );

   document.getElementById( "results" ).innerHTML =
      "<p>First occurrence is located at index " +
      letters.indexOf( inputField.value ) + "</p>" +
      "<p>Last occurrence is located at index " +
      letters.lastIndexOf( inputField.value ) + "</p>" +
      "<p>First occurrence from index 12 is located at index " +
      letters.indexOf( inputField.value, 12 ) + "</p>" +
      "<p>Last occurrence from index 12 is located at index " +
      letters.lastIndexOf( inputField.value, 12 ) + "</p>";
} // end function buttonPressed

// register click event handler for searchButton
function start()
{
   var searchButton = document.getElementById( "searchButton" );
   searchButton.addEventListener( "click", buttonPressed, false );
} // end function start

window.addEventListener( "load", start, false );
```


Searching Strings
file:///C:/books/2011/IW3HTP5/examples/ch11/fig11_06-07/Searchir
Search

The string to search is: abcdefghijklmnopqrstuvwxyzabcdefghijklm

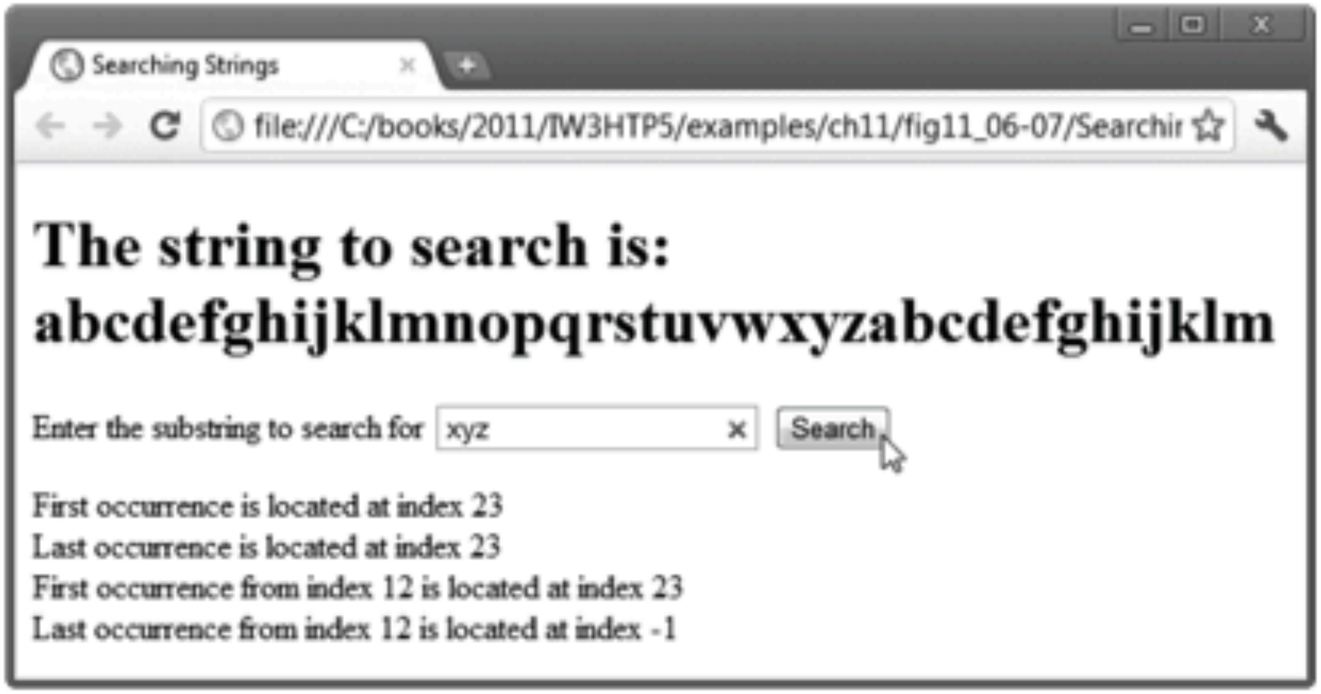
Enter the substring to search for def

First occurrence is located at index 3
Last occurrence is located at index 29
First occurrence from index 12 is located at index 29
Last occurrence from index 12 is located at index 3



