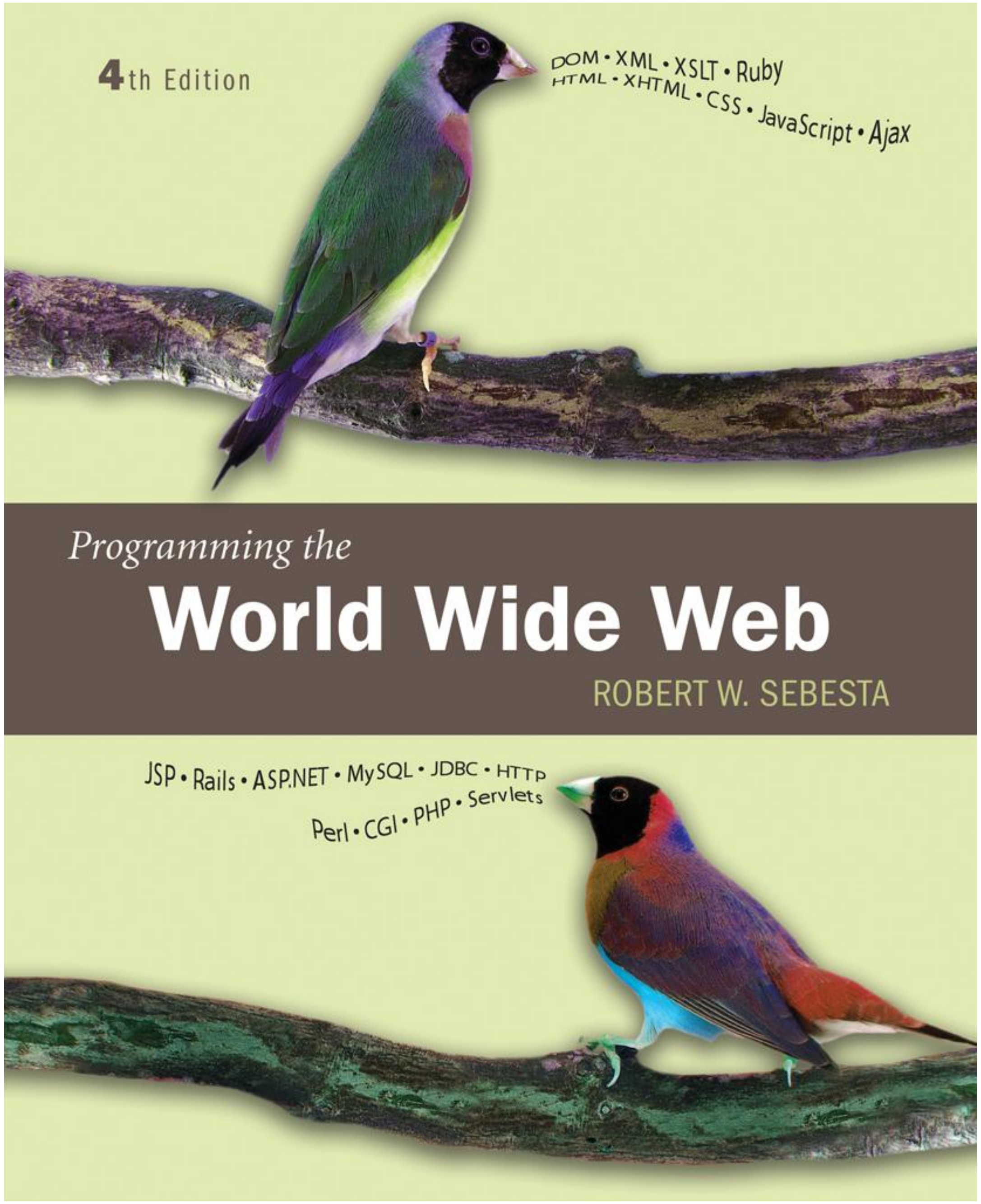

4th Edition
DOM • XML • XSLT • Ruby
HTML • XHTML • CSS • JavaScript • Ajax
Programming the
World Wide Web
ROBERT W. SEBESTA
JSP • Rails • ASP.NET • MySQL • JDBC • HTTP
Perl • CGI • PHP • Servlets

Chapter 4 Basics of JavaScript

JavaScript/EcmaScript References

- **The official EcmaScript, third edition, specification**
  - http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf
- **A working version of the 4th edition of the EcmaScript language specification**
  - http://www.carsoncheng.com/Ecma/tc39-tg1-2006-001.pdf
- **JavaScript as implemented in Mozilla products**
  - http://developer.mozilla.org/en/docs/JavaScript
- **Edition details can be found at the following site**
  - http://en.wikipedia.org/wiki/ECMAScript#cite_note-jsmoz-12
- **Details on how the browser and the engines work is given in the following site**
  - http://www.quora.com/JavaScript/How-does-a-JavaScript-engine-work#

4.1 Overview of JavaScript: Origins

- Originally developed by Netscape as LiveScript
- Joint Development with Sun Microsystems in 1995
- Standard 262 (ECMA-262) of the European Computer Manufacturers Association
- Supported by Netscape, Mozilla, Internet Explorer

Browsers, JavaScript and ECMA editions

Application	Latest version	ECMAScript edition
Google Chrome, the V8 engine	JavaScript	ECMA-262, edition 5
Mozilla Firefox, the Gecko layout engine, SpiderMonkey, and Rhino	JavaScript 1.8.5	ECMA-262, edition 5
Safari, the Nitro engine	JavaScript	ECMA-262, edition 5.1

4.1 JavaScript Components

- Core
  - The heart of the language
- Client-side
  - Library of objects supporting browser control and user interaction
- Server-side
  - Library of objects that support use in web servers
- Text focuses on Client-side

