Intro to Javascript

Click to add text

What is Javascript?

- a lightweight programming language ("scripting language")
  - used to make web pages interactive
  - insert dynamic text into HTML (ex: user name)
  - **react to events** (ex: page load user click)
  - get information about a user's computer (ex: browser type)
  - perform calculations on user's computer (ex: form validation)

What is Javascript?

- a web standard (but not supported identically by all browsers)
- NOT related to Java other than by name and some syntactic similarities

Linking to a JavaScript file: `script`

```
<script src="filename" type="text/javascript"></script>
```

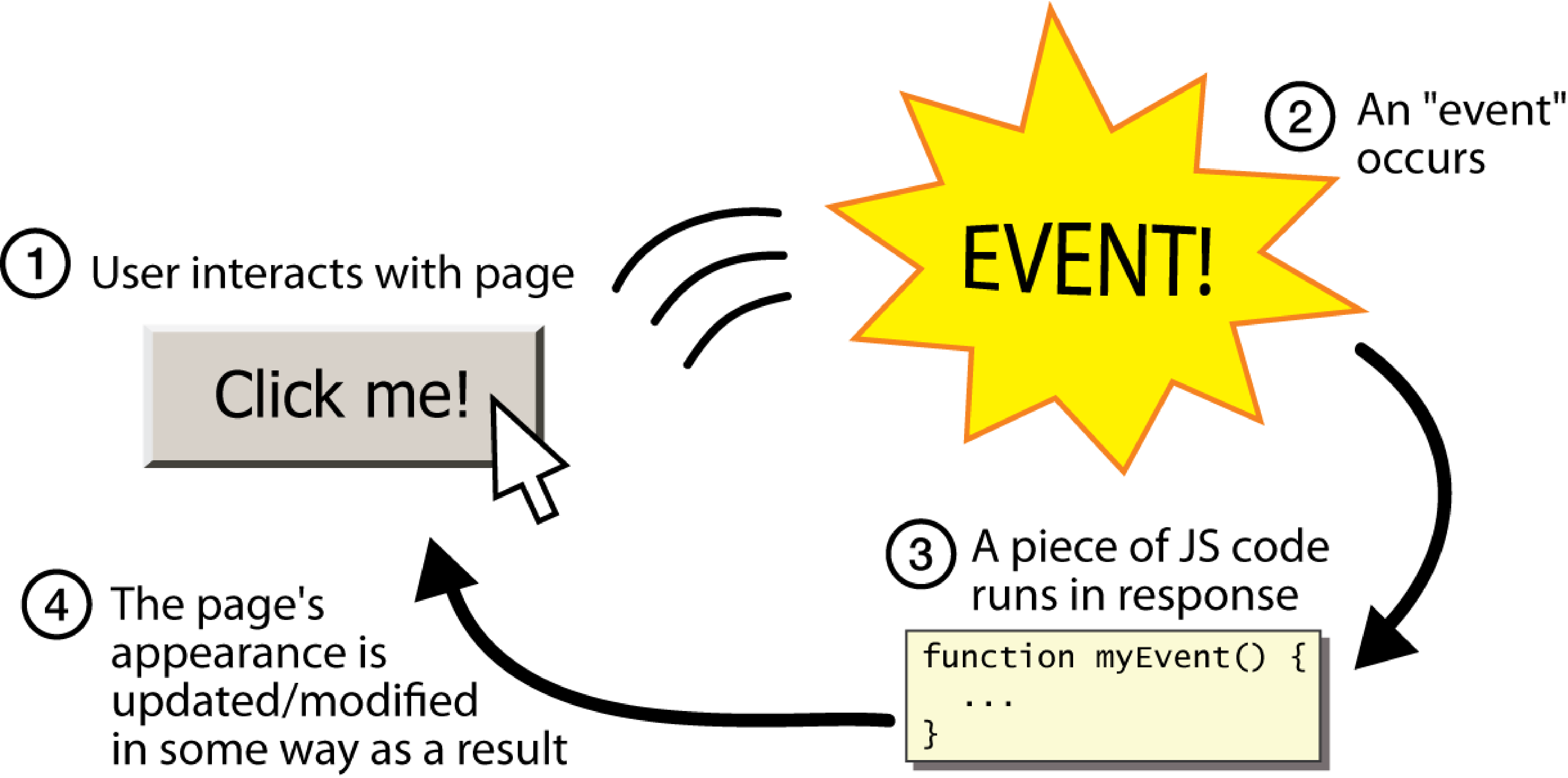
HTML

- script tag should be placed in HTML page's head
- script code is stored in a separate .js file
- JS code can be placed directly in the HTML file's body or head (like CSS)
  - but this is bad style (should separate content, presentation, and behavior

Example 1: Add Two Numbers

```
<html>
    …
 <p>  … </p>
<script>
        var num1, num2, sum
        num1 = prompt("Enter first number")
        num2 = prompt("Enter second number")
        sum = parseInt(num1) + parseInt(num2)
        alert("Sum = " + sum)
</script>
        …
</html>
```

Event-driven programming



1 User interacts with page

2 An "event" occurs

3 A piece of JS code runs in response

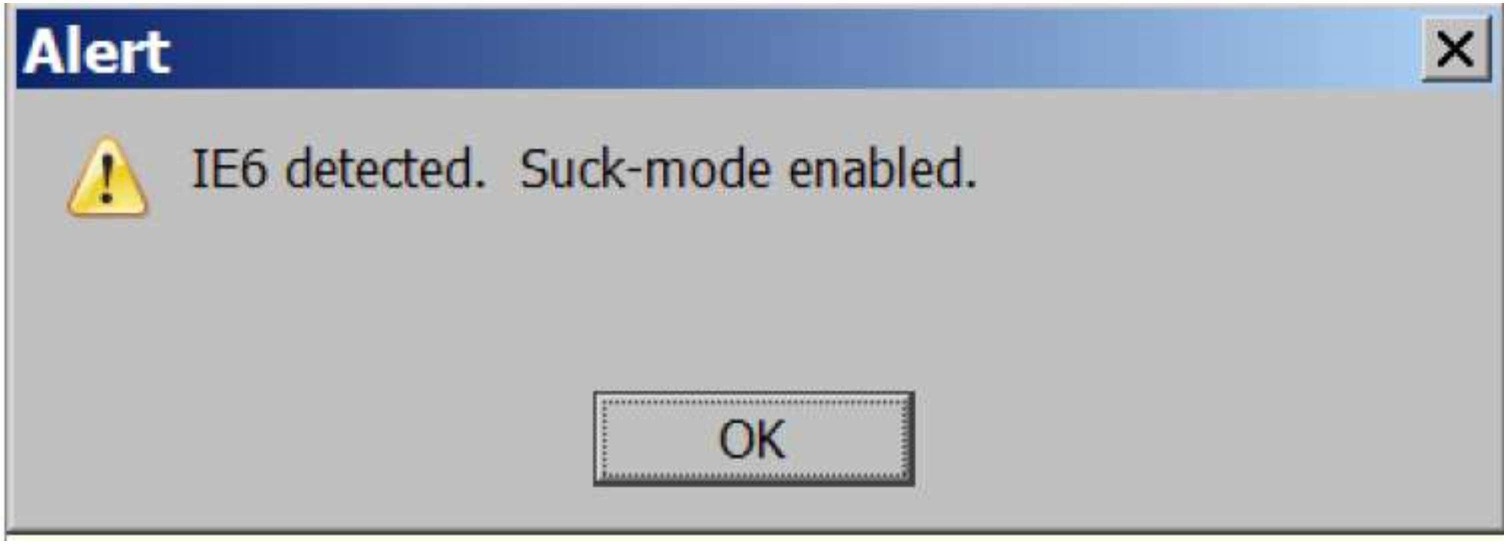
```
function myEvent() {
  ...
}
```

4 The page's appearance is updated/modified in some way as a result

A JavaScript statement: `alert`

```
alert("IE6 detected. Suck-mode enabled.");
```

JS



IE6 detected. Suck-mode enabled.

- a JS command that pops up a dialog box with a message

Event-driven programming

- you are used to programs start with a main method (or implicit main like in PHP)
- JavaScript programs instead wait for user actions called *events* and respond to them
- event-driven programming: writing programs driven by user events
- Let's write a page with a clickable button that pops up a "Hello, World" window...

Buttons

```
<button>Click me!</button>                                   HTML
```

□ button's text appears inside tag; can also contain images

□ To make a responsive button or other UI control:

1. choose the control (e.g. button) and event (e.g. mouse 1. click) of interest
2. write a JavaScript function to run when the event occurs
3. attach the function to the event on the control

JavaScript functions

```
function name() {
statement ;
statement ;
...
statement ;
}                                                  JS
```

```
function myFunction() {
      alert("Hello!");
      alert("How are you?");
}                                                  JS
```

- the above could be the contents of example.js linked to our HTML page
- statements placed into functions can be evaluated in response to user events

Event handlers

```
<element attributes onclick="function();">...
```

HTML

```
<button onclick="myFunction();">Click me!</button>
```

HTML

- JavaScript functions can be set as event handlers
  - when you interact with the element, the function will execute
- onclick is just one of many event HTML attributes we'll use
- but popping up an alert window is disruptive and annoying
  - A better user experience would be to have the message appear on the page...

Example 2: Browser Events

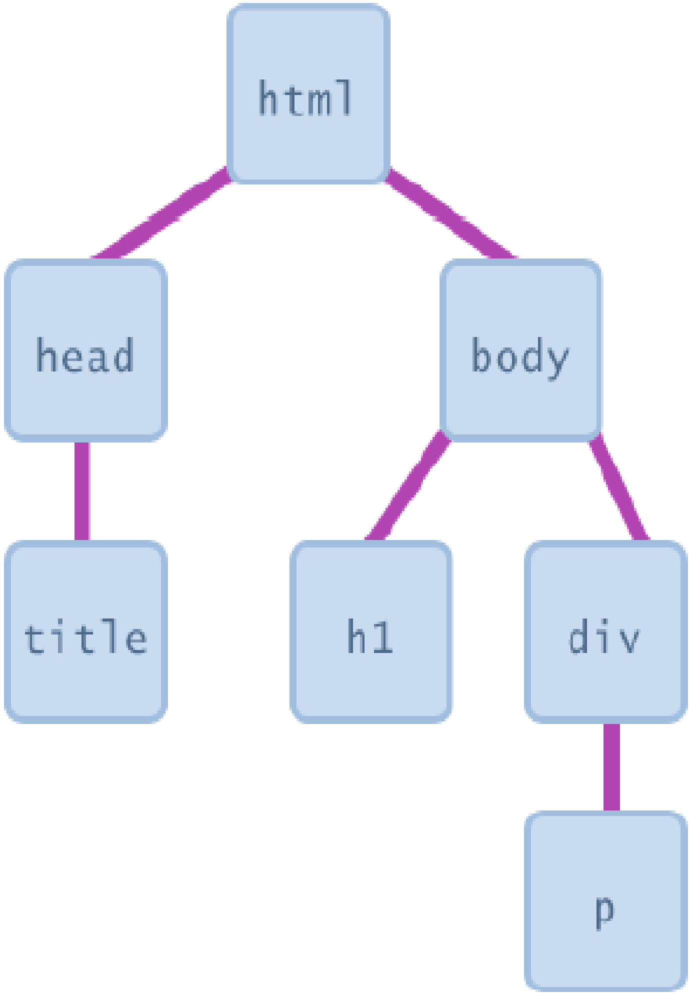
Mouse event causes page-defined function to be called

```
<script type="text/JavaScript">
   function whichButton(event) {
     if (event.button==1) {
              alert("You clicked the left mouse button!") }
     else {
              alert("You clicked the right mouse button!")
     }}
</script>
…
<body onmousedown="whichButton(event)">
…
</body>
```

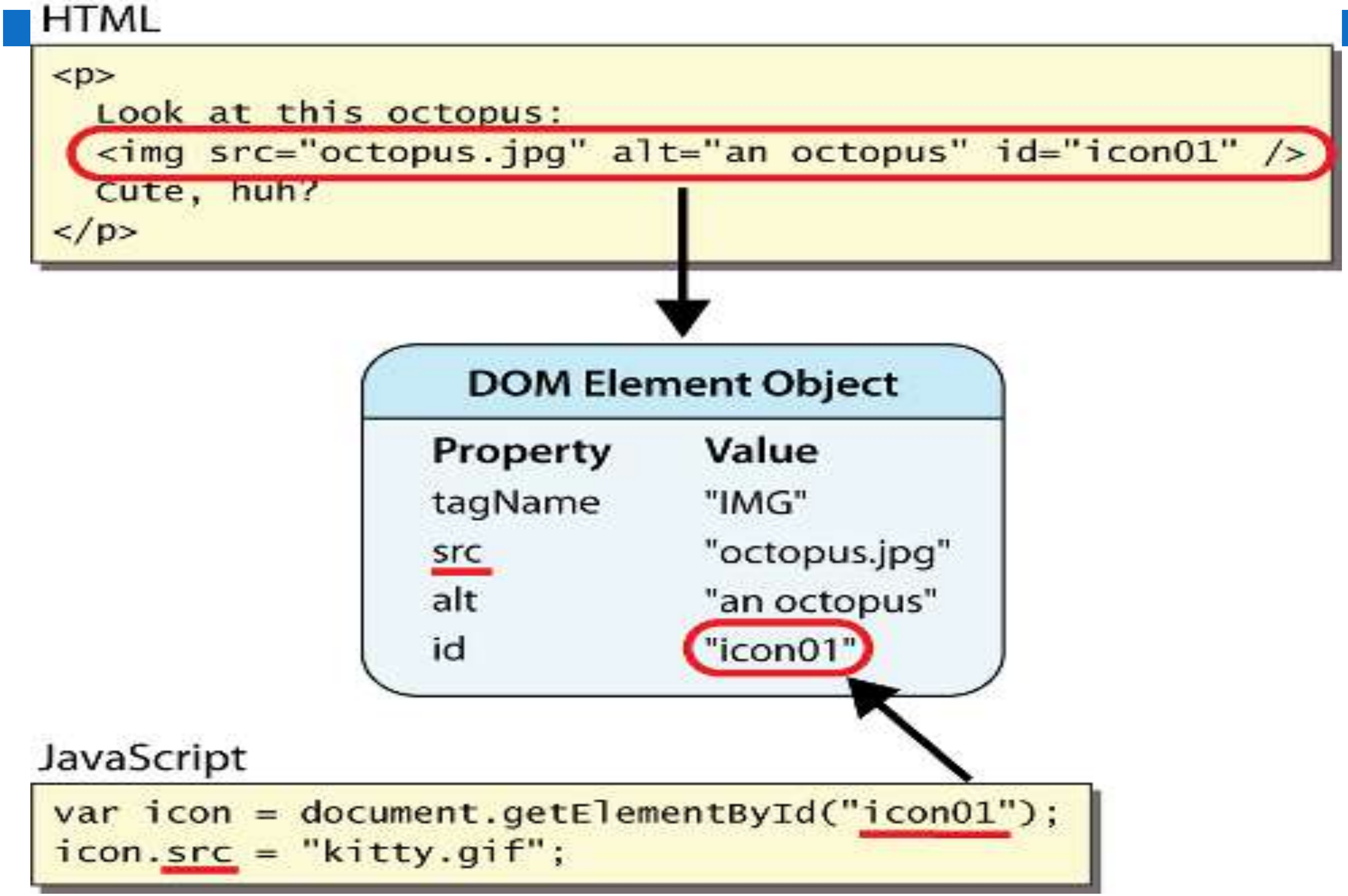
Other events: onLoad, onMouseMove, onKeyPress, onUnLoad

Document Object Model (DOM)

- most JS code manipulates elements on an HTML page
- we can examine elements' state
  - e.g. see whether a box is checked
- we can change state
  - e.g. insert some new text into a div
- we can change styles
  - e.g. make a paragraph red



DOM element objects



HTML

```
<p>
  Look at this octopus:
  <img src="octopus.jpg" alt="an octopus" id="icon01" />
  Cute, huh?
</p>
```

DOM Element Object

Property	Value
tagName	"IMG"
src	"octopus.jpg"
alt	"an octopus"
id	"icon01"

JavaScript

```
var icon = document.getElementById("icon01");
icon.src = "kitty.gif";
```

Accessing elements: document.getElementById

```
var name = document.getElementById("id");
```

JS

```
<button onclick="changeText();">Click me!</button>
<span id="output">replace me</span>
<input id="textbox" type="text" />
```

HTML

```
function changeText() {
        var span = document.getElementById("output");
        var textBox = document.getElementById("textbox");

         textbox.style.color = "red";

}
```

JS

Accessing elements: `document.getElementById`

- document.getElementById returns the DOM object for an element with a given id
- can change the text inside most elements by setting the innerHTML property
- can change the text in form controls by setting the value property

Changing element style: `element.style`

Attribute	Property or style object
color	color
padding	padding
background-color	backgroundColor
border-top-width	borderTopWidth
Font size	fontSize
Font famiy	fontFamily

Preetify

```
function changeText() {
        //grab or initialize text here

        // font styles added by JS:
        text.style.fontSize = "13pt";
        text.style.fontFamily = "Comic Sans MS";
        text.style.color = "red"; // or pink?
}
```

JS

Example 3: Page Manipulation

- Some possibilities
  - createElement(elementName)
  - createTextNode(text)
  - appendChild(newChild)
  - removeChild(node)
- Example: add a new list item

```
var list = document.getElementById('t1')
var newitem = document.createElement('li')
var newtext = document.createTextNode(text)
list.appendChild(newitem)
newitem.appendChild(newtext)
```

This uses the browser Document Object Model (DOM). We will focus on JavaScript as a language, not its use in the browser

Reading Properties with JavaScript

Sample HTML

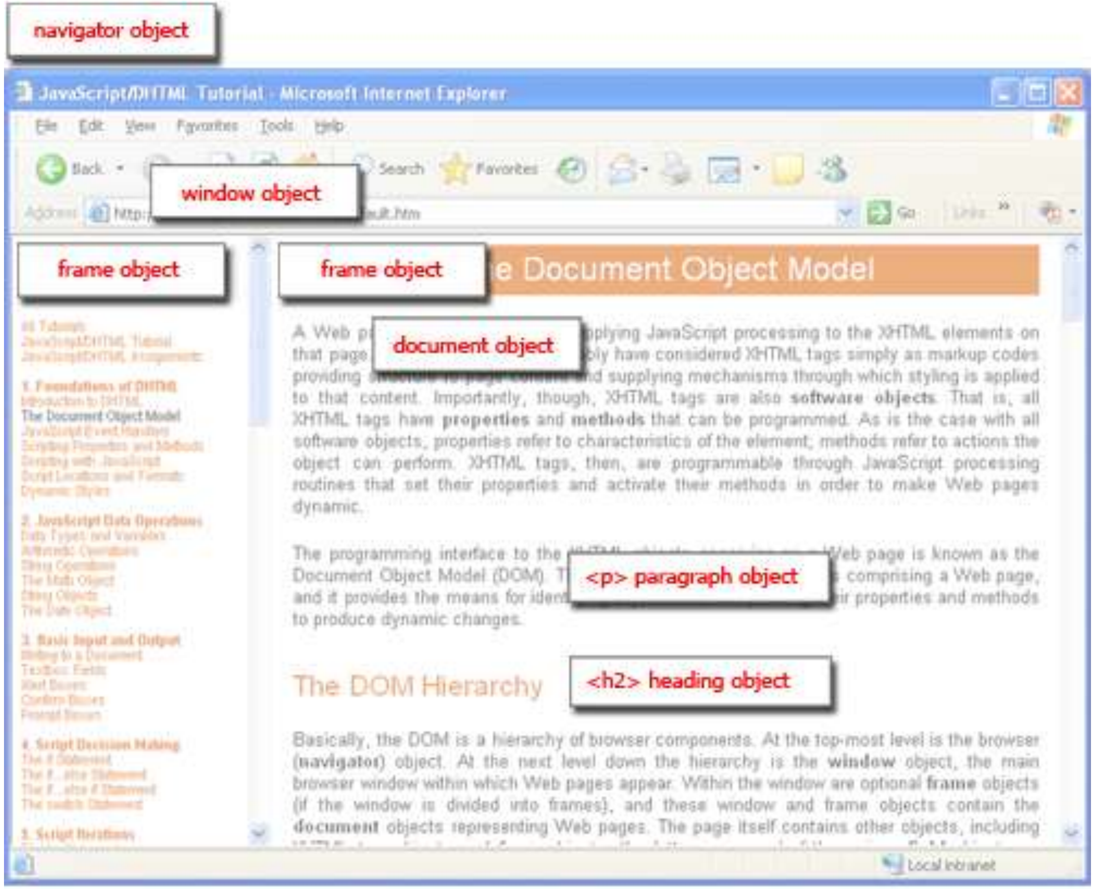
```
<ul id="t1">
<li> Item 1 </li>
</ul>
```

Sample script

```
1. document.getElementById('t1').nodeName
2. document.getElementById('t1').nodeValue
3. document.getElementById('t1').firstChild.nodeName
4. document.getElementById('t1').firstChild.firstChild.nodeName
5. document.getElementById('t1').firstChild.firstChild.nodeValue
```

- Example 1 returns "ul"
- Example 2 returns "null"
- Example 3 returns "li"
- Example 4 returns "text"
  - A text node below the "li" which holds the actual text data as its value
- Example 5 returns " Item 1 "

Browser and Document Structure



All Tutorials
JavaScript/DHTML Tutorial
JavaScript/DHTML Assignments

1. Foundations of DHTML
Introduction to DHTML
The Document Object Model
JavaScript Event Handlers
Scripting Properties and Methods
Scripting with JavaScript
Script Locations and Formats
Dynamic Styles

2. JavaScript Data Operations
Data Types and Variables
Arithmetic Operations
String Operations
The Math Object
String Objects
The Date Object

3. Basic Input and Output
Writing to a Document
Textbox Fields
Alert Boxes
Confirm Boxes
Prompt Boxes

4. Script Decision Making
The if Statement
The if...else Statement
The if...else if Statement
The switch Statement

5. Script Iterations

e Document Object Model

A Web p pplying JavaScript processing to the XHTML elements on that page bly have considered XHTML tags simply as markup codes providing d supplying mechanisms through which styling is applied to that content. Importantly, though, XHTML tags are also **software objects**. That is, all XHTML tags have **properties** and **methods** that can be programmed. As is the case with all software objects, properties refer to characteristics of the element; methods refer to actions the object can perform. XHTML tags, then, are programmable through JavaScript processing routines that set their properties and activate their methods in order to make Web pages dynamic.

The programming interface to the Web page is known as the Document Object Model (DOM). T comprising a Web page, and it provides the means for iden ir properties and methods to produce dynamic changes.

The DOM Hierarchy

Basically, the DOM is a hierarchy of browser components. At the top-most level is the browser (**navigator**) object. At the next level down the hierarchy is the **window** object, the main browser window within which Web pages appear. Within the window are optional **frame** objects (if the window is divided into frames), and these window and frame objects contain the **document** objects representing Web pages. The page itself contains other objects, including

W3C standard differs from models supported in existing browsers

More Javascript Syntax

JavaScript Primitive Datatypes

- Boolean: true and false
- Number: 64-bit floating point
  - Similar to Java double and Double
  - No integer type
  - Special values NaN (not a number) and Infinity
- String: sequence of zero or more Unicode chars
  - No separate character type (just strings of length 1)
  - Literal strings using ' or " characters (must match)
- Special objects: null and undefined

Variables

```
var name = expression;
```

JS

```
var clientName = "Connie Client";
var age = 32;
var weight = 127.4;
```

JS

- variables are declared with the var keyword (case sensitive)
- types are not specified, but JS does have types ("loosely typed")
  - `Number, Boolean, String, Array, Object, Function, Null, Undefined`
  - can find out a variable's type by calling `typeof`

Number type

```
var enrollment = 99;
var medianGrade = 2.8;
var credits = 5 + 4 + (2 * 3);
```

JS

- integers and real numbers are the same type (no int vs. double)
- same operators: + - * / % ++ -- = += -= *= /= %=
- similar precedence to Java
- many operators auto-convert types: "2" * 3 is 6

Comments (same as Java)

```
// single-line comment
/* multi-line comment */
```

JS

- identical to Java's comment syntax
- recall: 4 comment syntaxes
  - HTML: <!-- comment -->
  - CSS/JS/PHP: /* comment */
  - Java/JS/PHP: // comment
  - PHP: # comment

Math object

```
var rand1to10 = Math.floor(Math.random() * 10 + 1);
var three = Math.floor(Math.PI);
```

JS

- **methods:** `abs, ceil, cos, floor, log, max, min, pow, random, round, sin, sqrt, tan`
- **properties:** `E, PI`

Logical operators

- > < >= <= && || ! == != === !==
- most logical operators automatically convert types:
  - 5 < "7" is true
  - 42 == 42.0 is true
  - "5.0" == 5 is true
- === and !== are strict equality tests; checks both type and value
  - "5.0" === 5 is false

if/else statement (same as Java)

```
if (condition) {
      statements;
} else if (condition) {
      statements;
} else {
      statements;
}
```

JS

- identical structure to Java's if/else statement
- JavaScript allows almost anything as a condition

Boolean type

```
var iLike190M = true;
var ieIsGood = "IE6" > 0; // false
if ("web devevelopment is great") { /* true */ }
if (0) { /* false */ }
```

JS

- any value can be used as a Boolean
  - "falsey" values: 0, 0.0, NaN, "", null, and undefined
  - "truthy" values: anything else
- converting a value into a Boolean explicitly:
  - `var boolValue = Boolean(otherValue);`
  - `var boolValue = !!(otherValue);`

for loop (same as Java)

```
var sum = 0;
for (var i = 0; i < 100; i++) {
      sum = sum + i;
}
```

JS

```
var s1 = "hello";
var s2 = "";
for (var i = 0; i < s.length; i++) {
      s2 += s1.charAt(i) + s1.charAt(i);
}
// s2 stores "hheelllloo"
```

JS

while loops (same as Java)

```
while (condition) {
        statements;
}
```

JS

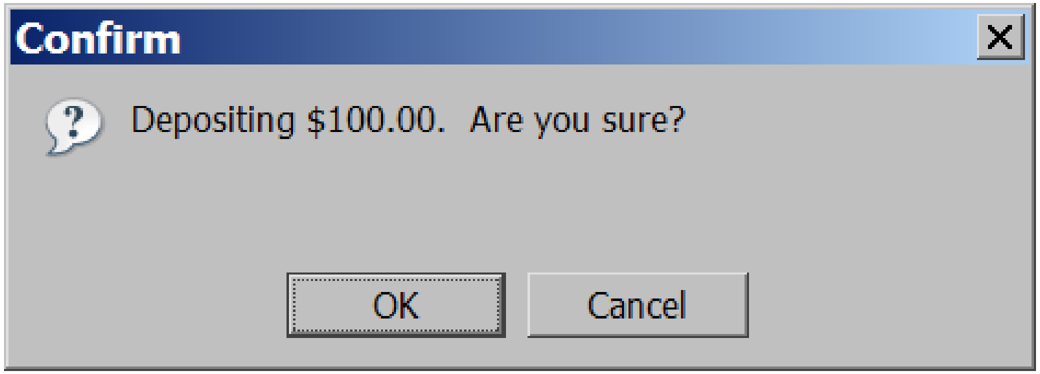
```
do {
    statements;
} while (condition);
```

JS

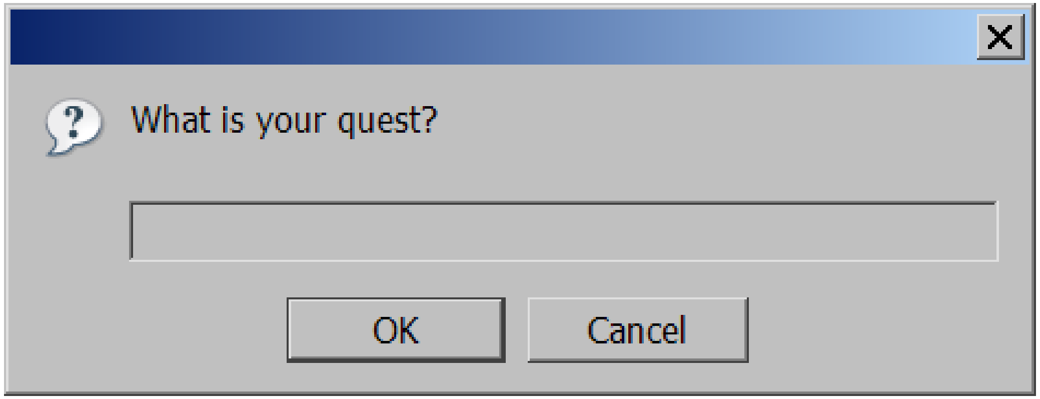
- □ break and continue keywords also behave as in Java

Popup boxes

```
alert("message"); // message
confirm("message"); // returns true or false
prompt("message"); // returns user input string
                                                    JS
```


Confirm

Depositing $100.00. Are you sure?


What is your quest?

Arrays

```
var name = []; // empty array
var name = [value, value, ..., value]; // pre-filled
name[index] = value; // store element
```

JS

```
var ducks = ["Huey", "Dewey", "Louie"];
var stooges = []; // stooges.length is 0
stooges[0] = "Larry"; // stooges.length is 1
stooges[1] = "Moe"; // stooges.length is 2
stooges[4] = "Curly"; // stooges.length is 5
stooges[4] = "Shemp"; // stooges.length is 5
```

JS

Array methods

```
var a = ["Stef", "Jason"]; // Stef, Jason
a.push("Brian"); // Stef, Jason, Brian
a.unshift("Kelly"); // Kelly, Stef, Jason, Brian
a.pop(); // Kelly, Stef, Jason
a.shift(); // Stef, Jason
a.sort(); // Jason, Stef
```

JS

- array serves as many data structures: list, queue, stack, ...
- **methods:** `concat, join, pop, push, reverse, shift, slice, sort, splice, toString, unshift`
  - push and pop add / remove from back
  - unshift and shift add / remove from front
  - shift and pop return the element that is removed

`String` type

```
var s = "Connie Client";
var fName = s.substring(0, s.indexOf(" ")); // "Connie"
var len = s.length; // 13
var s2 = 'Melvin Merchant';
```

JS

- **methods:** `charAt, charCodeAt, fromCharCode, indexOf, lastIndexOf, replace, split, substring, toLowerCase, toUpperCase`
  - charAt returns a one-letter String (there is no char type)
- length property (not a method as in Java)
- Strings can be specified with "" or ''
- concatenation with + :
  - 1 + 1 is 2, but "1" + 1 is "11"

More about String

- escape sequences behave as in Java: \' \" \& \n \t \\
- converting between numbers and Strings:

```
var count = 10;
var s1 = "" + count; // "10"
var s2 = count + " bananas, ah ah ah!"; // "10 bananas, ah
ah ah!"
var n1 = parseInt("42 is the answer"); // 42
var n2 = parseFloat("booyah"); // NaN                     JS
```

- accessing the letters of a String:

```
var firstLetter = s[0]; // fails in IE
var firstLetter = s.charAt(0); // does work in IE
var lastLetter = s.charAt(s.length - 1);                  JS
```

Splitting strings: split and join

```
var s = "the quick brown fox";
var a = s.split(" "); // ["the", "quick", "brown", "fox"]
a.reverse(); // ["fox", "brown", "quick", "the"]
s = a.join("!"); // "fox!brown!quick!the"
```

JS

- split breaks apart a string into an array using a delimiter
  - can also be used with regular expressions (seen later)
- join merges an array into a single string, placing a delimiter between them

JavaScript

Characteristics

- Case sensitive
- Object oriented
- Produces an HTML document
- Dynamically typed
- Standard operator precedence
- Overloaded operators
- Reserved words

Characteristics

- Division with / is not integer division
- Modulus (%) is not an integer operator
- 5 / 2 yields 2.5
- 5.1 / 2.1 yields 2.4285714285714284
- 5 % 2 yields 1
- 5.1 % 2.1 yields 0.8999999999999995

Characteristics

- " and ' can be used in pairs
- Scope rules for variables
- Strings are very common data types
- Rich set of methods available
- Arrays have dynamic length
- Array elements have dynamic type
- Arrays are passed by reference
- Array elements are passed by value

JavaScript Topics

- code placement
- document.writeln
- document tags
- window.alert
- user input/output
- parseInt and parseFloat
- arithmetic
- arithmetic comparisons
- for loops
- while loops
- do-while loops
- if-else
- variable values in tags
- math library
- switch
- break
- labeled break
- continue
- Booleans

JavaScript Topics

- functions
- random numbers
- rolling dice
- form input
- form output
- submit buttons
- games

- arrays
- searching
- strings
- substrings
- string conversions
- markup methods

JavaScript’s Uses Include:

- “Dynamic” web-pages
  - What’s DHTML? (in a second)
- Image manipulation
  - Swapping, rollovers, slide shows, etc.
- Date, time stuff (e.g. clocks, calendars)
- HTML forms processing
  - Verifying input; writing output to fields
- Cookies

What’s DHTML?

- Purpose: make dynamic / interactive web-pages on the client side
- Use of a collection of technologies together to do this, including
  - Markup language (HTML, XML, etc.)
  - Scripting language (JavaScript, etc.)
  - Presentation language (CSS etc.)

Other References

- CS453 Virtual Lab exercises
- *The Web Wizard's Guide To JavaScript*, Steven Estrella, Addison-Wesley
- *JavaScript for the World Wide Web*, Gesing and Schneider, Peachpit Press
- http://www.w3schools.com/js/
- *www.javascript.com*
- E-books in UVa's Safari On-line Books: http://proquest.safaribooksonline.com/search

Browser Compatability

- Use of:

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

// ends script hiding -->
</script>
```

- “language=“ for pre IE5 and NS6
- Comment for very old browsers (e.g. IE2)
  - BTW, comments in HTML vs. in JavaScript

Organization of JavaScript

- Create functions (non-OO style)
  - Define in header
  - Or load a .js file in header:

    ```
    <script type="text/javascript" language="javascript"
    src="mylib.js">
    ```
- Functions called in <BODY>
  - Often in response to events, e.g.

    ```
    <input type="button"… onclick="myFunc(…);">
    ```
- Global variables

JavaScript

- Programming by example

document.writeln

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<!– Welcome to JavaScript -->
<HEAD>
<TITLE> Welcome to JavaScript </TITLE>
<SCRIPT TYPE="text/javascript">
          document.writeln( "<FONT COLOR='magenta'><H1>Welcome to ",
           "JavaScript Programming!</H1></FONT>" );
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

document.write

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE> Using document.write </TITLE>
<SCRIPT TYPE="text/javascript">
        document.write (   "<H1>Welcome to ");
        document.writeln(   "JavaScript Programming!</H1>" );
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

window.alert

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE> Using window.alert </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>
```

User input/output

```
<SCRIPT TYPE="text/javascript">
        var firstNumber,   // first string entered by user
        secondNumber,   // second string entered by user
        number1,           // first number to add
        number2,           // second number to add
        sum;                // sum of number1 and number2
        // read in first number from user as a string
        firstNumber = window.prompt("Enter first integer", "0" );
        // read in second number from user as a string
        secondNumber = window.prompt( "Enter second integer", "0" );
        // convert numbers from strings to integers
        firstNumber = parseInt(firstNumber);
        number2 = parseInt( secondNumber );
        // add the numbers
        sum = firstNumber + number2;
        // display the results
        document.writeln( "<H1>The sum is " + sum + "</H1>" );
</SCRIPT>
```

Functions

```
<SCRIPT TYPE = "text/javascript">
        var input1 = window.prompt( "Enter first number", "0" );
        var input2 = window.prompt( "Enter second number", "0" );
        var input3 = window.prompt( "Enter third number", "0" );
        var value1 = parseFloat( input1 );
        var value2 = parseFloat( input2 );
        var value3 = parseFloat( input3 );
        var maxValue = maximum( value1, value2, value3 );
        document.writeln( "First number: " + value1 +
       "<BR>Second number: " + value2 +
          "<BR>Third number: " + value3 +
       "<BR>Maximum is: " + maxValue );
        // maximum method definition (called from above)
        function maximum( x, y, z ) {
                return Math.max( x, Math.max( y, z ) );
     }
</SCRIPT>
```

Random Numbers

```
<SCRIPT TYPE="text/javascript">
        var value;
        document.writeln( "<H1>Random Numbers</H1>" +
          "<TABLE BORDER = '1' WIDTH = '50%'><TR>" );
        for ( var i = 1; i <= 20; i++ ) {
          value = Math.floor( 1 + Math.random() * 6 );
          document.writeln( "<TD>" + value + "</TD>" );
          if ( i % 5 == 0 && i != 20 )
            document.writeln( "</TR><TR>" );
        }
        document.writeln( "</TR></TABLE>" );
</SCRIPT>
```

Roll the Die

```
<SCRIPT TYPE="text/javascript">
        var frequency1 = 0, frequency2 = 0,
        frequency3 = 0, frequency4 = 0,
        frequency5 = 0, frequency6 = 0, face;
        // summarize results
        for ( var roll = 1; roll <= 6000; ++roll ) {
          face = Math.floor( 1 + Math.random() * 6 );
          switch ( face ) {
          case 1: ++frequency1; break;
          case 2: ++frequency2; break;
          case 3: ++frequency3; break;
          case 4: ++frequency4; break;
          case 5: ++frequency5; break;
          case 6: ++frequency6; break;
          }
        }
        document.writeln( "<TABLE BORDER = '1' WIDTH = '50%'>" ); .....
```

Rules of Craps

- First roll:
  - 7 or 11 is a win
  - 2, 3, or 12 is a lose
  - otherwise, roll becomes your point
- Subsequent rolls:
  - rolling your point is a win
  - 7 or 11 is a lose
  - otherwise continue to roll

Craps

```
<SCRIPT TYPE="text/javascript">
// variables used to test the state of the game
var WON = 0, LOST = 1, CONTINUE_ROLLING = 2;
// other variables used in program
var firstRoll = true,    // true if first roll
sumOfDice = 0,         // sum of the dice
myPoint = 0,           // point if no win/loss on first roll
gameStatus = CONTINUE_ROLLING;  // game not over yet
```

Craps

```
// process one roll of the dice
function play() {
   if ( firstRoll ) {
      // first roll of the dice
      sumOfDice = rollDice();
      switch ( sumOfDice ) {
         case 7: case 11:
            // win on first roll
            gameStatus = WON;
            document.craps.point.value = ""; // clear point field
            break;
         case 2: case 3: case 12:
            // lose on first roll
            gameStatus = LOST;
            document.craps.point.value = ""; // clear point field
            break;
```

Craps

```
default:
        // remember point
        gameStatus = CONTINUE_ROLLING;
        myPoint = sumOfDice;
        document.craps.point.value = myPoint;
        firstRoll = false;
      }
  }
  else {
    sumOfDice = rollDice();
    if ( sumOfDice == myPoint ) gameStatus = WON;
    else if ( sumOfDice == 7 )  gameStatus = LOST;
  }
```

Craps

```
if ( gameStatus == CONTINUE_ROLLING ) window.alert ("Roll again");
   else {
      if ( gameStatus == WON ) {
         window.alert ("Player wins. " + "Click Roll Dice to play again.");
         document.craps.point.value = " ";
      }
      else {
         window.alert ("Player loses. " + "Click Roll Dice to play again.");
         document.craps.point.value = " ";
      }
  firstRoll = true;
  }
}
```

Craps

```
// roll the dice
function rollDice() {
   var die1, die2, workSum;
   die1 = Math.floor( 1 + Math.random() * 6 );
   die2 = Math.floor( 1 + Math.random() * 6 );
   workSum = die1 + die2;
   document.craps.firstDie.value = die1;
   document.craps.secondDie.value = die2;
   document.craps.sum.value = workSum;
   return workSum;
}
</SCRIPT>
```

Poker Hand

```
<SCRIPT TYPE="text/javascript">
function rand1toN(N) {
   return Math.floor( 1+Math.random()*N );
   }
function dealcard(card) {
   var rank = new Array(0,"A","2","3","4","5","6","7",
        "8","9","T","J","Q","K");
   var suit = new Array(0, "Spades", "Hearts", "Diamonds", "Clubs");
   card[0] = rank[rand1toN(13)];
   card[1] = suit[rand1toN(4)];
}
```