```
abcdefghijklmnopqrstuvwxyzabcdefghijklm

          1111111111222222222233333333
012345678901234567890123456789012345678
```

```
abcdefghijklmnopqrstuvwxyzabcdefghijklm

          1111111111222222222233333333
0123456789012345678901234567890123456789
```



The string to search is: abcdefghijklmnopqrstuvwxyzabcdefghijklm

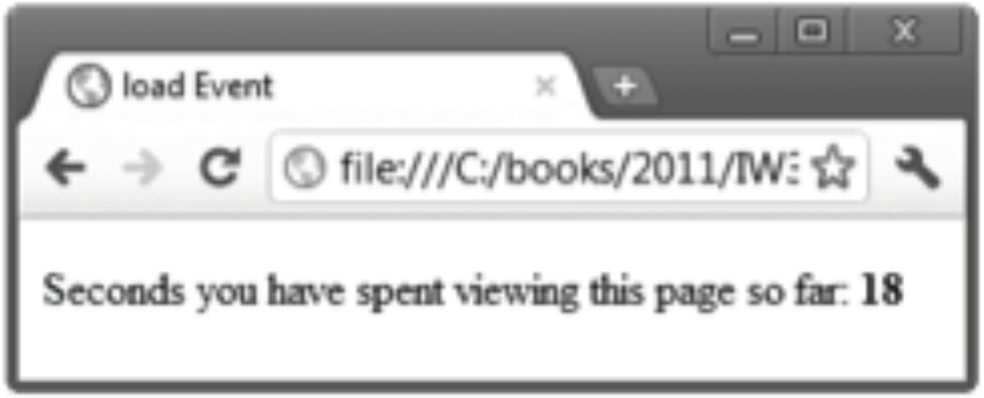
Enter the substring to search for xyz

```
First occurrence is located at index 23
Last occurrence is located at index 23
First occurrence from index 12 is located at index 23
Last occurrence from index 12 is located at index -1
```

Searching Strings
file:///C:/books/2011/IW3HTP5/examples/ch11/fig11_06-07/Searchir
Search

Window Events (example)

```
<html>
   <head>
      <meta charset = "utf-8">
      <title>load Event</title>
      <link rel = "stylesheet" type = "text/css" href = "style.css">
      <script src = "load.js"></script>
   </head>
   <body>
      <p>Seconds you have spent viewing this page so far:
      <span id = "soFar">0</span></p>
   </body>
</html>
```



Seconds you have spent viewing this page so far: 18

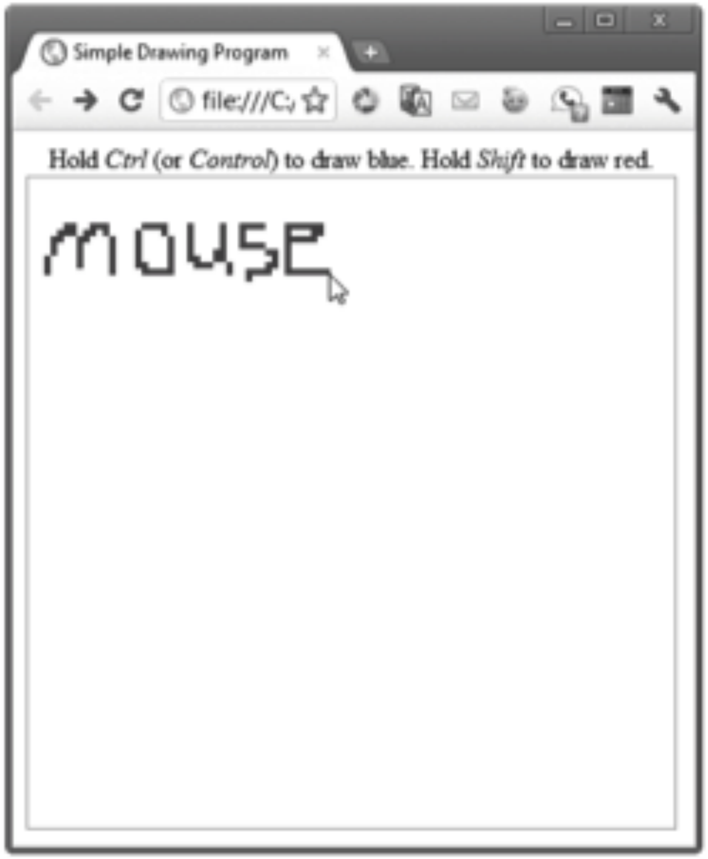
```
// load.js
var seconds = 0;

// called when the page loads to begin the timer
function startTimer()
{
   window.setInterval( "updateTime()", 1000 );
} // end function startTimer

// called every 1000 ms to update the timer
function updateTime()
{
   ++seconds;
   document.getElementById( "soFar" ).innerHTML = seconds;
} // end function updateTime

window.addEventListener( "load", startTimer, false );
```

The `load` event's handler creates an interval timer that updates a `span` with the number of seconds that have elapsed since the document was loaded.

mouseMove example