4.1 Java and JavaScript

- **Differences**
  - **JavaScript has a different object model from Java**
  - **JavaScript is not strongly typed**
  - **Objects are dynamic in JavaScript**

- **Java 1.6 has support for scripting**
  - **http://java.sun.com/javase/6/docs/technotes/guides/scripting/index.html**

- ***SpiderMonkey* is the code name for the first-ever JavaScript engine, written by Brendan Eich at Netscape Communications**

- **Mozilla Rhino is an implementation of JavaScript in Java**
  - **http://www.mozilla.org/rhino/**

4.1 Uses of JavaScript

- Provides alternative to server-side programming
    - Servers are often overloaded
    - Client processing has quicker reaction time
- JavaScript can work with forms
- JavaScript can access and modify the CSS properties and content of any element of a displayed XHTML document
- JavaScript is used to provide more complex user interface than plain forms with HTML/CSS can provide
- JavaScript can be used as a replacement to Java applets

4.1 Event-Driven Computation

- Users actions, such as mouse clicks and key presses, are referred to as *events*
- The main task of most JavaScript programs is to respond to events
- For example, a JavaScript program could validate data in a form before it is submitted to a server
  - *Caution:* It is important that crucial validation be done by the server. It is relatively easy to bypass client-side controls
  - For example, a user might create a copy of a web page but remove all the validation code.

4.1 XHTML/JavaScript Documents

- When JavaScript is embedded in an XHTML document, the browser must interpret it
- Two locations for JavaScript server different purposes
  - JavaScript in the head element will react to user input and be called from other locations
  - JavaScript in the body element will be executed once as the page is loaded
- Various strategies must be used to ‘protect’ the JavaScript from the browser
  - For example, comparisons present a problem since < and > are used to mark tags in XHTML
  - JavaScript code can be enclosed in XHTML comments

4.2 Object Orientation and JavaScript

- **JavaScript does not have classes**
- **JavaScript is *object-based***
  - **JavaScript defines objects that encapsulate both data and processing**
  - **However, JavaScript does not have true inheritance**
- **JavaScript provides *prototype-based inheritance***
- ***No polymorphism***

4.2 JavaScript Objects

- Objects are collections of *properties*
- Properties are either *data properties* or *method properties*
- Data properties are either primitive values or references to other objects
- Primitive values are often implemented directly in hardware (nonobject types)
- The root object in JavaScript is `object`, is the ancestor of all objects in a program
  - `object` has no data properties, but several method properties
- Object appears both internally or externally as the collection of property/value pairs
- Collection of properties of a JavaScript is dynamic

4.3 JavaScript in XHTML

- **Directly embedded**

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

- **Indirect reference**

```
<script type="text/javascript" src="tst_number.js"/>
```

  - **This is the preferred approach**

4.3 General Syntactic Characteristics

- JavaScript can appear directly as the content of a **`<script>`** tag
- The `type` attribute must be set to “`text/javascript`”
- It can be indirectly embedded in XHTML document using the `src` attribute of <`script>`
  - Example

    ```
    <script type="“text/javascript”" src=“test_num.js”>
    </script>
    ```
- And requires the closing tag always

4.3 General Syntactic Characteristics contd…

- Identifiers
  - Start with $, _, letter
  - Continue with $, _, letter or digit
  - Case sensitive
- Reserved words are 25 example: break, case catch, throw etc…
- Another collection of words is reserved for future use in JavaScript--- Can be found in ECMA web site.
- Comments
  - //
  - /* … */

4.3 Statement Syntax

- Statements can be terminated with a semicolon
- However, the interpreter will insert the semicolon if missing at the end of a line and the statement seems to be complete
- Can be a problem:

```
return
x;
```

- If a statement must be continued to a new line, make sure that the first line does not make a complete statement by itself
- Example hello.html

4.4 Primitive Types

- Five primitive types
  - Number
  - String
  - Boolean
  - Undefined
  - Null
- There are five classes corresponding to the five primitive types
  - Wrapper objects for primitive values
  - Place for methods and properties relevant to the primitive types
  - Primitive values are *coerced* to the wrapper class as necessary, and vice-versa