Poker Hand

```
var card = new Array(2);
var player = new Array(10);
var dealer = new Array(10);
for (var i=0; i<=4; i++) {
   dealcard(card);
   player[i*2] = card[0];
   player[i*2+1] = card[1];
   dealcard(card);
   dealer[i*2] = card[0];
   dealer[i*2+1] = card[1];
}
```

Poker Hand

```
document.writeln("<H1> PLAYER </H1>");
document.writeln("<TABLE BORDER='1' >");
for (var i=0; i<=4; i++) {
   document.writeln("<TR><TD><P>" + player[i*2] + "</TD>"
     + "<TD><P>" + player[i*2+1] + "</TD></TR>");
}
document.writeln("</TABLE> </HTML>");
</SCRIPT>
```

Character Processing

```
<SCRIPT TYPE="text/javascript">
var s = "ZEBRA";
var s2 = "AbCdEfG";
document.writeln( "<P> Character at index 0 in '"+
   s + "' is " + s.charAt( 0 ) );
document.writeln( "<BR>Character code at index 0 in '" +
   s + "' is " + s.charCodeAt( 0 ) + "</P>" );
document.writeln( "<P>'" + String.fromCharCode( 87, 79, 82, 68 ) +
   "' contains character codes 87, 79, 82 and 68</P>" );
document.writeln( "<P>'" + s2 + "' in lowercase is '" +
   s2.toLowerCase() + "'" );
document.writeln( "<BR>'" + s2 + "' in uppercase is '" +
   s2.toUpperCase() + "'</P>" );
</SCRIPT>
```

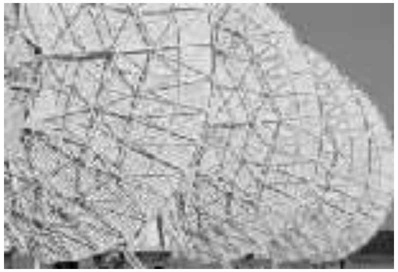
Dates and Times

```
<SCRIPT LANGUAGE = "JavaScript">
var current = new Date();
document.writeln(current);
document.writeln( "<H1>String representations and valueOf</H1>" );
document.writeln( "toString: " + current.toString() +
    "<BR>toLocaleString: " + current.toLocaleString() +
    "<BR>toUTCString: " + current.toUTCString() +
    "<BR>valueOf: " + current.valueOf() );
document.writeln( "<H1>Get methods for local time zone</H1>" );
document.writeln( "getDate: " + current.getDate() +
   "<BR>getDay: " + current.getDay() + "<BR>getMonth: " +
   current.getMonth() + "<BR>getFullYear: " + current.getFullYear() +
   "<BR>getTime: " + current.getTime() + "<BR>getHours: " +
   current.getHours() + "<BR>getMinutes: " + current.getMinutes() +
   "<BR>getSeconds: " + current.getSeconds() +  "<BR>getMilliseconds: " +
   current.getMilliseconds() + "<BR>getTimezoneOffset: " +
   current.getTimezoneOffset() );
```

Dates and Times

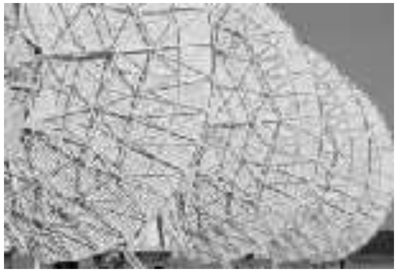
```
document.writeln( "<H1>Specifying arguments for a new Date</H1>" );
var anotherDate = new Date( 1999, 2, 18, 1, 5, 3, 9 );
document.writeln( "Date: " + anotherDate );
document.writeln( "<H1>Set methods for local time zone</H1>" );
anotherDate.setDate( 31 );
anotherDate.setMonth( 11 );
anotherDate.setFullYear( 1999 );
anotherDate.setHours( 23 );
anotherDate.setMinutes( 59 );
anotherDate.setSeconds( 59 );
document.writeln( "Modified date: " + anotherDate );
</SCRIPT>
```

End of Examples



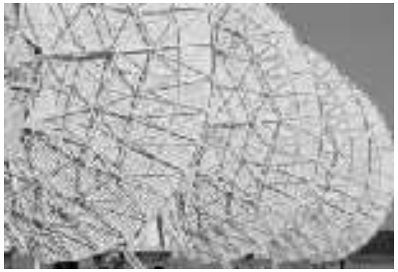
New Perspectives on Creating Web Pages with HTML

Programming with JavaScript



Tutorial Objectives

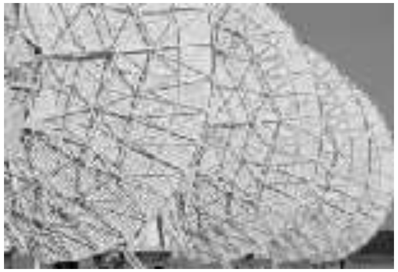
- Learn about the features of JavaScript
- Send output to a Web page
- Work with variables and data
- Work with expressions and operators
- Create a JavaScript function
- Work with arrays and conditional statements
- Learn about program loops



Introduction to JavaScript

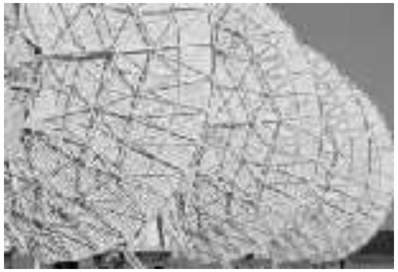
- **JavaScript** is an interpreted programming or script language from Netscape.
- JavaScript is used in Web site development to such things as:
  - automatically change a formatted date on a Web page
  - cause a linked-to-page to appear in a popup window
  - cause text or a graphic image to change during a mouse rollover

Resource: www.whatis.com



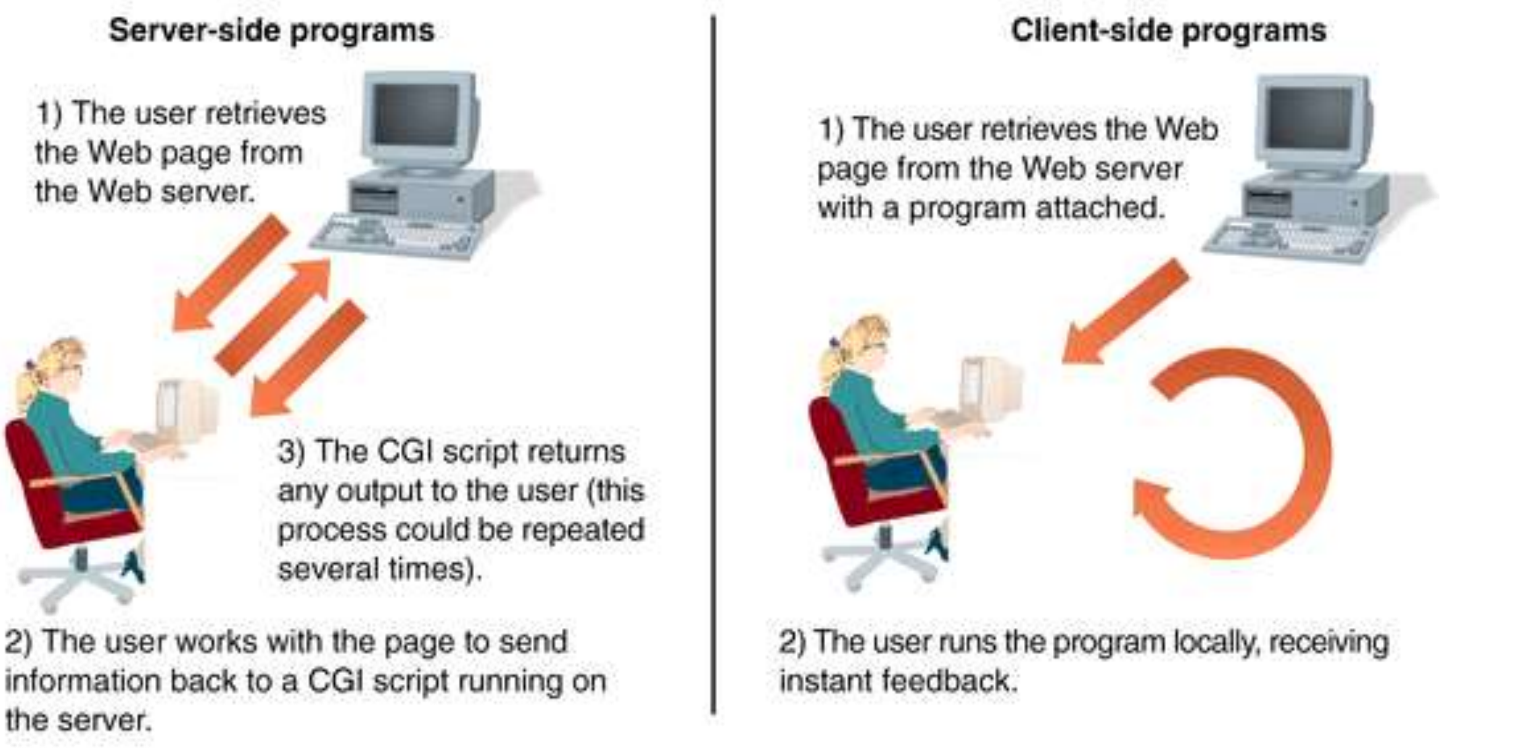
Server-Side and Client-Side Programs

- CGI scripts run from a Web server. There are some disadvantages to this approach:
  - must be connected to the Web server to run the script
  - only the programmer can create or alter the script
  - the Web server’s system administrator can place limitations on how users access the script
  - the system administrator has to be concerned about users continually accessing the server and potentially overloading the system



Server-Side and Client-Side Programming

This figure shows the issues like those previously mentioned that led to the development of programs, or scripts, that could be run from the Web browser (the client).



Server-side programs

1) The user retrieves the Web page from the Web server.

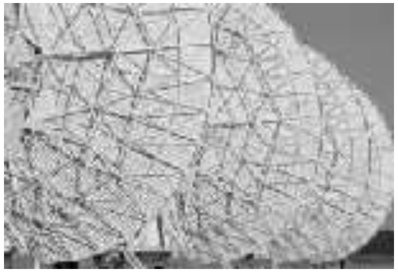
2) The user works with the page to send information back to a CGI script running on the server.

3) The CGI script returns any output to the user (this process could be repeated several times).

Client-side programs

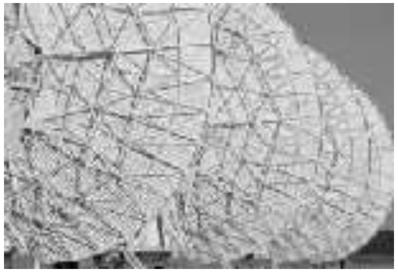
1) The user retrieves the Web page from the Web server with a program attached.

2) The user runs the program locally, receiving instant feedback.



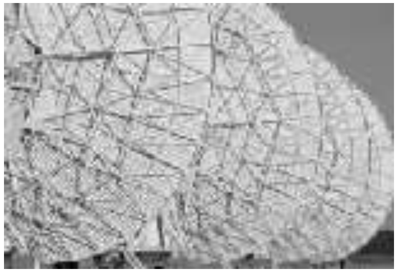
Client-Side Programs

- **Client-side** programs:
  - solve many of the problems associated with server-side scripts
  - computing is distributed over the Web, so that no one server is overloaded with programming requests
  - can be tested locally without first uploading it to a Web server
  - are likely to be more responsive to the user
  - can never completely replace CGI scripts



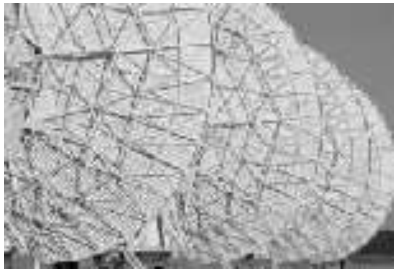
The Development of JavaScript

- JavaScript created by Netscape in 1995
- JScript created by Microsoft in 1996
- IE and Netscape renderings are slightly different
- Standardized by European Computer Manufacturers Association (ECMA)
- http://www.ecma-international. org/publications /standards/Ecma-262.htm
- conforms to the ECMAScript specification



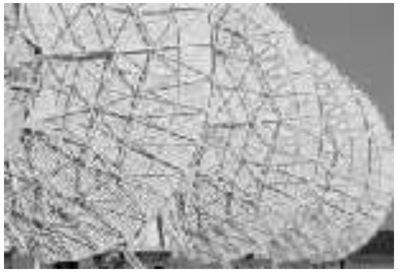
JavaScript

- There are several important differences between Java and JavaScript:
  - users don’t need to work with a developers kit to compile a JavaScript program
  - JavaScript commands can be inserted directly into an HTML file rather than being placed in a separate program file
  - simpler to use
  - JavaScript meets the needs of most users who want to create programmable Web pages



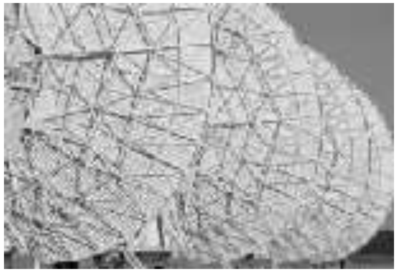
JavaScript

- Internet Explorer supports a slightly different version of JavaScript called **JScript**.
- JScript is identical to JavaScript, but there are some JavaScript commands not supported in JScript, and vice versa.
- Always test JavaScript programs on a variety of Web browsers.



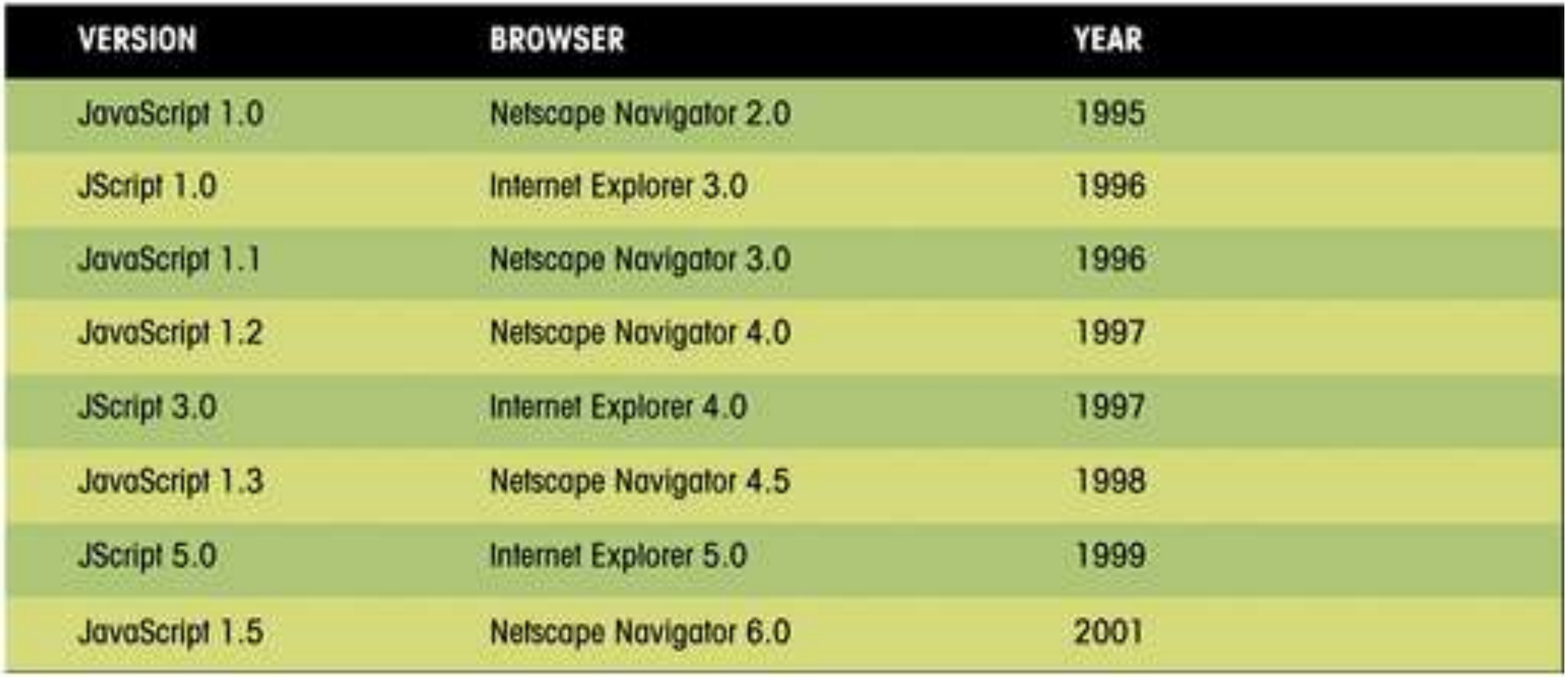
ECMAScript

- The responsibility for the development of a scripting standard has been transferred to an international body called the **European Computer Manufacturers Association (ECMA)**.
- The standard developed by the ECMA is called ECMAScript, though browsers still refer to it as JavaScript.
- The latest version is ECMA-262, which is supported by the major browsers.

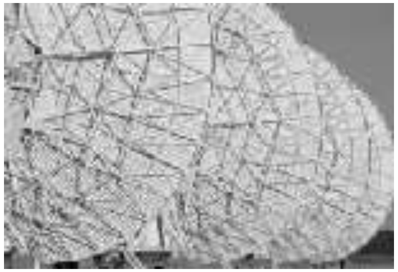


Versions of JavaScript and JScript

This figure lists the versions of JavaScript or JScript and the corresponding browser support.



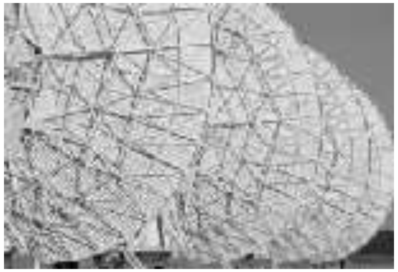
VERSION	BROWSER	YEAR
JavaScript 1.0	Netscape Navigator 2.0	1995
JScript 1.0	Internet Explorer 3.0	1996
JavaScript 1.1	Netscape Navigator 3.0	1996
JavaScript 1.2	Netscape Navigator 4.0	1997
JScript 3.0	Internet Explorer 4.0	1997
JavaScript 1.3	Netscape Navigator 4.5	1998
JScript 5.0	Internet Explorer 5.0	1999
JavaScript 1.5	Netscape Navigator 6.0	2001



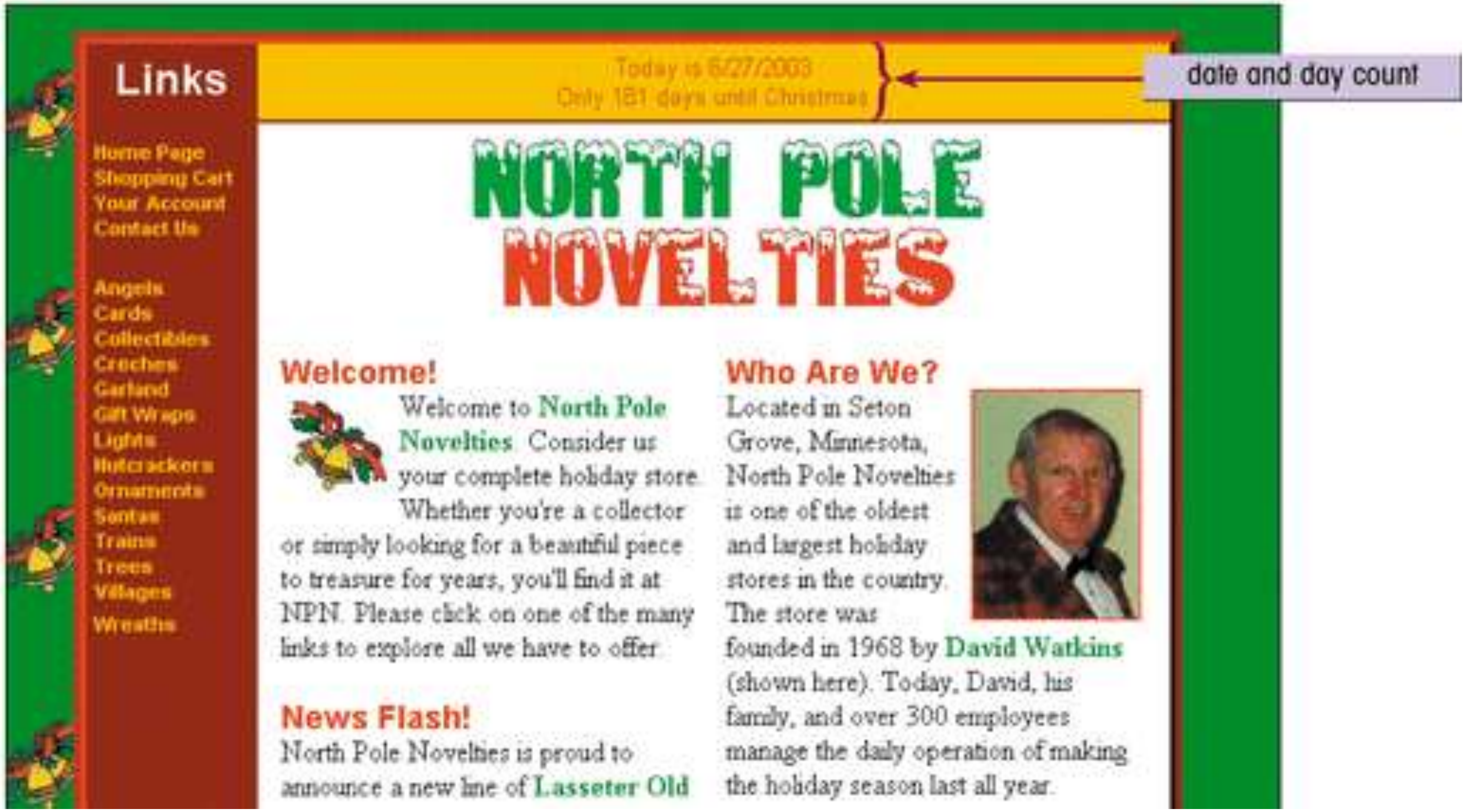
Other Client-Side Programming Languages

- Other client-side programming languages are also available to Web page designers, such as the Internet Explorer scripting language, **VBScript**.
- VBScript is an interpreted script language from Microsoft, it is similar to Netscape’s JavaScript.*

**Resource: www.whatis.com*



Example of Web Site using JavaScript



Today is 6/27/2003
Only 181 days until Christmas

date and day count

Links

- Home Page
- Shopping Cart
- Your Account
- Contact Us

- Angels
- Cards
- Collectibles
- Creches
- Garland
- Gift Wraps
- Lights
- Nutcrackers
- Ornaments
- Santas
- Trains
- Trees
- Villages
- Wreaths

NORTH POLE NOVELTIES

Welcome!

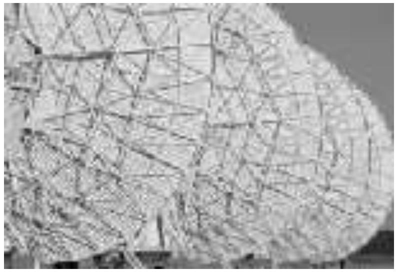
Welcome to North Pole Novelties. Consider us your complete holiday store. Whether you're a collector or simply looking for a beautiful piece to treasure for years, you'll find it at NPN. Please click on one of the many links to explore all we have to offer.

News Flash!

North Pole Novelties is proud to announce a new line of Lasseter Old

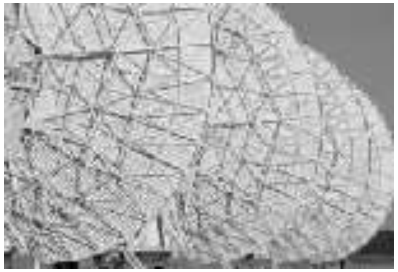
Who Are We?

Located in Seton Grove, Minnesota, North Pole Novelties is one of the oldest and largest holiday stores in the country. The store was founded in 1968 by David Watkins (shown here). Today, David, his family, and over 300 employees manage the daily operation of making the holiday season last all year.



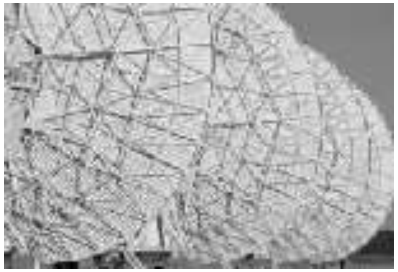
Writing a JavaScript Program

- Before writing a program, it’s a good idea to outline the main tasks you want the program to perform.
- The Web browser runs a JavaScript program when the Web page is first loaded, or in response to an event.
- JavaScript programs can either be placed directly into the HTML file or they can be saved in external files.
  - placing a program in an external file allows you to hide the program code from the user
  - source code placed directly in the HTML file can be viewed by anyone
- More complicated and larger the JavaScript program are usually placed in an external file.



Writing a JavaScript Program Continued

- Your program can be placed anywhere within the HTML file.
- Many programmers favor placing their programs between **`<head>`** tags in order to separate the programming code from the Web page content and layout.
- Some programmers prefer placing programs within the body of the Web page at the location where the program output is generated and displayed.

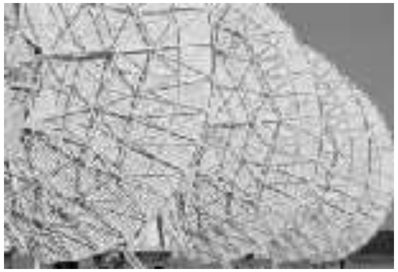


Using the **`<script>`** Tag

- To distinguish JavaScript code from the text in an HTML file, use the **`<script>`** tag.
- The **`<script>`** tag is a two-sided tag that identifies the beginning and end of a client-side program.
- The general syntax for this tag is:

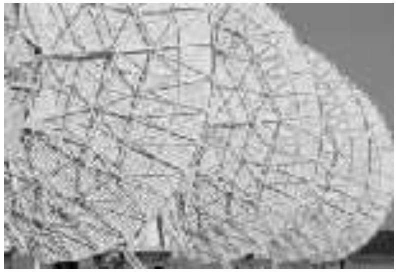
```
<script src=“URL” language=“language”>
  Script command and comments
<script>
```

  - ***URL*** is the location of an external document containing the program code
  - ***language*** is the language that the program is written in
  - the ***src*** attribute is required only if the program is placed in a separate file



Browser and JavaScript

- The default language value is “**JavaScript**.”
- Internet Explorer interprets “**JavaScript**” as being identical to “**JScript**.”
- If you omit the language attribute, the browser assumes that the program is written in JavaScript.

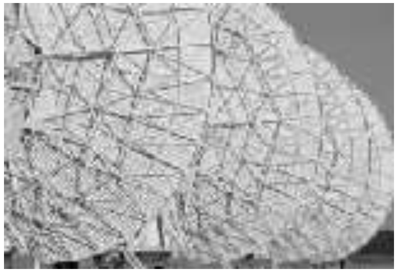


Sending Output to a Web Page

- JavaScript provides two methods to display text on a Web page:
  - the document.write() method
  - the document.writeIn() method
- The syntax for these commands is:

```
document.write(“text”);
and
document.writeIn(“text”);
```

  - ***test*** is a string of characters for display on the Web page.

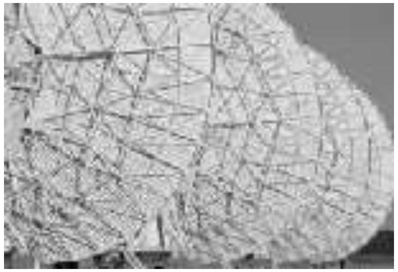


The document.write() and document.writeIn() Methods

- The following method shows how to display the text “Only 45 days until Christmas” in a Web page:

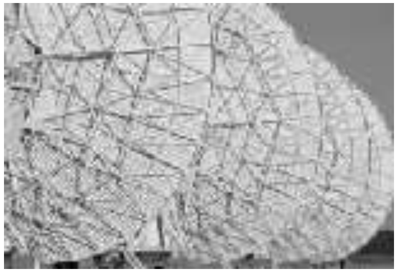
```
document.write (“Only 45 days until Christmas”);
```

  - *document* is an object (the page that the Web browser is accessing)
  - ***write()*** or ***writeIn()*** are actions that can be applied to the document
- The document.write() and document.writeIn() methods reflect the object-oriented nature of the JavaScript language.
- The term “**method**” means an action applied to something existing on a Web page or in the Web browser.



The document.write() and document.writeIn() Methods Continued

- Most of the time you’ll use the document.write() method.
- The document.writeIn() method differs from document.write() in that it attaches a carriage return to the end of each text string sent to the Web page.
  - this becomes relevant only when the text string is formatted with the **`<pre>`** tag for which the browser recognizes the existence of carriage returns

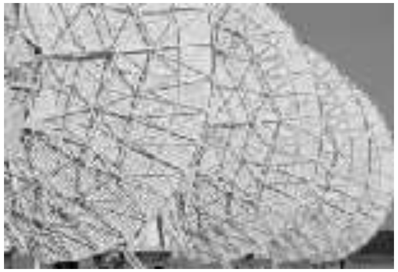


The document.write() and document.writeIn() Methods Continued

- You’re not limited to displaying text; you can also include HTML tags in the text string to format the text and to insert images.
  - for example, the following command displays the text “News Flash!” formatted with the **`<h3>`** header tag:

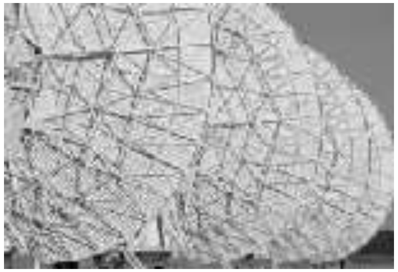
```
document.write(“<h3>News Flash!</h3>”);
```

  - the text string specified by the document.write() method can be enclosed within either double or single quotation marks



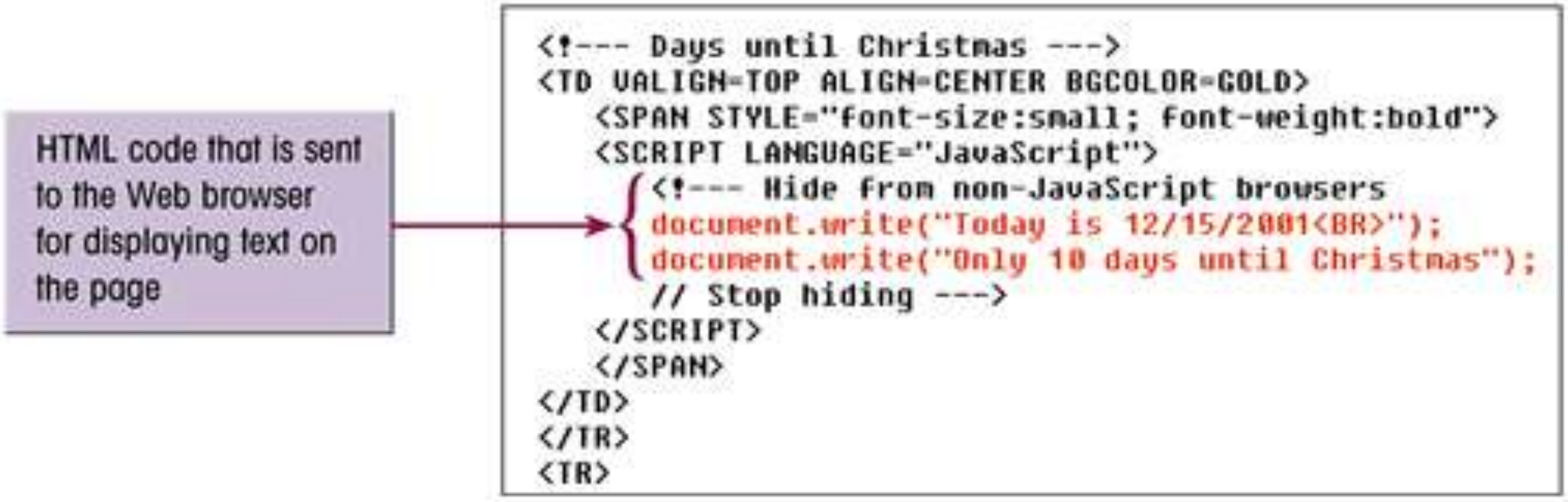
JavaScript Syntax Issues

- JavaScript commands and names are case-sensitive.
- You can use the command “document.write()”, but you cannot replace that command with “Document.Write()” without JavaScript generating an error message.
- JavaScript command lines end with a semicolon to separate it from the next command line in the program.
  - in some situations, the semicolon is optional
  - semicolons are useful to make your code easier to follow and interpret



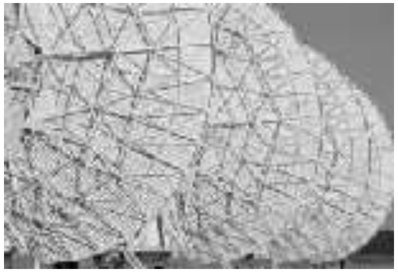
Using JavaScript to Display Text on a Web Page

This figure shows an example of using JavaScript to display text on a Web page. The `<br>` tag is used to create a line break between the date and the number of days until Christmas.



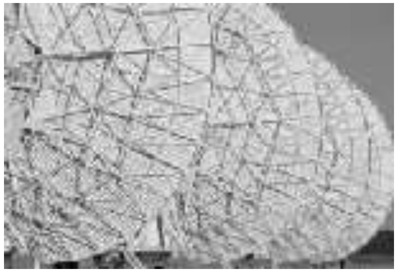
HTML code that is sent to the Web browser for displaying text on the page

```
<!--- Days until Christmas --->
<TD VALIGN=TOP ALIGN=CENTER BGCOLOR=GOLD>
   <SPAN STYLE="font-size:small; font-weight:bold">
   <SCRIPT LANGUAGE="JavaScript">
     <!--- Hide from non-JavaScript browsers
     document.write("Today is 12/15/2001<BR>");
     document.write("Only 10 days until Christmas");
     // Stop hiding --->
   </SCRIPT>
   </SPAN>
</TD>
</TR>
<TR>
```



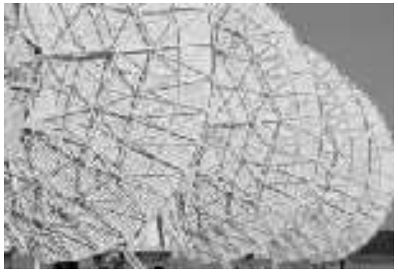
Working with Variables and Data

- A **variable** is a named element in a program that stores information.
- Variables are useful because they can store information created in one part of your program and use that information in another.
- The following restrictions apply to variable names:
  - the first character must be either a letter or an underscore character (_)
  - the remaining characters can be letters, numbers, or underscore characters
  - variable names cannot contain spaces
  - you cannot use words that JavaScript has reserved for other purposes
- Variable names are case-sensitive.



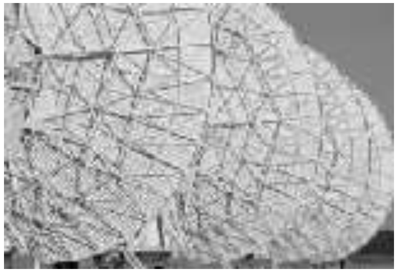
An Example of a Variable

- For example, you can create a variable named “**Year**” to store the value of the current year, and then use the Year variable at different locations in the program: **`Year=2003;`**
- With the Year variable assigned a value, you can use the document.write() method to display the value on the Web page: **`document.write(Year);`**
  - this code displays the text “**2003**” on the Web page
- You can also combine text with the variable value by using a plus symbol (+): **`document.write(“The year is “ + Year);`**
  - this command displays the text “**The year is 2003**” on the Web page



Types of Variables

- JavaScript supports four different types of variables:
  - ***numeric variables*** can be a number, such as 13, 22.5, or -3.14159
  - ***string variables*** is any group of characters, such as “Hello” or “Happy Holidays!”
  - ***Boolean variables*** are variables that accept one of two values, either true or false
  - ***null variables*** is a variable that has no value at all

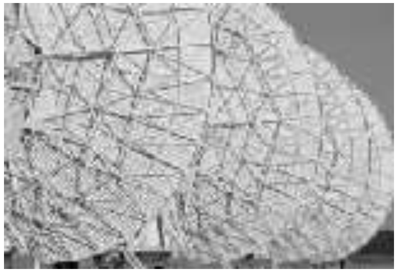


Declaring a Variable

- Before you can use a variable in your program, you need to create it; also known as **declaring a variable**.
- You declare a variable in JavaScript using the **var** command or by assigning the variable a value.
- Any of the following commands is a legitimate way of creating a variable named “Month”:

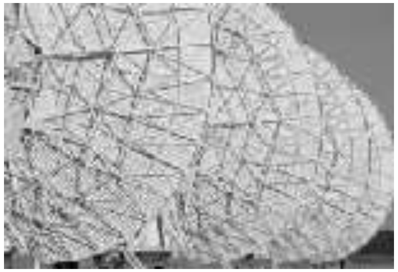
```
var Month;
var Month = “December”;
Month = “December”;
```

  – the first command creates the variable without assigning it a value, while the second and third commands both create the variable and assign it a value



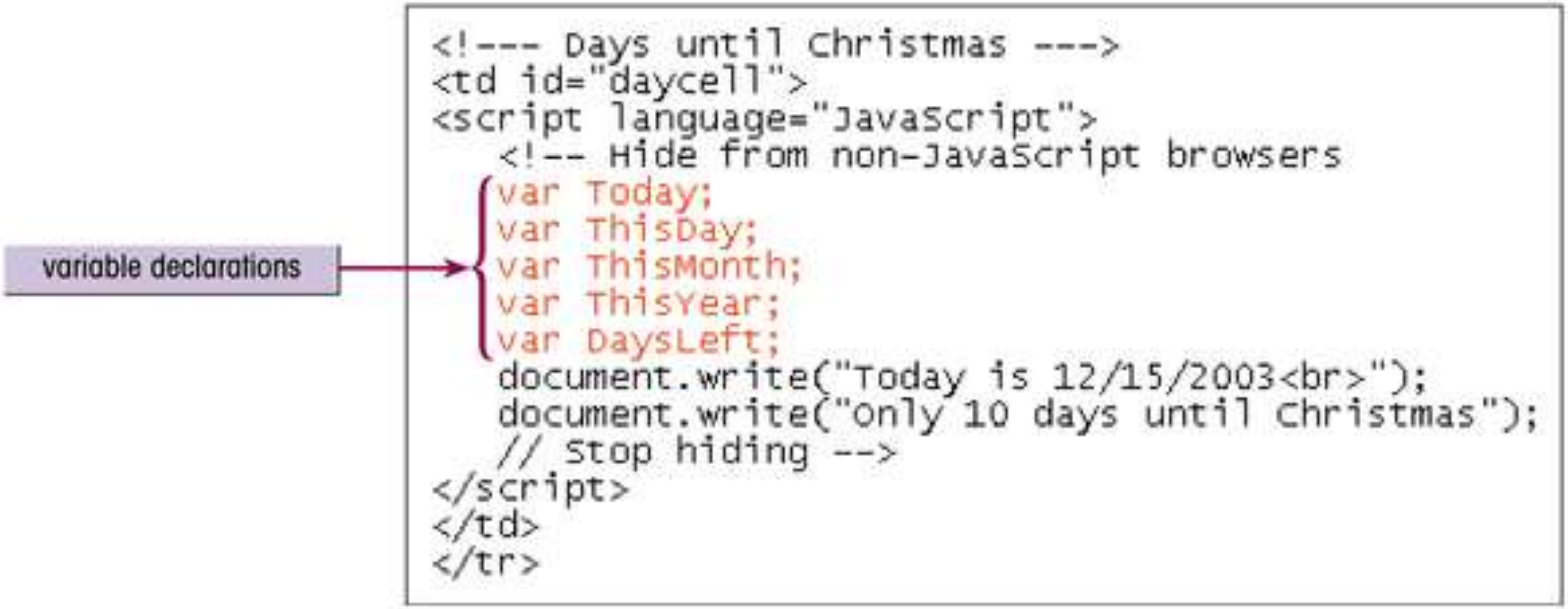
Declaring a Variable Continued

- It’s good programming to include the **`var`** command whenever you create a variable.
- Many Web designers place all of their variable declarations at the beginning of the program along with comments describing the purpose of each variable in the program.
- The following are some JavaScript variables:
  - **Today** - containing information about the current date and time
  - **ThisDay** - storing the current day of the month
  - **ThisMonth** - storing a number indicating the current month
  - **ThisYear** - storing a number indicating the current year
  - **DaysLeft** - storing number of days until a selected date

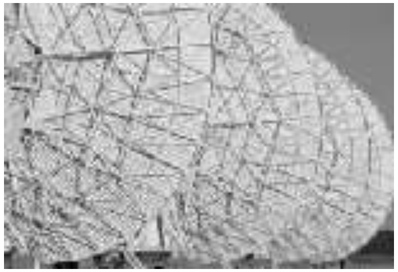


Declaring JavaScript Variables

This figure shows an example of declaring JavaScript variables.



```
<!--- Days until Christmas --->
<td id="daycell">
<script language="JavaScript">
   <!-- Hide from non-JavaScript browsers
   var Today;
   var ThisDay;
   var ThisMonth;
   var ThisYear;
   var DaysLeft;
   document.write("Today is 12/15/2003<br>");
   document.write("Only 10 days until Christmas");
   // Stop hiding -->
</script>
</td>
</tr>
```



Working with Dates

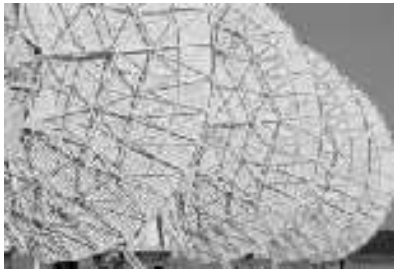
- JavaScript does not provide a date data type.
- JavaScript allows you to create a **date object**, which is an object containing date information.
- There are two ways to create a date object:

```
variable = new Date(“month, day, year,
  hours:minutes: seconds”)
```

or

```
variable = new Date(“month, day, year, minutes,
  seconds”)
```

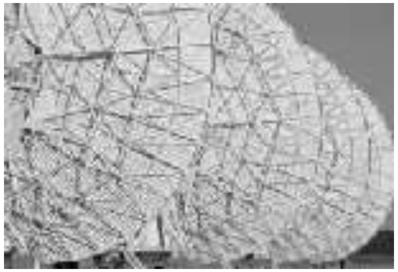
  - ***variable*** is the name of the variable that contains the date information
  - ***month***, ***day***, ***year***, ***hours***, ***minutes***, and ***seconds*** indicate the date and time



Retrieving the Day Value

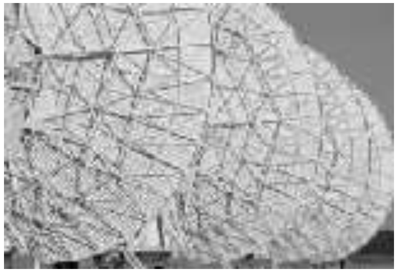
- The **Today** variable has all the date and time information you need.
- JavaScript stores dates and times as the number of milliseconds since 6 p.m on 12/31/69.
- Use built in JavaScript date methods to do calculations.
- For each part of the date, or used in a calculation, you need a date method to retrieve its value.
  - For example, if you want the **ThisDay** variable to store the day of the month. To get that information, apply the **getDate()** method. The general syntax is:

```
DayValue = DateObject.getDate()
```



Retrieving the Day Value

- ***DayValue*** is the name of a variable that contains the day of the month
- ***DateObject*** is a date object or a date variable that contains the complete date and time information

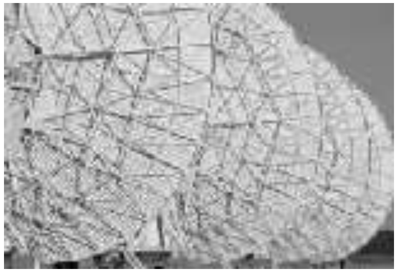


Retrieving the Month Value

- The **getMonth()** method extracts the value of the current month.
- JavaScript starts counting months with 0 for January, you may want to add 1 to the month number returned by the getMonth() method.
- The following JavaScript code extracts the current month number, increases it by 1, and stores it in a variable named **ThisMonth**:

```
ThisMonth = Today.getMonth()+1;
```

  - for a date of October 15, the ThisMonth variable would have a value of “10”.