```
<html>
   <head>
      <meta charset="utf-8">
      <title>Simple Drawing Program</title>
      <link rel = "stylesheet" type = "text/css" href = "style.css">
      <script src = "draw.js"></script>
   </head>
   <body>
      <table id = "canvas">
         <caption>Hold <em>Ctrl</em> (or <em>Control</em>) to draw blue.
            Hold <em>Shift</em> to draw red.</caption>
         <tbody id = "tablebody"></tbody>
      </table>
   </body>
</html>
```

a) User holds the *Shift* key and moves the mouse to draw in red.

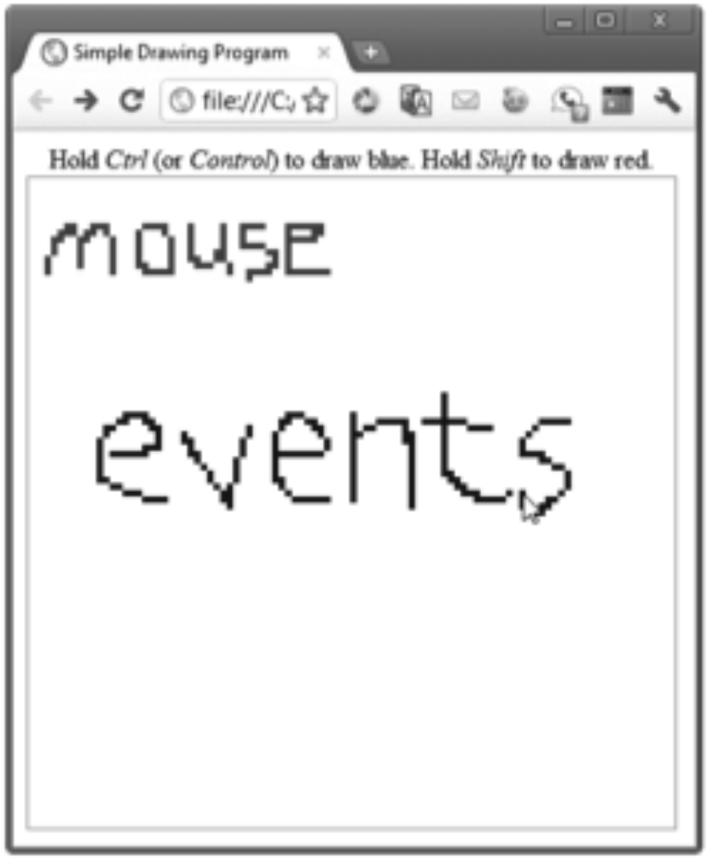


Hold *Ctrl* (or *Control*) to draw blue. Hold *Shift* to draw red.

mouseMove example