4.4 Primitive and Object Storage

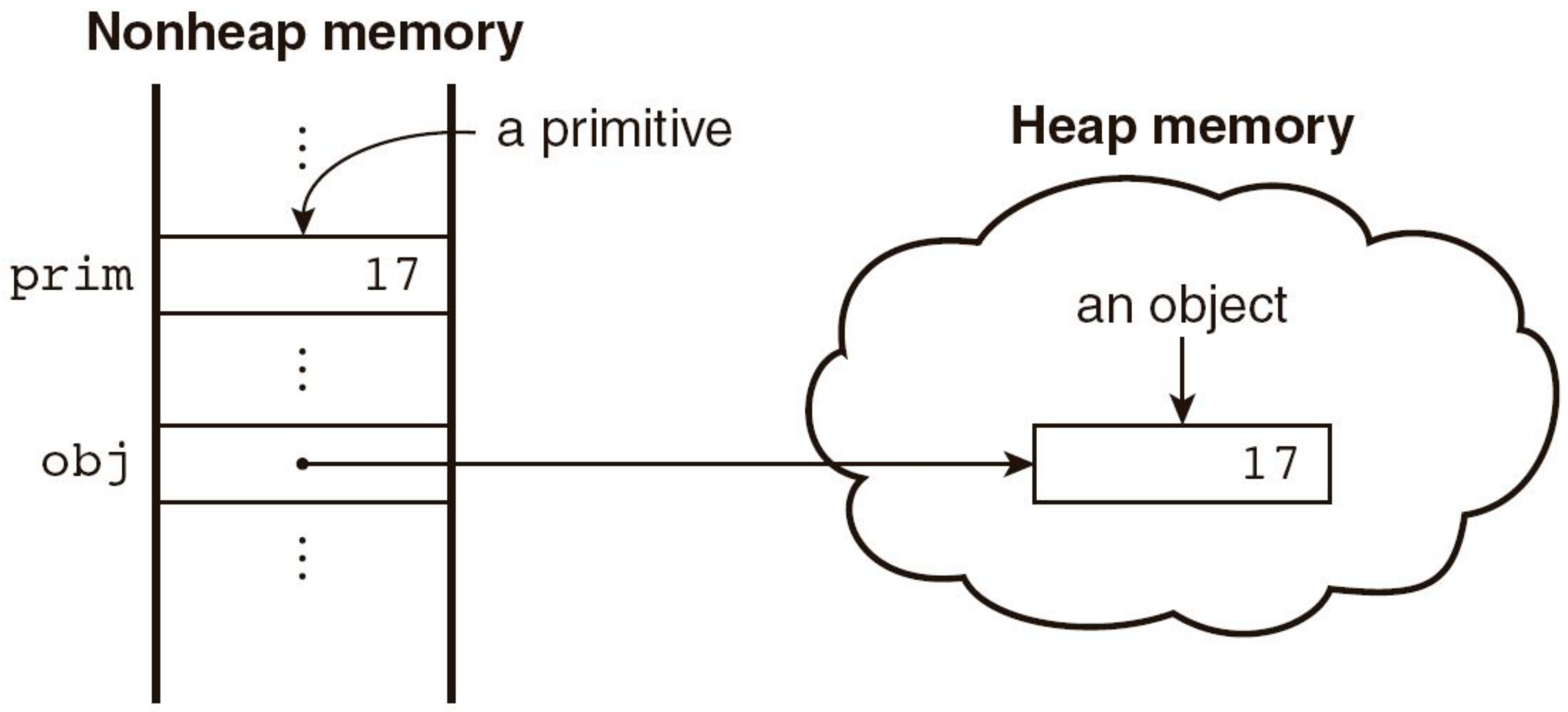


Figure 4.1 Primitives and objects

4.4 Numeric and String Literals

- **Number values are represented internally as double-precision floating-point values.**
- **This format stores numbers in 64 bits, where the number (the fraction) is stored in bits 0 to 51, the exponent in bits 52 to 62, and the sign in bit 63**
  - **Number literals can be either integer or float**
  - **Float values may have a decimal and/or exponent**

    eg: 12 1.2 .12 12. 1E2 1e2 .1e2 1.e2 1.2E-2
  - **These can also be written in hex form by preceding their first digit with 0x or OX.**
- **A String literal is delimited by either single or double quotes**
  - **There is no difference between single and double quotes**
  - **Certain characters may be *escaped* in strings**
    - **\’ or \” to use a quote in a string delimited by the same quotes**
    - **\\ to use a literal backspace**
  - **The empty string ‘’ or “” has no characters**

4.4 Other Primitive Types

- Null
  - A single value, null
  - `null` is a reserved word
  - A variable that is used but has not been declared nor been assigned a value has a null value
  - Using a null value usually causes an error
- Undefined
  - A single value, undefined
  - However, undefined is not, itself, a reserved word
  - The value of a variable that is declared but not assigned a value
- Boolean
  - Two values: true and false

4.4 Declaring Variables

- JavaScript is *dynamically typed*, that is, variables do not have declared types
  - A variable can hold different types of values at different times during program execution
- A variable can be declared either by assigning it a value or by listing it in a declaration statement that begins with a keyword var

```
var counter,
index,
pi = 3.14159265,
quarterback = "Elway",
stop_flag = true;
```

4.4 Numeric Operators

- Standard arithmetic
  - `+ * - / %`
- Increment and decrement
  - `-- ++`
  - Increment and decrement differ in effect when used before and after a variable
  - Assume that `a` has the value 3, initially
  - `(++a) * 3` has the value 24
  - `(a++) * 3` has the value 27
  - `a` has the final value 8 in either case

4.4 Precedence of Operators

Operators	Associativity
`++`, `--`, unary `-`	Right
`*`, `/`, `%`	Left
`+`, `-`	Left
`>`, `<`, `>=` ,`<=`	Left
`==`, `!=`	Left
`===`,`!==`	Left
`&&`	Left
`||`	Left
`=`, `+=`, `-=`, `*=`, `/=`, `&&=`, `||=`, `%=`	Right

4.4 Example of Precedence

```
var a = 2,
b = 4,
c,
d;
c = 3 + a * b;
// * is first, so c is now 11 (not 24)
d = b / a / 2;
// / associates left, so d is now 1 (not 4)
```

4.4 The `Math` Object

- Provides a collection of properties of Number object and methods which operate on Number objects
- This includes the trigonometric functions such as `sin` and `cos`
- Other methods are floor, round, max etc.
- When used, the methods must be referenced through the Math object, as in `Math.sin(x)`

4.4 The `Number` Object

- Properties
  - `MAX_VALUE`
  - `MIN_VALUE`
  - `NaN`
  - `POSITIVE_INFINITY`
  - `NEGATIVE_INFINITY`
  - `PI`
- Operations resulting in errors return `NaN`
  - Dividing zero by zero for example
- `toString()` method converts a number to a string

4.4 String Catenation

- The operation + is the string catenation operation
- In many cases, other types are automatically converted to string

Example:

```
var str = “first”;
var str1 = str + “second”;
```

4.4 Implicit Type Conversion

- JavaScript attempts to convert values in order to be able to perform operations
- "August " + 1977 causes the number to be converted to string and a concatenation to be performed
- 7 * "3" causes the string to be converted to a number and a multiplication to be performed
- null is converted to 0 in a numeric context, undefined to NaN
- 0 is interpreted as a Boolean false, all other numbers are interpreted as true
- The empty string is interpreted as a Boolean false, all other strings (including "0"!) as Boolean true
- undefined, Nan and null are all interpreted as Boolean false

4.4 Explicit Type Conversion

- Type conversion primarily between strings and numbers
- String that contain numbers can be converted to numbers using the String constructor

  Example:

  var str = String(value);
- Explicit conversion of string to number
  - var number = Number(aString)
  - var number = aString – 0
  - Number in the string must not be followed by any character apart from space
- Number can also be converted to string by catenating it with the empty string
- parseInt and parseFloat convert the beginning of a string but do not cause an error if a non-space follows the numeric part

4.4 `String` Properties and Methods

- One property: length
    - Note to Java programmers, this is not a method!