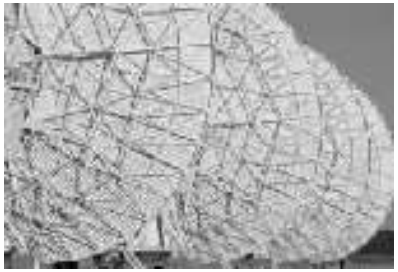


Retrieving the Year Value

- The **getFullYear()** method extracts the year value from the date variable.
- The following code shows how you would store the value of the current year in a variable you name ThisYear:

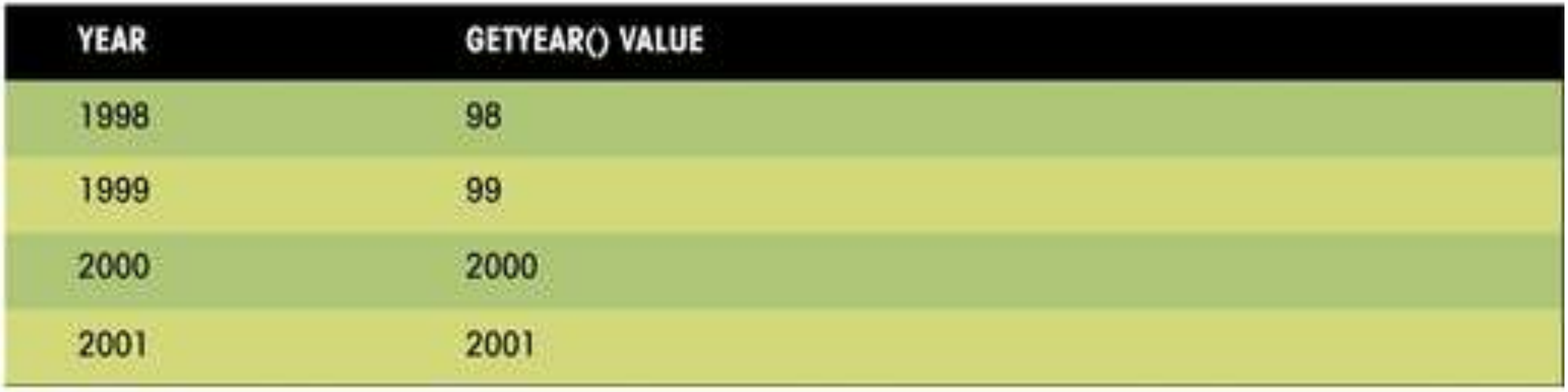
```
ThisYear = Today.getFullYear();
```

  – if the date stored in the Today variable is October 15, 2003, the value of the getFullYear variable is “2003”

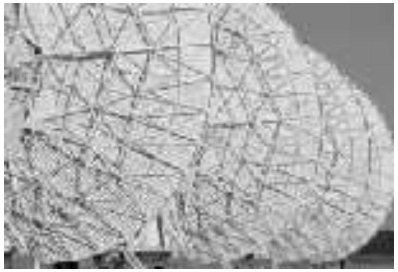


Values of the `getYear()` method from 1998 to 2001

The `getYear()` method returns only the last two digits of the year for years prior to 2000. This figure shows values of the `getYear()` method from 1998 to 2001.

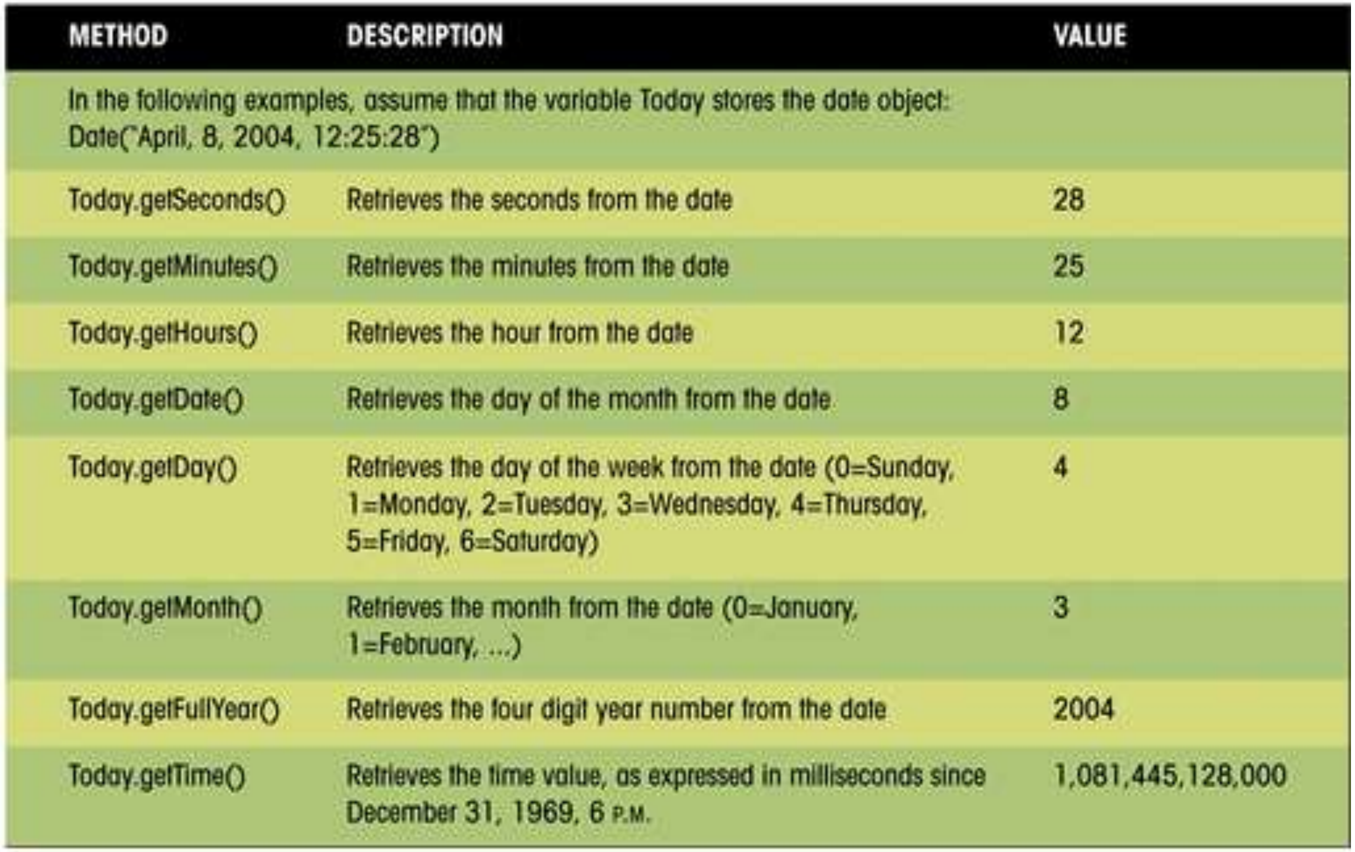


YEAR	GETYEAR() VALUE
1998	98
1999	99
2000	2000
2001	2001

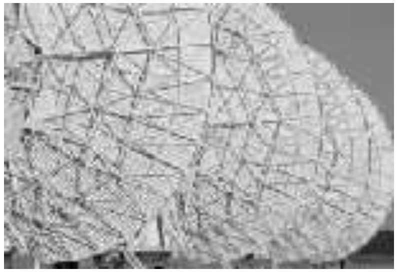


Date Methods

This figure shows most of the date methods you can use with JavaScript.

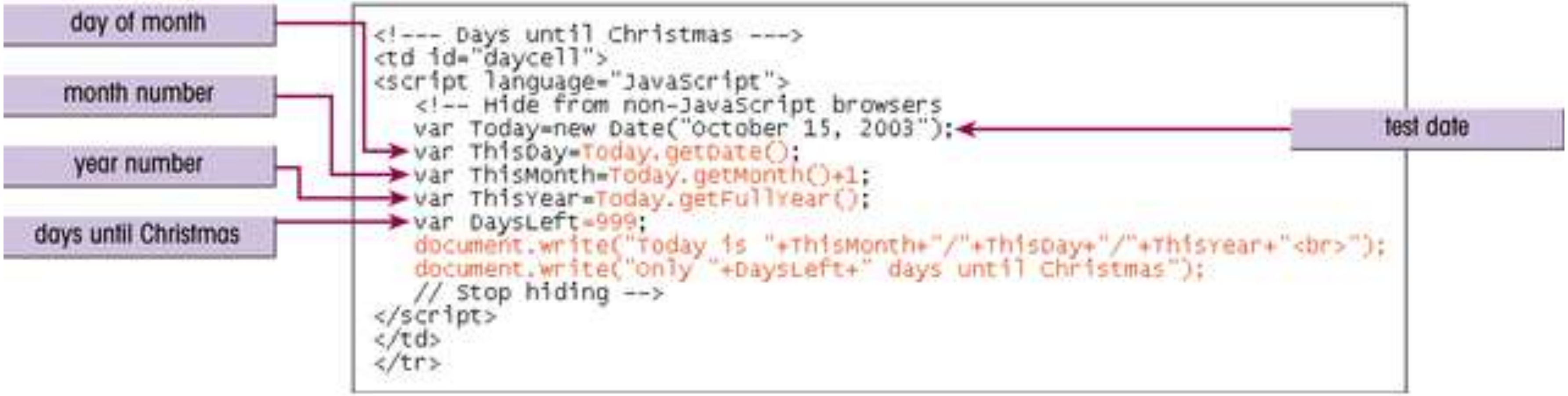


METHOD	DESCRIPTION	VALUE
In the following examples, assume that the variable Today stores the date object: Date("April, 8, 2004, 12:25:28")		
Today.getSeconds()	Retrieves the seconds from the date	28
Today.getMinutes()	Retrieves the minutes from the date	25
Today.getHours()	Retrieves the hour from the date	12
Today.getDate()	Retrieves the day of the month from the date	8
Today.getDay()	Retrieves the day of the week from the date (0=Sunday, 1=Monday, 2=Tuesday, 3=Wednesday, 4=Thursday, 5=Friday, 6=Saturday)	4
Today.getMonth()	Retrieves the month from the date (0=January, 1=February, ...)	3
Today.getFullYear()	Retrieves the four digit year number from the date	2004
Today.getTime()	Retrieves the time value, as expressed in milliseconds since December 31, 1969, 6 P.M.	1,081,445,128,000

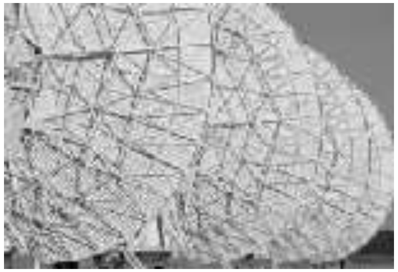


Retrieving Date Information with JavaScript

This figure shows an example of how to retrieve date information with JavaScript.

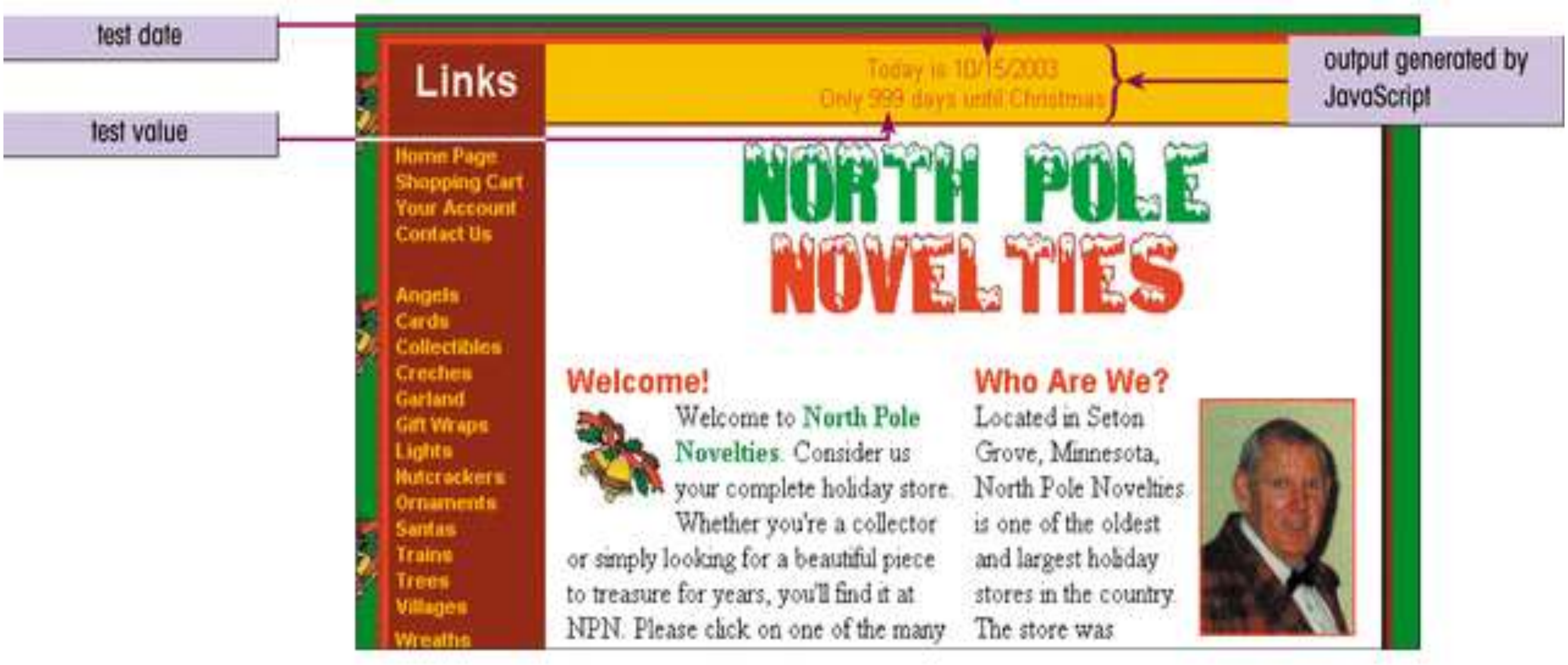


```
<!--- Days until Christmas --->
<td id="daycell">
<script language="JavaScript">
   <!-- Hide from non-JavaScript browsers
   var Today=new Date("October 15, 2003");
   var ThisDay=Today.getDate();
   var ThisMonth=Today.getMonth()+1;
   var ThisYear=Today.getFullYear();
   var DaysLeft=999;
   document.write("Today is "+ThisMonth+"/"+ThisDay+"/"+ThisYear+"<br>");
   document.write("Only "+DaysLeft+" days until Christmas");
   // Stop hiding -->
</script>
</td>
</tr>
```



Displaying the Date Values

This figure shows an example of a website displaying date values.



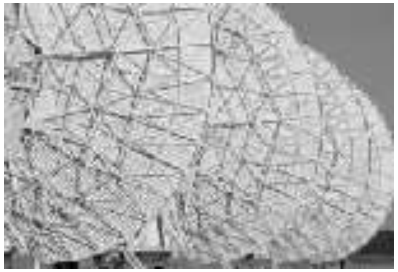
Today is 10/15/2003
Only 999 days until Christmas

Welcome!

Welcome to North Pole Novelties. Consider us your complete holiday store. Whether you're a collector or simply looking for a beautiful piece to treasure for years, you'll find it at NPN. Please click on one of the many

Who Are We?

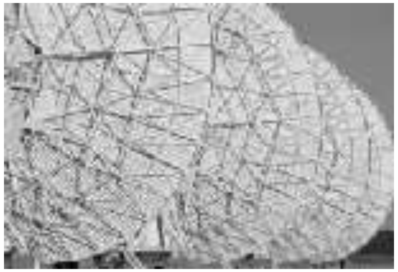
Located in Seton Grove, Minnesota, North Pole Novelties is one of the oldest and largest holiday stores in the country. The store was



Working with Expressions and Operators

- **Expressions** are JavaScript commands that assign values to variables.
  - for example, use the expression, DaysLeft=999, to assign to the value 999 to the DaysLeft variable
- Expressions are created using variables, values, and **operators** (elements that perform actions within the expression).
- One of the most commonly used operators is the **+ operator**, which performs the action of adding or combining two elements.
  - for example, use the plus operator in a program with the following command:

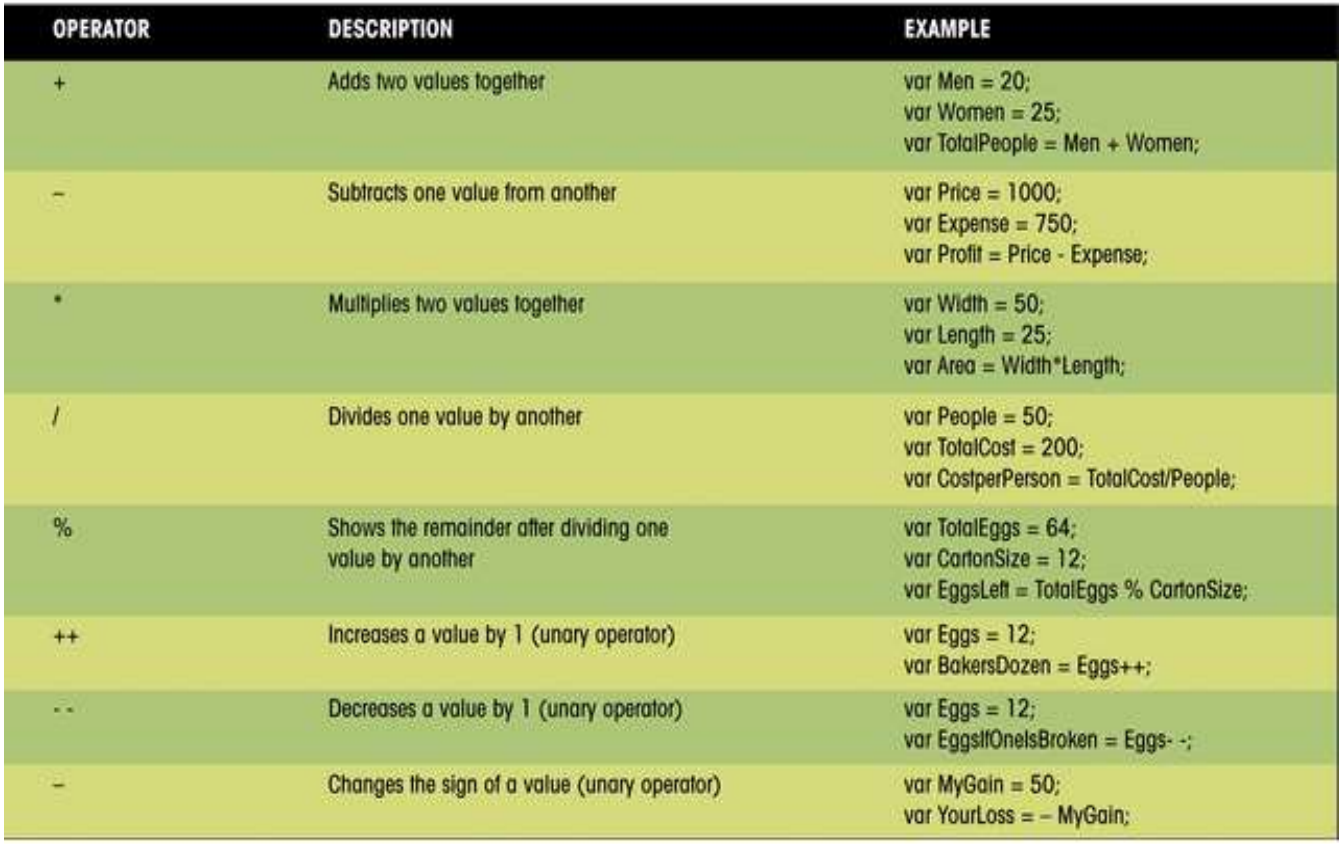
```
var ThisMonth = Today.getMonth()+1;
```



Arithmetic Operators

The + operator belongs to a group of operators called arithmetic operators, which perform simple mathematical calculations.

This figure lists some of the arithmetic operators and gives examples of how they work.



OPERATOR	DESCRIPTION	EXAMPLE

+	Adds two values together	var Men = 20;
		var Women = 25;
		var TotalPeople = Men + Women;

–	Subtracts one value from another	var Price = 1000;
		var Expense = 750;
		var Profit = Price - Expense;

*	Multiplies two values together	var Width = 50;
		var Length = 25;
		var Area = Width*Length;

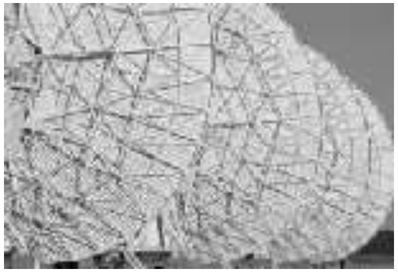
/	Divides one value by another	var People = 50;
		var TotalCost = 200;
		var CostperPerson = TotalCost/People;

%	Shows the remainder after dividing one value by another	var TotalEggs = 64;
		var CartonSize = 12;
		var EggsLeft = TotalEggs % CartonSize;

++	Increases a value by 1 (unary operator)	var Eggs = 12;
		var BakersDozen = Eggs++;

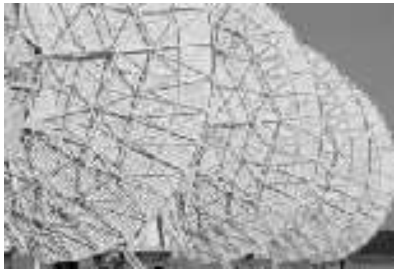
- -	Decreases a value by 1 (unary operator)	var Eggs = 12; var EggsIfOneIsBroken = Eggs- -;

–	Changes the sign of a value (unary operator)	var MyGain = 50;
		var YourLoss = – MyGain;



Operators

- **Binary operators** work on two elements in an expression.
- **Unary operators** work on only one variable.
  - unary operators include: the increment (++), decrement (--), and negation (-) operators.
- The **increment operator** can be used to increase the value of variable by 1.

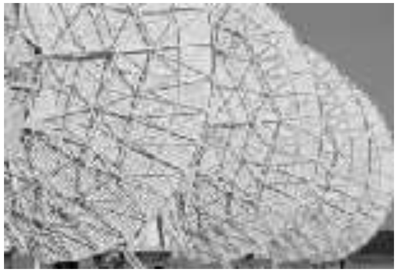


An Example of the Increment Operator

- In the following code, an **increment operator** is used to increase the value of the ***x*** variable by one.

```
x = 100;
y = x++;
```

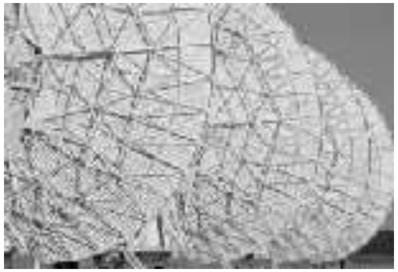
  - thus, after both commands are run, the value of the ***x*** variable is 100 and the value of the ***y*** variable is 101



An Example of the Decrement Operator

- The **decrement operator** has the opposite effect, reducing the value of a variable by 1.
- The following JavaScript code assigns the value 100 to the ***x*** variable and 99 to the ***y*** variable:

```
x = 100;
y = x--;
```

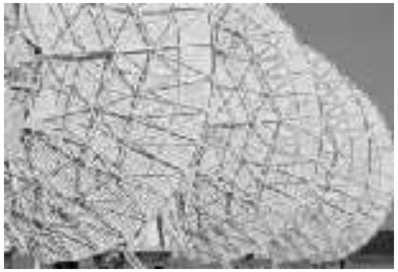


An Example of the Negation Operator

- The **negation operator** changes the sign of a variable:

```
x = -100;
y = -x;
```

  - the value of the ***x*** variable is –100, and the value of the **y** variable is opposite that, or 100

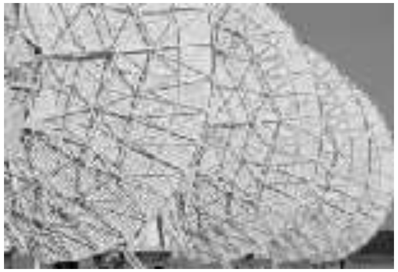


Assignment Operators

- Expressions assign values using **assignment operators**.
  - the most common assignment operator is the equals (=) sign
- JavaScript provides additional assignment operators that manipulate elements in an expression and assign values within a single operation.
  - one of these is the += operator
- In JavaScript, the following two expressions create the same results:

```
x = x + y;
x += y
```

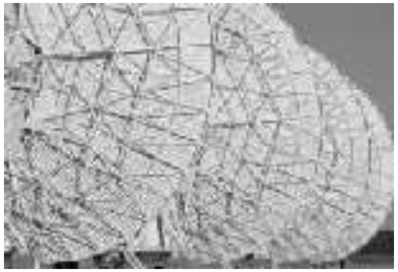
  - in both expressions, the value of the **x** variable is added to the value of the ***y*** variable and then the new variable is stored back into the x variable



Assignment Operators Continued

- An assignment operator also can be used with numbers to increase a variable by a specific amount.
  - for example, to increase the value of the ***x*** variable by 2, you can use either of the following two expressions:

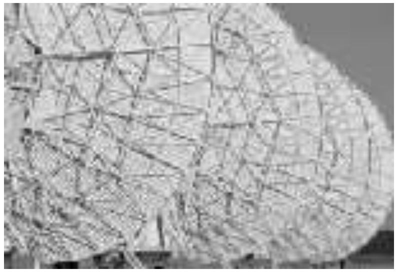
```
x = x + 2;
x += 2
```



The += Operator

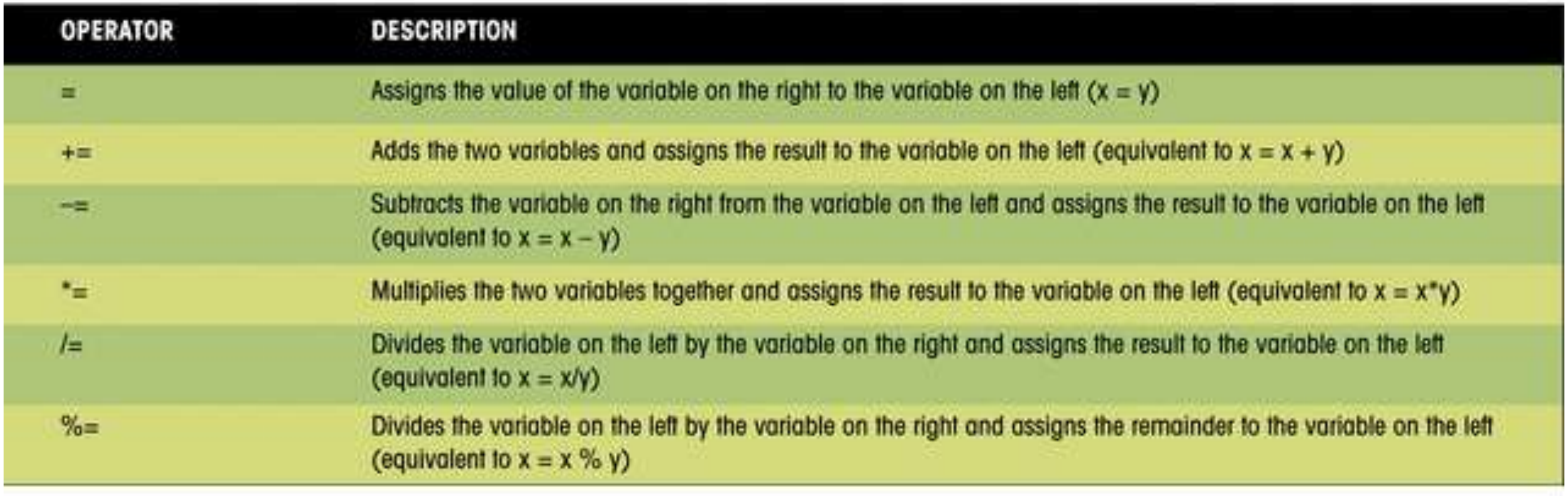
- A common use of the **+= operator** is to create extended text strings.
  - for example, if you have a text string that covers several lines, you may find it difficult to store the text in a variable using a single command. However, you can do so in the following manner:

```
quote = “To be or not to be.”;
quote +=“That is the question. “;
quote +=“Whether tis nobler of the mind
  to suffer the lings and arrows of
  outrageous fortune, “;
. . .
```

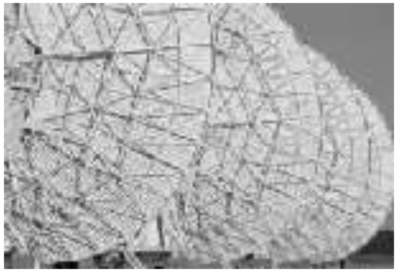


Assignment Operators

This figure shows additional assignment operators that can be used. Assignment operators allow you to create expressions that are both efficient and compact.



OPERATOR	DESCRIPTION
=	Assigns the value of the variable on the right to the variable on the left (x = y)
+=	Adds the two variables and assigns the result to the variable on the left (equivalent to x = x + y)
−=	Subtracts the variable on the right from the variable on the left and assigns the result to the variable on the left (equivalent to x = x − y)
*=	Multiplies the two variables together and assigns the result to the variable on the left (equivalent to x = x*y)
/=	Divides the variable on the left by the variable on the right and assigns the result to the variable on the left (equivalent to x = x/y)
%=	Divides the variable on the left by the variable on the right and assigns the remainder to the variable on the left (equivalent to x = x % y)

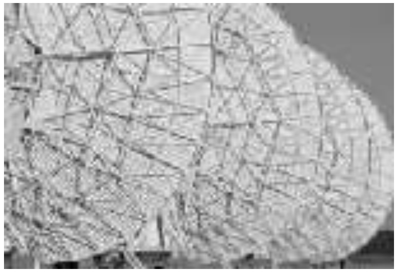


The Math Object and Math Methods

- Another way of performing a calculation is to use on the JavaScript built-in Math methods.
- These methods are applied to an object called the **Math object**.
- The syntax for applying a Math method is:

```
value = Math.method(variable);
```

  - ***method*** is the method you'll apply to a variable
  - ***value*** is the resulting value

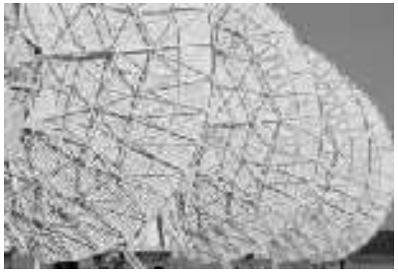


An Example of a Math Method

- For example, to calculate the absolute value of a variable named NumVar, you use the “**abs**” method as follows:

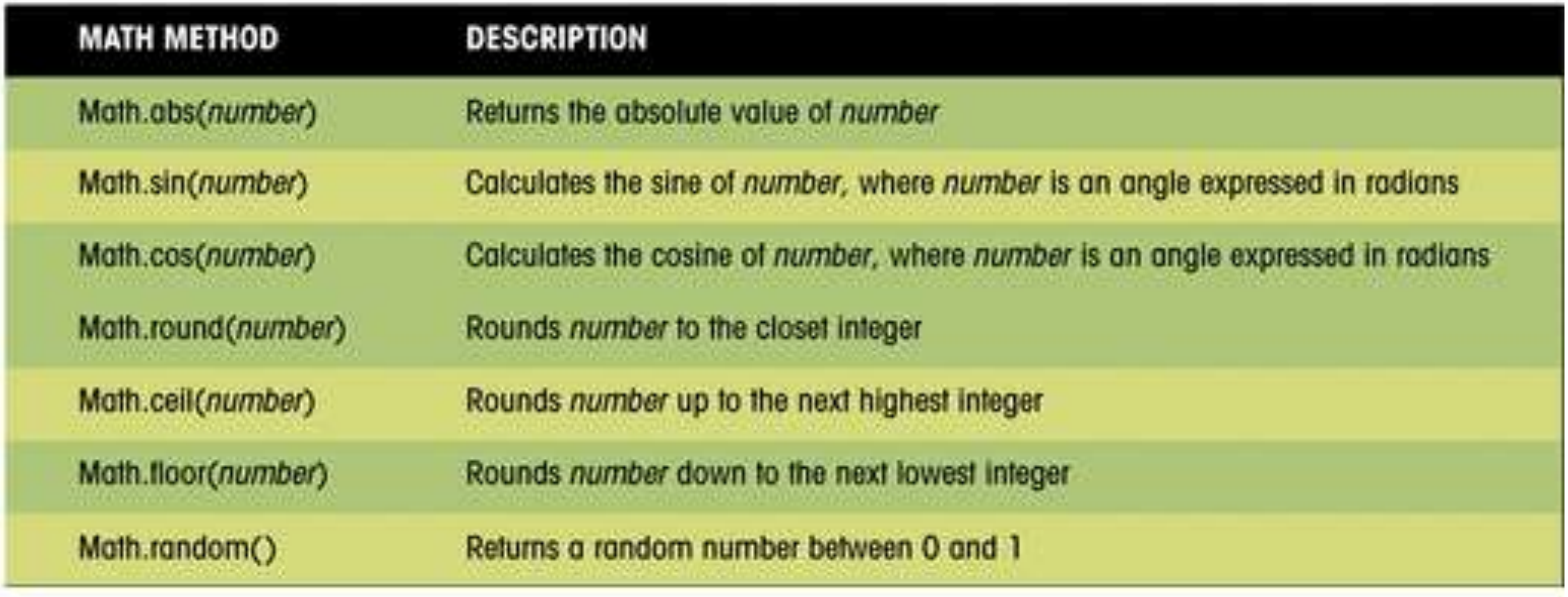
```
AbsValue = Math.abs(NumVar);
```

  – the value of the AbsValue variable is set to the absolute value of the NumVar variable

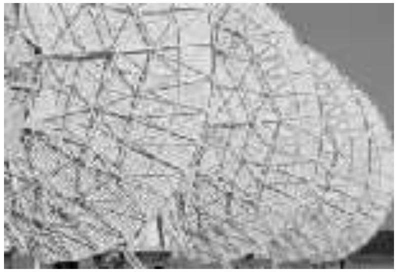


Math Methods

This figure lists some additional math methods supported by JavaScript. Case is important with JavaScript commands. You must type "Math" (with an uppercase M) instead of "math" when using these commands.



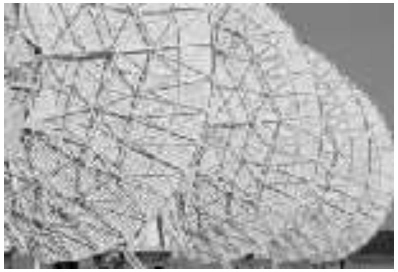
MATH METHOD	DESCRIPTION
Math.abs(*number*)	Returns the absolute value of *number*
Math.sin(*number*)	Calculates the sine of *number*, where *number* is an angle expressed in radians
Math.cos(*number*)	Calculates the cosine of *number*, where *number* is an angle expressed in radians
Math.round(*number*)	Rounds *number* to the closet integer
Math.ceil(*number*)	Rounds *number* up to the next highest integer
Math.floor(*number*)	Rounds *number* down to the next lowest integer
Math.random()	Returns a random number between 0 and 1



Creating JavaScript Functions

- A **function** is a series of commands that performs an action or calculates a value.
- A function consists of the **function name**, which identifies it; **parameters**.
- Parameters are values used by the function; and a set of commands that are run when the function is used.
- Not all functions require parameters.
- The general syntax of a JavaScript function is:

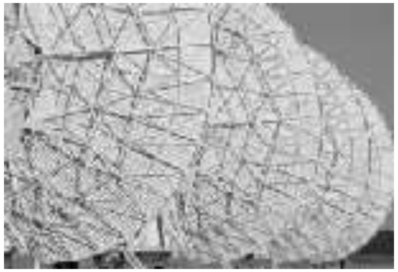
```
function function_name(parameters) {
  JavaScript commands
}
```



Creating JavaScript Functions Continued

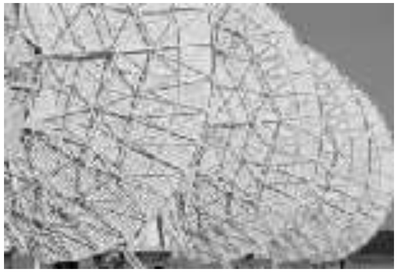
  - ***function_name*** is the name of the function
  - ***parameters*** are the values sent to the function
  - ***JavaScript commands*** are the actual commands and expressions used by the function

- Curly braces } are used to mark the beginning and end of the commands in the function.
- The group of commands set off by the curly braces is called a **common block**.