```
<html>
   <head>
      <meta charset="utf-8">
      <title>Simple Drawing Program</title>
      <link rel = "stylesheet" type = "text/css" href = "style.css">
      <script src = "draw.js"></script>
   </head>
   <body>
      <table id = "canvas">
         <caption>Hold <em>Ctrl</em> (or <em>Control</em>) to draw blue.
            Hold <em>Shift</em> to draw red.</caption>
         <tbody id = "tablebody"></tbody>
      </table>
   </body>
</html>
```

b) User holds the *Ctrl* key and moves the mouse to draw in blue.



Hold *Ctrl* (or *Control*) to draw blue. Hold *Shift* to draw red.

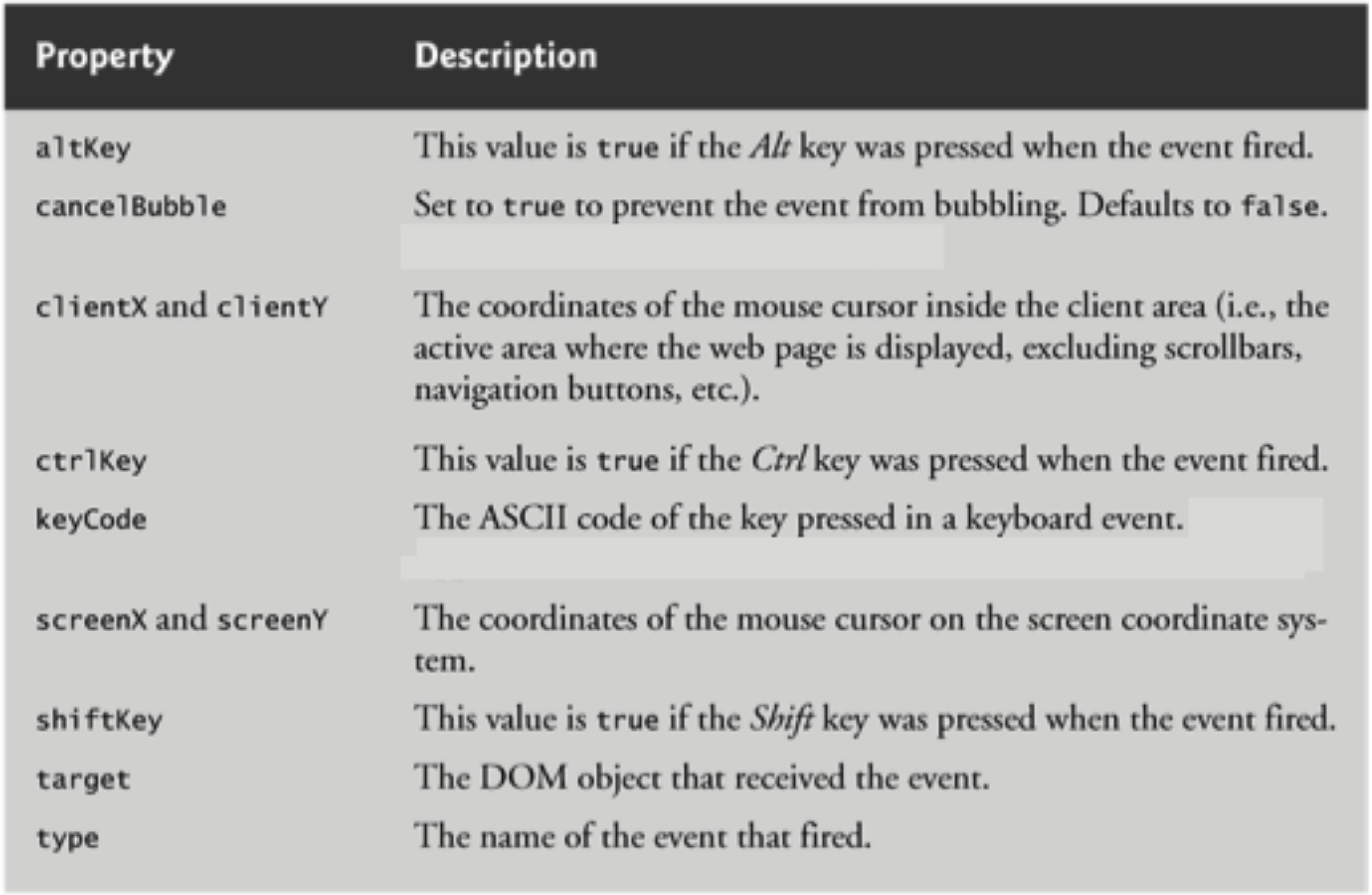
mouseMove example:

Drawing in the table body

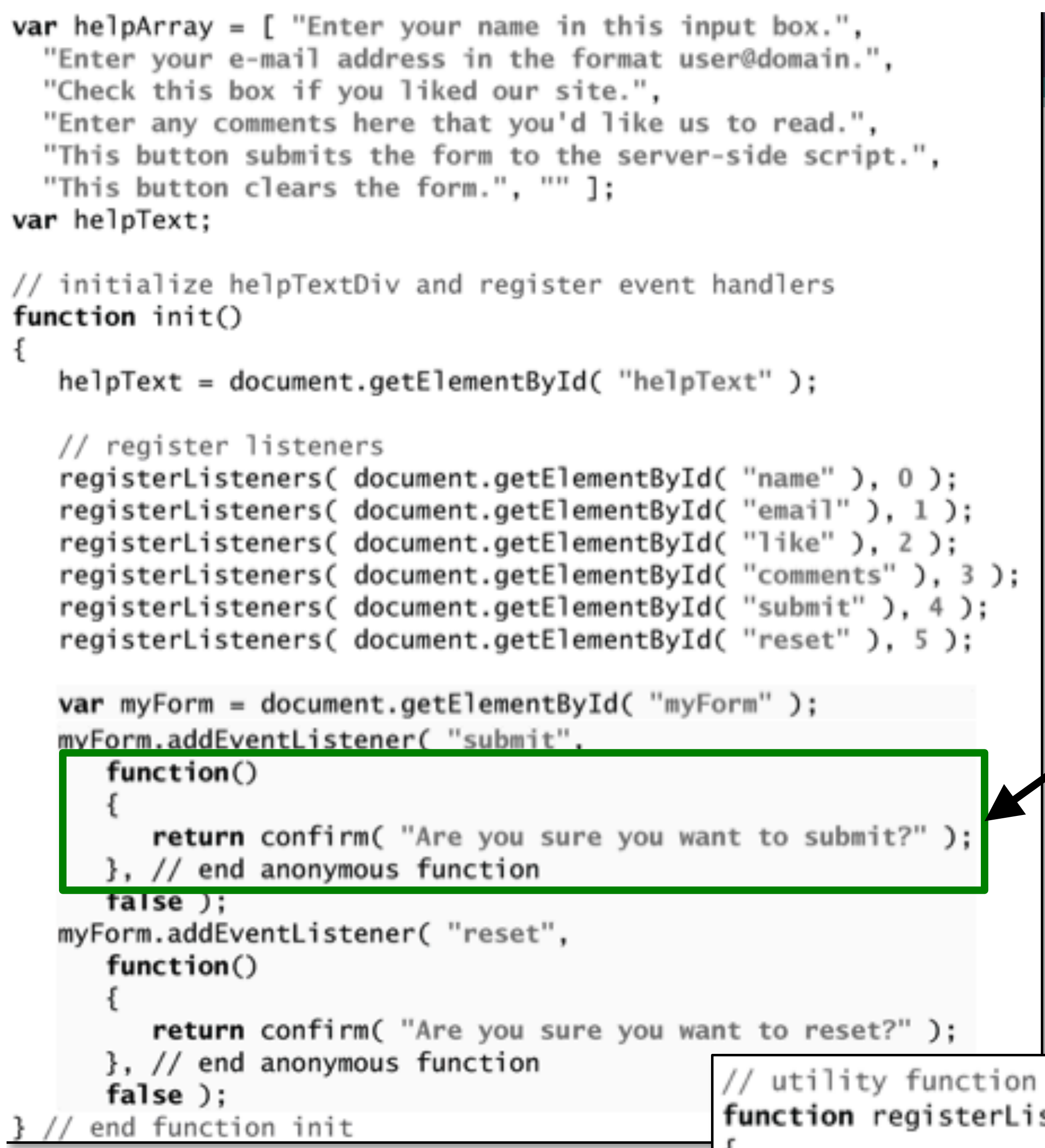
```
// draw.js
// A simple drawing program.
// initialization function to insert cells into the table
function createCanvas()
{
   var side = 100;
   var tbody = document.getElementById( "tablebody" );