Example:

```
var str= "george";
var len = str.length;
```

- Large collection of methods
- Character positions in strings begin at index 0

4.4.11 String Methods

Method	Parameters	Result
charAt	A number	Returns the character in the String object that is at the specified position
indexOf	One-character string	Returns the position in the String object of the parameter
substring	Two numbers	Returns the substring of the String object from the first parameter position to the second
toLowerCase	None	Converts any uppercase letters in the string to lowercase
toUpperCase	None	Converts any lowercase letters in the string to uppercase

4.4 The `typeof` Operator

- **Returns “number” or “string” or “boolean” for primitive types**
- **Returns “object” for an object or null**
- **Two syntactic forms**
  - `typeof x`
  - `typeof(x)`
- **If the operand is a variable which has not been assigned a value,** `typeof` **evaluates to “undefined”**
- `typeof` **operator always returns a string**

4.4 Assignment Statements

- Plain assignment indicated by =
- Compound assignment with
  - `+=     -=     /=     *=    %=`   …
- a += 7 means the same as
- a = a + 7

4.4 The Date Object

- A Date object represents a *time stamp*, that is, a point in time
- A Date object is created with the new operator
  - var now= new Date();
  - This creates a Date object for the time at which it was created
- The date and time properties of a Date object are in two forms: local and UTC

4.4 The `Date` Object: Methods

toLocaleString	A string of the Date information
getDate	The day of the month
getMonth	The month of the year, as a number in the range of 0 to 11
getDay	The day of the week, as a number in the range of 0 to 6
getFullYear	The year
getTime	The number of milliseconds since January 1, 1970
getHours	The number of the hour, as a number in the range of 0 to 23
getMinutes	The number of the minute, as a number in the range of 0 to 59
getSeconds	The number of the second, as a number in the range of 0 to 59
getMilliseconds	The number of the millisecond, as a number in the range of 0 to 999

4.5 Screen Output and Keyboard Input

- **Standard output for JavaScript is the same as the screen in which the content of the host XHTML document is displayed**
- **JavaScript models the XHTML document with the** `Document` **object.**
- **The** `write` **method of the Document object writes its parameters to the browser window**
- **If a line break is needed in the output, use <br /> into the output**

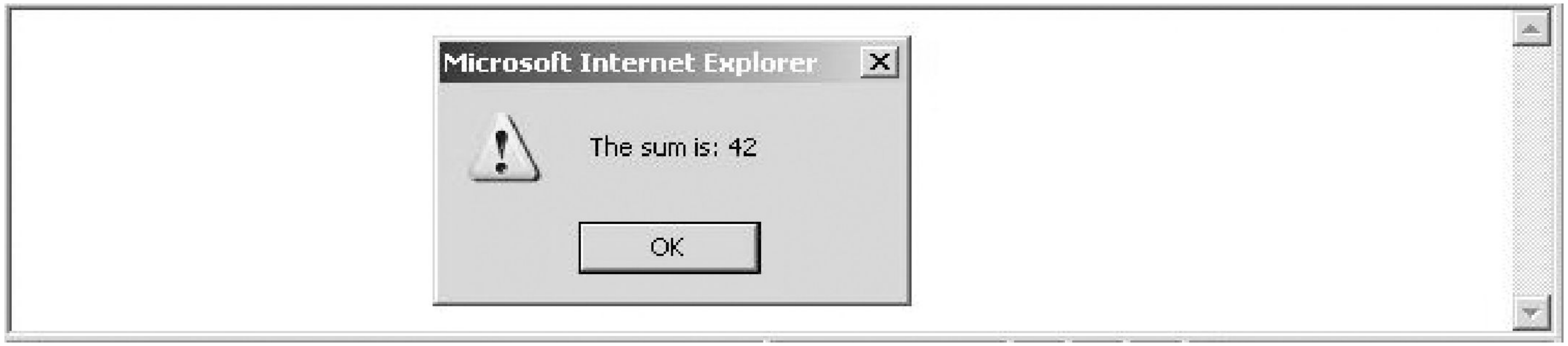
4.5 Window and Document

- **The** `Window` **object represents the window in which the document containing the script is being displayed**
- **The** `Document` **object represents the document being displayed using DOM**
- `Window` **has two properties**
  - `window` **refers to the Window object itself**
  - `document` **refers to the Document object**
- **The** `Window` **object is the default object for JavaScript, so properties and methods of the** `Window` **object may be used without qualifying with the class name**

4.5 The alert Method

- The alert method opens a dialog box with a message
- The output of the alert is *not* XHTML, so use new lines rather than <br/>

```
alert("The sum is:" + sum + "\n");
```