Creating JavaScript Functions Continued

- Function names are case-sensitive.
- The function name must begin with a letter or underscore (_) and cannot contain any spaces.
- There is no limit to the number of function parameters that a function may contain.
- The parameters must be placed within parentheses, following the function name, and the parameters must be separated by commas.



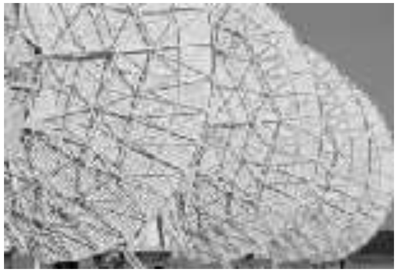
Performing an Action with a Function

- The following function displays a message with the current date:

```
function ShowDate(date) {
  document.write(“Today is “ + date + “<br>”);
}
```

  – the function name is ShowDate, and it has one parameter, date

  – there is one line in the function’s command block, which displays the current date along with a text string

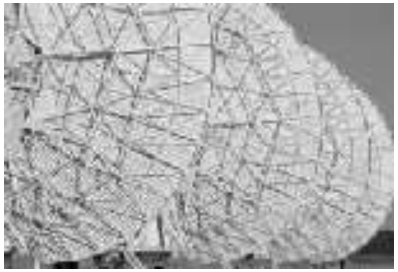


Performing an Action with a Function Continued

- To run a function, insert a JavaScript command containing the function name and any parameters it requires, this process is known as **calling** a function.
- To call the ShowDate function, enter the following commands:

```
var Today = “3/25/2003”;
ShowDate(Today);
```

  - the first command creates a variable named “**Today**” and assigns it the text string, “**3/25/2003**”
  - the second command runs the ShowDate function, using the value of the Today variable as a parameter
  - result is “**Today is 3/25/2003**”

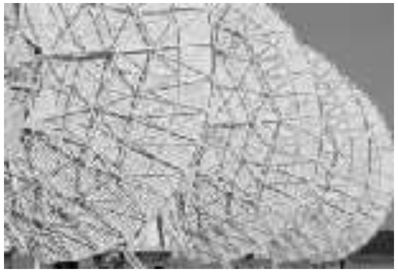


Returning a Value from a Function

- To use a function to calculate a value use the **return** command along with a variable or value.
- The following is a example using the Area function:

```
function Area(Width, Length) {
   var Size = Width*Length;
   return Size;
}
```

  - the ***Area*** function calculates the area of a rectangular region and places the value in a variable named “**Size**”
  - the value of the Size variable is returned by the function

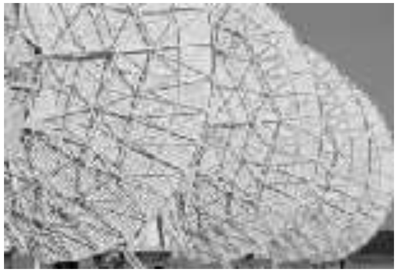


The Area Function

- A simple JavaScript program is:

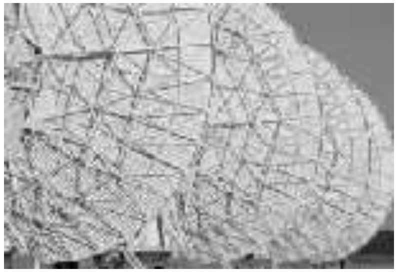
```
var x = 8;
var y = 6;
var z = Area(x,y);
```

  - the first two commands assign the values 8 and 6 to the x and y variables
  - the values of both of these variables are then sent to the Area function, corresponding to the Width and Length parameters
  - the ***Area*** function uses these values to calculate the area, assigning the value to the z variable
  - result is “**48**”, which is assigned to the value of the z variable



Placing a Function in an HTML File

- Where you place a function in the HTML file is important.
- The function definition must be placed before the command that calls the function.
- One programming convention is to place all of the function definitions used between the **`<head>`** and **`</head>`** tags.
- A function is executed only when called by another JavaScript command.



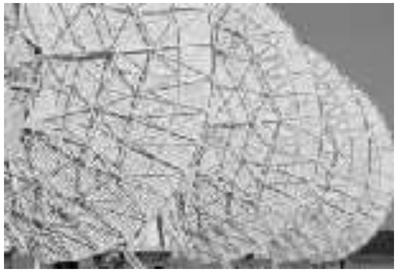
Placing a Function in an HTML File

- To use a function on several Web pages, place the function in a separate file and access the function from each Web page.
- To access the externally located function, insert the following command in the head section of the HTML file:

```
<script src=“URL”
  language=“JavaScript”></script>
```

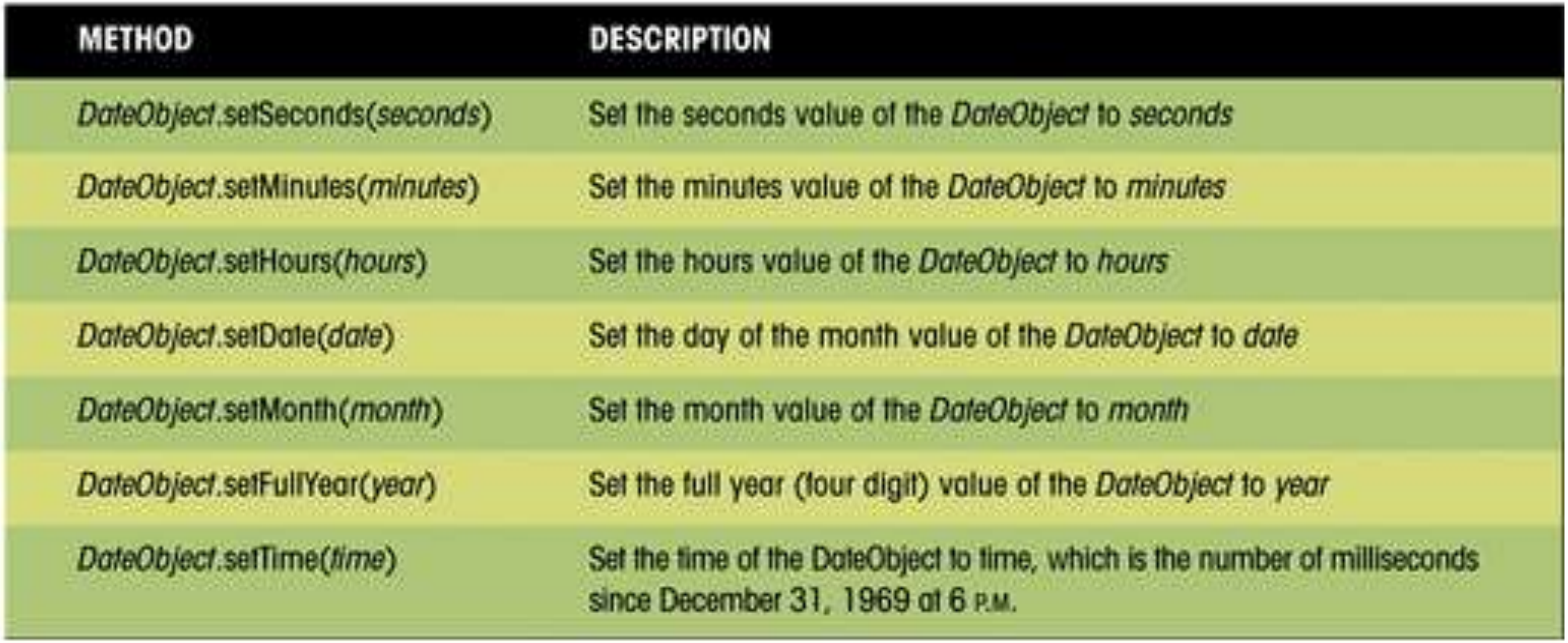
  - ***URL*** is the filename and location of the external file containing the functions

- It’s common practice for JavaScript programmers to create libraries of functions located in external files.

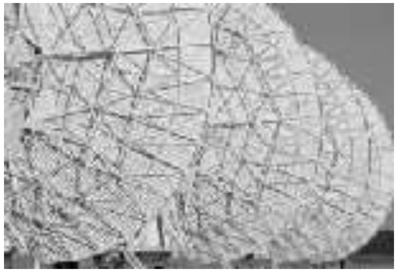


Setting Date Values

This figure shows additional JavaScript functions that allow you to set or change the values of date objects.



METHOD	DESCRIPTION
DateObject.setSeconds(*seconds*)	Set the seconds value of the *DateObject* to *seconds*
DateObject.setMinutes(*minutes*)	Set the minutes value of the *DateObject* to *minutes*
DateObject.setHours(*hours*)	Set the hours value of the *DateObject* to *hours*
DateObject.setDate(*date*)	Set the day of the month value of the *DateObject* to *date*
DateObject.setMonth(*month*)	Set the month value of the *DateObject* to *month*
DateObject.setFullYear(*year*)	Set the full year (four digit) value of the *DateObject* to *year*
DateObject.setTime(*time*)	Set the time of the DateObject to time, which is the number of milliseconds since December 31, 1969 at 6 P.M.



The XMASDAYS Function

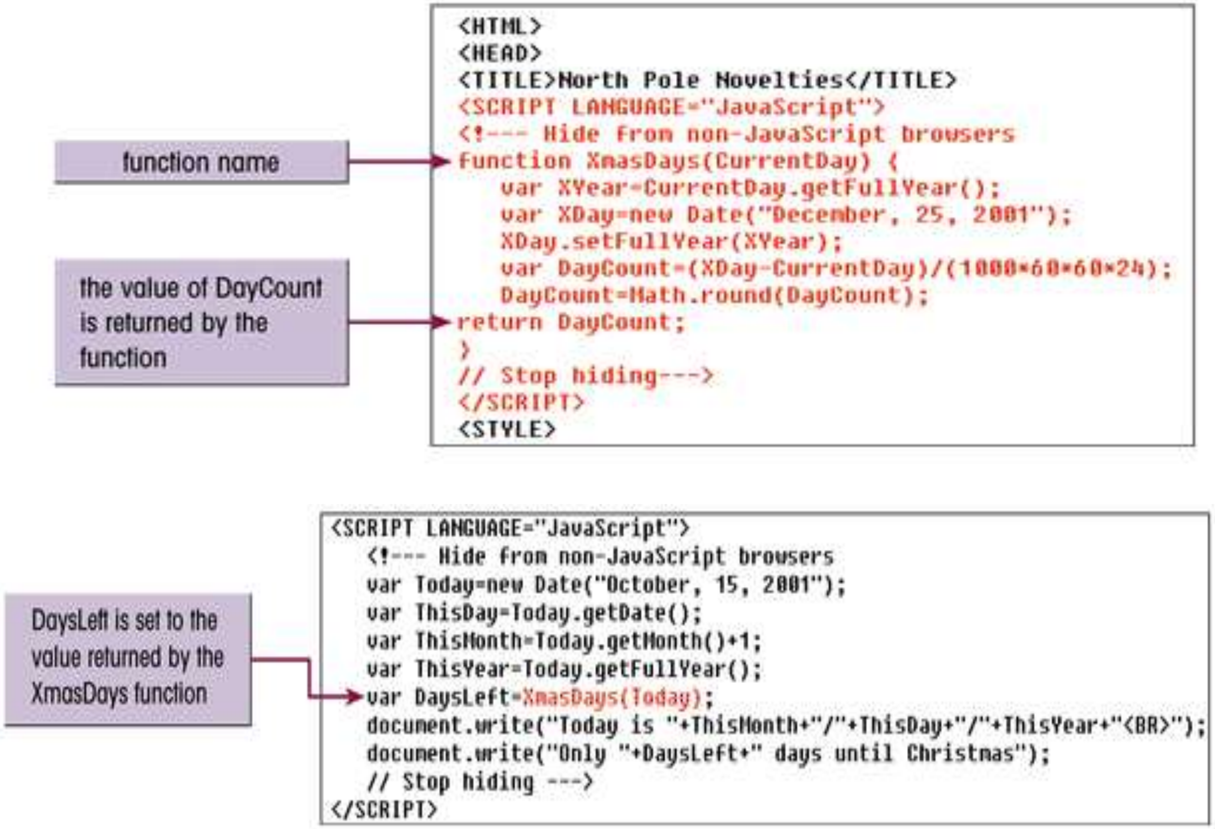
This figure show an example of the XMASDAYS function.

The function has three variables:

XYear: The current year

XDay: The date of Christmas. The initial value of this variable is the date "December 25, 2003."

DayCount: The number of days between current date and December 25. This is the value that is returned by the function.



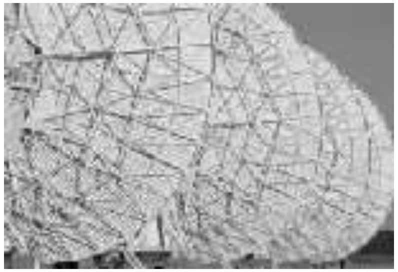
function name

the value of DayCount is returned by the function

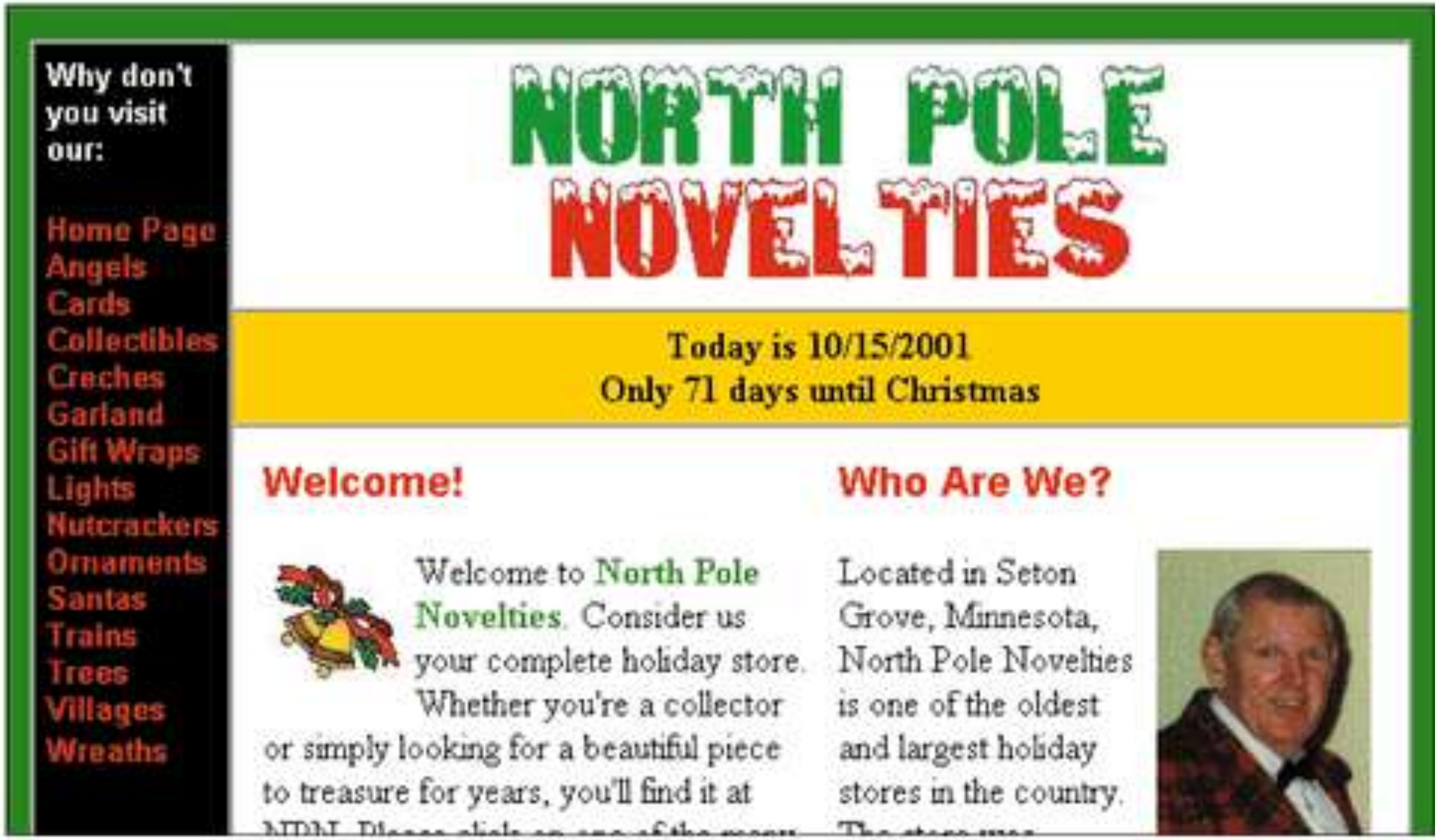
```
<HTML>
<HEAD>
<TITLE>North Pole Novelties</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--- Hide from non-JavaScript browsers
function XmasDays(CurrentDay) {
   var XYear=CurrentDay.getFullYear();
   var XDay=new Date("December, 25, 2001");
   XDay.setFullYear(XYear);
   var DayCount=(XDay-CurrentDay)/(1000*60*60*24);
   DayCount=Math.round(DayCount);
return DayCount;
}
// Stop hiding--->
</SCRIPT>
<STYLE>
```

DaysLeft is set to the value returned by the XmasDays function

```
<SCRIPT LANGUAGE="JavaScript">
   <!--- Hide from non-JavaScript browsers
   var Today=new Date("October, 15, 2001");
   var ThisDay=Today.getDate();
   var ThisMonth=Today.getMonth()+1;
   var ThisYear=Today.getFullYear();
   var DaysLeft=XmasDays(Today);
   document.write("Today is "+ThisMonth+"/"+ThisDay+"/"+ThisYear+"<BR>");
   document.write("Only "+DaysLeft+" days until Christmas");
   // Stop hiding --->
</SCRIPT>
```



Example of Web Page with "Days Until Christmas"



Why don't you visit our:

Home Page
Angels
Cards
Collectibles
Creches
Garland
Gift Wraps
Lights
Nutcrackers
Ornaments
Santas
Trains
Trees
Villages
Wreaths

NORTH POLE NOVELTIES

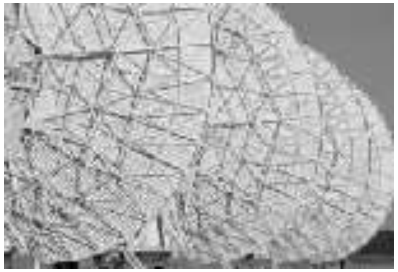
Today is 10/15/2001
Only 71 days until Christmas

Welcome!

Welcome to North Pole Novelties. Consider us your complete holiday store. Whether you're a collector or simply looking for a beautiful piece to treasure for years, you'll find it at NPN. Please click on one of the menu

Who Are We?

Located in Seton Grove, Minnesota, North Pole Novelties is one of the oldest and largest holiday stores in the country. The store was

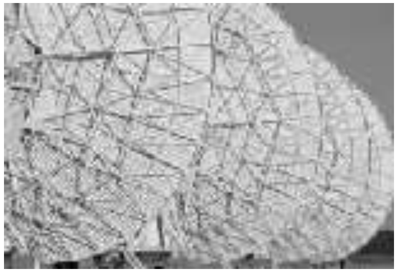


Working with Conditional Statements

- A **conditional statement** is one that runs only when specific conditions are met i.e. If statement.
- An If statement has the following general syntax:

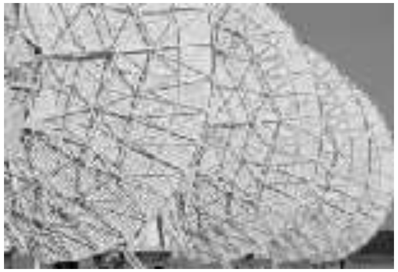
```
 if (condition) {
    JavaScript Commands
 }
```

  - *condition* is an expression that is either true or false
    - if the condition is true, the JavaScript Commands in the command block are executed
    - if the condition is not true, then no action is taken



Comparison, Logical, and Conditional Operators

- To create a condition in JavaScript, you need one of three types of operators: comparison operators, logical operators, and conditional operators
  - a **comparison operator** compares the value of one element with that of another, which creates a **Boolean expression** that is either true or false
  - a **logical operator** connects two or more Boolean expressions
  - a **conditional operator** tests whether a specific condition is true and returns one value if the condition is true and a different value if the condition is false



An Example of Boolean Expressions

- Here are two examples of Boolean expressions:

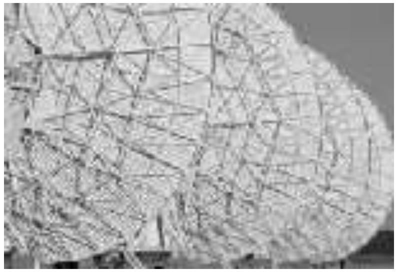
  `x < 100;`

  – if ***x*** is less than 100, this expression returns the value true; however, if ***x*** is 100 or greater, the expression is false

  `y == 20;`

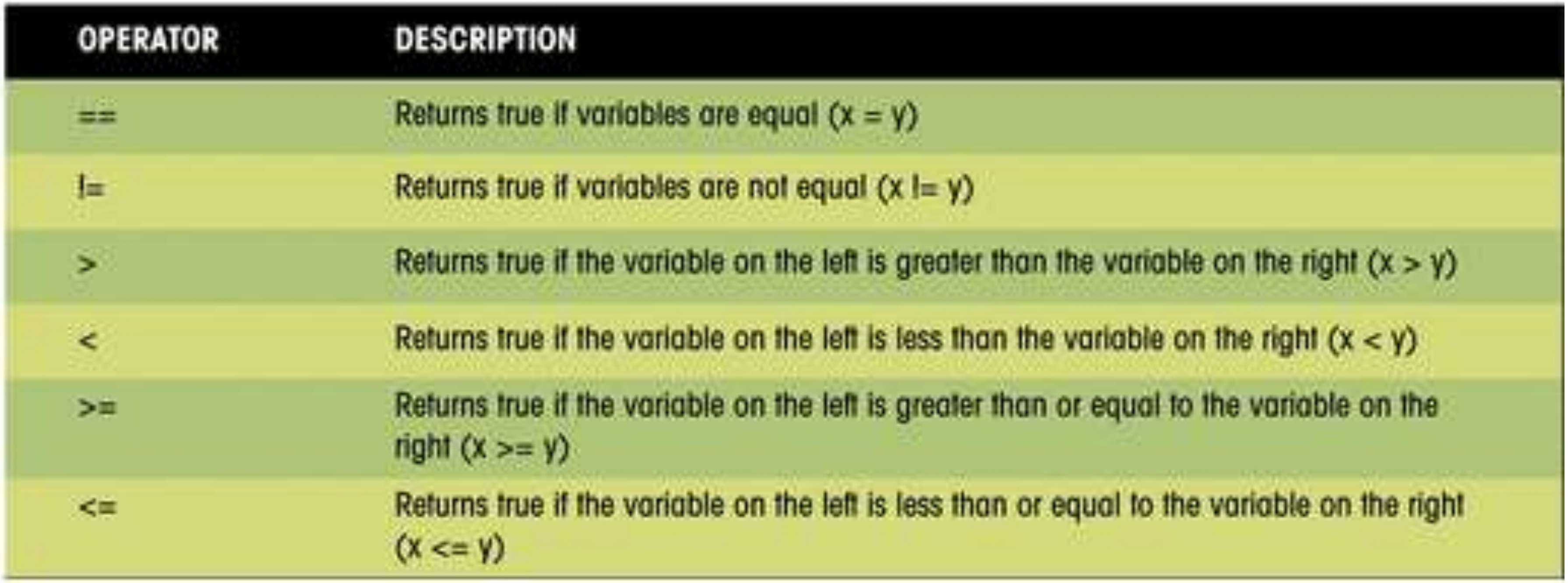
  – the ***y*** variable must have an exact value of 20 for the expression to be true

  – comparison operator uses a double equal sign (==) rather than a single one (a single equal sign is an assignment operator and is not used for making comparisons)

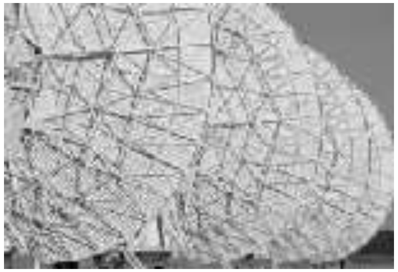


Comparison Operators

This figure lists some of the other comparison operators used in JavaScript.



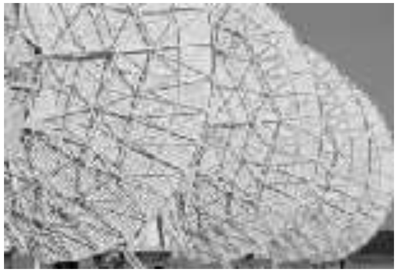
OPERATOR	DESCRIPTION
==	Returns true if variables are equal (x = y)
!=	Returns true if variables are not equal (x != y)
>	Returns true if the variable on the left is greater than the variable on the right (x > y)
<	Returns true if the variable on the left is less than the variable on the right (x < y)
>=	Returns true if the variable on the left is greater than or equal to the variable on the right (x >= y)
<=	Returns true if the variable on the left is less than or equal to the variable on the right (x <= y)



A Logical Operator

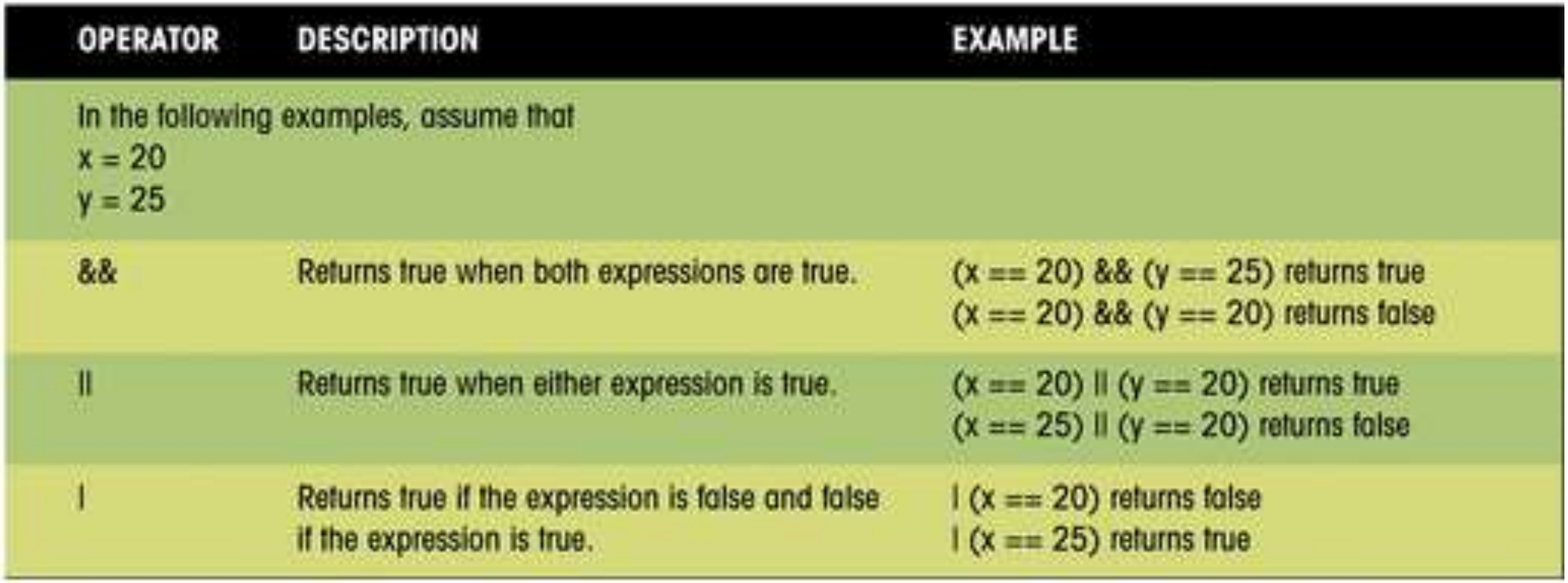
- The **logical operator &&** returns a value of true only if all of the Boolean expressions are true.
  - for example, the following expression is true only if x is less than 100 and y is equal to 20:

```
(x < 100) && (y == 20);
```



Logical Operators

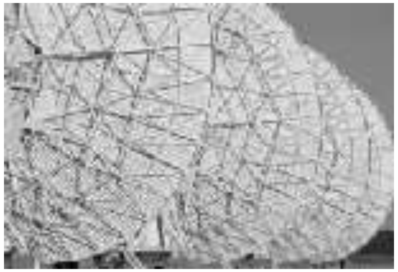
This figure lists some of the logical operators used by JavaScript.



OPERATOR	DESCRIPTION	EXAMPLE

In the following examples, assume that
x = 20
y = 25		

&&	Returns true when both expressions are true.	(x == 20) && (y == 25) returns true (x == 20) && (y == 20) returns false
||	Returns true when either expression is true.	(x == 20) || (y == 20) returns true (x == 25) || (y == 20) returns false
!	Returns true if the expression is false and false if the expression is true.	! (x == 20) returns false ! (x == 25) returns true

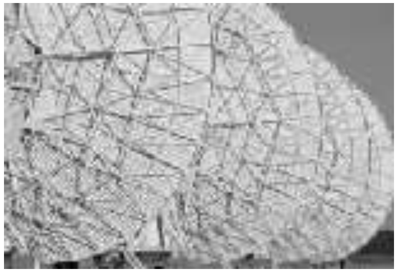


A Conditional Operator

- A **conditional operator** tests whether a specific condition is true and returns one value if the condition is true and a different value if the condition is false.
  - for example, the following statement:

    ```
    Message = (mail == “Yes”) ? “You have
      mail”: “No mail”;
    ```

  - tests whether the mail variable is equal to the value “**Yes**”
    - if it is, the message variable has the value “**You have mail**”;
    - otherwise, the message variable has the value “**No mail**”.

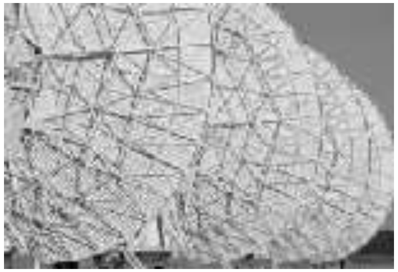


Using an If...Else Statement

- The If statement runs a set of commands if the condition is true.
- To run the If statement for one set of commands if the condition is true and another set of commands if the condition is false use the If...Else statement.
- The general syntax is:

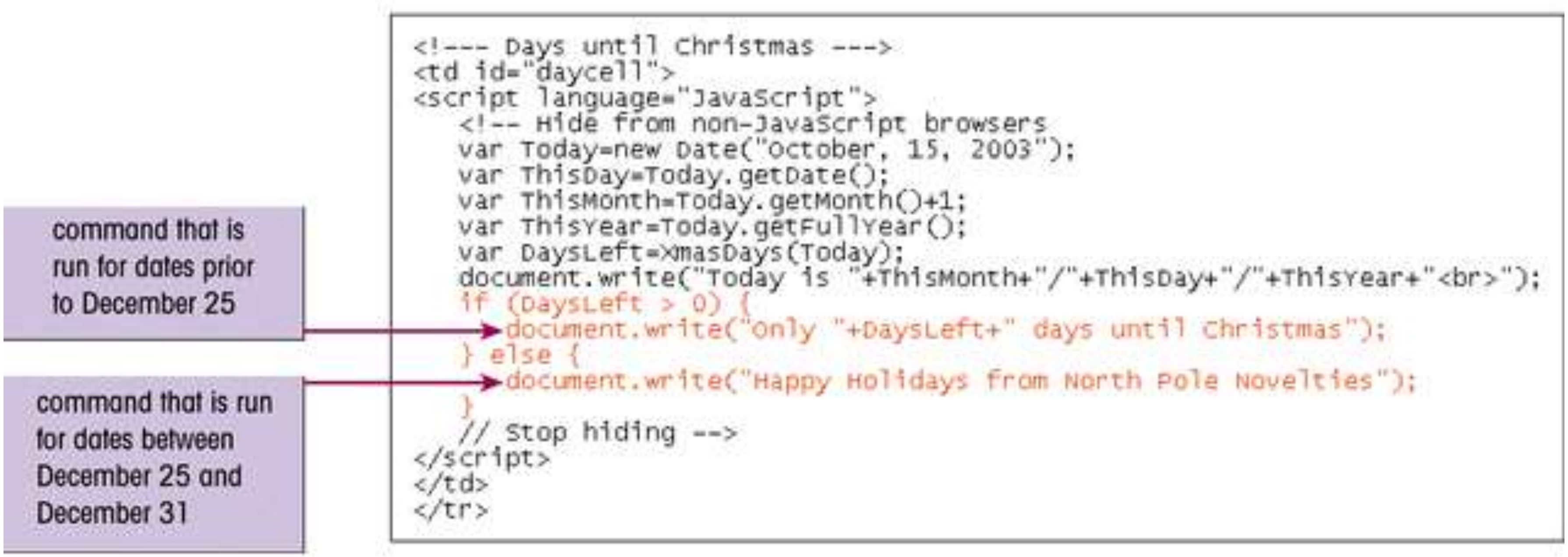
```
if (condition) {
   JavaScript Commands if true
} else
   JavaScript Commands if false
}
```

  - ***condition*** is an expression that is either true or false, and one set of commands is run if the expression is true, and another is run if the expression is false



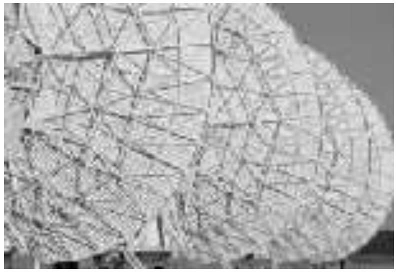
Using an If...Else Conditional Statement

This figure shows an example of an If...Else conditional statement.



```
<!--- Days until Christmas --->
<td id="daycell">
<script language="JavaScript">
   <!-- Hide from non-JavaScript browsers
   var Today=new Date("October, 15, 2003");
   var ThisDay=Today.getDate();
   var ThisMonth=Today.getMonth()+1;
   var ThisYear=Today.getFullYear();
   var DaysLeft=XmasDays(Today);
   document.write("Today is "+ThisMonth+"/"+ThisDay+"/"+ThisYear+"<br>");
   if (DaysLeft > 0) {
      document.write("Only "+DaysLeft+" days until Christmas");
   } else {
      document.write("Happy Holidays from North Pole Novelties");
   }
   // Stop hiding -->
</script>
</td>
</tr>
```

command that is run for dates prior to December 25
command that is run for dates between December 25 and December 31

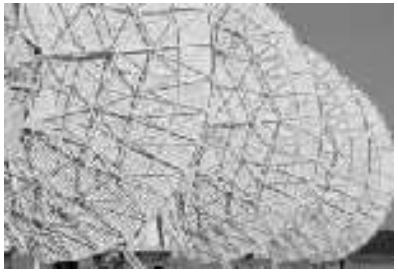


Using Arrays

- An **array** is an ordered collection of values referenced by a single variable name.
- The syntax for creating an array variable is:

```
var variable = new Array(size);
```

  - ***variable*** is the name of the array variable
  - ***size*** is the number of elements in the array (optional)
- Once an array is created, you create values for each individual element in the array.



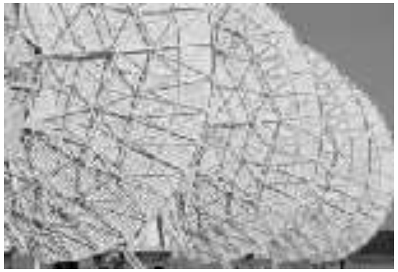
Using Arrays Continued

- A more efficient way of populating an array is to specify the array contents in the new **Array()statement**.
- In this form, the syntax is:

```
var variable = new Array(contents);
```

  - *contents* are the array elements enclosed in quotes and separated by commas
  - For example, the following statement creates an array of the names of the seven days of the week:

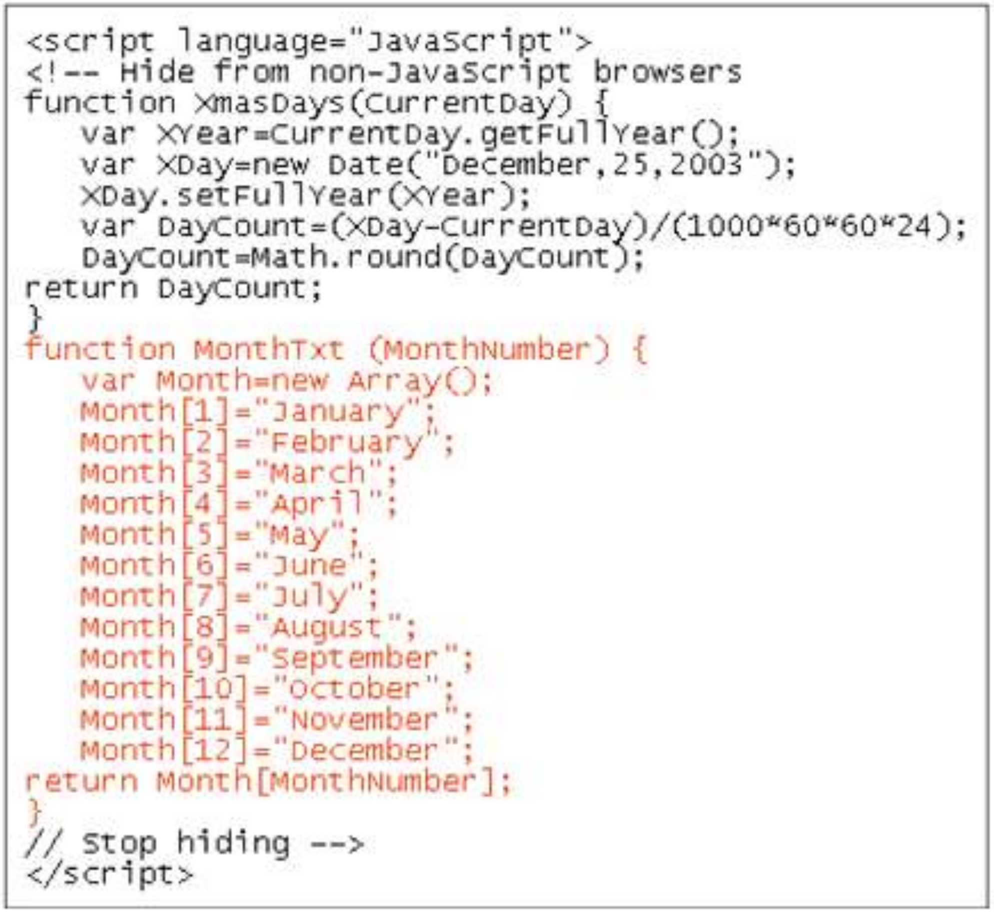
```
var Wday=new Array('Sun', 'Mon', 'Tue',
  'Wed', 'Thu', 'Fri', 'Sat');
```



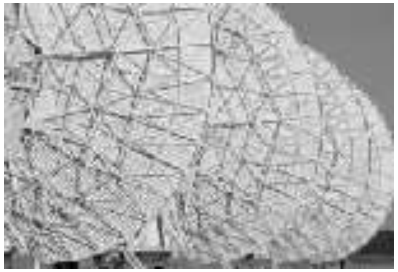
Creating the MonthTxt Function

The “MonthTxt” function has one parameter, “MonthNumber”, which is the number of a month that the function uses to return the name of the corresponding month.