   for ( var i = 0; i < side; ++i )
   {
      var row = document.createElement( "tr" );

      for ( var j = 0; j < side; ++j )
      {
         var cell = document.createElement( "td" );
         row.appendChild( cell );
      } // end for

      tbody.appendChild( row );
   } // end for

   // register mousemove listener for the table
   document.getElementById( "canvas" ).addEventListener(
      "mousemove", processMouseMove, false );
} // end function createCanvas
```

```
// processes the onmousemove event
function processMouseMove( e )
{
   if ( e.target.tagName.toLowerCase() == "td" )
   {
      // turn the cell blue if the Ctrl key is pressed
      if ( e.ctrlKey )
      {
         e.target.setAttribute( "class", "blue" );
      } // end if

      // turn the cell red if the Shift key is pressed
      if ( e.shiftKey )
      {
         e.target.setAttribute( "class", "red" );
      } // end if
   } // end if
} // end function processMouseMove

window.addEventListener( "load", createCanvas, false );
```

Some Event Object Properties



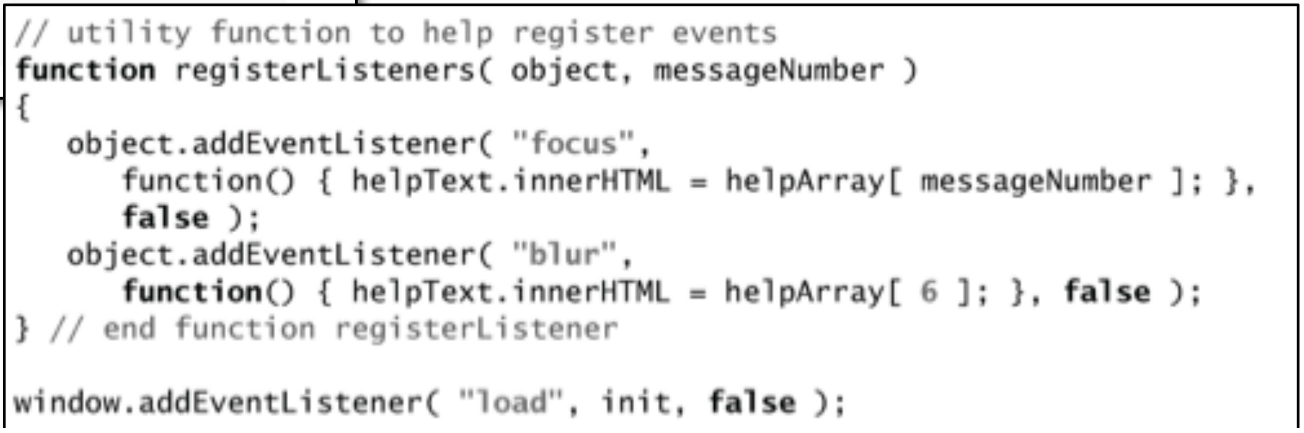
Property	Description
`altKey`	This value is `true` if the *Alt* key was pressed when the event fired.
`cancelBubble`	Set to `true` to prevent the event from bubbling. Defaults to `false`.
`clientX` and `clientY`	The coordinates of the mouse cursor inside the client area (i.e., the active area where the web page is displayed, excluding scrollbars, navigation buttons, etc.).
`ctrlKey`	This value is `true` if the *Ctrl* key was pressed when the event fired.
`keyCode`	The ASCII code of the key pressed in a keyboard event.
`screenX` and `screenY`	The coordinates of the mouse cursor on the screen coordinate system.
`shiftKey`	This value is `true` if the *Shift* key was pressed when the event fired.
`target`	The DOM object that received the event.
`type`	The name of the event that fired.