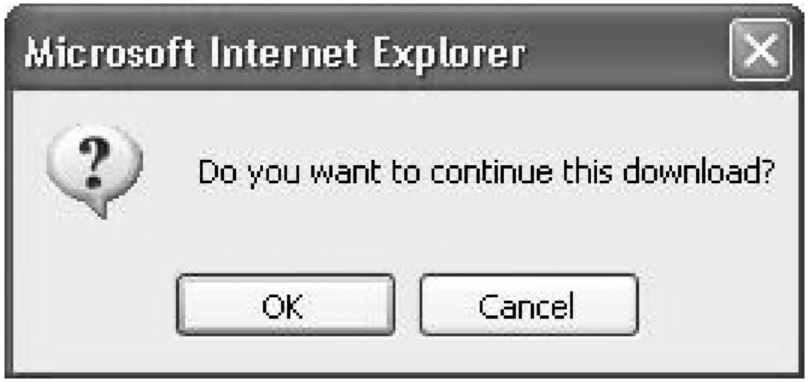


The sum is: 42

4.5 The confirm Method

- **The confirm methods displays a message provided as a parameter**
  - **The confirm dialog has two buttons: OK and Cancel**
- **If the user presses OK, true is returned by the method**
- **If the user presses Cancel, false is returned**

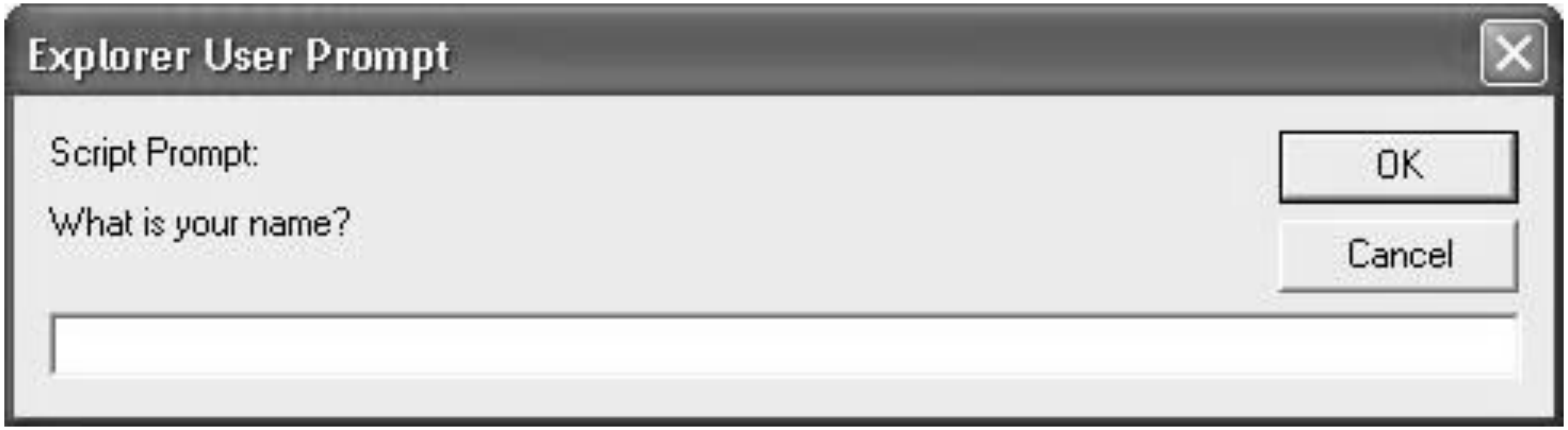
```
var question =
    confirm("Do you want to continue this download?");
```


Microsoft Internet Explorer
Do you want to continue this download?
OK
Cancel

4.5 The prompt Method

- This method displays its string argument in a dialog box
  - A second argument provides a default content for the user entry area
- The dialog box has an area for the user to enter text
- The method returns a String with the text entered by the user

```
name = prompt("What is your name?", "");
```



Script Prompt:

What is your name?

4.6 Control Statements

- A *compound statement* in JavaScript is a sequence of 0 or more statements enclosed in curly braces
  - Compound statements can be used as components of control statements allowing multiple statements to be used where, syntactically, a single statement is specified
- A *control construct* is a control statement including the statements or compound statements that it contains

4.6 Control Expressions

- A control expression has a Boolean value
  - An expression with a non-Boolean value used in a control statement will have its value converted to Boolean automatically
- Comparison operators
  - `==     !=    <    <=    >    >=`
  - `===` compares identity of values or objects
  - 3 == ‘3’ is true due to automatic conversion
  - 3 === ‘3’ is false
- Boolean operators
  - `&&     ||      !`
- Warning! A Boolean object evaluates as true
  - Unless the object is null or undefined

4.6 Selection Statements

- The if-then and if-then-else are similar to that in other programming languages, especially C/C++/Java

4.6 switch Statement Syntax

```
switch (expression) {
case value_1:
   // statement(s)
case value_2:
   // statement(s)
...
[default:
   // statement(s)]
}
```

4.6 switch Statement Semantics

- **The expression is evaluated**
- **The value of the expressions is compared to the value in each case in turn**
- **If no case matches, execution begins at the default case**
- **Otherwise, execution continues with the statement following the case**
- **Execution continues until either the end of the switch is encountered or a `break` statement is executed**

4.6 Loop Statements

- **Loop statements in JavaScript are similar to those in C/C++/Java**
- **While**

  `while` (*control expression*)

  statement or compound statement
- **For**

  `for` (*initial expression; control expression; increment expression*)

  statement or compound statement
- **do/while**

  `do` *statement or compound statement*

  `while` (*control expression*)

4.6 date.js Example

- Uses Date objects to time a calculation
- Displays the components of a Date object
- Illustrates a for loop