The figure shows the code for the MonthTxt function.

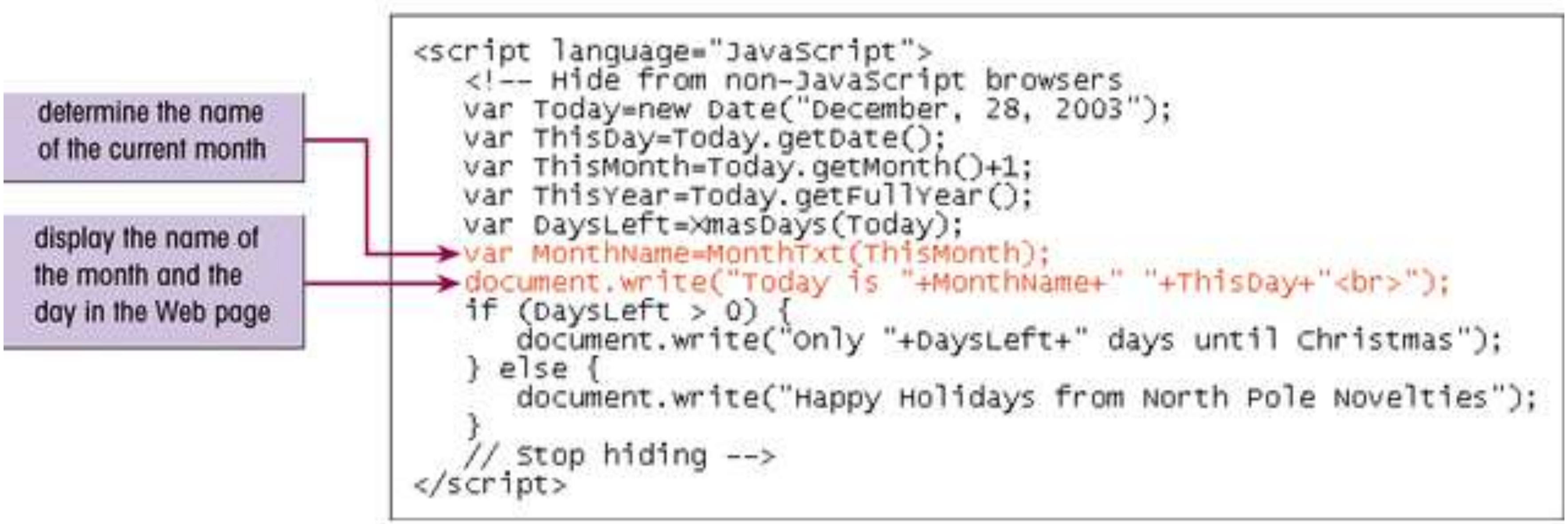


```
<script language="JavaScript">
<!-- Hide from non-JavaScript browsers
function XmasDays(CurrentDay) {
   var XYear=CurrentDay.getFullYear();
   var XDay=new Date("December,25,2003");
   XDay.setFullYear(XYear);
   var DayCount=(XDay-CurrentDay)/(1000*60*60*24);
   DayCount=Math.round(DayCount);
return DayCount;
}
function MonthTxt (MonthNumber) {
   var Month=new Array();
   Month[1]="January";
   Month[2]="February";
   Month[3]="March";
   Month[4]="April";
   Month[5]="May";
   Month[6]="June";
   Month[7]="July";
   Month[8]="August";
   Month[9]="September";
   Month[10]="October";
   Month[11]="November";
   Month[12]="December";
return Month[MonthNumber];
}
// Stop hiding -->
</script>
```



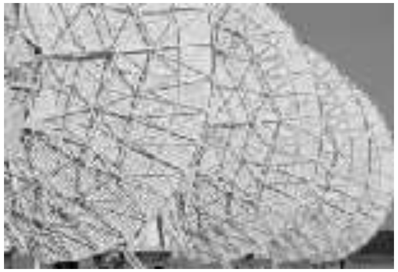
Calling the MonthTxt Function

This figure shows the use of the `ThisMonth` variable to call the `MonthTxt` function and then stores the result in a new variable named "`MonthName.`".



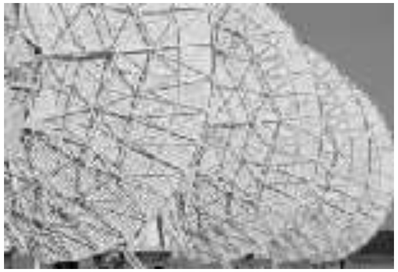
determine the name of the current month
display the name of the month and the day in the Web page

```
<script language="JavaScript">
   <!-- Hide from non-JavaScript browsers
   var Today=new Date("December, 28, 2003");
   var ThisDay=Today.getDate();
   var ThisMonth=Today.getMonth()+1;
   var ThisYear=Today.getFullYear();
   var DaysLeft=XmasDays(Today);
   var MonthName=MonthTxt(ThisMonth);
   document.write("Today is "+MonthName+" "+ThisDay+"<br>");
   if (DaysLeft > 0) {
      document.write("Only "+DaysLeft+" days until Christmas");
   } else {
      document.write("Happy Holidays from North Pole Novelties");
   }
   // Stop hiding -->
</script>
```



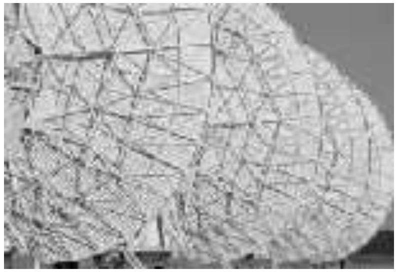
Working with Loops

- A **program loop** is a set of instructions that is executed repeatedly.
- There are two types of loops:
  - loops that repeat a set number of times before quitting
  - loops that repeat as long as a certain condition is met



The For Loop

- The **For loop** allows you to create a group of commands to be executed a set number of times through the use of a **counter** that tracks the number of times the command block has been run.
- Set an initial value for the counter, and each time the command block is executed, the counter changes in value.
- When the counter reaches a value above or below a certain stopping value, the loop ends.

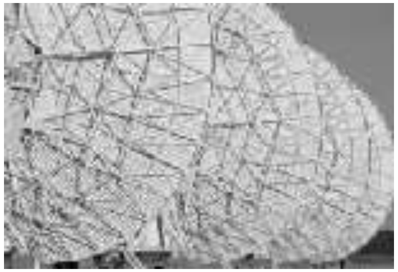


The For Loop Continued

- The general syntax of the For loop is:

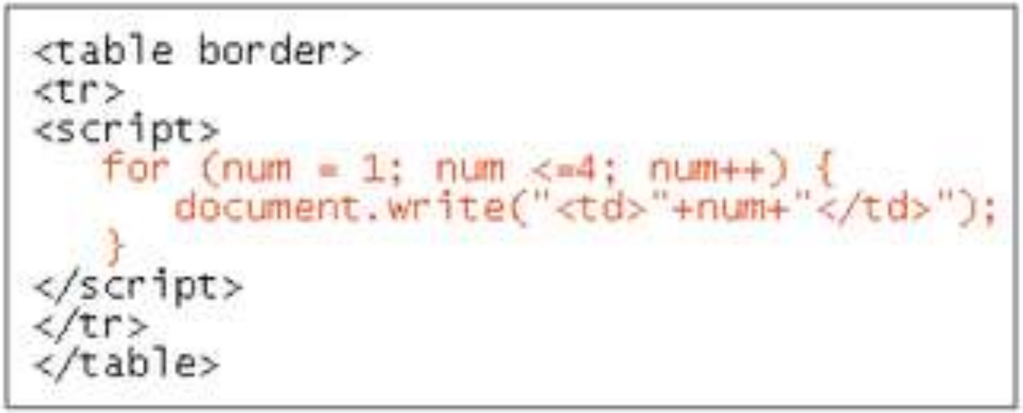
```
for (start; condition; update) {
    JavaScript Commands
}
```

  - ***start*** is the starting value of the counter
  - ***condition*** is a Boolean expression that must be true for the loop to continue
  - ***update*** specifies how the counter changes in value each time the command block is executed



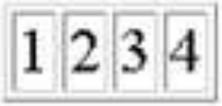
Creating a For Loop

This figure shows an example of a For loop used to write a row of table cells.



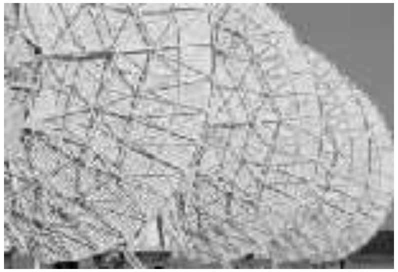
```
<table border>
<tr>
<script>
   for (num = 1; num <=4; num++) {
      document.write("<td>"+num+"</td>");
   }
</script>
</tr>
</table>
```

For loop



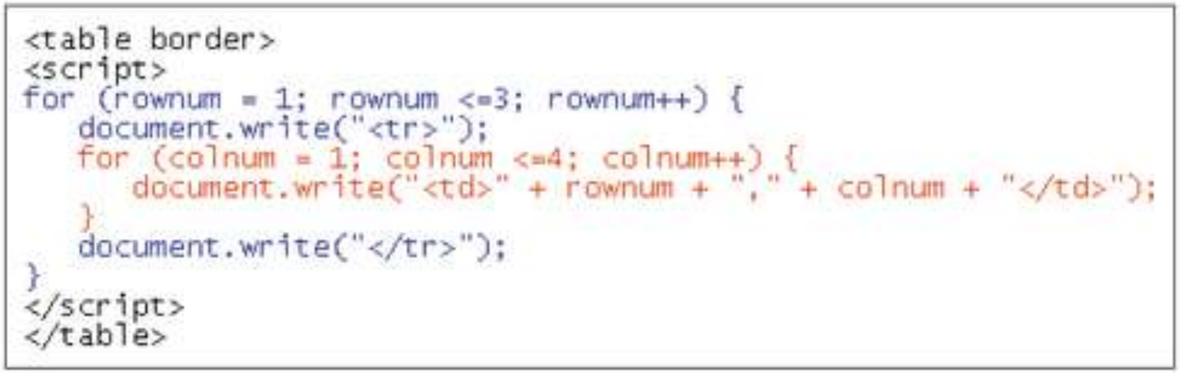
1	2	3	4

resulting table



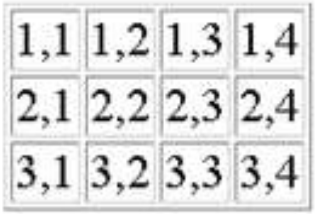
Nesting a For Loop

This figure shows code used to write a table containing three rows and four columns.



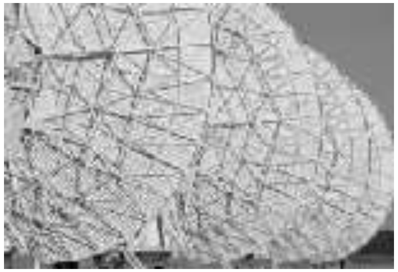
```
<table border>
<script>
for (rownum = 1; rownum <=3; rownum++) {
   document.write("<tr>");
   for (colnum = 1; colnum <=4; colnum++) {
      document.write("<td>" + rownum + "," + colnum + "</td>");
   }
   document.write("</tr>");
}
</script>
</table>
```

nested For loop



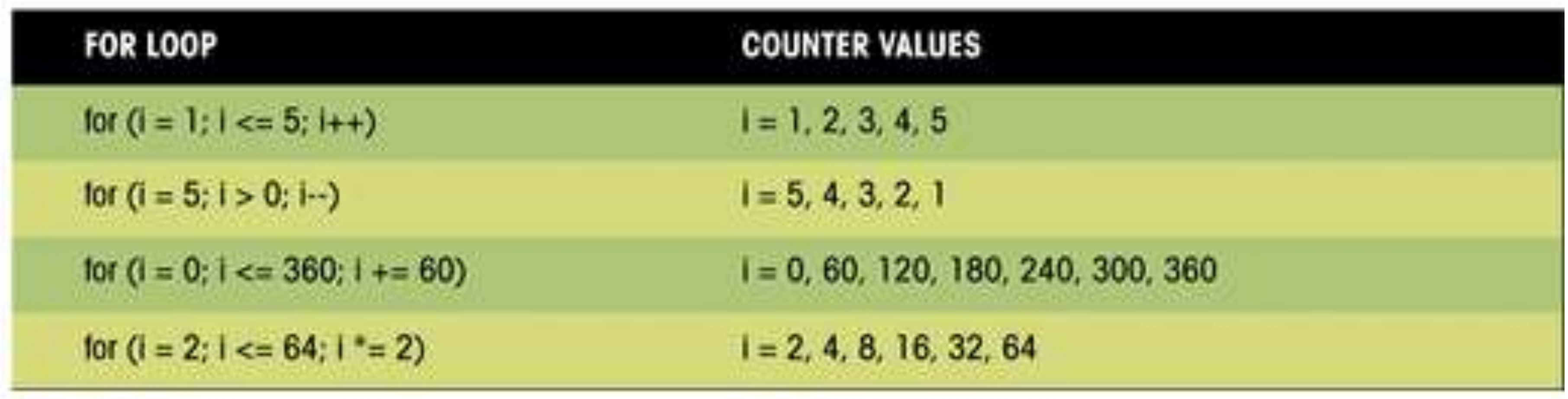
1,1	1,2	1,3	1,4
2,1	2,2	2,3	2,4
3,1	3,2	3,3	3,4

resulting table

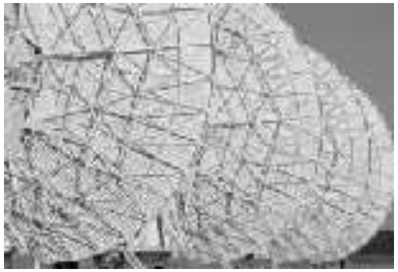


Specifying Counter Values in a For Loop

The For loop is not limited to incrementing the value of the counter by 1. This figure shows examples of other ways of incrementing the counter in a For loop.



FOR LOOP	COUNTER VALUES
for (i = 1; i <= 5; i++)	i = 1, 2, 3, 4, 5
for (i = 5; i > 0; i--)	i = 5, 4, 3, 2, 1
for (i = 0; i <= 360; i += 60)	i = 0, 60, 120, 180, 240, 300, 360
for (i = 2; i <= 64; i *= 2)	i = 2, 4, 8, 16, 32, 64

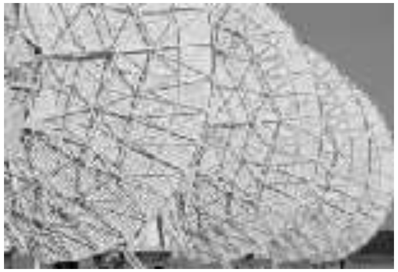


The While Loop

- The **While loop** runs a command group as long as a specific condition is met, but it does not employ any counters.
- The general syntax of the While loop is:

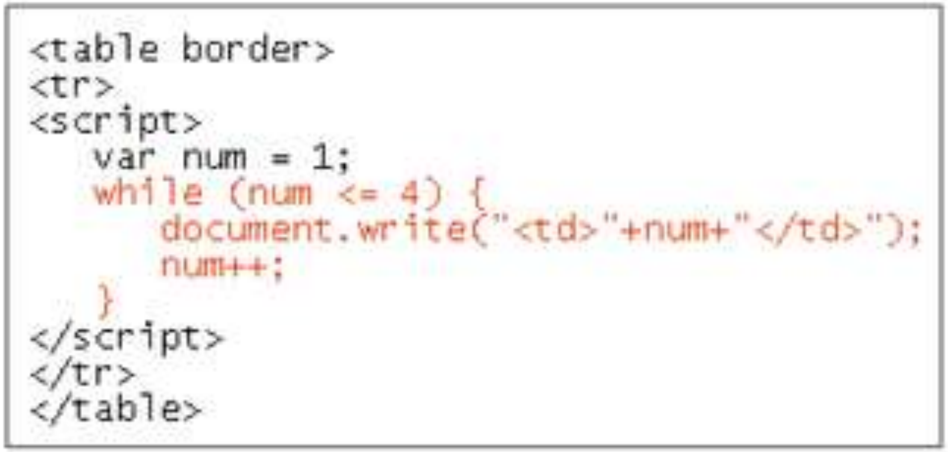
```
while (condition) {
   JavaScript Commands
}
```

  - *condition* is a Boolean expression that can be either true or false



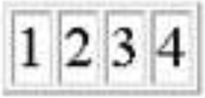
Creating a While Loop

This figure shows how you can create a set of table cells using a While loop.



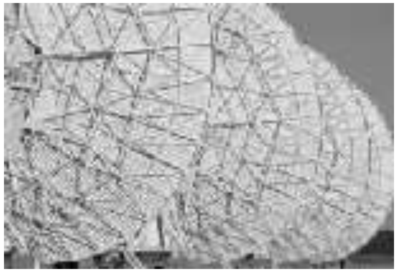
```
<table border>
<tr>
<script>
   var num = 1;
   while (num <= 4) {
      document.write("<td>"+num+"</td>");
      num++;
   }
</script>
</tr>
</table>
```

While loop



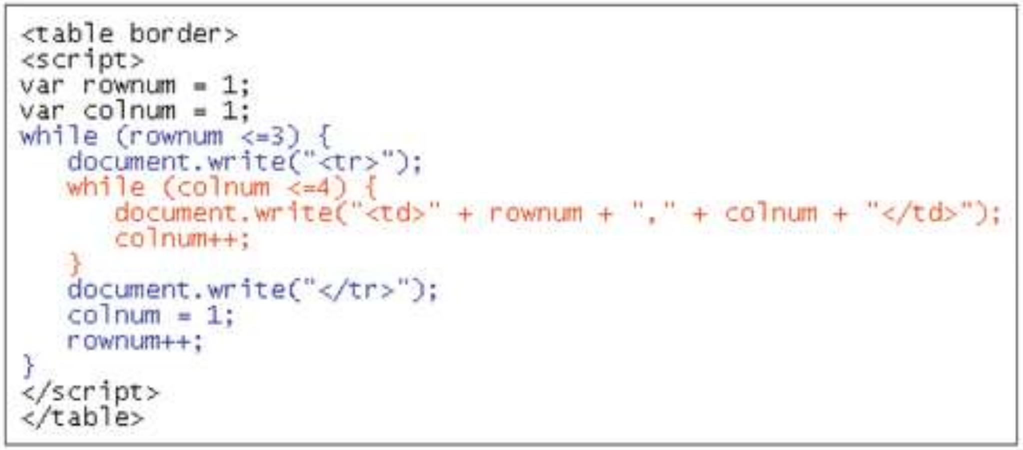
1	2	3	4

resulting table



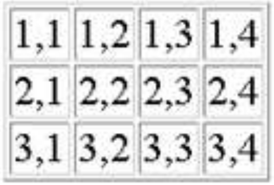
Nesting a While Loop

As with For loops, this figure shows that While loops can be nested inside one another.



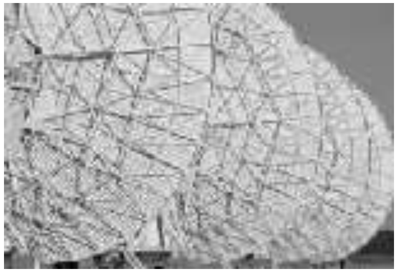
```
<table border>
<script>
var rownum = 1;
var colnum = 1;
while (rownum <=3) {
   document.write("<tr>");
   while (colnum <=4) {
      document.write("<td>" + rownum + "," + colnum + "</td>");
      colnum++;
   }
   document.write("</tr>");
   colnum = 1;
   rownum++;
}
</script>
</table>
```

nested While loop



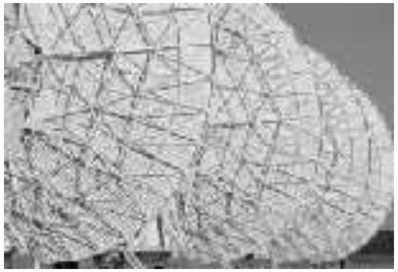
1,1	1,2	1,3	1,4
2,1	2,2	2,3	2,4
3,1	3,2	3,3	3,4

resulting table



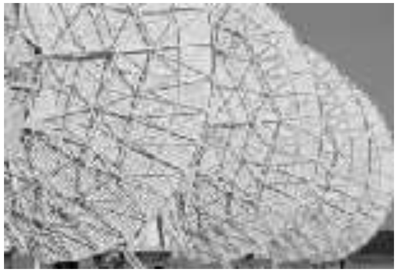
Tutorial 8 Summary

- Discussed JavaScript history and philosophy.
- Covered the basic features of JavaScript including syntax, operators, expressions and functions.
- Learned about the **`<SCRIPT>`** tag and the **`document.write()`** method.
- Learned how to write a simple JavaScript program.
- Learned about various JavaScript data types, operators and expressions.



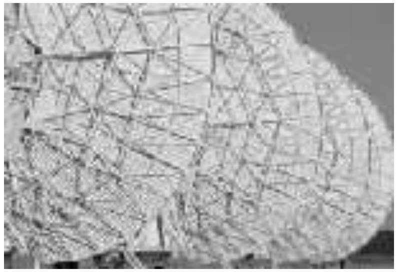
Tutorial 8 Summary Continued

- Learned about JavaScript tools used to work with dates.
- Learned how to create a function to calculate some simple date values.
- Covered the topics of conditional statements, arrays, and program loops.
- Learned how to create IF statements and IF...ELSE statements and observe how to nest one set of conditional statements within another.
- Learned about FOR loops and WHILE loops.



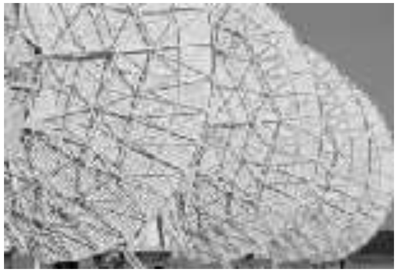
New Perspectives on Creating Web Pages with HTML

Working with JavaScript Objects and Events



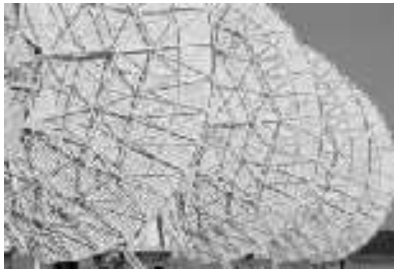
Tutorial Objectives

- Learn about form validation
- Study the object-based nature of the JavaScript language
- Work with objects, properties, methods, and events of your Web page
- Create a program to calculate a value



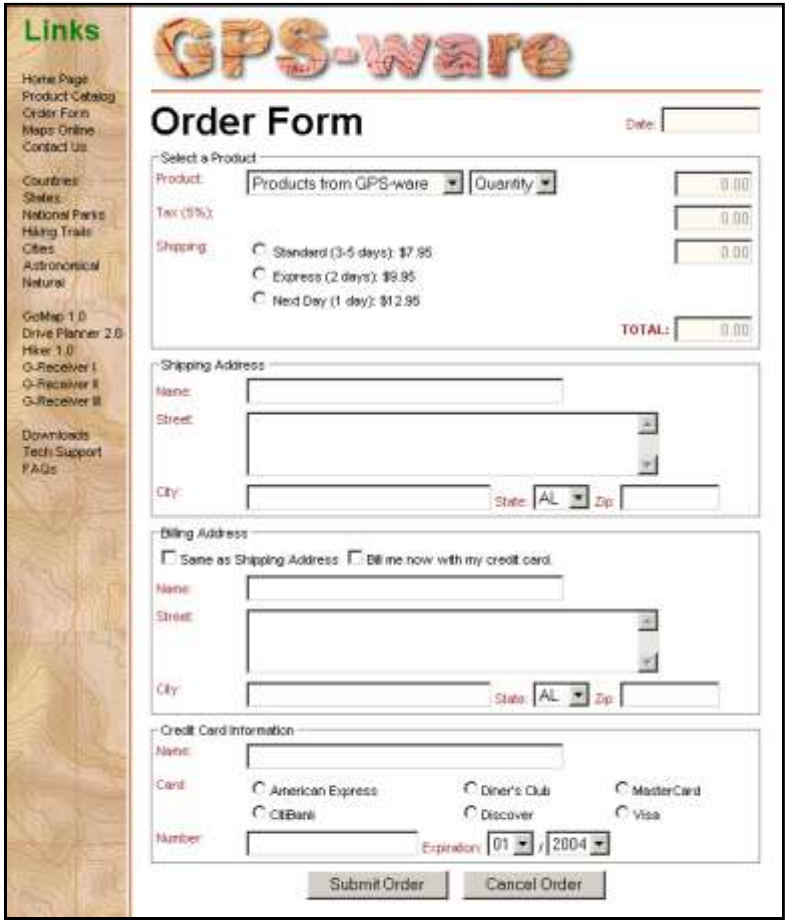
Tutorial Objectives Continued

- Copy a value into a form field
- Extract a value from a selection list and radio button
- Display a message box to the user
- Control a form submission



Sample Order Form

This figure shows a sample order form.



Order Form

Date:

Select a Product

Product: Products from GPS-ware ▾ Quantity ▾ 0.00

Tax (5%): 0.00

Shipping: ○ Standard (3-5 days): $7.95 0.00

○ Express (2 days): $9.95

○ Next Day (1 day): $12.95

TOTAL: 0.00

Shipping Address

Name:

Street:

City: State: AL ▾ Zip:

Billing Address

□ Same as Shipping Address □ Bill me now with my credit card.

Name:

Street:

City: State: AL ▾ Zip:

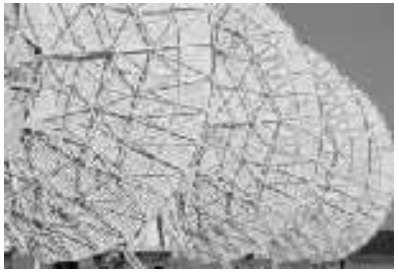
Credit Card Information

Name:

Card: ○ American Express ○ Diner's Club ○ MasterCard

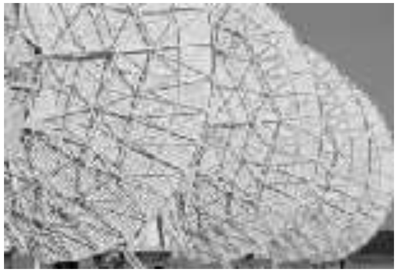
○ CitiBank ○ Discover ○ Visa

Number: Expiration: 01 ▾ / 2004 ▾



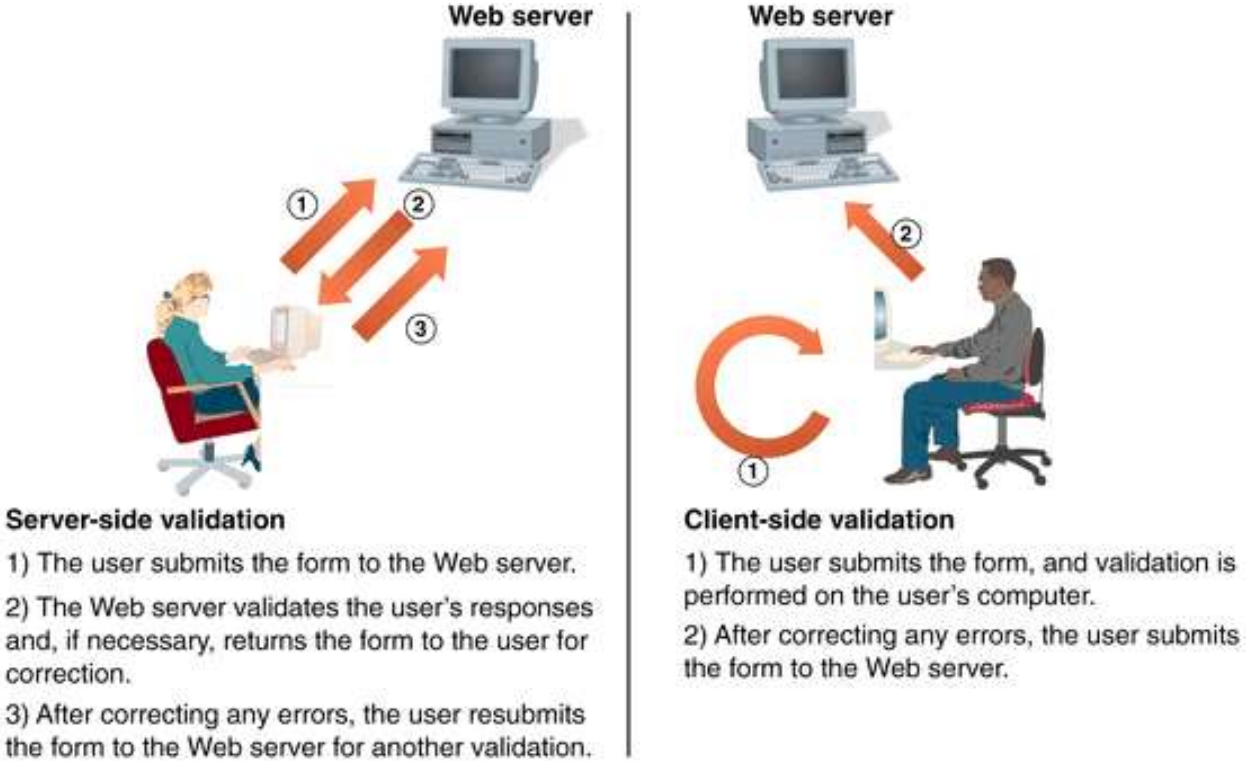
Understanding Form Validation

- A **form validation** is a process by which the server on the browser checks form entries and, where possible, eliminates errors.
- On the Web, validation can occur on the client or server side.
- Form validation is a critical aspect of data entry.
- A properly designed form reduces the possibility of faulty data being entered.



Server-Side and Client-Side Validation

This figure shows server-side and client-side validation.



Server-side validation

1) The user submits the form to the Web server.

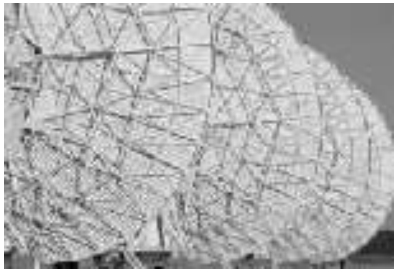
2) The Web server validates the user's responses and, if necessary, returns the form to the user for correction.

3) After correcting any errors, the user resubmits the form to the Web server for another validation.

Client-side validation

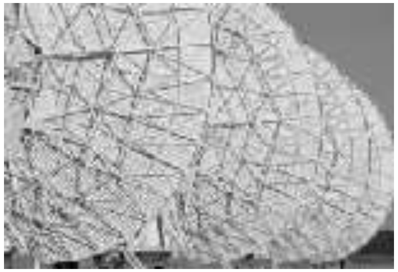
1) The user submits the form, and validation is performed on the user's computer.

2) After correcting any errors, the user submits the form to the Web server.



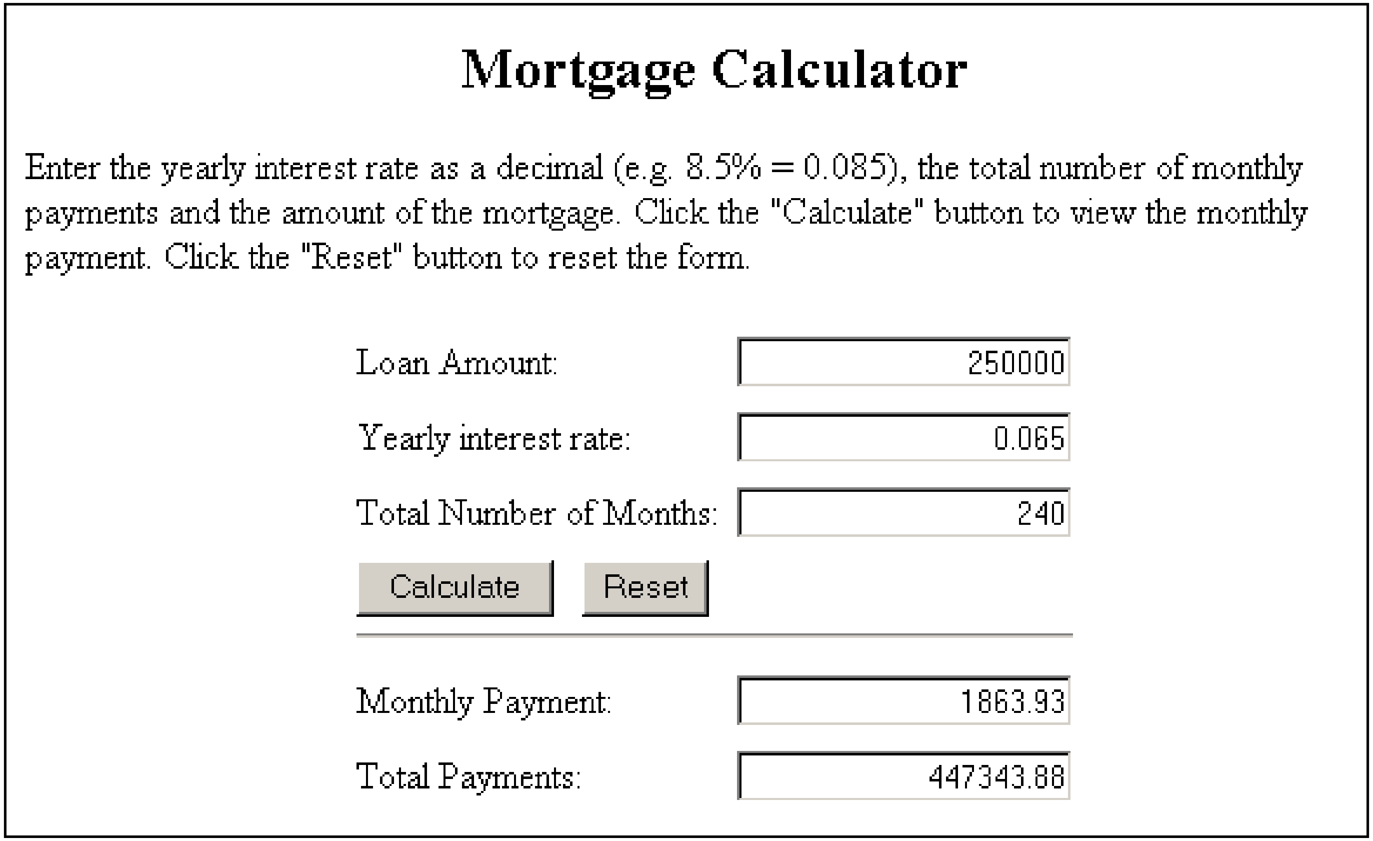
JavaScript and Client-Side Validation

- A powerful use of **JavaScript** is to provide client-side validation.
- Using a script built into the Web page form provides immediate feedback to users as they enter data.
- **Client-side validation** can reduce the network traffic between users and the Web server.



An Example of the Use of JavaScript

This figure shows an example of the use of JavaScript, which will provide users with immediate feedback.

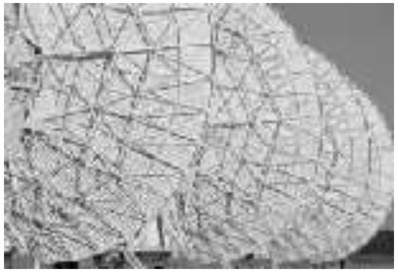


Mortgage Calculator

Enter the yearly interest rate as a decimal (e.g. 8.5% = 0.085), the total number of monthly payments and the amount of the mortgage. Click the "Calculate" button to view the monthly payment. Click the "Reset" button to reset the form.

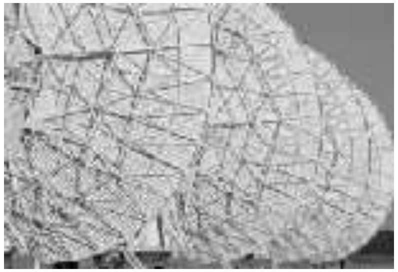
Loan Amount:	250000
Yearly interest rate:	0.065
Total Number of Months:	240

Monthly Payment:	1863.93
Total Payments:	447343.88



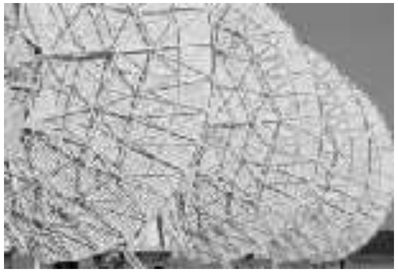
Working with JavaScript Objects

- JavaScript is an object-based language.
- JavaScript is based on manipulating objects by modifying an object’s properties or by applying methods to an object.
  - ***objects*** are items that have a defined existence
  - each object has ***properties*** that describe its appearance, purpose, or behavior
  - each object has ***methods***, which are actions that can be performed with the object or to it



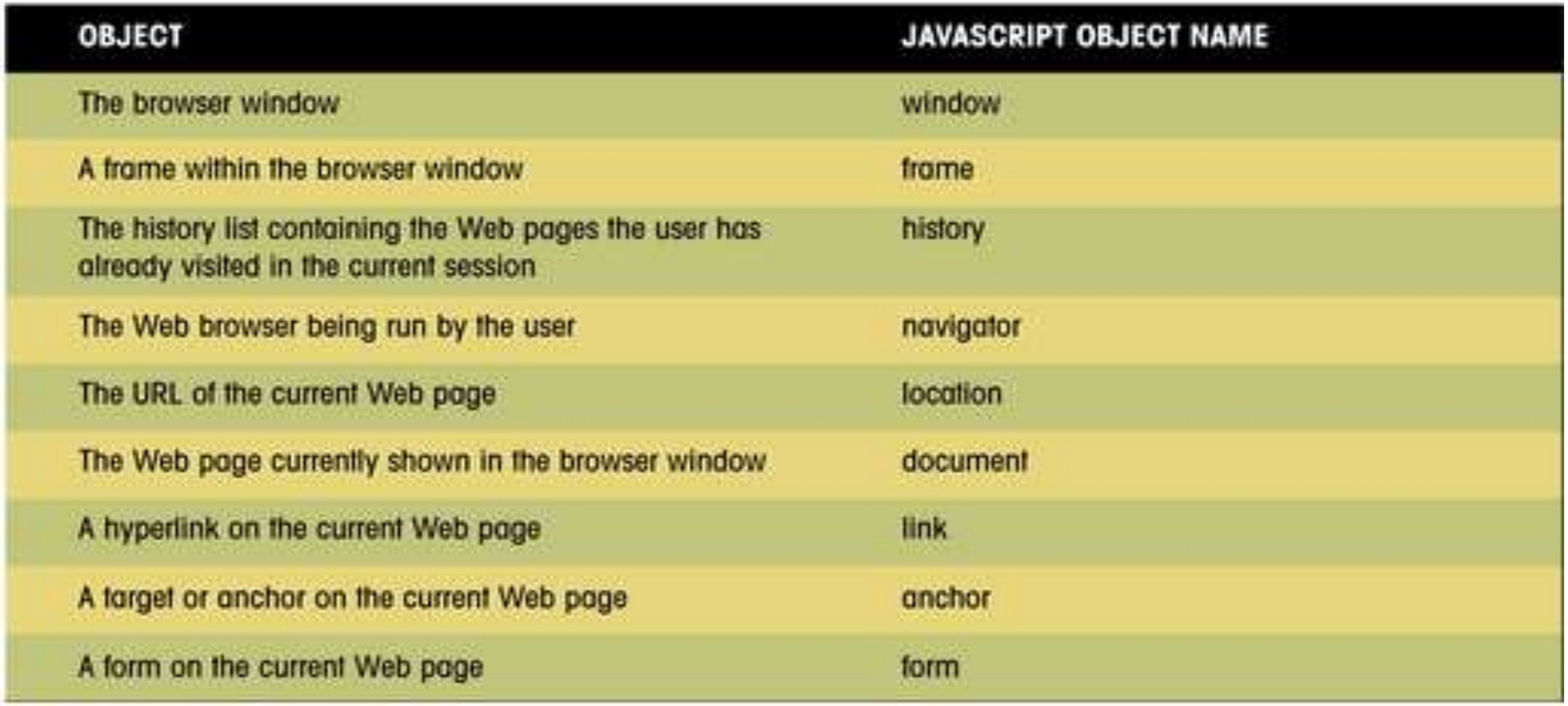
Understanding JavaScript Objects and Object Names

- In JavaScript, each object is identified by an object name.
  - for example, when you want to use JavaScript to manipulate the current window, you use the object name “**window**”
  - operations that affect the current Web page use the “**document**” object name
  - the object name can also be based on the name assigned to the object by the user

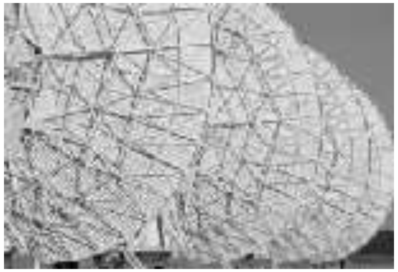


Some JavaScript Objects and Their Object Names

This figure shows a list of the many objects available in JavaScript and their corresponding object names.