```
var helpArray = [ "Enter your name in this input box.",
  "Enter your e-mail address in the format user@domain.",
  "Check this box if you liked our site.",
  "Enter any comments here that you'd like us to read.",
  "This button submits the form to the server-side script.",
  "This button clears the form.", "" ];
var helpText;

// initialize helpTextDiv and register event handlers
function init()
{
   helpText = document.getElementById( "helpText" );

   // register listeners
   registerListeners( document.getElementById( "name" ), 0 );
   registerListeners( document.getElementById( "email" ), 1 );
   registerListeners( document.getElementById( "like" ), 2 );
   registerListeners( document.getElementById( "comments" ), 3 );
   registerListeners( document.getElementById( "submit" ), 4 );
   registerListeners( document.getElementById( "reset" ), 5 );

   var myForm = document.getElementById( "myForm" );
   myForm.addEventListener( "submit",
      function()
      {
         return confirm( "Are you sure you want to submit?" );
      }, // end anonymous function
      false );
   myForm.addEventListener( "reset",
      function()
      {
         return confirm( "Are you sure you want to reset?" );
      }, // end anonymous function
      false );
} // end function init
```

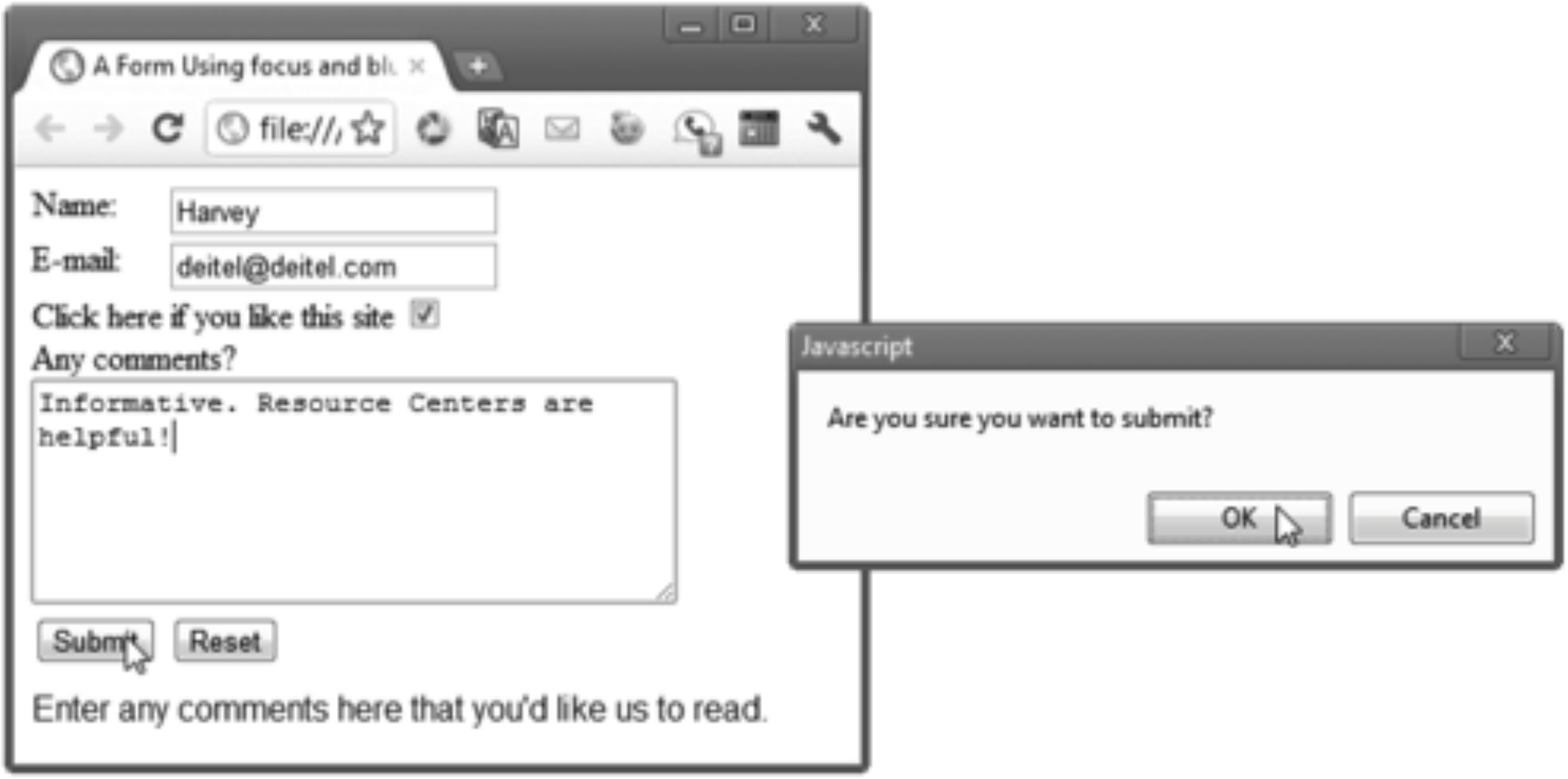
The anonymous function executes in response to the user's submitting the form by clicking the **Submit** button or pressing the *Enter* key.

`confirm(…)` asks the user a question (the argument) & presents an OK and Cancel button
If OK clicked, confirm(…) returns true,
Else confirm(…) returns false