4.6 `while` Statement Semantics

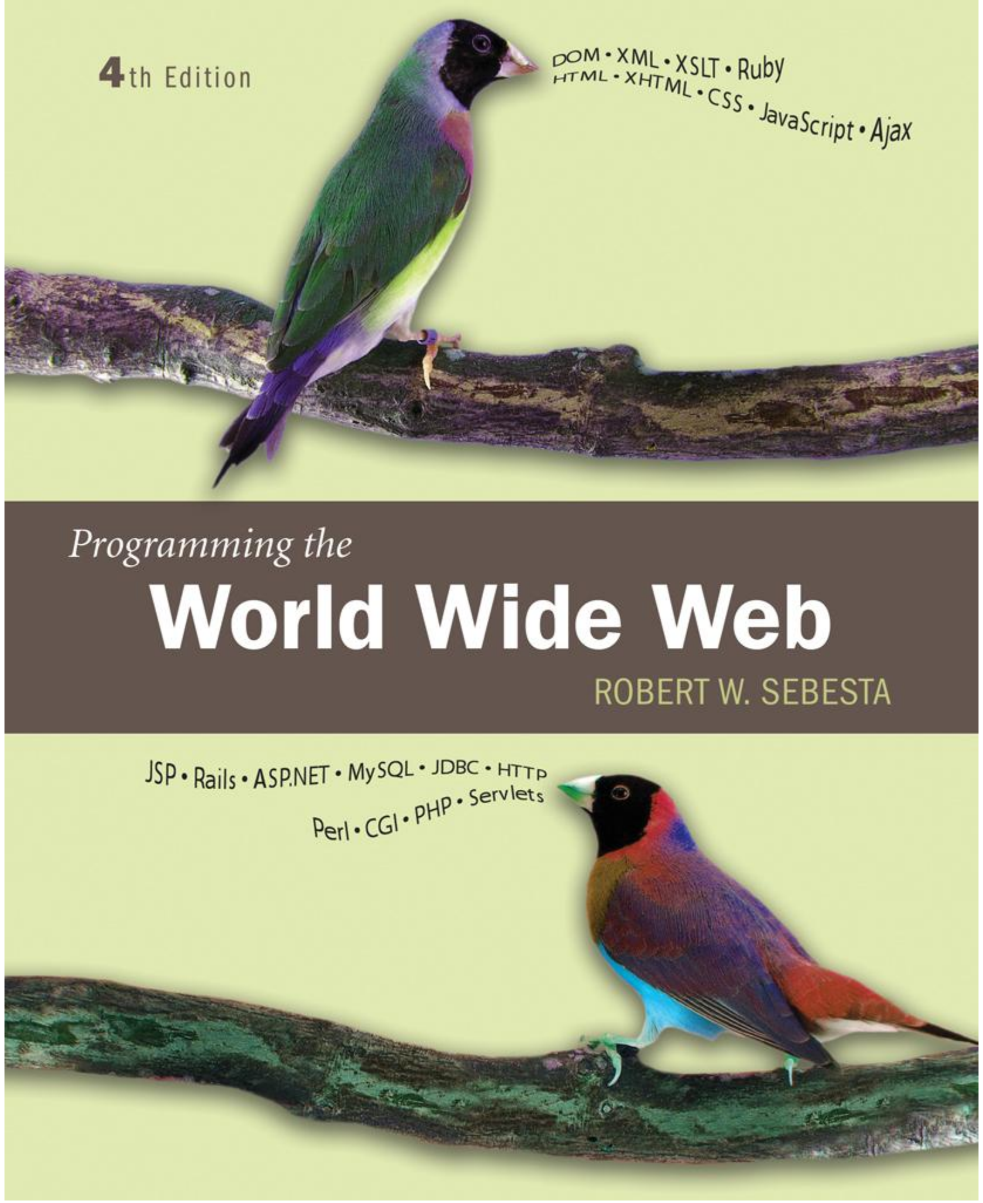
- The control expression is evaluated
- If the control expression is true, then the statement is executed
- These two steps are repeated until the control expression becomes false
- At that point the while statement is finished

4.6 `for` Statement Semantics

- The initial expression is evaluated
- The control expression is evaluated
- If the control expression is true, the statement is executed
- Then the increment expression is evaluated
- The previous three steps are repeated as long as the control expression remains true
- When the control expression becomes false, the statement is finished executing

4.6 `do/while` Statement Semantics

- The statement is executed
- The control expression is evaluated
- If the control expression is true, the previous steps are repeated
- This continues until the control expression becomes false
- At that point, the statement execution is finished


4th Edition
DOM • XML • XSLT • Ruby
HTML • XHTML • CSS • JavaScript • Ajax
Programming the
World Wide Web
ROBERT W. SEBESTA
JSP • Rails • ASP.NET • MySQL • JDBC • HTTP
Perl • CGI • PHP • Servlets

JavaScript (Continued)

4.7 Object Creation and Modification

- The `new` expression is used to create an object
  - This includes a call to a *constructor*
  - The new operator creates a blank object, or one with <u>no properties</u>
- Properties of an object are accessed using a dot notation: `object.property`
- Properties are not variables, just names of values and are not declared
- The number of properties of an object may vary dynamically in JavaScript

4.7 Dynamic Properties

- **The number of properties in JavaScript object is dynamic**

Example : Create my_car and add some properties

```
// Create an Object object
var my_car = new Object();
// Create and initialize the make property
my_car.make = "Ford";
// Create and initialize model
my_car.model = "Contour SVT";


          OR
Var my_car= {make: "Ford", model: "Contour SVT"};
```

4.7 Dynamic Properties contnd..

- Objects can also be nested
- A new object can be created that is a property of my_car with properties of its own as follows:

Example:

```
my_car.engine= new Object();
my_car.engine.config = “V6”;
my_car.engine.hp = 200;
```

- Properties can be accessed in two ways:
  1. object-dot-property notation
  2. As if they were elements of an array using the property name as the subscript