OBJECT	JAVASCRIPT OBJECT NAME
The browser window	window
A frame within the browser window	frame
The history list containing the Web pages the user has already visited in the current session	history
The Web browser being run by the user	navigator
The URL of the current Web page	location
The Web page currently shown in the browser window	document
A hyperlink on the current Web page	link
A target or anchor on the current Web page	anchor
A form on the current Web page	form

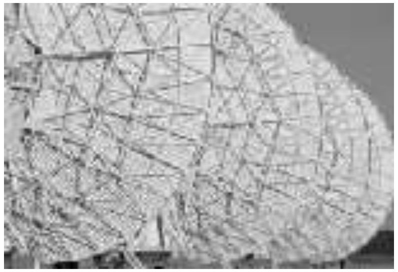


Introducing the Document Object Model

- JavaScript arranges objects in a **Document Object Model** or **DOM**.
- The DOM defines the logical structure of objects and the way an object is accessed and manipulated.

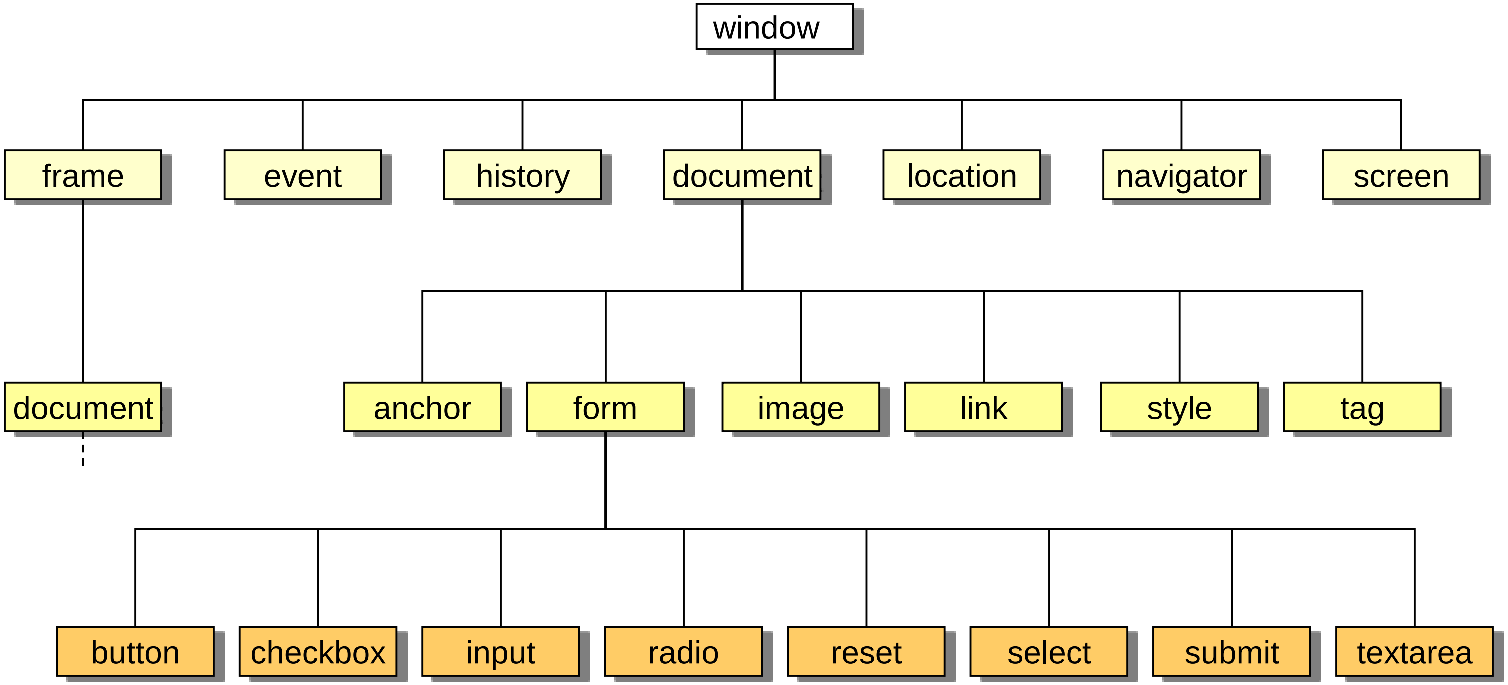
Click to add text

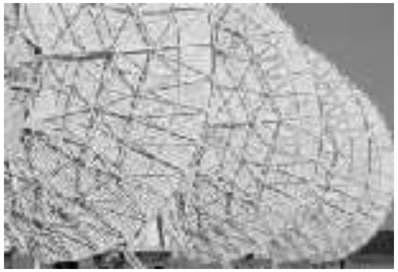
- The document object model can be thought of as a hierarchy moving from the most general object to the most specific.



A Part of the Document Object Model

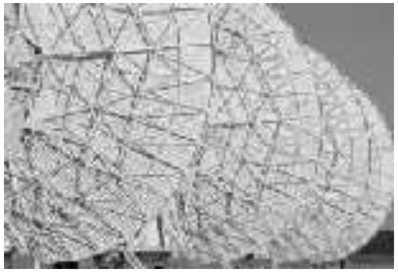
This figure shows a section of the entire Document Object Model (DOM). The full DOM would be a much larger figure.





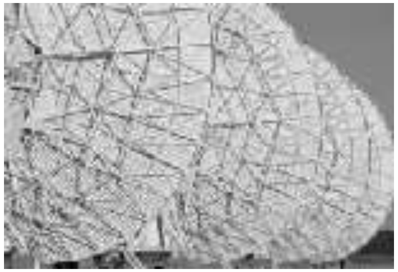
DOM Hierarchy

- The topmost object in the **hierarchy** is the window object, which contains the other objects in the list, such as the **current frame**, **history list**, and the **Web page document**.
- The Web page document contains its own set of objects, including **links**, **anchors**, and **forms**.
- Within each form are form objects, such as **input boxes**, **radio buttons**, or **selection lists**.



Object Names and Browsers

- Include the DOM hierarchy when referring to an object i.e window.document.order.
- Some browsers cannot interpret the object names without the complete hierarchy.

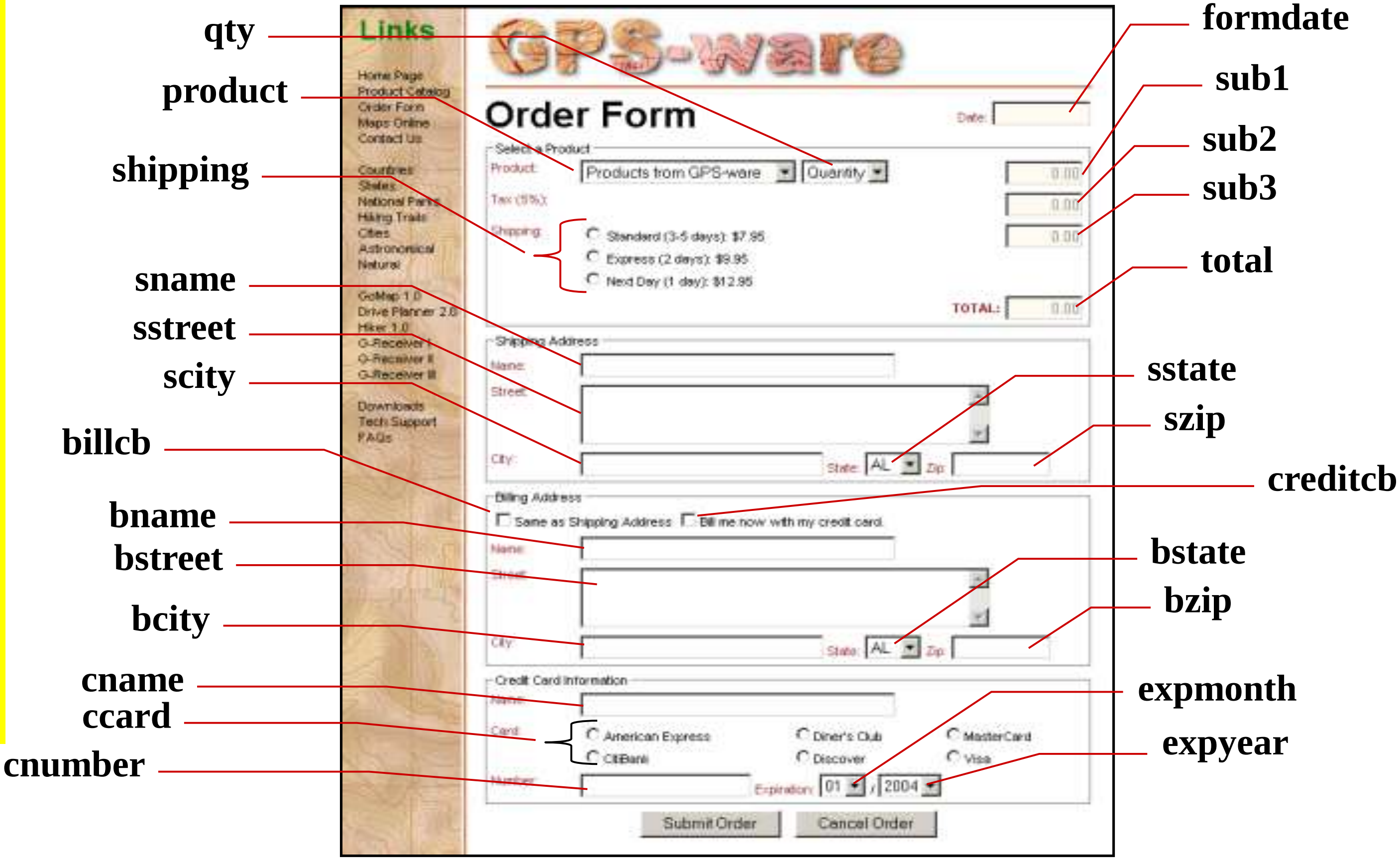


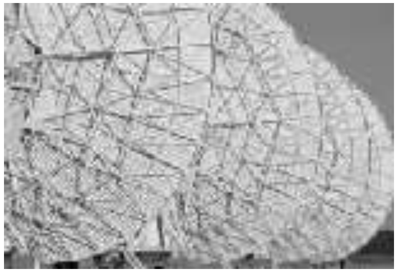
Field Names in a Order Form

This figure shows that each field in the order form has been given a name.

To refer to a particular field, you attach the field name to the JavaScript reference for the form.

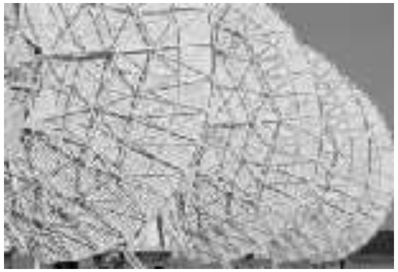
<form name = “order”>





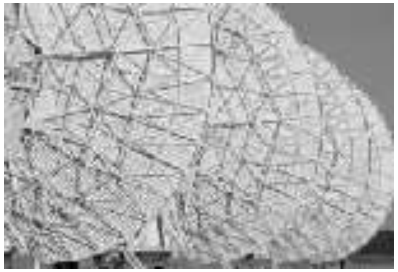
Field Names

- To refer to a particular field, attach the **field name** to the JavaScript reference for the form.
  - for example, in the order form to display the current date in the formdata field use the following JavaScript object reference: document.order.formdate



Object Collections

- There is another way to reference an object and that is with an **object collection**.

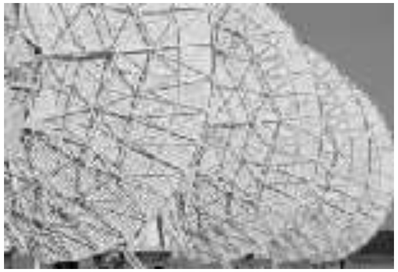


Object Collections Continued

- An **object collection** is an array of all objects of a particular type, such as all of the hyperlinks for a single document or all of the elements within a single form.
- An item from an object collection can be referenced in one of three ways:

```
collection[i]
collection[“name”]
collection.name
```

  - *collection* is the JavaScript name of the collection
  - *i* is an index number of the item in the collection
  - *name* is the name assigned to the object using the name attribute

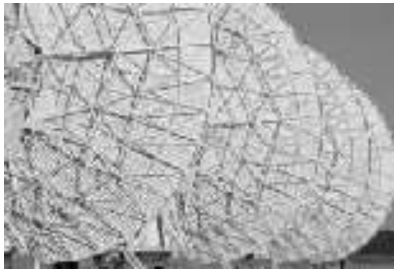


Some JavaScript Object Collections

This figure lists some of the more commonly used JavaScript object collections.

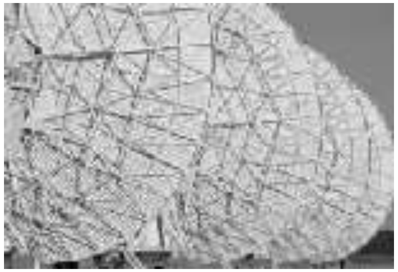
Not all object collections are supported by all browsers or browser versions.

Collection	Description	Browser Support	
		Netscape	IE
document.all	All HTML elements in the document		4.0
document.anchors	All anchor elements in the document	3.0	3.0
document.applets	All Java applets in the document. The applet must be started before being recognized as part of the DOM	3.0	3.0
document.embeds	All embedded objects in the document	3.0	4.0
document.*form*.elements	All of the elements in the form named *form*.		
document.forms	All forms in the document	2.0	3.0
document.frames	All internal frames in the document		4.0
document.images	All inline images in the document	2.0	3.0
document.links	All hyperlinks in the document	2.0	3.0
document.plugins	All plug-ins in the document		4.0
document.scripts	All scripts (created with the <script> tag) in the document		4.0



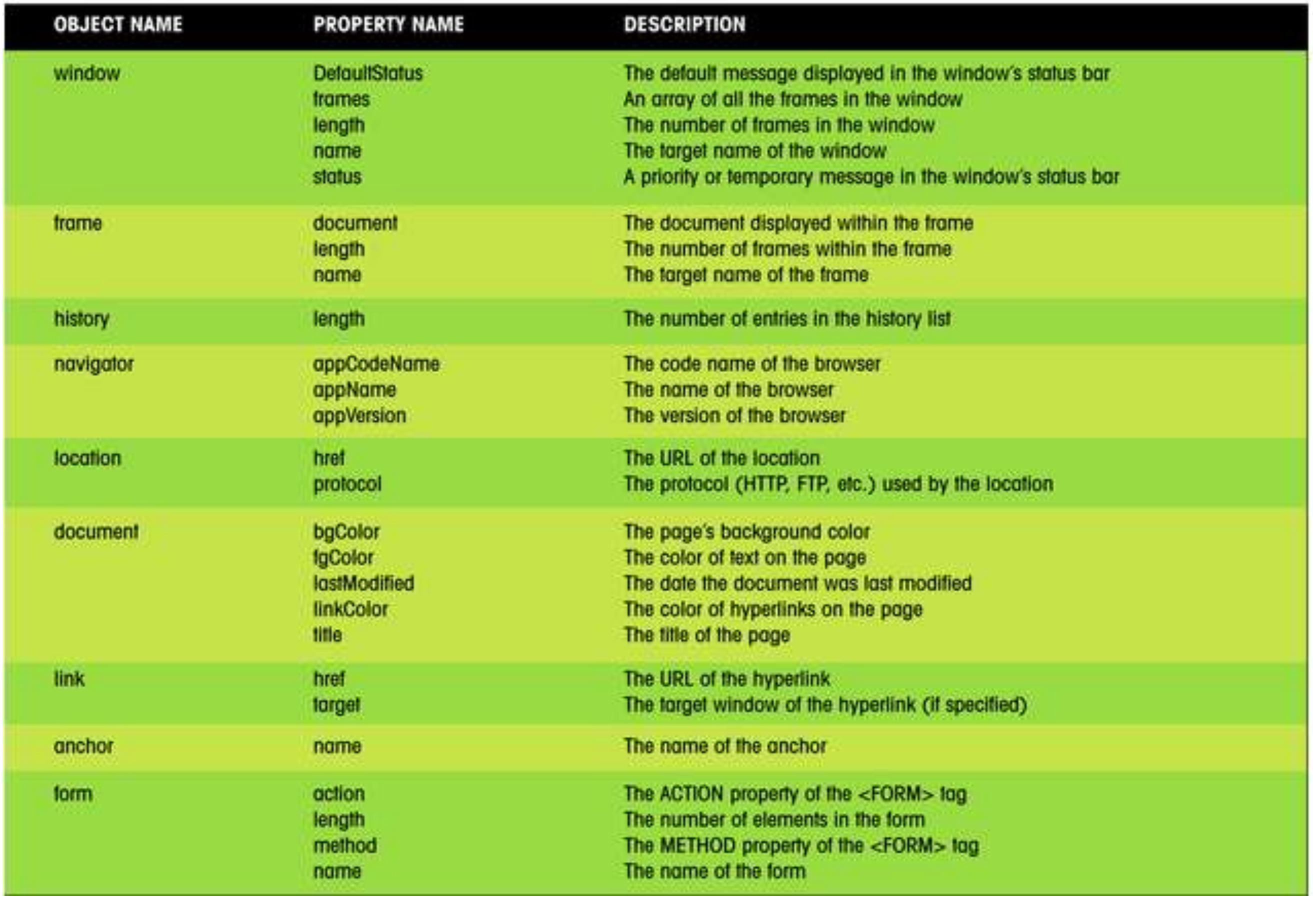
Working with Object Properties

- Each object in JavaScript has properties associated with it.
- The number of properties depends on the particular object; some objects have only a few properties, while others have many.
- As with object names, certain keywords identify properties.

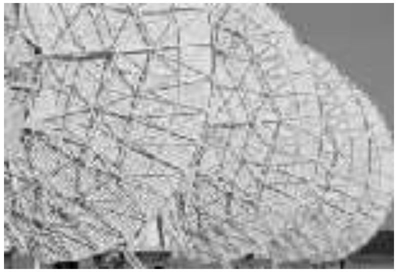


JavaScript Objects and Properties

This figure shows a partial list of objects and their properties.

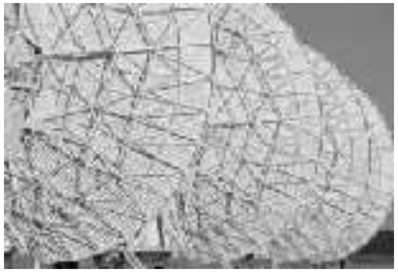


OBJECT NAME	PROPERTY NAME	DESCRIPTION
window	DefaultStatus	The default message displayed in the window's status bar
	frames	An array of all the frames in the window
	length	The number of frames in the window
	name	The target name of the window
	status	A priority or temporary message in the window's status bar
frame	document	The document displayed within the frame
	length	The number of frames within the frame
	name	The target name of the frame
history	length	The number of entries in the history list
navigator	appCodeName	The code name of the browser
	appName	The name of the browser
	appVersion	The version of the browser
location	href	The URL of the location
	protocol	The protocol (HTTP, FTP, etc.) used by the location
document	bgColor	The page's background color
	fgColor	The color of text on the page
	lastModified	The date the document was last modified
	linkColor	The color of hyperlinks on the page
	title	The title of the page
link	href	The URL of the hyperlink
	target	The target window of the hyperlink (if specified)
anchor	name	The name of the anchor
form	action	The ACTION property of the <FORM> tag
	length	The number of elements in the form
	method	The METHOD property of the <FORM> tag
	name	The name of the form



JavaScript Properties

- There are several ways of working with properties.
  - the value of a property can be changed
  - store the property’s value in a variable
  - test whether the property equals a specified value in an If…Then expression

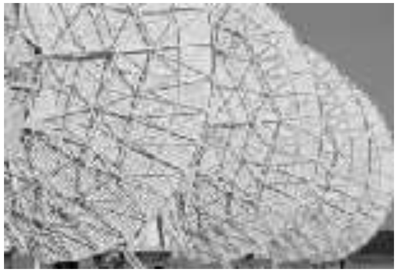


Modifying a Property’s Value

- The syntax for changing the value of a property is:

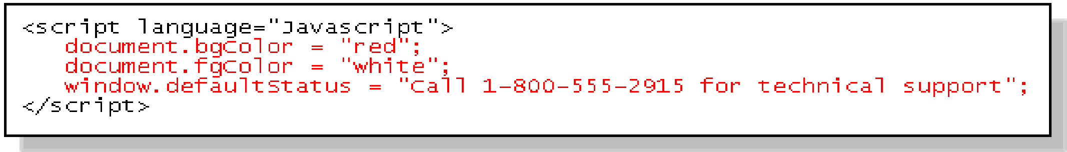
```
object.property = expression
```

  - ***object*** is the JavaScript name of the object you want to manipulate
  - ***property*** is a property of that object
  - ***expression*** is a JavaScript expression that assigns a value to the property



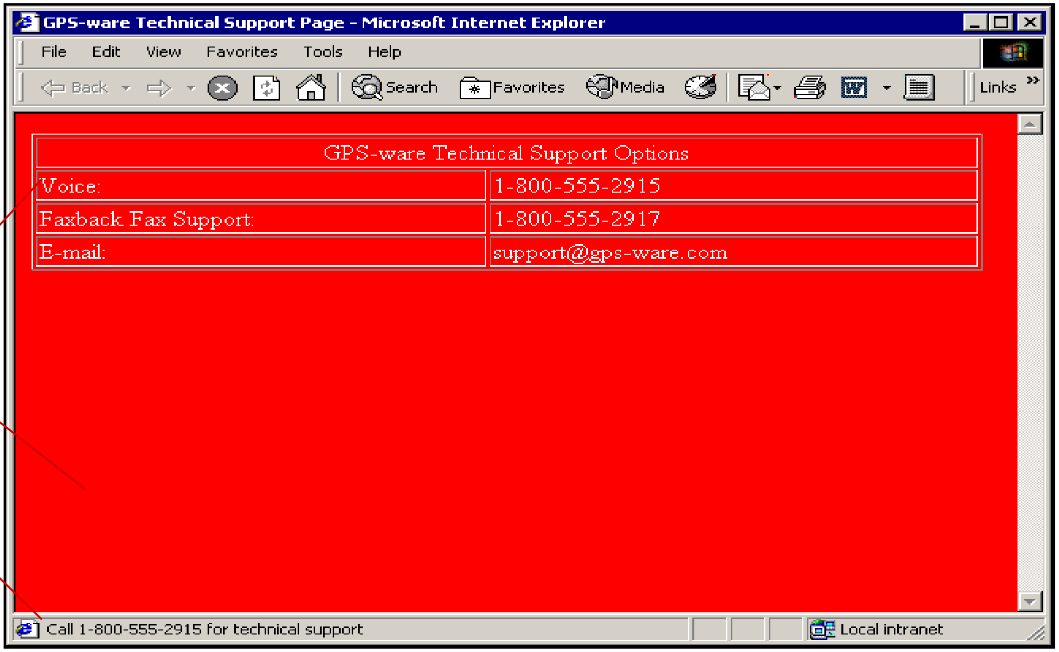
Setting an Object's Property Value

This figure shows how you can use objects and properties to modify a Web page and Web browser.



```
<script language="Javascript">
   document.bgColor = "red";
   document.fgColor = "white";
   window.defaultStatus = "Call 1-800-555-2915 for technical support";
</script>
```

JavaScript commands



GPS-ware Technical Support Options	
Voice:	1-800-555-2915
Faxback Fax Support:	1-800-555-2917
E-mail:	support@gps-ware.com

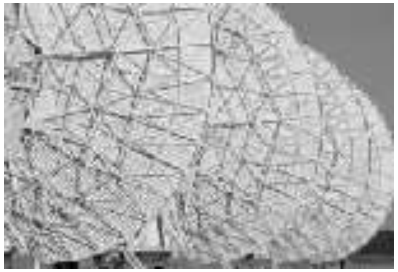
Call 1-800-555-2915 for technical support

document.fgColor

document.bgColor

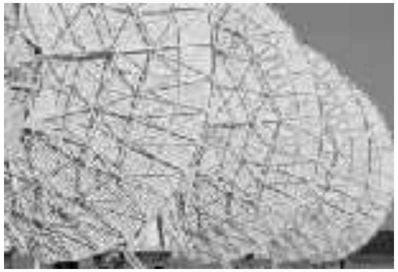
window.defaultStatus

resulting Web page



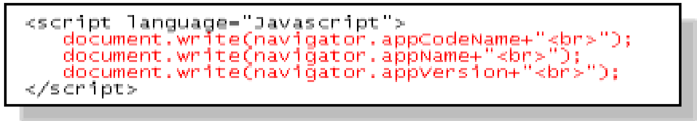
Changing Properties

- Not all properties can be changed.
- Some properties are read-only, which means that you can read the property value, but cannot modify it.



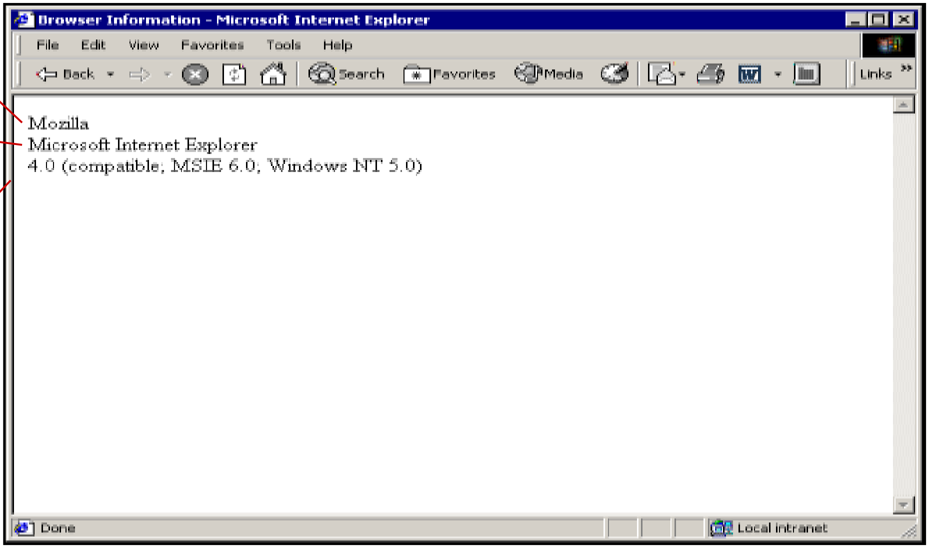
Displaying Some Read-Only Browser Properties

This figure shows how you can use JavaScript to display additional read-only information about your browser.



```
<script language="Javascript">
   document.write(navigator.appCodeName+"<br>");
   document.write(navigator.appName+"<br>");
   document.write(navigator.appVersion+"<br>");
</script>
```

JavaScript commands



Browser Information - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Media
Links
Done
Local intranet

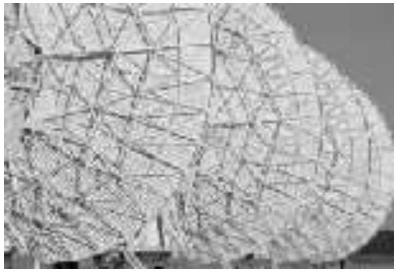
```
Mozilla
Microsoft Internet Explorer
4.0 (compatible; MSIE 6.0; Windows NT 5.0)
```

browser code name

browser name

browser version

resulting Web page

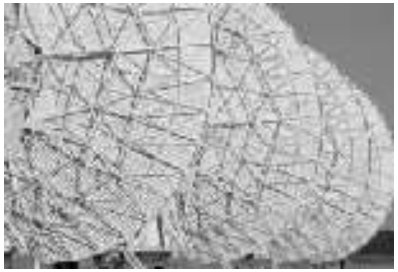


Assigning a Property to a Variable

- Although you cannot change the value of read-only properties, you can assign a value to a variable in your JavaScript program.
- The syntax for assigning a property to a variable is:

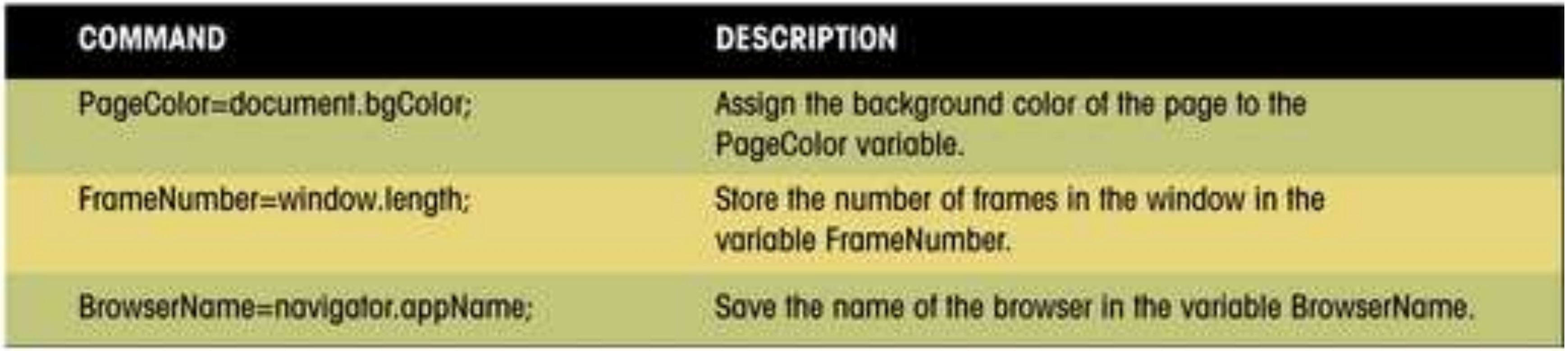
```
variable = object.property
```

  - ***variable*** is the variable name
  - ***object*** is the name of the object
  - ***property*** is the name of its property

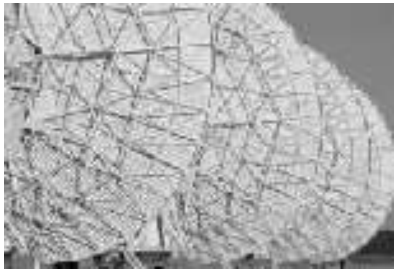


Assigning Property Values to Variables

This figure shows three examples of property values being assigned to JavaScript variables.



COMMAND	DESCRIPTION
PageColor=document.bgColor;	Assign the background color of the page to the PageColor variable.
FrameNumber=window.length;	Store the number of frames in the window in the variable FrameNumber.
BrowserName=navigator.appName;	Save the name of the browser in the variable BrowserName.

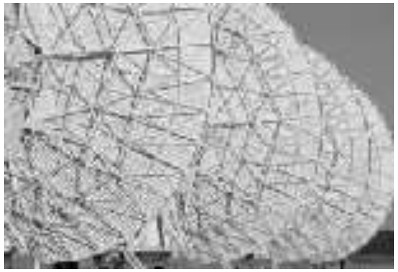


Using Property Values to Variables

- A conditional statement changes how the Web page behaves based on the value of an object property.
- The following JavaScript code shows how you can incorporate object properties into a simple conditional expression:

```
If (document.bgColor==“black”) {
  document.fgColor=“white”;
} else {
  document.fgColor=“black”;
}
```

- Using objects, properties, and conditional statement provides a great deal of control over the appearance of a Web page.

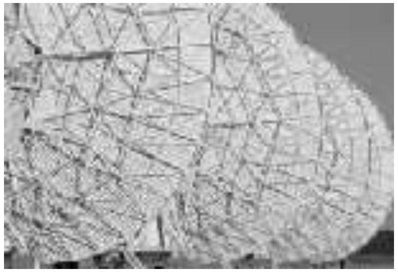


Working with Object Methods

- Another way to control a Web page is to use methods.
- **Methods** are either actions that objects perform or actions applied to objects.
- The syntax for applying a method to an object is:

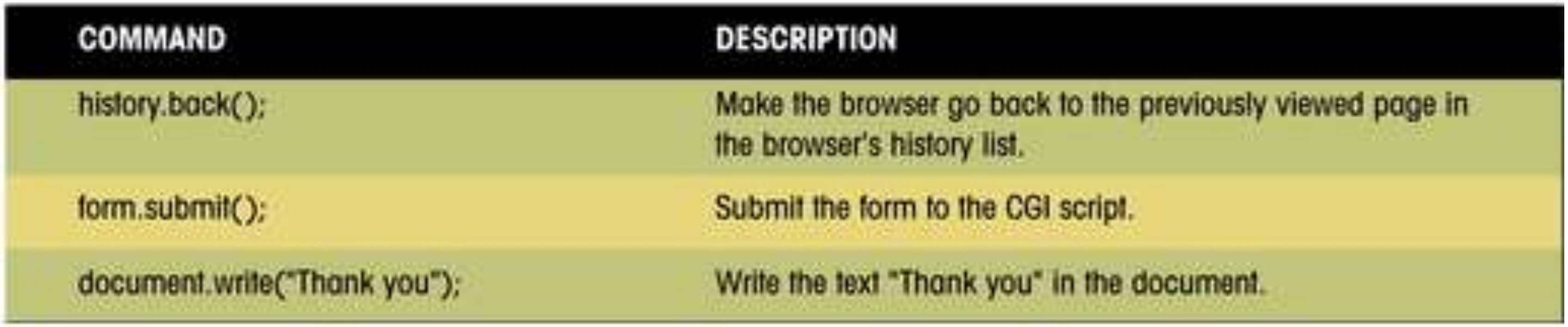
```
object.method(parameters);
```

  - ***object*** is the name of the object
  - ***method*** is the method to be applied
  - ***parameters*** are any values used in applying the method to the object

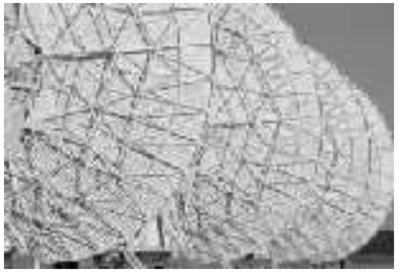


Examples of JavaScript Objects and Methods

This figure shows three examples of objects and methods.

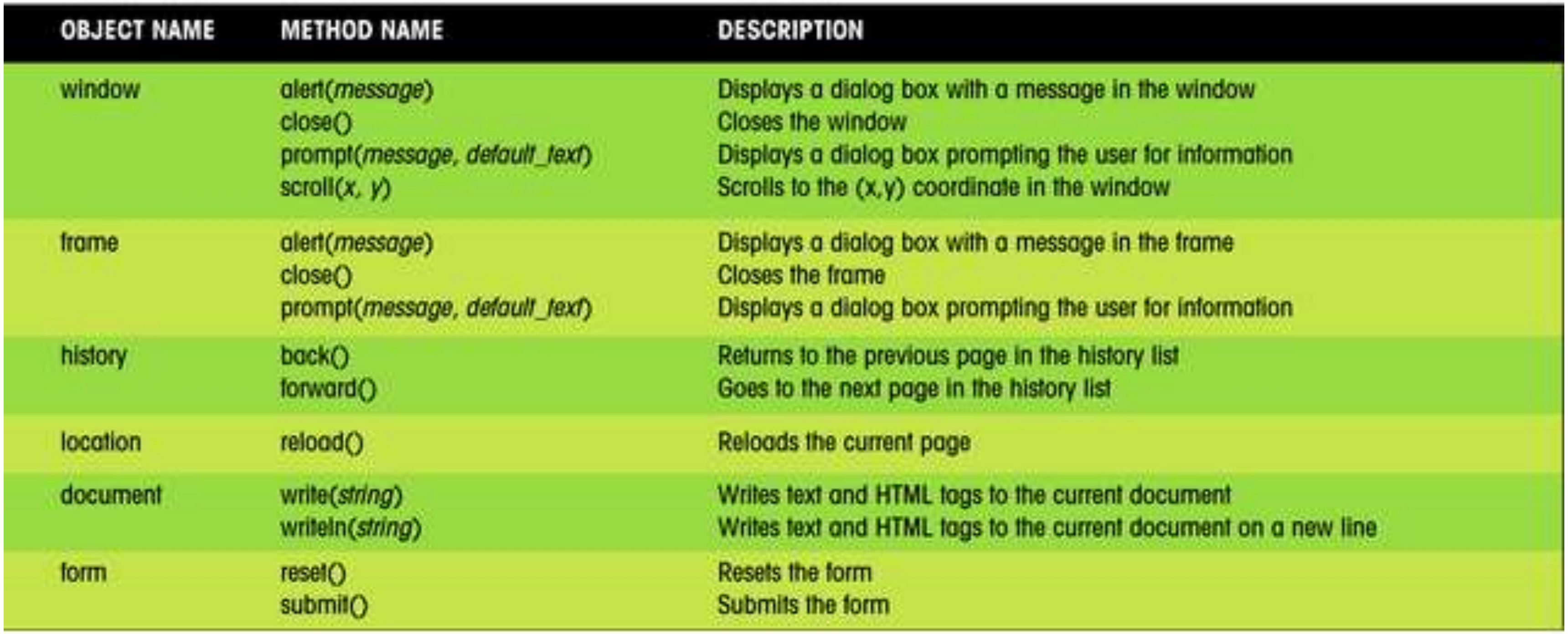


COMMAND	DESCRIPTION
history.back();	Make the browser go back to the previously viewed page in the browser's history list.
form.submit();	Submit the form to the CGI script.
document.write("Thank you");	Write the text "Thank you" in the document.



JavaScript Objects and Their Methods

This figure lists some additional JavaScript objects and some of the methods associated with them. A more complete list of objects, properties, and methods is included in Appendix G.



OBJECT NAME	METHOD NAME	DESCRIPTION

window	alert(*message*)
	close()
	prompt(*message, default_text*)
	scroll(*x, y*)
		Displays a dialog box with a message in the window
		Closes the window
		Displays a dialog box prompting the user for information
		Scrolls to the (x,y) coordinate in the window

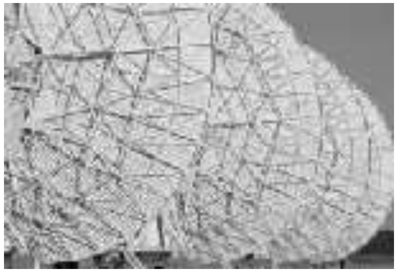
frame	alert(*message*)
	close()
	prompt(*message, default_text*)
		Displays a dialog box with a message in the frame
		Closes the frame
		Displays a dialog box prompting the user for information

history	back()
	forward()
		Returns to the previous page in the history list
		Goes to the next page in the history list

location	reload()	Reloads the current page

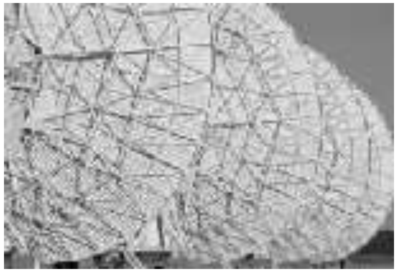
document	write(*string*)
	writeln(*string*)
		Writes text and HTML tags to the current document
		Writes text and HTML tags to the current document on a new line

form	reset()
	submit()
		Resets the form
		Submits the form



Managing Events

- An **event** is a specific occurrence within the Web browser. For example:
  - opening up a Web page
  - positioning the mouse pointer over a location on that page
- Events are an important part of JavaScript programming, you can write scripts that run in response to the actions of the user, even after the Web page has been opened.