```
// utility function to help register events
function registerListeners( object, messageNumber )
{
   object.addEventListener( "focus",
      function() { helpText.innerHTML = helpArray[ messageNumber ]; },
      false );
   object.addEventListener( "blur",
      function() { helpText.innerHTML = helpArray[ 6 ]; }, false );
} // end function registerListener

window.addEventListener( "load", init, false );
```

If an event handler returns false, the event's default action is not taken
Can also use `preventDefault()` in the Event object to suppress default behavior



Name: Harvey

E-mail: deitel@deitel.com

Click here if you like this site ✓

Any comments?

```
Informative. Resource Centers are
helpful!
```

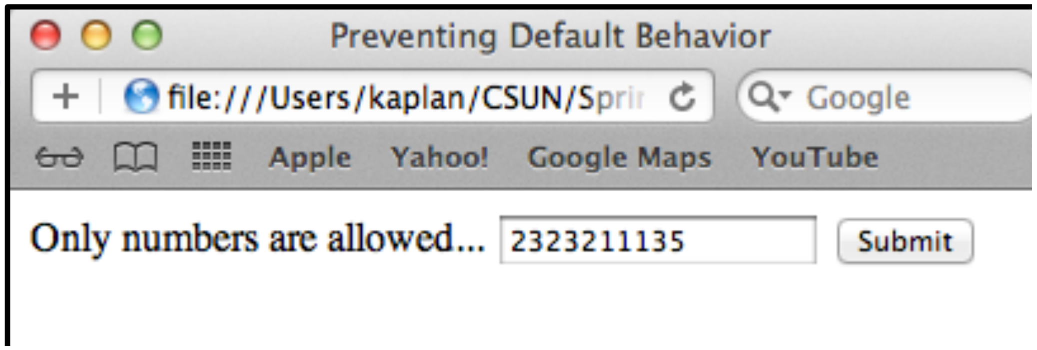
Enter any comments here that you'd like us to read.

Are you sure you want to submit?

```
<!DOCTYPE html>
<html>
<head>
  <title>Preventing Default Behavior</title>
  <script>
      var numberinput;
      function keyHandler( e )
      {
          var keycode = e.keyCode;
          var keychar = String.fromCharCode(keycode);
          if (keychar < '0' || keychar > '9')
          {
              e.preventDefault();
          }
      }

      function submitHandler()
      {
          alert("You entered " + numberinput.value + "\n");
      }

      function init ()
      {
          numberinput = document.getElementById("numbersonly");
          numberinput.addEventListener("keypress", keyHandler, false);
          var form = document.getElementById("theform");
          form.addEventListener("submit", submitHandler, false);
      }

      window.addEventListener("load", init, false);
  </script>
</head>
<body>
    <form id="theform" action="#">
        <label>Only numbers are allowed...</label>
        <input id="numbersonly" type="text">
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

Every time a key is pressed, check to see that its char is between ‘0’ and ‘9’

– if not, preventDefault(), which for this event will prevent key from affecting text field