4.7 Dynamic Properties contnd..

Example:

```
var prop1 = my_car.make;
var prop2 = my_car [ “make” ];
```

- If an attempt is made to access a property of an object that does not exist, the value undefined is used

- The delete operator can be used to delete a property from an object
- `delete my_car.model`

4.7 The for-in Loop

- **Used in listing the properties of an object**
- **Syntax**

  for (*identifier* in *object*)

  statement or compound statement
- The loop lets the identifier take on each property in turn in the object
- Printing the properties in my_car:

```
for (var prop in my_car)
  document.write("Name: ", prop, "; Value: ",
    my_car[prop], "<br />");
```

- Result:

```
Name: make; Value: Ford
Name: model; Value: Contour SVT
```

4.8 Arrays

- Arrays are lists of elements indexed by a numerical value
- Array indexes in JavaScript begin at 0
- Arrays can be modified in size even after they have been created (dynamic)

4.8 `Array` Object Creation

- **Arrays can be created using the new Array method**
  - new Array with one parameter creates an empty array of the specified number of elements
    - var my_list=new Array(10)
  - new Array with two or more parameters creates an array with the specified parameters as elements
    - var my_list=new Array(10, 20)
- **Arrays can also be created with Literal arrays which is a list of values enclosed in square brackets**
  - var alist = [1, “ii”, “gamma”, “4”];
- **Elements of an array do not have to be of the same type**

4.8 Characteristics of `Array` Objects

- The length of an array is one more than the highest index to which a value has been assigned or the initial size (using Array with one argument), whichever is larger
- Assignment to an index greater than or equal to the current length simply increases the length of the array
- Only assigned elements of an array occupy space
  - Suppose an array were created using new Array(200)
  - Suppose only elements 150 through 174 were assigned values
  - Only the 25 assigned elements would be allocated storage, the other 175 would not be allocated storage

4.8 Example insert_names.js

- This example shows the dynamic nature of arrays in JavaScript

4.8 `Array` Methods

1. *join*

   -- converts all of the elements of an array to strings and catenates them into a single string

2. *reverse*

   --reverses the order of the elements of the Array object through which it is called

3. *sort*

   --coerces the elements of the array to strings, if they are not already strings, and sorts them alphabetically

4. *concat*

   --catenates its actual parameters to the end of the Array object on which it is called

4.8 Dynamic List Operations

5. ***slice*** **---**it returns the part of the Array object specified by its parameter
6. ***push*** **--**Adds element to the high end of the array
7. ***pop*** **--**Removes from the high end of the array
8. ***shift*** **--**Removes element from the beginning of the array
9. ***unshift*** **–**Adds elements to the beginning of the array

Example:

```
var list = [“first”, ”second”, ”third”];
 var temp= list.pop();      // temp=“third”
 list.push(“third”);
 var temp = list.shift();   //temp=“first”
  list.unshift(“first”);        //puts “first” back on the list
```

4.8 Two-dimensional Arrays

- **A two-dimensional array in JavaScript is an array of arrays**
  - **This need not even be rectangular shaped: different rows could have different length**
- **Example nested_arrays.js illustrates two-dimensional arrays**

4.9 Function Fundamentals

- **Function definition syntax**
  - **A function definition consist of a header followed by a compound statement**
  - **A function header:**
    - **function *function-name*(*optional-formal-parameters*)**
- **return statements**
  - **A return statement causes a function to cease execution and control to pass to the caller**
  - **A return statement may include a value which is sent back to the caller**
    - **This value may be used in an expression by the caller**
  - **A return statement without a value implicitly returns undefined**
- **Function call syntax**
  - **Function name followed by parentheses and any actual parameters**
  - **Function call may be used as an expression or part of an expression**
- **Functions must defined before use in the page header**

4.9 Functions are Objects

- **Functions are objects in JavaScript**
- **Function's addresses can be properties in other objects, in which case they act as methods**
- **Example**

```
function fun() {
  document.write(    "This surely is fun! <br/>");
}
  ref_fun = fun; // Now, ref_fun refers to the fun object
  fun(); // A call to fun
  ref_fun(); // Also a call to fun
```

4.9 Local Variables

- “The *scope* of a variable is the range of statements over which it is visible”
- A variable not declared using var has global scope, visible throughout the page, even if used inside a function definition
- A variable declared with var outside a function definition has global scope
- A variable declared with var inside a function definition has local scope, visible only inside the function definition
  - If a global variable has the same name, it is hidden inside the function definition and the local variable has the precedence over the global variable

4.9 Parameters

- Parameters named in a function header are called "*formal parameters*"
- Parameter values that appear in a call to a function are called "*actual parameters*"
- Parameters are passed by value
  - For an object parameter, the reference is passed, so the function body can actually change the object
  - However, an <u>assignment</u> to the formal parameter will not change the actual parameter

4.9 Parameter Passing Example

```
function fun1(my_list) {
   var list2 = new Array(1, 3, 5);
   my_list[3] = 14;
   ...
   my_list = list2;
   ...
}
...
var list = new Array(2, 4, 6, 8)
fun1(list);
```

- **The first assignment changes list in the caller**
- **The second assignment has no effect on the list object in the caller**
- **Pass by reference can be simulated by passing an array containing the value**

4.9 Parameter Checking

- JavaScript checks neither the type nor number of parameters in a function call
  - Formal parameters have no type specified
  - Extra actual parameters are ignored (however, see below)
  - If there are fewer actual parameters than formal parameters, the extra formal parameters remain undefined
- A property array named `arguments` holds all of the actual parameters, whether or not there are more actual parameters than number of formal parameters
- There is no proper way to pass a primitive value by reference
- Improper way is to put the value into an array and pass the array

4.9 Parameter Checking contd…

Example:

```
function fun(a){
a[0] *= 10; }
…..
var x;
var listx = new Array(1);
…..
listx[0]=x;
fun(listx);
x=listx[0];
```