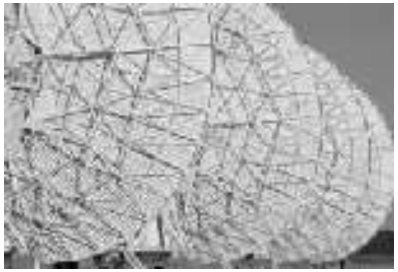


Working with Event Handlers

- Events are controlled in JavaScript using **event handlers** that indicate what actions the browser takes in response to an event.
- Event handlers are created as attributes added to the HTML tags in which the event is triggered.
- The general syntax is:

```
< tag onevent = “JavaScript commands;”>
```

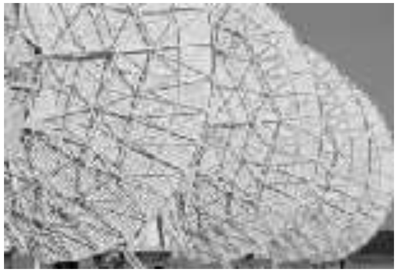
  - ***tag*** is the name of the HTML tag
  - ***onevent*** is the name of the event that occurs within the tag
  - ***JavaScript commands*** are the commands the browser runs in response to the event



JavaScript Event Holders

This figure describes event handlers that JavaScript provides.

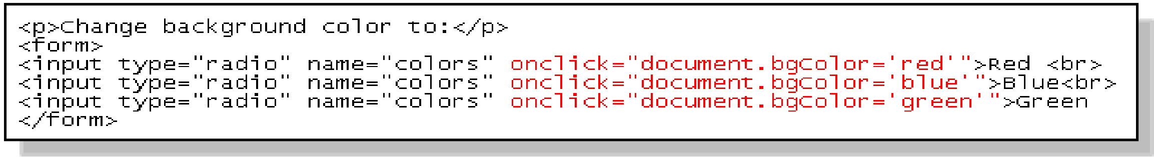
Category	Event Handler	Description	Netscape	IE
Window and Document events	onload	The browser has completed loading the document.	2.0	3.0
	onunload	The browser has completed unloading the document.	2.0	3.0
	onabort	The transfer of an image as been aborted.	3.0	4.0
	onerror	An error has occurred in the JavaScript program.	3.0	4.0
	onmove	The user has moved the browser window.	4.0	3.0
	onresize	The user has resized the browser window.	4.0	4.0
	onscroll	The user has moved the scrollbar.		4.0
Form events	onfocus	The user has entered an input field.	2.0	3.0
	onblur	The user has exited an input field.	2.0	3.0
	onchange	The content of an input field has changed.	2.0	3.0
	onselect	The user has selected text in an input or textarea field.	2.0	3.0
	onsubmit	A form has been submitted.	2.0	3.0
	onreset	The user has clicked the Reset button.	3.0	4.0
Keyboard and Mouse events	onkeydown	The user has begun pressing a key.	4.0	4.0
	onkeyup	The user has released a key.	4.0	4.0
	onkeypress	The user has pressed and released a key.	4.0	4.0
	onclick	The user has clicked the mouse button.	2.0	3.0
	ondblclick	The user has double-clicked the mouse button.	4.0	4.0
	onmousedown	The user has begun pressing the mouse button.	4.0	4.0
	onmouseup	The user has released the mouse button.	4.0	4.0
	onmousemove	The user has moved the mouse pointer.	4.0	4.0
	onmouseover	The user has moved the mouse over an element.	2.0	3.0
	onmouseout	The user has moved the mouse out from an element.	3.0	4.0



Using the Onclick Event Handler

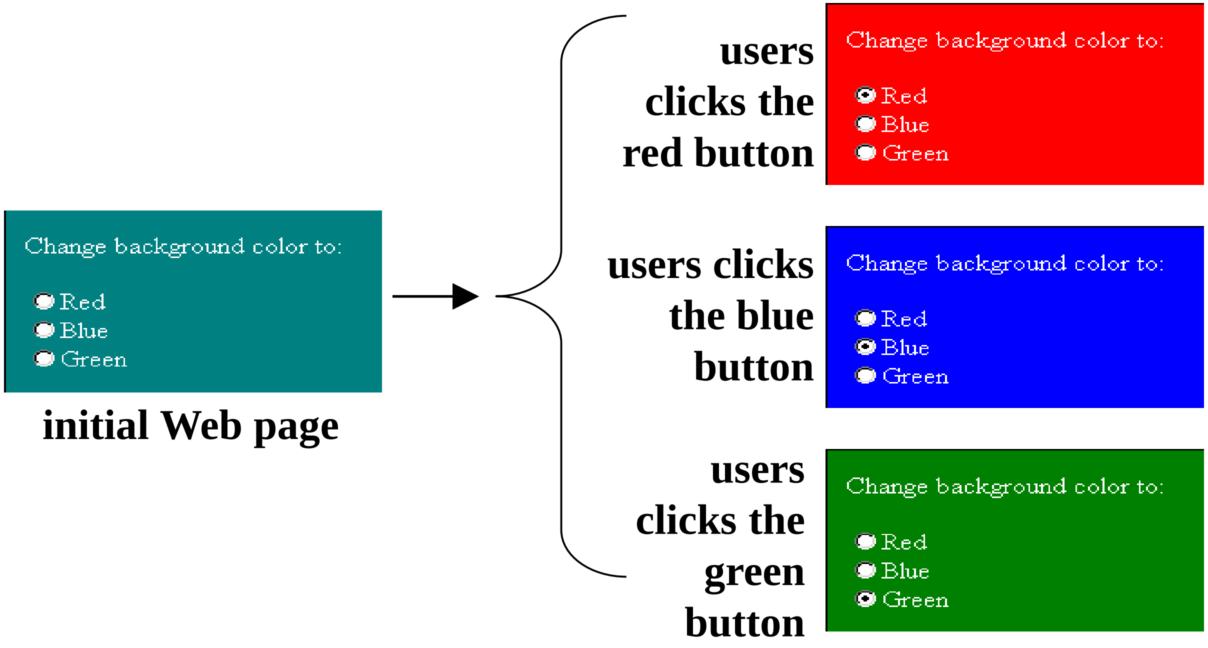
This figure shows an example of the onclick event handler used with a collection of radio buttons.

When the user clicks a radio button, the click event is initiated and the onclick event handler instructs the browser to run a JavaScript command to change the background color of the Web page.



```
<p>Change background color to:</p>
<form>
<input type="radio" name="colors" onclick="document.bgColor='red'">Red <br>
<input type="radio" name="colors" onclick="document.bgColor='blue'">Blue<br>
<input type="radio" name="colors" onclick="document.bgColor='green'">Green
</form>
```

JavaScript commands



Change background color to:

○ Red
○ Blue
○ Green

initial Web page

users clicks the red button

Change background color to:

● Red
○ Blue
○ Green

users clicks the blue button

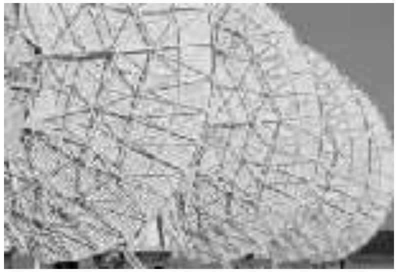
Change background color to:

○ Red
● Blue
○ Green

users clicks the green button

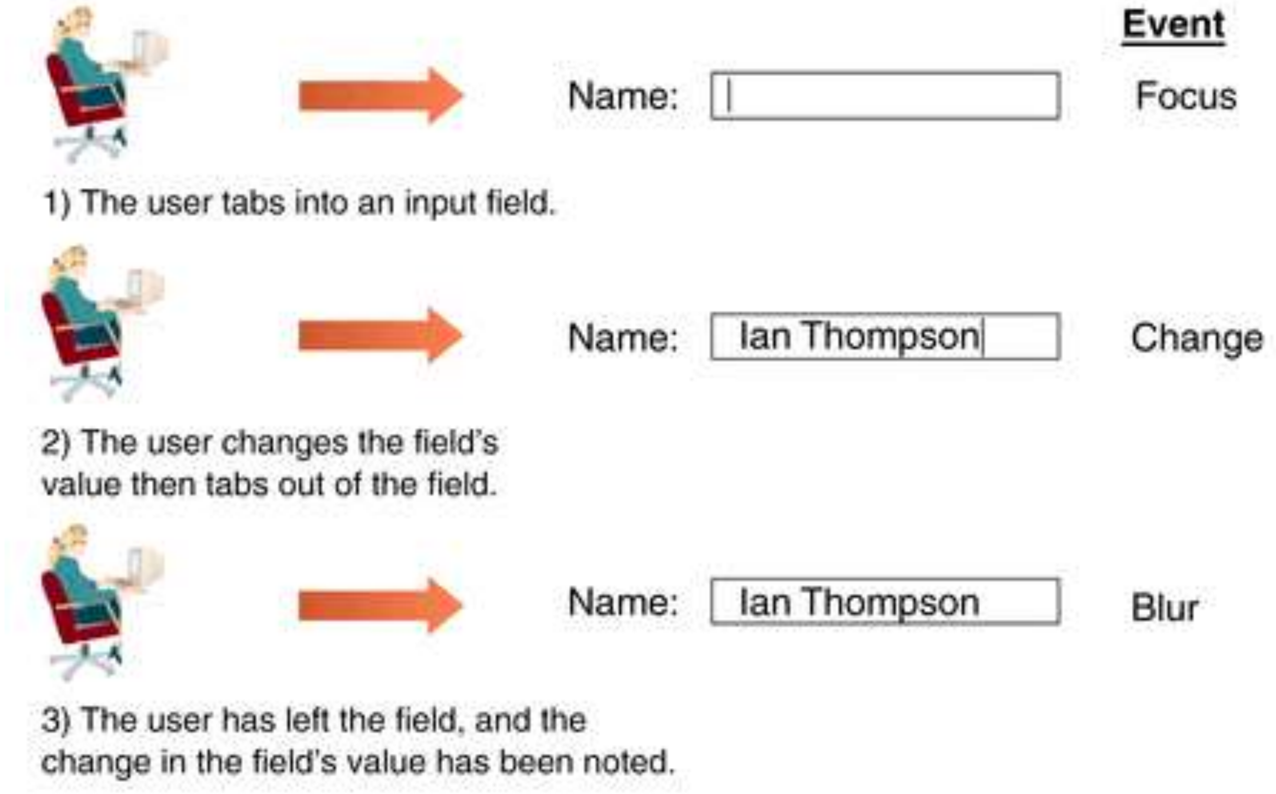
Change background color to:

○ Red
○ Blue
● Green



Events Initiated by the User During Data Entry

This figure shows that events often take place in rapid succession.



Event

Name: | → Focus

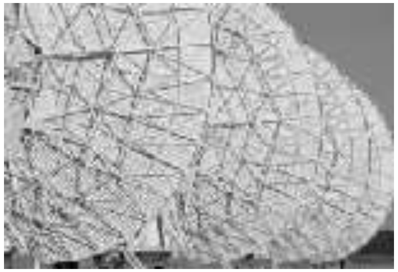
1) The user tabs into an input field.

Name: Ian Thompson| → Change

2) The user changes the field's value then tabs out of the field.

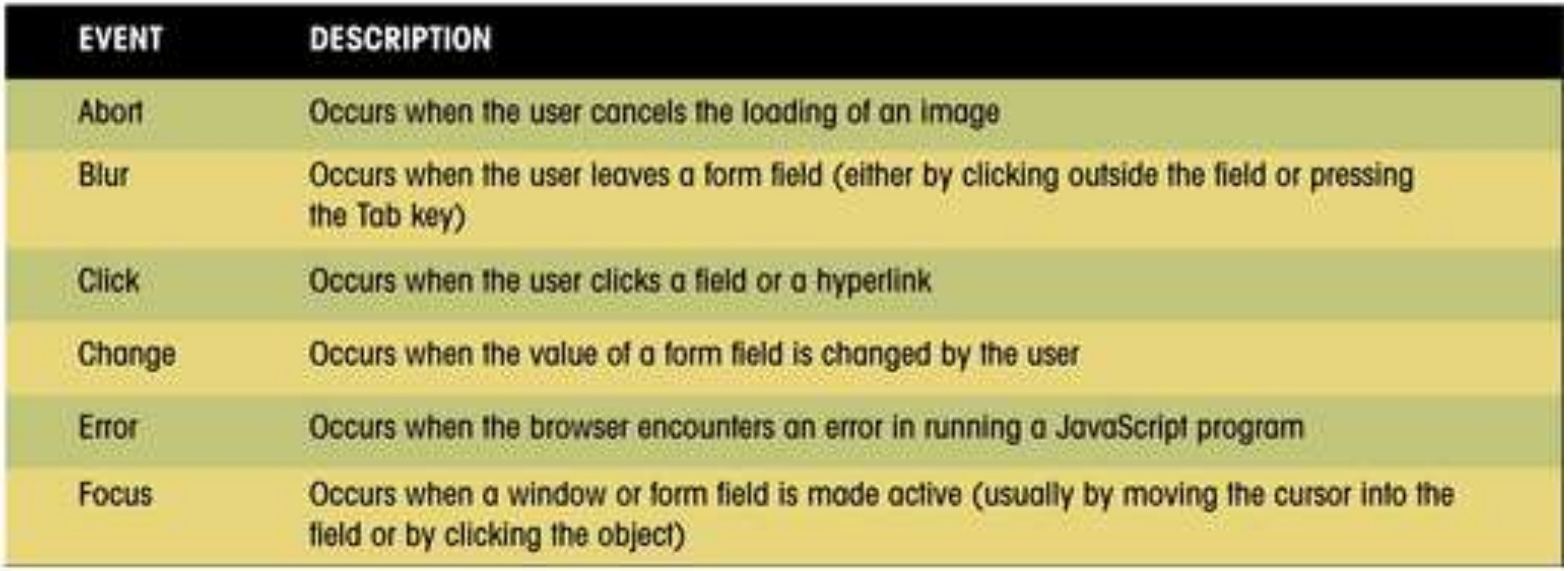
Name: Ian Thompson → Blur

3) The user has left the field, and the change in the field's value has been noted.

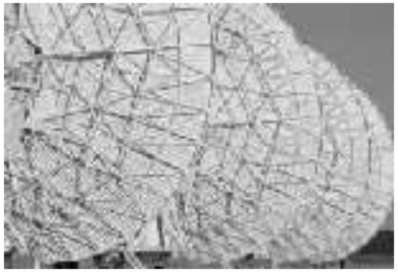


JavaScript Events

This figure shows JavaScript events.

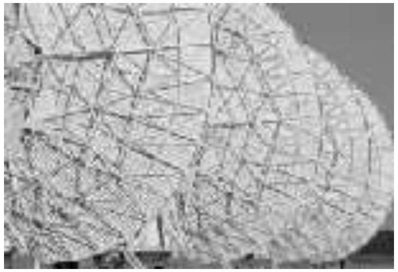


EVENT	DESCRIPTION
Abort	Occurs when the user cancels the loading of an image
Blur	Occurs when the user leaves a form field (either by clicking outside the field or pressing the Tab key)
Click	Occurs when the user clicks a field or a hyperlink
Change	Occurs when the value of a form field is changed by the user
Error	Occurs when the browser encounters an error in running a JavaScript program
Focus	Occurs when a window or form field is made active (usually by moving the cursor into the field or by clicking the object)



Browser and Event Handlers

- Generally, Internet Explorer and Netscape 6.0 can apply event handlers to most HTML tags.
- Versions of Netscape prior to 6.0 apply event handlers to a smaller number of HTML tags.
- Test Web pages with a variety of browsers and browser versions, especially if the Web page relies on JavaScript functions to operate correctly.

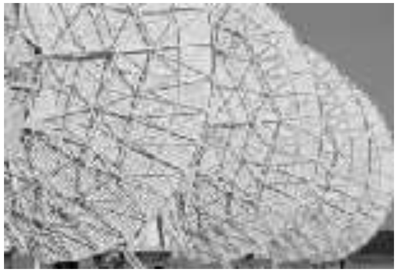


Running JavaScript Commands as Hyperlinks

- To run a command in response to the click event, an easy way of doing this is to create a hyperlink around the object to receive the mouse click.
- The syntax for doing this is:

```
<a href=“javascript:JavaScript
  commands”>Hypertext</a>
```

  – ***JavaScript commands*** are the commands you want to run when the text link Hypertext is clicked by the user

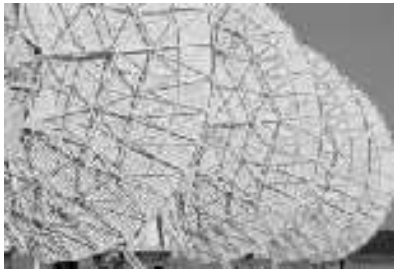


Running JavaScript Commands as Hyperlinks Continued

- The following code changes the Web page’s background color to red when the hypertext “Change background to red” is clicked.

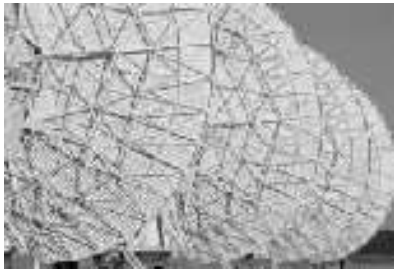
```
<a href=“javascript:document.bgcolor= ‘red’;”>
   Change background to red
</a>
```

- One advantage of this technique is that you can apply it to objects that might not support the onclick event handler in all browsers or browser versions.



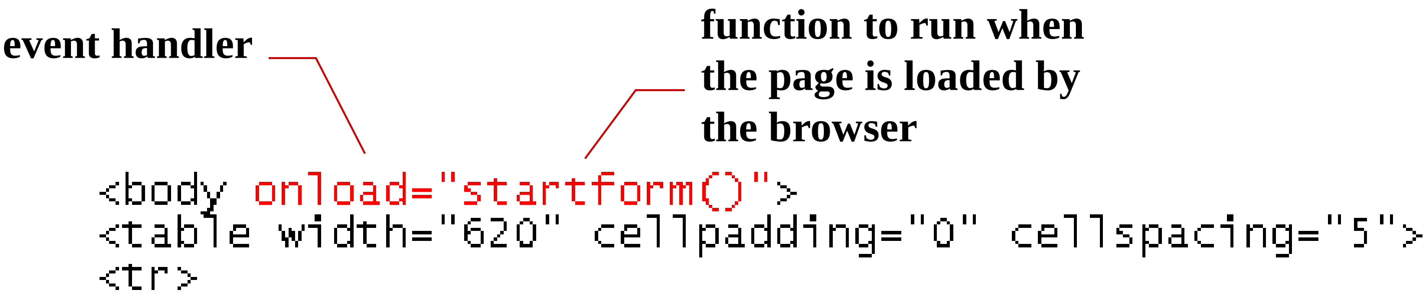
Using the `onload` Event Handler

- The event handler for loading the Web page is the **`onload`** event handler.
- This handler is associated with the document object and must be placed in the **`<body>`** tag of the HTML file.
- When the browser encounters the load event, it runs the **`startform()`** function.



Events Initiated by the User During Data Entry

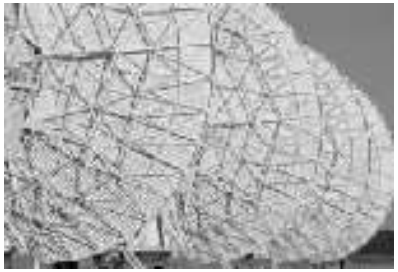
This figure shows events initiated by the user during data entry.



event handler

function to run when the page is loaded by the browser

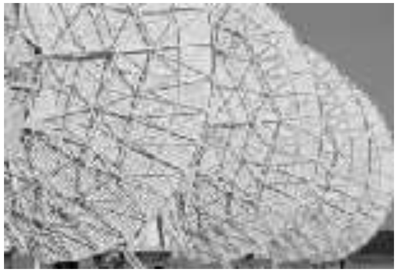
```
<body onload="startform()">
<table width="620" cellpadding="0" cellspacing="5">
<tr>
```



The `startform()` Function

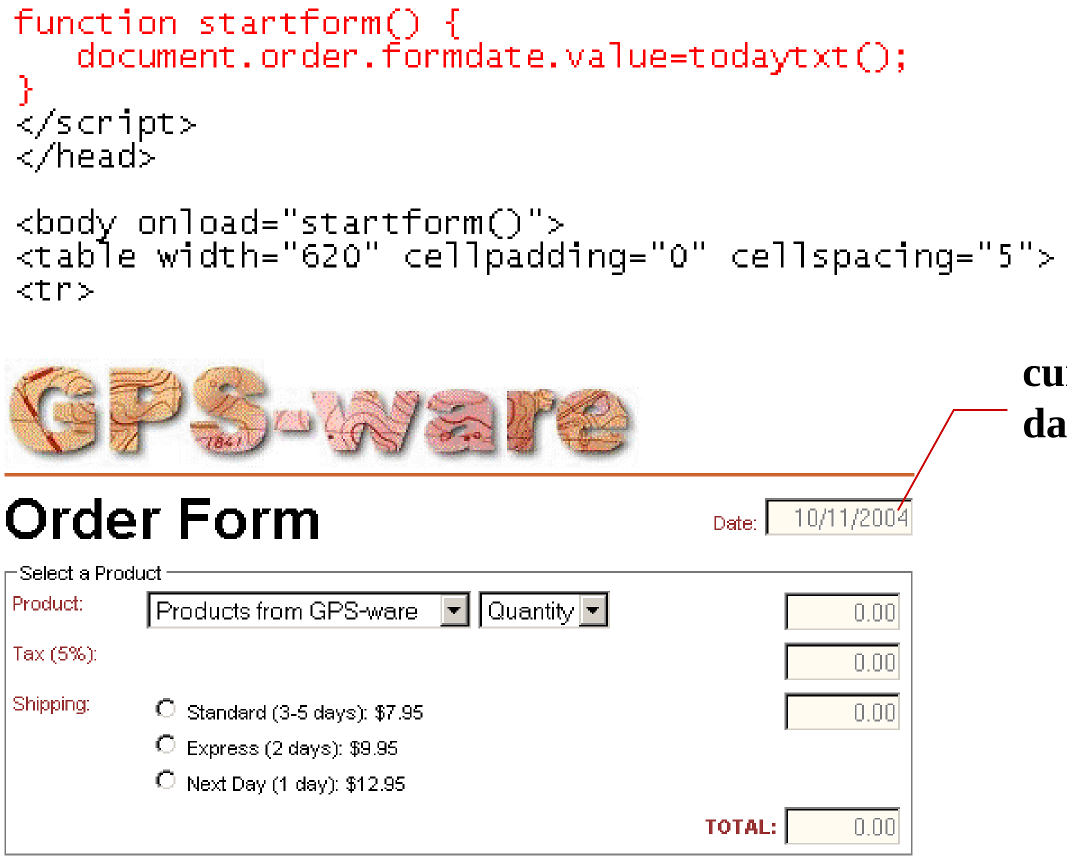
- The **`startform()`** function relies on another JavaScript function named **`todaytxt()`**.
- The code for the **`todaytxt()`** function is as follows:

```
function todaytxt() {
  var Today=new Date();
  return
  today.getMonth()+1+”/”+Today.getDate()+”
  /”+Today.
getFullYear();
}
```



Creating The `startform()` Function

This figure shows that one of the purposes of the **`startform()`** function, is to retrieve the date string and display it in the formdate field of the order form.



```
function startform() {
   document.order.formdate.value=todaytxt();
}
</script>
</head>

<body onload="startform()">
<table width="620" cellpadding="0" cellspacing="5">
<tr>
```

Order Form

Date: 10/11/2004

Select a Product

Product: Products from GPS-ware ▾ Quantity ▾ 0.00

Tax (5%): 0.00

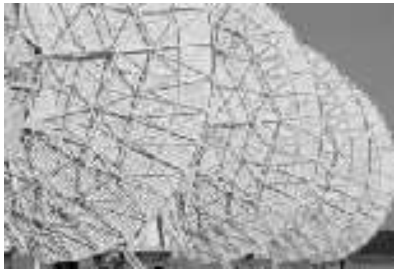
Shipping: ○ Standard (3-5 days): $7.95 0.00

○ Express (2 days): $9.95

○ Next Day (1 day): $12.95

TOTAL: 0.00

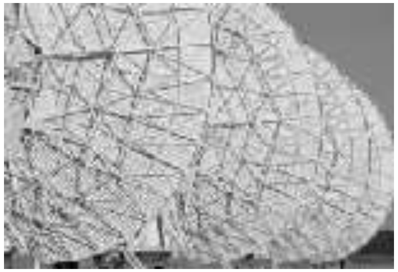
current date



Properties, Methods, and Event Handlers of Input Fields

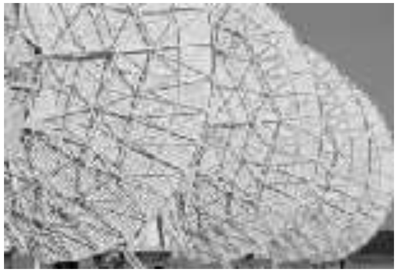
This figure shows additional properties and methods that can be associated with fields.

Property	**Description**	**IE**	**Netscape**
defaultvalue	Default value of the field	3.0	2.0
maxlength	Maximum number of characters in the field	4.0	6.0
name	The name of the field	3.0	2.0
size	The width of the field in characters	4.0	6.0
type	The type of input field	4.0	3.0
value	The value of the input field	3.0	2.0
Method	**Description**	**IE**	**Netscape**
blur()	Remove the focus from the field	3.0	2.0
focus()	Give focus to the field	3.0	2.0
select()	Select the field	3.0	2.0
Event Handler	**Description**	**IE**	**Netscape**
onfocus()	Run when the field receives the focus	3.0	2.0
onblur()	Run when the field loses the focus	3.0	2.0
onchange()	Run when the value of the field changes	3.0	2.0



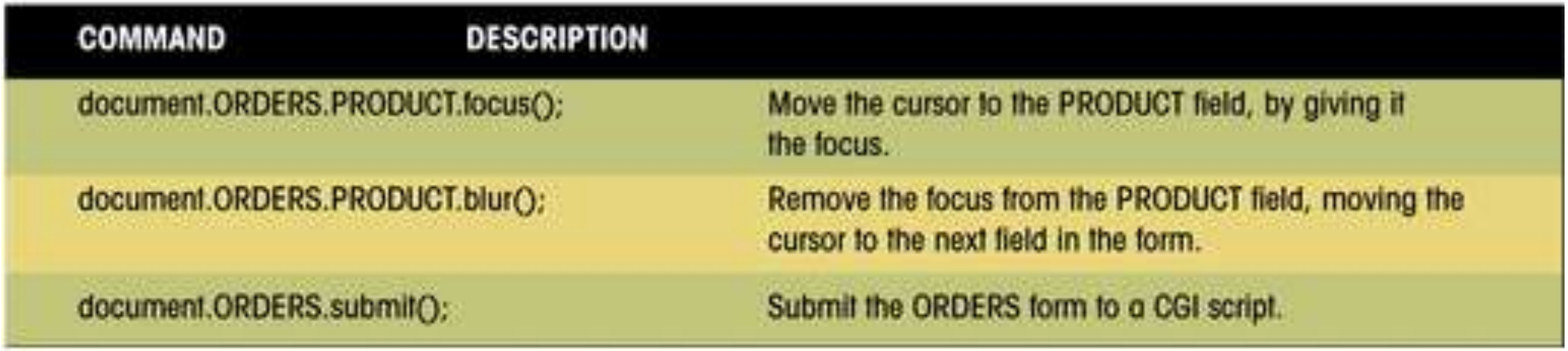
Initiating Events and JavaScript

- When using JavaScript to initiate an event, you are instructing the Web page to perform an action that a user would normally do.
  - for example, such as moving the cursor to a specific field in the form

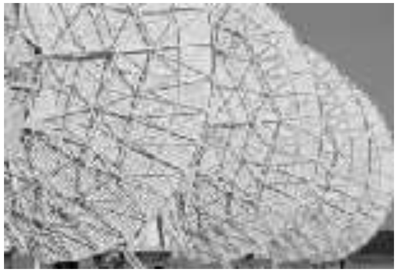


Initiating an Event with JavaScript

This figure shows three examples of JavaScript commands that initiate events in a order form.

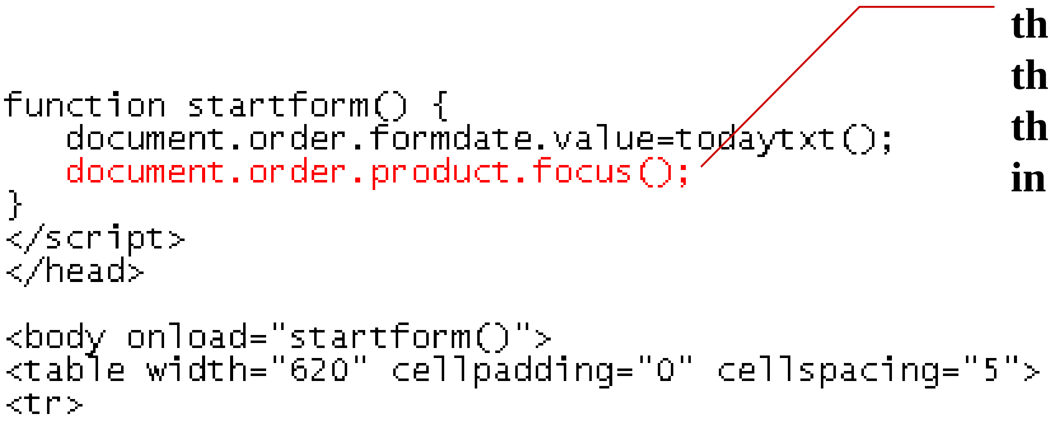


COMMAND	DESCRIPTION
document.ORDERS.PRODUCT.focus();	Move the cursor to the PRODUCT field, by giving it the focus.
document.ORDERS.PRODUCT.blur();	Remove the focus from the PRODUCT field, moving the cursor to the next field in the form.
document.ORDERS.submit();	Submit the ORDERS form to a CGI script.



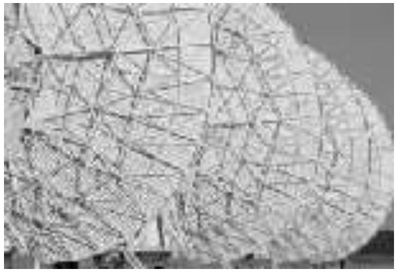
Moving the Focus to the Product Field

This figure shows an example of moving the focus to the product field.



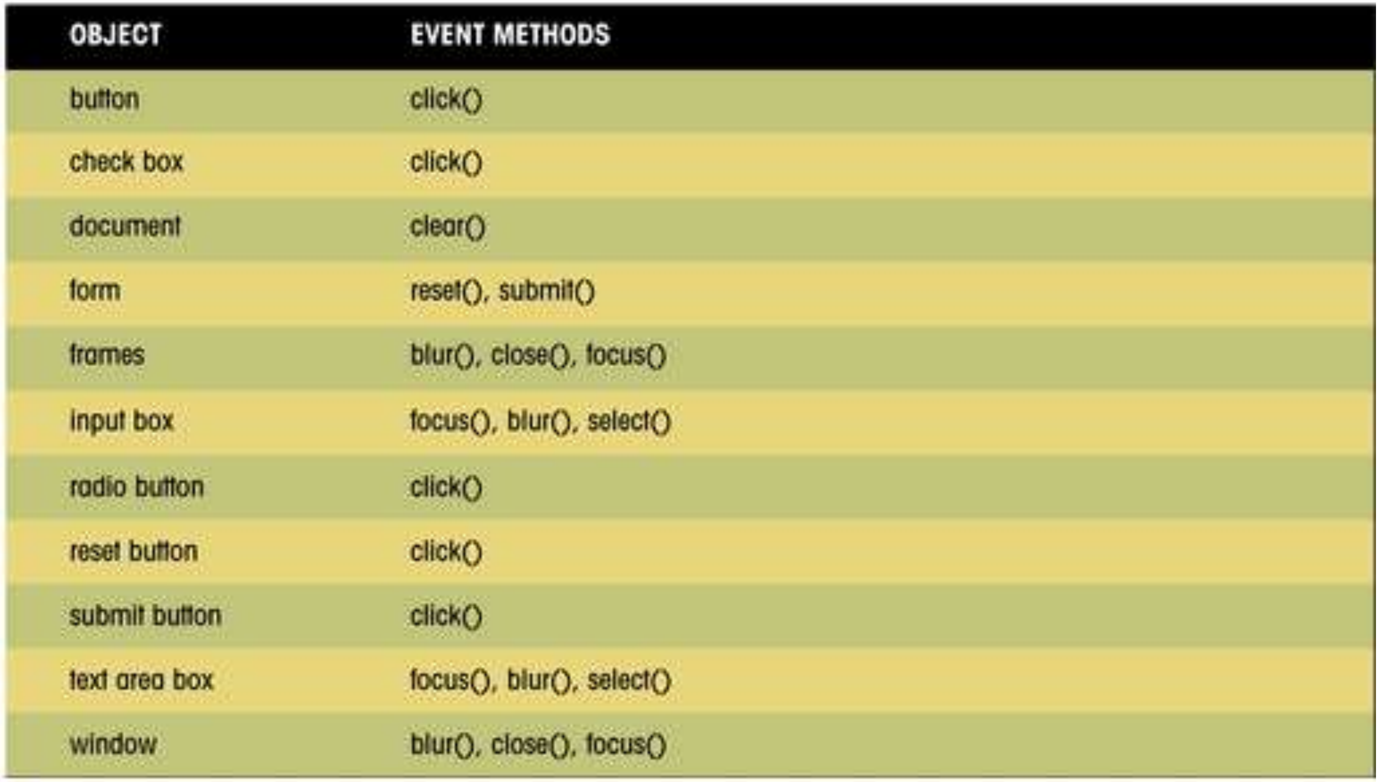
```
function startform() {
   document.order.formdate.value=todaytxt();
   document.order.product.focus();
}
</script>
</head>

<body onload="startform()">
<table width="620" cellpadding="0" cellspacing="5">
<tr>
```

the product field receives the focus of the cursor after the current date is entered in the formdate field

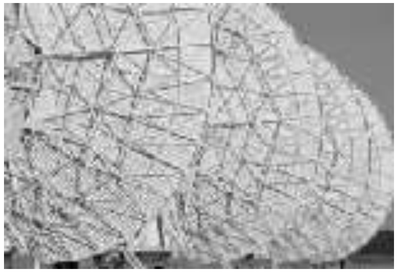


Emulating an Event with Event Methods

This figure shows additional events you can emulate in forms.

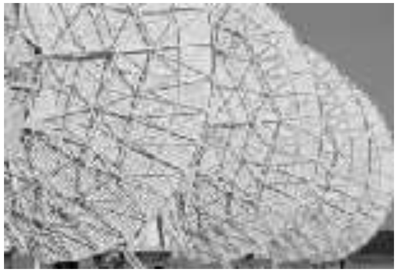


OBJECT	EVENT METHODS
button	click()
check box	click()
document	clear()
form	reset(), submit()
frames	blur(), close(), focus()
input box	focus(), blur(), select()
radio button	click()
reset button	click()
submit button	click()
text area box	focus(), blur(), select()
window	blur(), close(), focus()



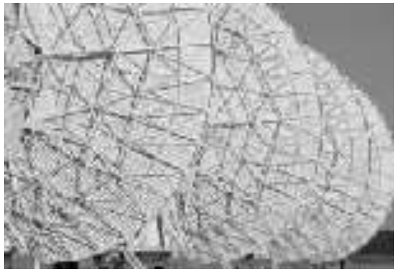
Calculate the Cost of a Customer’s Order

- You can use JavaScript to calculate the cost of a customer’s order based on product purchased, quantity, sales tax, and shipping costs.



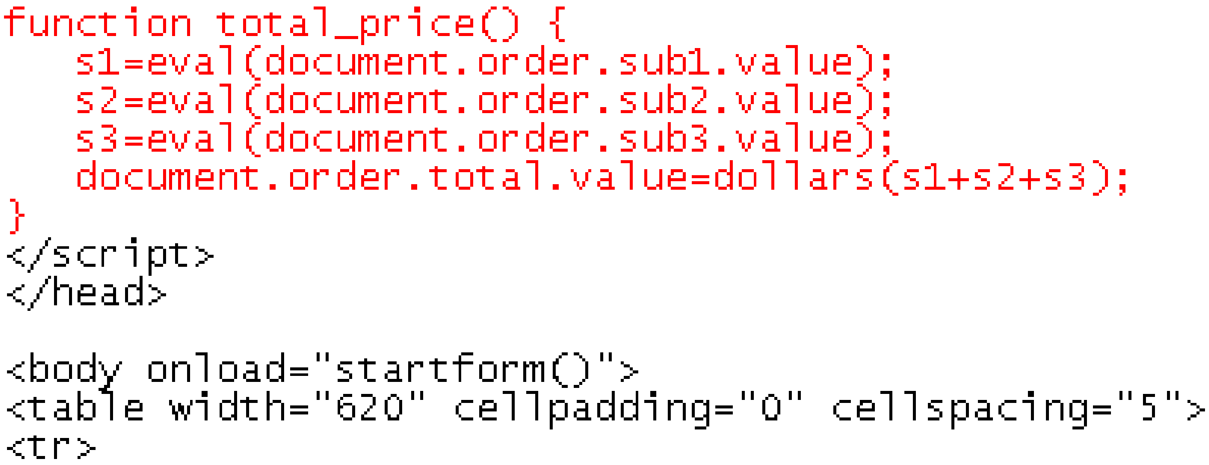
Creating a Calculated Field

- JavaScript:
  - treats the values of input fields as text strings
  - does not round off the values to nice digits
  - displays calculated values to several digits
- The **`dollar()`** function takes a value, n, and rounds it to two digits to the right of the decimal point.

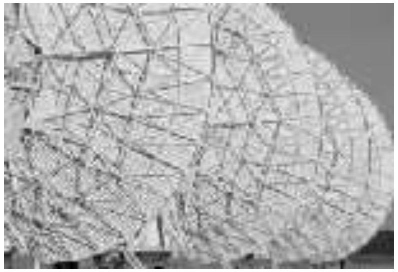


Inserting the `total_cost()` Function

This figure shows the function to calculate the total cost of a customer’s order.

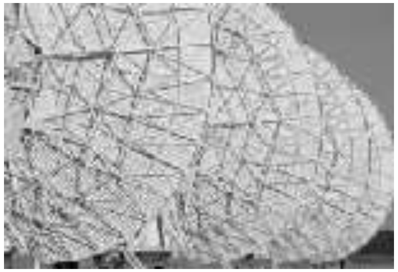


```
function total_price() {
   s1=eval(document.order.sub1.value);
   s2=eval(document.order.sub2.value);
   s3=eval(document.order.sub3.value);
   document.order.total.value=dollars(s1+s2+s3);
}
</script>
</head>

<body onload="startform()">
<table width="620" cellpadding="0" cellspacing="5">
<tr>
```



Working with a Selection List

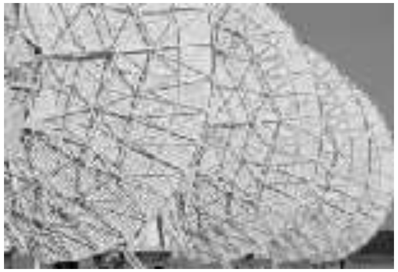
- JavaScript treats a selection list as an array of option values.