Only numbers are allowed... 2323211135

Event Bubbling

- ***Event bubbling***
  - The process whereby events fired on *child* elements "bubble" up to their *parent* elements and ancestors
    - This happens when parent and child both handle same event…which handler gets called
  - When an event is fired on an element, it is first delivered to the element's event handler (if any), then to the parent element's event handler (if any)
    - If using ***event capture*** --- addEventListener(… , … , **true**)
      - Goes the other way…parent first, then child
      - Hardly ever used!
- *If you intend to handle an event in a child element alone, you should cancel the bubbling of the event in the child's event-handler by using the* `cancelBubble` *property of the event object*

```
function doSomething(e)
{
  if (!e) var e = window.event;  // handle event
  e.cancelBubble = true;         // cancel the bubbling
}
```

Displaying Random Images

```
<script>
   // variables used to interact with the i mg elements
   var die1Image;
   var die2Image;
   var die3Image;
   var die4Image;

   // register button listener and get the img elements
   function start()
   {
      var button = document.getElementById( "rollButton" );
      button.addEventListener( "click", rollDice, false  );
      die1Image = document.getElementById( "die1" );
      die2Image = document.getElementById( "die2" );
      die3Image = document.getElementById( "die3" );
      die4Image = document.getElementById( "die4" );
   } // end function rollDice

   // roll the dice
   function rollDice()
   {
      setImage( die1Image );
      setImage( die2Image );
      setImage( die3Image );
      setImage( die4Image );
   } // end function rollDice

   // set image source for a die
   function setImage( dieImg )
   {
      var dieValue = Math.floor( 1 + Math.random() * 6 );
      dieImg.setAttribute( "src", "die" + dieValue + ".png" );
      dieImg.setAttribute( "alt",
         "die image with " + dieValue + " spot(s)" );
   } // end function setImage

   window.addEventListener( "load", start, false );
</script>
```

Each variable is assigned an object from the HTML document

Math.random() returns a float in the range [0, 1)
Thus, dieValue assigned an integer between 1 and 6

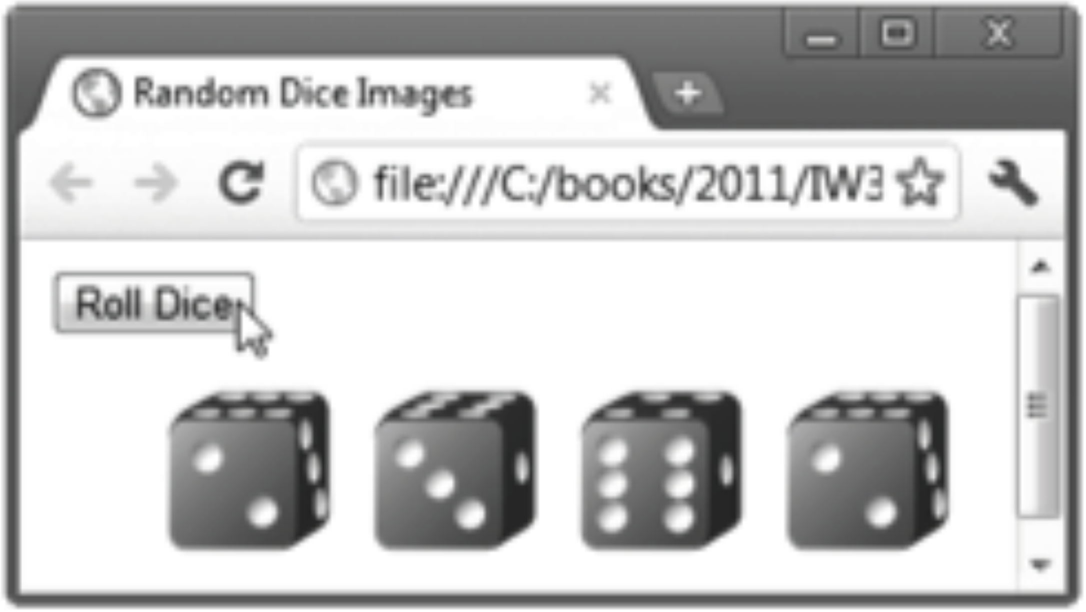
Then, the passed-in image is assigned a src from die1.png to die6.png

Displaying Random Images (part 2)

```
<body>
   <form action = "#">
      <input id = "rollButton" type = "button" value = "Roll Dice">
   </form>
   <ol>
      <li><img id = "die1" src = "blank.png" alt = "die 1 image"></li>
      <li><img id = "die2" src = "blank.png" alt = "die 2 image"></li>
      <li><img id = "die3" src = "blank.png" alt = "die 3 image"></li>
      <li><img id = "die4" src = "blank.png" alt = "die 4 image"></li>
   </ol>
</body>
```

# symbol by itself represents the current page

The four img elements will display the randomly selected dice
Initially, they display blank.png (empty white image) when page first rendered


Random Dice Images
file:///C:/books/2011/IW3
Roll Dice