4.9 Parameter Checking contd…

- **Another way is to return the new value from the function to the caller**

  Example:

```
function fun(a){
    return 10*a;
    }
    var x;
    x= fun(x);
```

4.9 The `sort` Method, Revisited

- If you need to sort something other than strings, or if you want an array to be sorted in some order other than alphabetically as strings, the comparison operation must be supplied to the sort method by the caller.
- A parameter can be passed to the sort method to specify how to sort elements in an array
  - The parameter is a function that takes two parameters
  - The function returns a negative value to indicate the parameters are in the desired order
  - The function returns a positive value to indicate that the numbers must be interchanged
  - The function returns 0 to indicate the first parameter and the second parameter are equivalent as far as the ordering is concerned

4.11 Constructors

- **Constructors are functions that create and initialize properties for new objects**
- **A constructor uses the keyword** `this` **in the body to reference the object being initialized**
- **A method can be inserted in the object in the same way as a data property.**

4.12 Pattern Using Regular Expressions

- Regular expressions are used to specify patterns in strings
- JavaScript provides two methods to use regular expressions in pattern matching
  - String methods
  - RegExp objects (not covered in the text)
- A literal regular expression pattern is indicated by enclosing the pattern in slashes
- The `search` method returns the position of a match, if found, or -1 if no match was found

4.12 Example Using search

```
var str = "Rabbits are furry";
var position = str.search(/bits/);
if (position > 0)
   document.write("'bits' appears in position",
      position, "<br />");
else
   document.write(
      "'bits' does not appear in str <br />");
```

- **This uses a pattern that matches the string ‘bits’**
- **The output of this code is as follows:**

  'bits' appears in position 3

4.12 Characters and Character-Class Patterns

- *Metacharacters* have special meaning in regular expressions
  - `\ | ( ) [ ] { } ^ $ * + ? .`
  - These characters may be used literally by escaping them with \
- Other characters represent themselves
- A period matches any character other than newline
  - /f.r/ matches for and far and fir but not fr
  - if you want to match /3.4/?
- A character class matches one of a specified set of characters
  - [*character set*]
  - List characters individually: [abcdef]
  - Give a range of characters: [a-z]
  - Beware of [A-z]
  - circumflex (^) at the beginning negates the class

4.12 Predefined character classes

Name	Equivalent Pattern	Matches
\d	[0-9]	A digit
\D	[^0-9]	Not a digit
\w	[A-Za-z_0-9]	A word character (alphanumeric)
\W	[^A-Za-z_0-9]	Not a word character
\s	[\r\t\n\f]	A whitespace character
\S	[^ \r\t\n\f]	Not a whitespace character

4.12 Repeated Matches

- A pattern can be repeated a fixed number of times by following it with a pair of curly braces enclosing a count
- A pattern can be repeated by following it with one of the following 3 symbolic quantifiers:
  - * indicates zero or more repetitions of the previous pattern
  - + indicates one or more of the previous pattern
  - ? indicates zero or none of the previous pattern
- Examples
  - `/\(\d{3}\)\d{3}-\d{4}/` might represent a telephone number
  - `/[A-Za-z]\w*/` matches identifiers
  - /x*y+z?/-----???
  - /\d+\.\d*/-----???

4.12 Repeated Matches contd…

- One additional named pattern \b (boundary) that matches the boundary position between a word character (\w) and a nonword character (\W), in either order
- Example:

  `/\bis\b/` matches

  “ A tulip is a flower” but not “A frog isn’t”
- Boundary pattern does not match a character it matches the position between two characters

4.12 Anchors

- Anchors in regular expressions match positions rather than characters
  - Anchors are 0 width and may not take multiplicity modifiers
- Anchoring to the end of a string
  - ^ at the beginning of a pattern matches the beginning of a string
  - $ at the end of a pattern matches the end of a string
    - The $ in /a$b/ matches a $ character
- Anchoring at a word boundary
  - \b matches the position between a word character and a non-word character or the beginning or the end of a string
  - /\bthe\b/ will match ‘the’ but not ‘theatre’ and will also match ‘the’ in the string ‘one of the best’

4.12 Pattern Modifiers

- Pattern modifiers are specified by characters that follow the right delimiter / of a pattern
- Modifiers modify the way a pattern is interpreted or used
- The `x` modifier causes whitespace in the pattern to be ignored
  - This allows better formatting of the pattern
  - \s still retains its meaning
- The `i` modifier makes the letters in the pattern match either lowercase or uppercase

  Example: `/Apple/i` matches any combination of uppercase and lowercase spellings of the word “apple”

- The `g` modifier is explained in the following

4.12 Other Pattern Matching Methods

- **The `replace` method takes a pattern parameter and a string parameter**
  - **The method replaces a match of the pattern in the target string with the second parameter**
  - **A `g` modifier on the pattern causes multiple replacements**
- **Parentheses can be used in patterns to mark sub-patterns**
  - **The pattern matching machinery will remember the parts of a matched string that correspond to sub-patterns**
- **The `match` method takes one pattern parameter**
  - **Without a `g` modifier, the return is an array of the match and parameterized sub-matches**
  - **With a `g` modifier, the return is an array of all matches**
- **The `split` method splits the object string using the pattern to specify the split points**

4.13 An Example

- forms_check.js
- Using javascript to check the validity of input data
- Note, a server program may need to check the data sent to it since the validation can be bypassed in a number of ways

4.14 Errors in Scripts

- JavaScript errors are detected by the browser
- Different browsers report this differently
  - Firefox uses a special console
- Support for debugging is provided
  - In IE 7, the debugger is part of the browser
  - For Firefox 2, plug-ins are available
    - These include Venkman and Firebug