Select List Array Text and Values

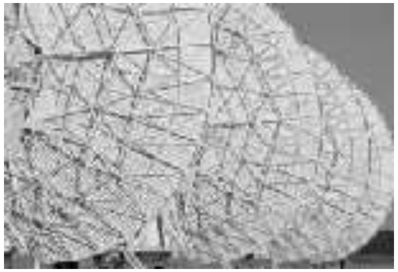
This figure shows the JavaScript object references and property values for the items in a product selection list. The array of selection options starts with an index value of 0.

Object	Object Properties	
	.text	.value
document.order.product.options[0]	Products from GPS-ware	0
document.order.product.options[1]	GoMap 1.0 ($19.95)	19.95
document.order.product.options[2]	Drive Planner 2.0 ($29.95)	29.95
document.order.product.options[3]	Hiker 1.0 ($29.95)	29.95
document.order.product.options[4]	G-Receiver I ($149.50)	149.50
document.order.product.options[5]	G-Receiver II ($199.50)	199.50
document.order.product.options[6]	G-Receiver III ($249.50)	249.50



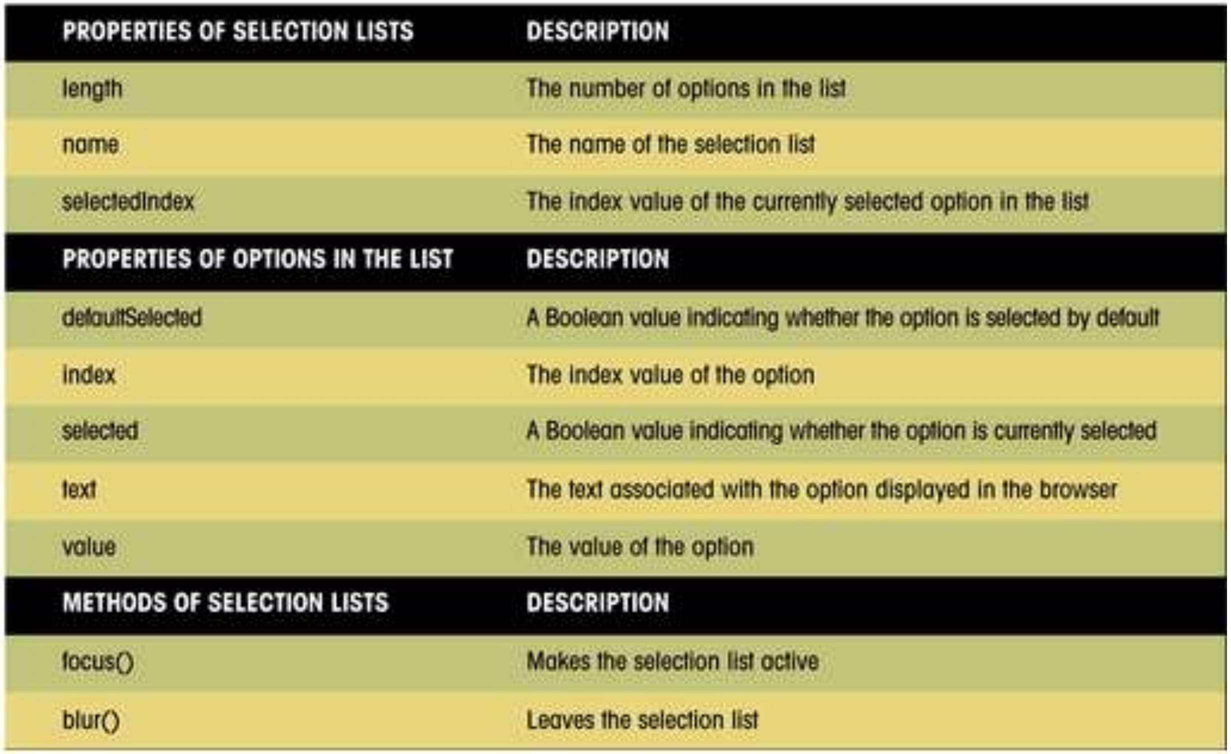
The `selectedIndex` Property

- There is no value property for the selection list itself, only for the options within the list.
- The **`selectedIndex`** property indicates the index number of the selected option.
- The index number of the selected item is stored in the item_index variable.
- The item_index variable is then used to determine the value of the selected item and stores the value in the item_value variable.
- The text of the selected item is stored in the item_text variable.

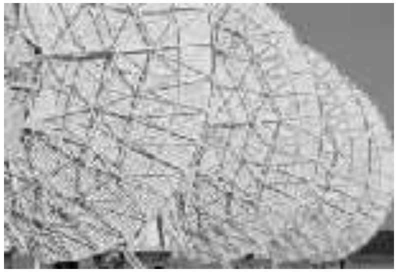


Selection Lists and Selection Options

This figure shows some of the other properties and methods associated with selection lists and selection options.

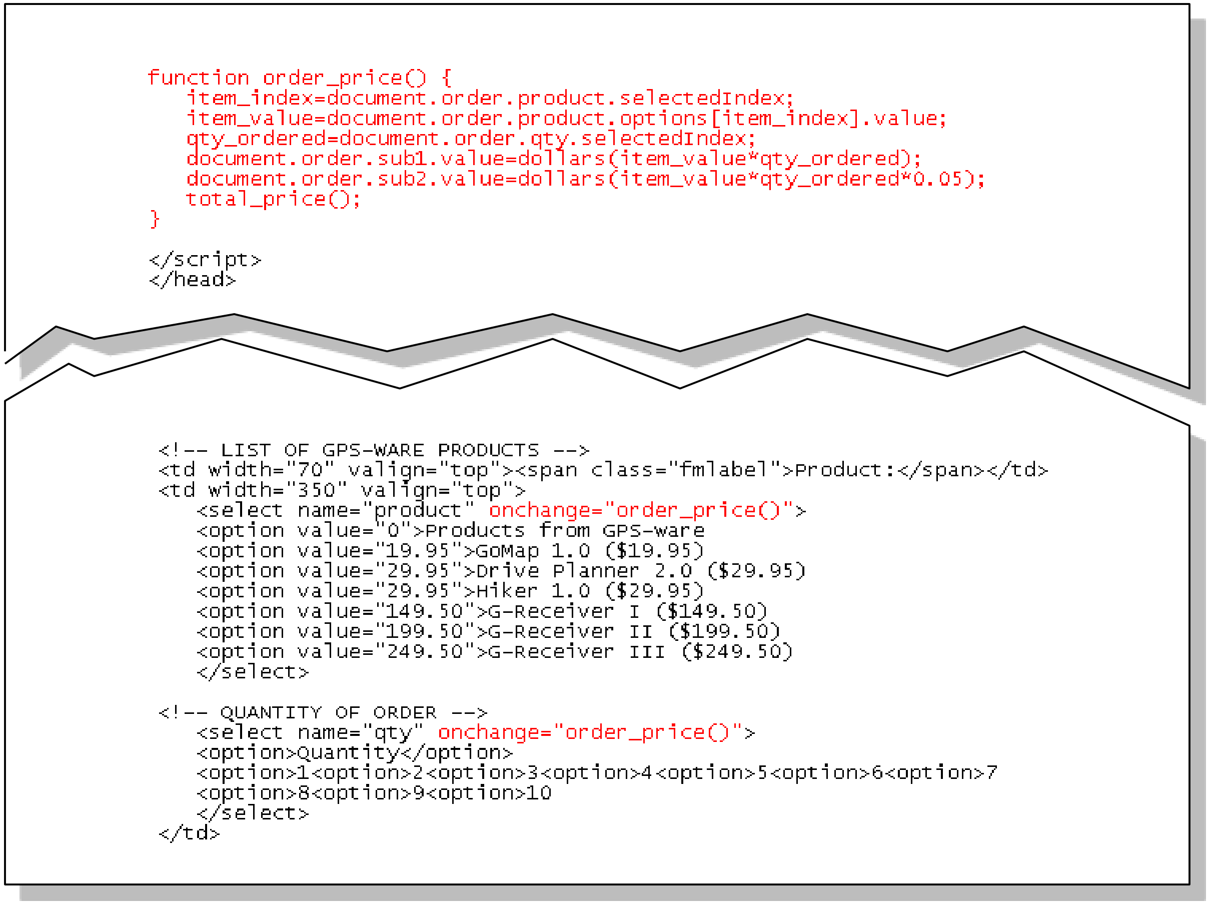


PROPERTIES OF SELECTION LISTS	DESCRIPTION
length	The number of options in the list
name	The name of the selection list
selectedIndex	The index value of the currently selected option in the list
PROPERTIES OF OPTIONS IN THE LIST	**DESCRIPTION**
defaultSelected	A Boolean value indicating whether the option is selected by default
index	The index value of the option
selected	A Boolean value indicating whether the option is currently selected
text	The text associated with the option displayed in the browser
value	The value of the option
METHODS OF SELECTION LISTS	**DESCRIPTION**
focus()	Makes the selection list active
blur()	Leaves the selection list



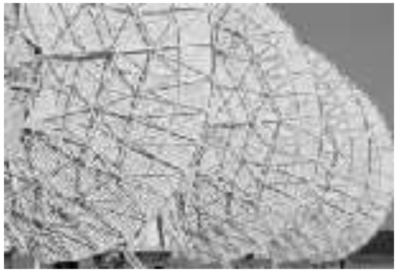
Creating the `order_price()` Function

This figure shows the `order_price()` function.



```
function order_price() {
   item_index=document.order.product.selectedIndex;
   item_value=document.order.product.options[item_index].value;
   qty_ordered=document.order.qty.selectedIndex;
   document.order.sub1.value=dollars(item_value*qty_ordered);
   document.order.sub2.value=dollars(item_value*qty_ordered*0.05);
   total_price();
}

</script>
</head>
```

```
<!-- LIST OF GPS-WARE PRODUCTS -->
<td width="70" valign="top"><span class="fmlabel">Product:</span></td>
<td width="350" valign="top">
   <select name="product" onchange="order_price()">
   <option value="0">Products from GPS-ware
   <option value="19.95">GoMap 1.0 ($19.95)
   <option value="29.95">Drive Planner 2.0 ($29.95)
   <option value="29.95">Hiker 1.0 ($29.95)
   <option value="149.50">G-Receiver I ($149.50)
   <option value="199.50">G-Receiver II ($199.50)
   <option value="249.50">G-Receiver III ($249.50)
   </select>

<!-- QUANTITY OF ORDER -->
   <select name="qty" onchange="order_price()">
   <option>Quantity</option>
   <option>1<option>2<option>3<option>4<option>5<option>6<option>7
   <option>8<option>9<option>10
   </select>
</td>
```



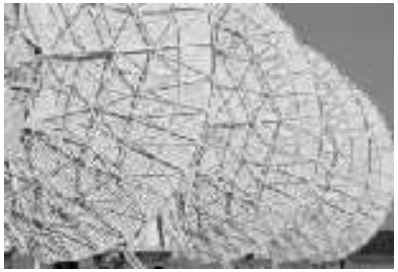
Working with Radio Buttons

- The JavaScript reference for a radio button is:

```
document.form.field[i]
```

  - ***form*** is the name of the Web page form
  - ***field*** is the name assigned to the radio button
  - ***i*** is the index number of specific radio button
- The first radio button has an index value of 0, the second button has an index value of 1, and so on.
  - the JavaScript object references for three shipping radio buttons are:

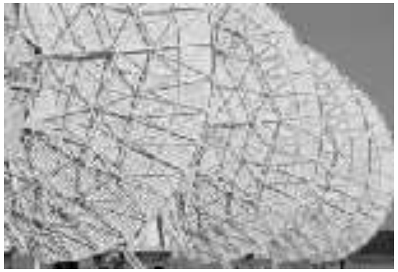
```
document.order.shipping[0]
document.order.shipping[1]
document.order.shipping[2]
```



Properties, Methods, and Event Handlers of Radio Buttons

This figure describes some of the properties, methods, and event handlers associated with radio buttons.

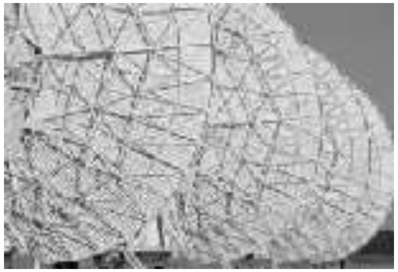
Property	**Description**	**IE**	**Netscape**
checked	A Boolean value indicating whether the radio button has been checked	3.0	2.0
name	The name of the radio button field	3.0	2.0
value	The value of radio button	3.0	2.0
Method	**Description**	**IE**	**Netscape**
focus()	Give focus to the radio button	3.0	2.0
blur()	Remove focus from the radio button	3.0	2.0
click()	Click the radio button	3.0	2.0
Event Handler	**Description**	**IE**	**Netscape**
onfocus()	Run when the radio button receives the focus	3.0	2.0
onblur()	Run when the radio button loses the focus	3.0	2.0
onclick()	Run when the radio button is clicked	3.0	2.0



Working with Radio Buttons Continued

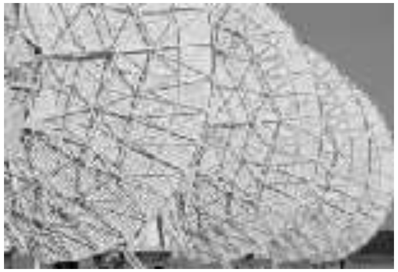
- For example, the values of the three shipping radio buttons can be expressed as follows in JavaScript:

```
document.order.shipping[0].value = “7.95”;
document.order.shipping[1].value = “9.95”;
document.order.shipping[2].value = “12.95”;
```



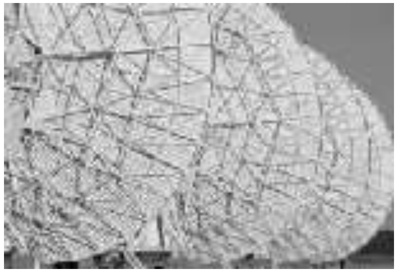
A Problem with Radio Buttons

- There is no JavaScript object that refers to the entire collection of radio buttons; thus there is no single value property that tells us which button was selected.
- There are only value properties for the individual radio buttons.
- You could treat each radio button as a different field and run a different function for each button.
- You could use an If…Then statement to test which radio button was selected.
- There is an easier way: use the "**this**" keyword.



Working the “this” Keyword

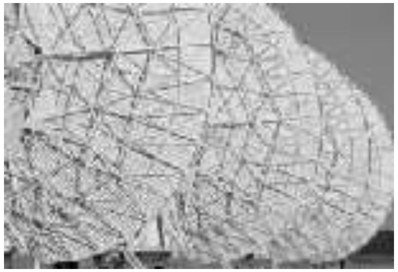
- The “**this**” keyword is a JavaScript object name that refers to the currently selected object.
- Useful in situations where several different objects on the page might access the same function.
  - in that situation, the “**this**” keyword can pass along information about the object that initiated the function



Properties, Methods, and Event Handlers of Check Boxes

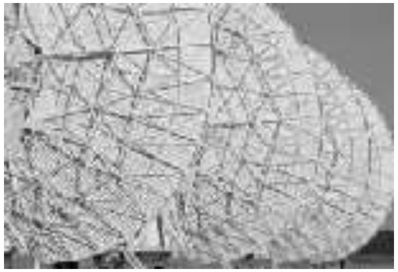
This figure lists some of the properties, methods, and event handlers of check box objects.

Property	**Description**	**IE**	**Netscape**
checked	A Boolean value indicating whether the check box has been checked	3.0	2.0
name	The name of the check box field	3.0	2.0
value	The value of the check box	3.0	2.0
Method	**Description**	**IE**	**Netscape**
focus()	Give focus to the check box	3.0	2.0
blur()	Remove focus from the check box	3.0	2.0
click()	Click the check box	3.0	2.0
Event Handler	**Description**	**IE**	**Netscape**
onfocus()	Run when the check box receives the focus	3.0	2.0
onblur()	Run when the check box loses the focus	3.0	2.0
onclick()	Run when the check box is clicked	3.0	2.0



Submitting a Form

- If a condition of the form is not met, the browser should refuse the form submission and indicate to the user why the form was not submitted.



Creating the `check_form()` Function

```
function checkform() {
   product_ok=true;
   if (document.order.sub1.value == "0.00") product_ok=false;
   if (document.order.sub2.value == "0.00") product_ok=false;
   if (document.order.sub3.value == "0.00") product_ok=false;

   shipping_ok=true;
   if (document.order.sname.value == "") shipping_ok=false;
   if (document.order.sstreet.value == "") shipping_ok=false;
   if (document.order.scity.value == "") shipping_ok=false;
   if (document.order.szip.value == "") shipping_ok=false;

   billing_ok=true;
   if (document.order.bname.value == "") billing_ok=false;
   if (document.order.bstreet.value == "") billing_ok=false;
   if (document.order.bcity.value == "") billing_ok=false;
   if (document.order.bzip.value == "") billing_ok=false;

   credit_ok=true;
   if (document.order.cname.value == "") credit_ok=false;
   if (document.order.cnumber.value == "") credit_ok=false;
   cardchecked=false;
   for (i=0;i<=5;i++) {
      if (document.order.ccard[i].checked) cardchecked=true;
   }
   if (cardchecked == false) credit_ok=false;

   payment_ok=(credit_ok || billing_ok);

   form_ok=(product_ok && shipping_ok && payment_ok);
}

</script>
</head>
```

test that a product, quantity, and shipping method has been selected

test that a shipping address has been entered

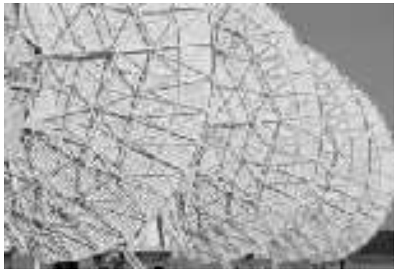
test whether a billing address has been entered

test whether a card name and number has been entered

test whether a credit card type has been selected

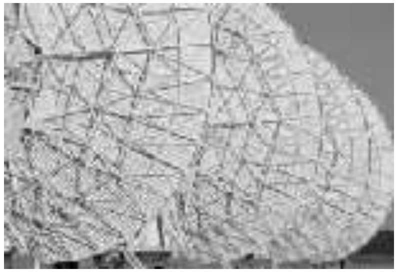
test whether the user has entered a billing address or complete credit information

test whether the entire form has been completed properly



Controlling Form Submission

- When a user completes a form and then clicks the submit button, a **submit event** is initiated.
- JavaScript provides the **`onsubmit`** event handler that allows you to run a program in response to this action.
- The submit event is associated with the form object, the event handler must be placed within the **`<form>`** tag.
- The **`onsubmit`** event handler must be able to override the act of submitting the form if the form fails a validation test.



The `onsubmit` Event Handler

- The syntax for doing this is:

  `<form onsubmit=“return function();”>`

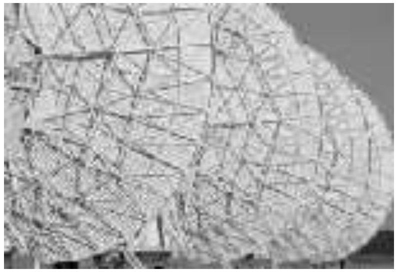
  – ***function*** is the name of the function that validates your form

  – the function must return a value of true or false

    - if the value is true, the form is submitted
    - if the value is false, the submission is canceled, and the user is returned to the form

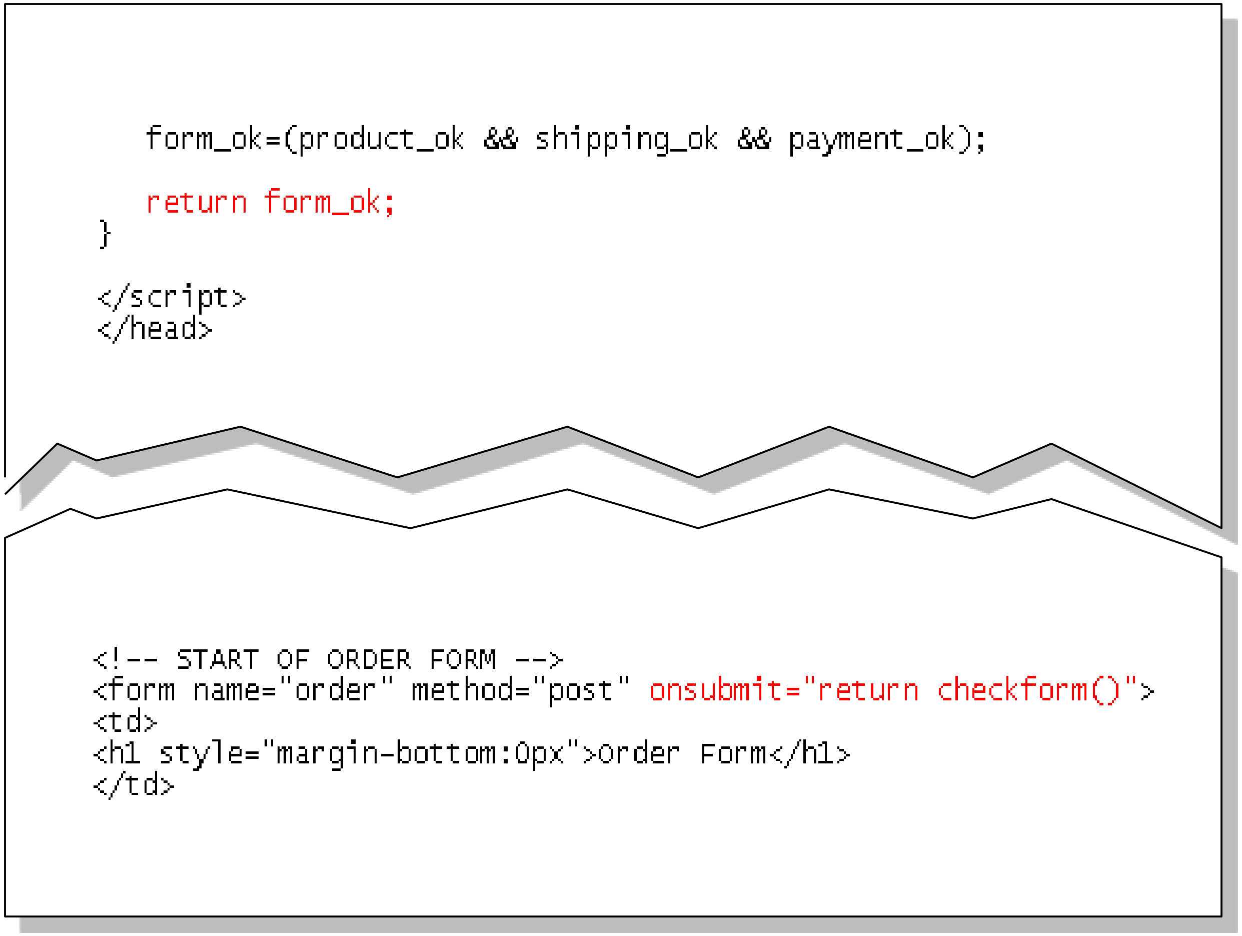
  – the keyword ***return*** in this syntax.

    - if the return keyword is not included, the browser submits the form whether or not it passes the validation test



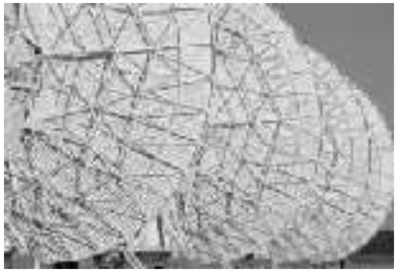
Using the `onsubmit` Event Handler

This figure shows the code to return the value of the `form_ok` variable that indicates whether or not the form was completed properly.



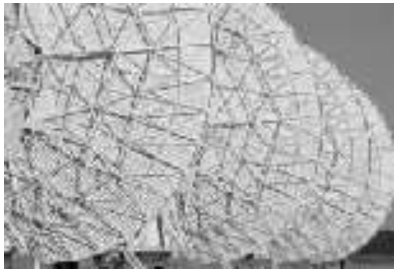
```
   form_ok=(product_ok && shipping_ok && payment_ok);

   return form_ok;
}

</script>
</head>
```

```
<!-- START OF ORDER FORM -->
<form name="order" method="post" onsubmit="return checkform()">
<td>
<h1 style="margin-bottom:0px">Order Form</h1>
</td>
```



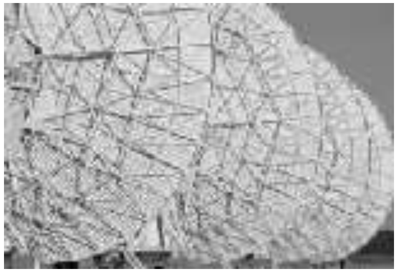
Dialog Boxes

- You may want to use a dialog box to display a message to the user to indicate that the form was not properly completed.



Creating a Dialog Box

- JavaScript supports three types of dialog boxes: alert, prompt, and confirm.
  - an **alert dialog box** displays a message, usually alerting the user to a problem.
  - the **prompt dialog box** displays both a message and a text box.
  - the **confirm dialog box** display a message along with OK and Cancel buttons.

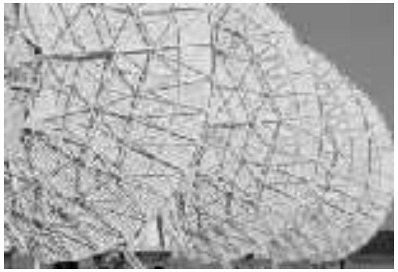


The Dialog Boxes Syntax

- The syntax for creating these dialog boxes is:

```
alert(“message”);
prompt (“message”, “default”);
confirm (“message”);
```

  - *message* is the text displayed in the dialog box
  - *default* is the default text for the prompt dialog box



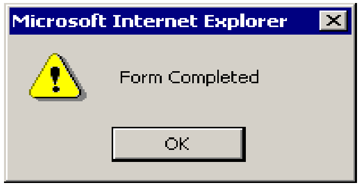
JavaScript Dialog Boxes Displayed by Internet Explorer

This figure shows examples of JavaScript dialog boxes.

Different browsers display their dialog boxes with subtle differences, but all dialog boxes share the common features of a title bar, default value, OK button, and Cancel button.

alert dialog box

alert("Form Completed")

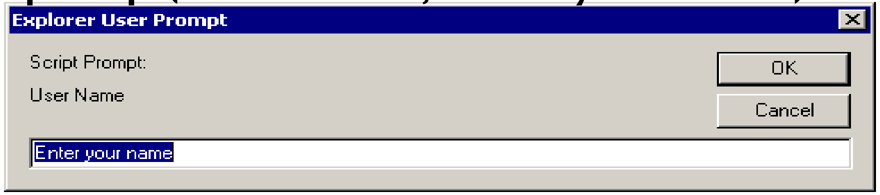


Microsoft Internet Explorer

Form Completed

prompt dialog box

prompt("User Name", "Enter your name")



Explorer User Prompt

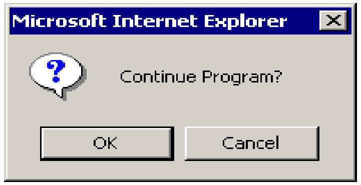
Script Prompt:

User Name

Enter your name

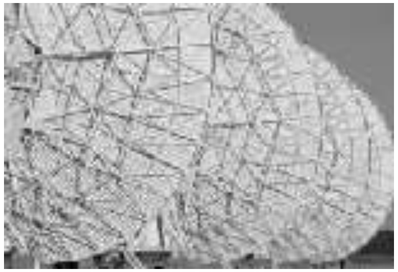
confirm dialog box

confirm("Continue Program?")



Microsoft Internet Explorer

Continue Program?

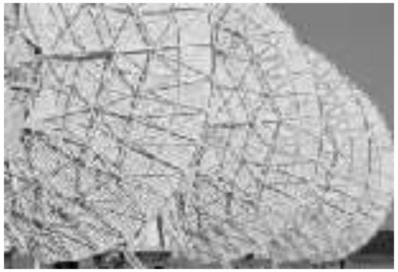


Responses to Dialog Boxes

- You can store the response of the user for both the prompt and the confirm dialog boxes.
- The syntax is:

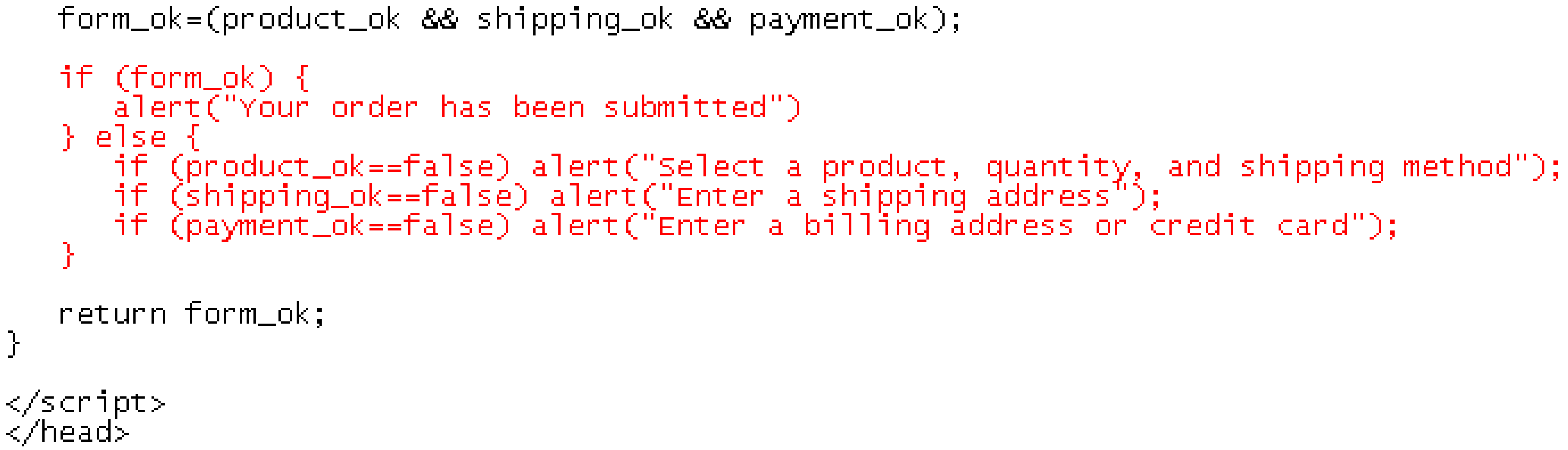
```
variable = prompt(“message”, “default”);
variable = confirm(“message”);
```

  – ***variable*** is a variable that stores the user’s response
    - in the case of the prompt dialog box, this is the contents of the text box
    - for the confirm dialog box, variable has a value of true if the user clicks the OK button and false if the user clicks the Cancel button

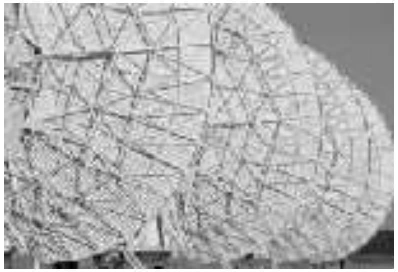


Creating Alert Dialog Boxes

This figure shows how to create an alert dialog box.

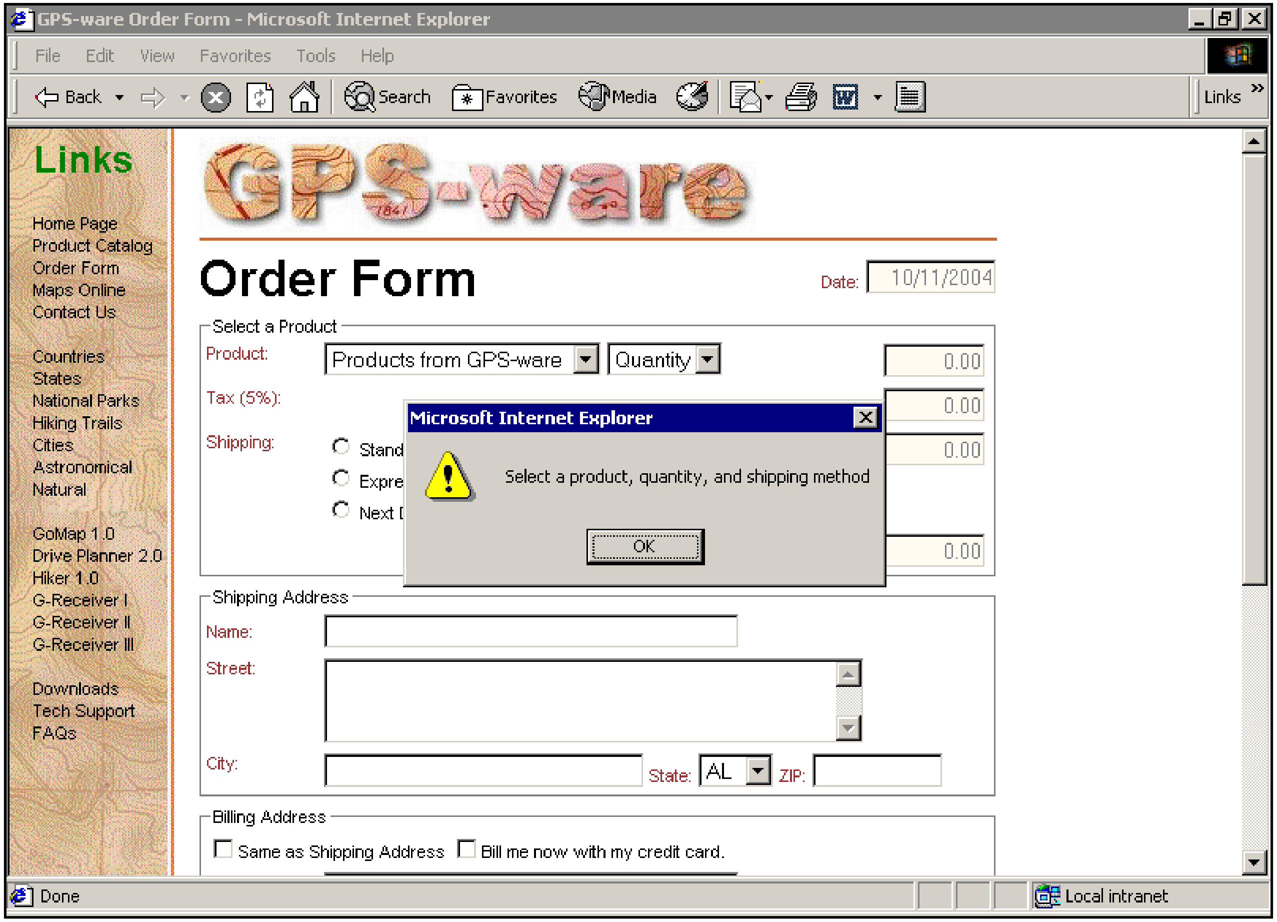


```
   form_ok=(product_ok && shipping_ok && payment_ok);

   if (form_ok) {
      alert("Your order has been submitted")
   } else {
      if (product_ok==false) alert("Select a product, quantity, and shipping method");
      if (shipping_ok==false) alert("Enter a shipping address");
      if (payment_ok==false) alert("Enter a billing address or credit card");
   }

   return form_ok;
}

</script>
</head>
```



Displaying an Alert Dialog Box

This figure shows an example of an alert dialog box.



Order Form

Date: 10/11/2004

Select a Product

Product: Products from GPS-ware Quantity 0.00

Tax (5%): 0.00

Shipping: ○ Stand ○ Expre ○ Next [0.00

0.00

Microsoft Internet Explorer

Select a product, quantity, and shipping method

Shipping Address

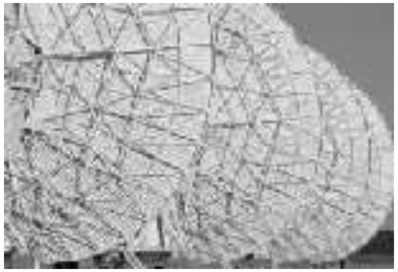
Name:

Street:

City: State: AL ZIP:

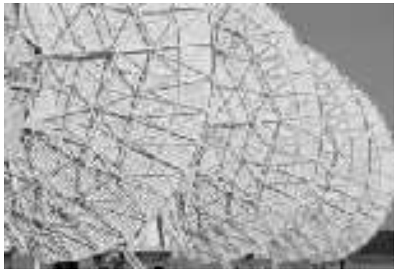
Billing Address

□ Same as Shipping Address □ Bill me now with my credit card.



Resetting a Form

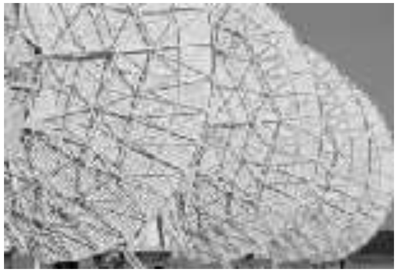
- When designing a form, it is important to allow the user to reset the form.



Resetting a Form

- Resetting a form does not load the page.
- Use JavaScript to reload the page.
  - this has the effect of resetting all field values and rerunning the startform() function that inserts the current date.
  - use the **location object** to reload a Web page
- One of the methods associated with the location object is the reload()method, which reloads the current page.
- The syntax is simply:

```
location.reload();
```

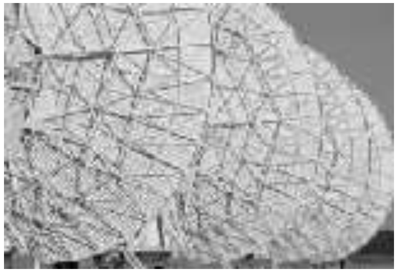


Resetting a Form Continued

- Use JavaScript to load a different page, the command is:

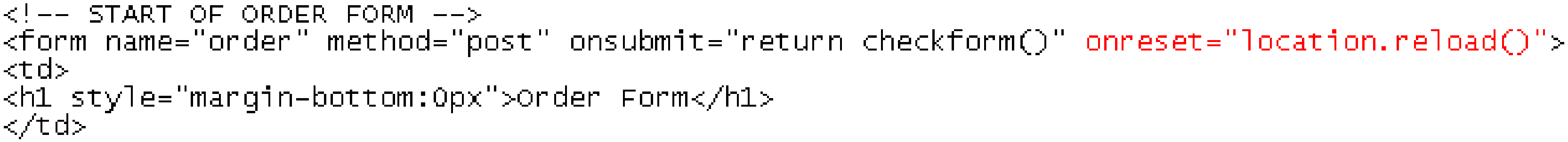
  `location="URL";`

  - ***URL*** is the address of the Web page you want to display in the browser
- To control the reset event, use the onreset handler and apply it to the **`<form>`** tag.

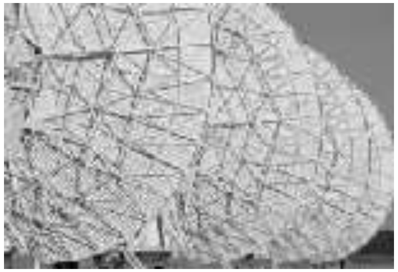


Specifying an Action for the Form Reset

This figure shows how specify an action for the form reset.

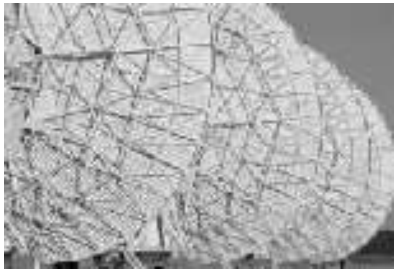


```
<!-- START OF ORDER FORM -->
<form name="order" method="post" onsubmit="return checkform()" onreset="location.reload()">
<td>
<h1 style="margin-bottom:0px">Order Form</h1>
</td>
```



Tutorial 9 Summary

- Learned the object-oriented nature of the JavaScript language.
- Used JavaScript as a validation tool for online forms.
- Learned the basic concepts of form validation.
- Introduced to object-based programming concepts i.e. objects, properties, and methods.
- Learned about events, and how to run programs in response to events.



Tutorial 9 Summary Continued

- Used event handlers and simulating events with event methods.
- Learned about form elements.
- Learned how to request information from the user by creating dialog boxes.
- Learned how to create calculated fields by working with a field’s **`value`** property.