

# Basics of JavaScript

JavaScript

# Basics of JavaScript

---

- ▶ Originally developed by Netscape
- ▶ Joint Development with Sun Microsystems in 1995
- ▶ Standard 262 (ECMA-262) of the European Computer Manufacturers Association
- ▶ Edition 6 is the current standard
- ▶ Supported by Netscape, Mozilla, Internet Explorer



# JavaScript Components

---

- ▶ Core

- ▶ The heart of the language
- ▶ Operators, expressions, statements, and subprograms

- ▶ Client-side

- ▶ Collection of objects supporting browser control and user interaction
- ▶ i.e. XHTML document can be made responsive to user inputs like mouse click using JavaScript

- ▶ Server-side

- ▶ Collection of objects that support use in web servers
- ▶ i.e. support communication with a database server

- ▶ We focus on Client-side!!



# Java and JavaScript

---

- ▶ Java and JavaScript are not the same
- ▶ Differences
  - ▶ JavaScript has a different object model from Java
  - ▶ JavaScript is not a strongly typed
    - ▶ Variables in JavaScript need not be declared and are dynamically typed

# Uses of JavaScript (Client-side)

---

- ▶ JavaScript is embedded in XHTML documents and interpreted by browsers
  - ▶ Provide alternative to server-side programming
    - ▶ Servers are often overloaded
    - ▶ Client processing has quicker reaction time
- ▶ JavaScript can **work with forms**
- ▶ JavaScript can interact with the internal model of the web page (**Document Object Model**)
  - ▶ Allows access and modify CSS properties and content of any element displayed
- ▶ JavaScript is used to **provide more complex user interface** than plain forms with HTML/CSS can provide



# Event-Driven Computation

---

- ▶ Users actions, such as mouse clicks and key presses, are referred to as events
- ▶ The main task of most JavaScript programs is **to respond to events**

- ▶ For example:

A JavaScript program could validate data in a form before it is submitted to a server

- ▶ **Caution: It is important that crucial validation be done by the server.**
  - ▶ It is relatively easy to bypass client-side controls
- ▶ **For example, a user might create a copy of a web page but remove all the validation code.**



# XHTML/JavaScript Documents

---

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



# An Example of JavaScript code enclosed in a CDATA section

---

```
<script>
  <xsl:comment>
    <![CDATA[
      var xslStylesheet = null;
      var xmlSource = null;
      var sortColumn = null;
      function display(column) {
        sortColumn.nodeValue = column;
        standings.innerHTML =
          xmlSource.documentElement.transformNode(xslStylesheet);
      }
    ]]>
  </xsl:comment>
</script>
```





# Object Orientation and JavaScript

---

- ▶ JavaScript is **not an object-oriented programming!!**
  - ▶ No classes like in C++ or Java
- ▶ It is an **object-based language**
  - ▶ JavaScript defines objects that encapsulate both data and processing
  - ▶ However, JavaScript does not have true inheritance nor subtyping
- ▶ JavaScript provides prototype-based inheritance



# JavaScript Objects

---

- ▶ Objects are collections of properties
  - ▶ Correspond to members of classes in Java and C++
- ▶ Properties are either **data properties** or **method properties**
- ▶ Data properties are either primitive values or references to other objects
- ▶ Primitive values are often implemented directly in hardware



# JavaScript in XHTML

---

- ▶ Directly embedded

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

- ▶ However, note that `<!--` will not be allowed here!

- ▶ Indirect reference

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

- ▶ Hides the script from the browser user

# JavaScript in XHTML: CDATA

---

- ▶ The `<![CDATA[ ... ]]>` block is intended to hold data that should not be interpreted as XHTML
- ▶ Using this should allow any data (including special symbols and --) to be included in the script
- ▶ This, however does not work, at least in Firefox:

```
<script type="text/javascript">  
  <![CDATA[  
    ...JavaScript here...  
  ]]>  
</script>
```

- ▶ The problem seems to be that the CDATA tag causes an internal JavaScript error



# JavaScript in XHTML

---

- ▶ This does work in Firefox

```
<script type="text/javascript">  
  /* */<br/>    ...JavaScript here...<br/>  /*]]&gt; */<br/>&lt;/script&gt;</pre></div><div data-bbox="55 562 907 791" data-label="List-Group"><ul><li>▶ The comment symbols do not bother the XML parser (only <code>/*</code> and <code>*/</code> are 'visible' to it)</li><li>▶ The comment symbols protect the CDATA markers from the JavaScript parser</li></ul></div><div data-bbox="48 938 62 965" data-label="Page-Footer"><hr/><img alt="red triangle" data-bbox="48 938 62 965"/></div><div data-bbox="884 933 948 954" data-label="Page-Footer"><p>EGCI427</p></div>
```

# General Syntactic Characteristics

---

- ▶ Identifiers

- ▶ Start with \$, \_, or letter
- ▶ Continue with \$, \_, letter or digit
- ▶ Case sensitive

- ▶ Reserved words

- ▶ i.e. break, case, catch, continue, if, else, return, void, etc.

- ▶ Comments

- ▶ //
- ▶ /\* ... \*/



# Statement Syntax

---

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

- ▶ Can be a problem:

```
return
```

```
x;
```

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



# Primitive Types

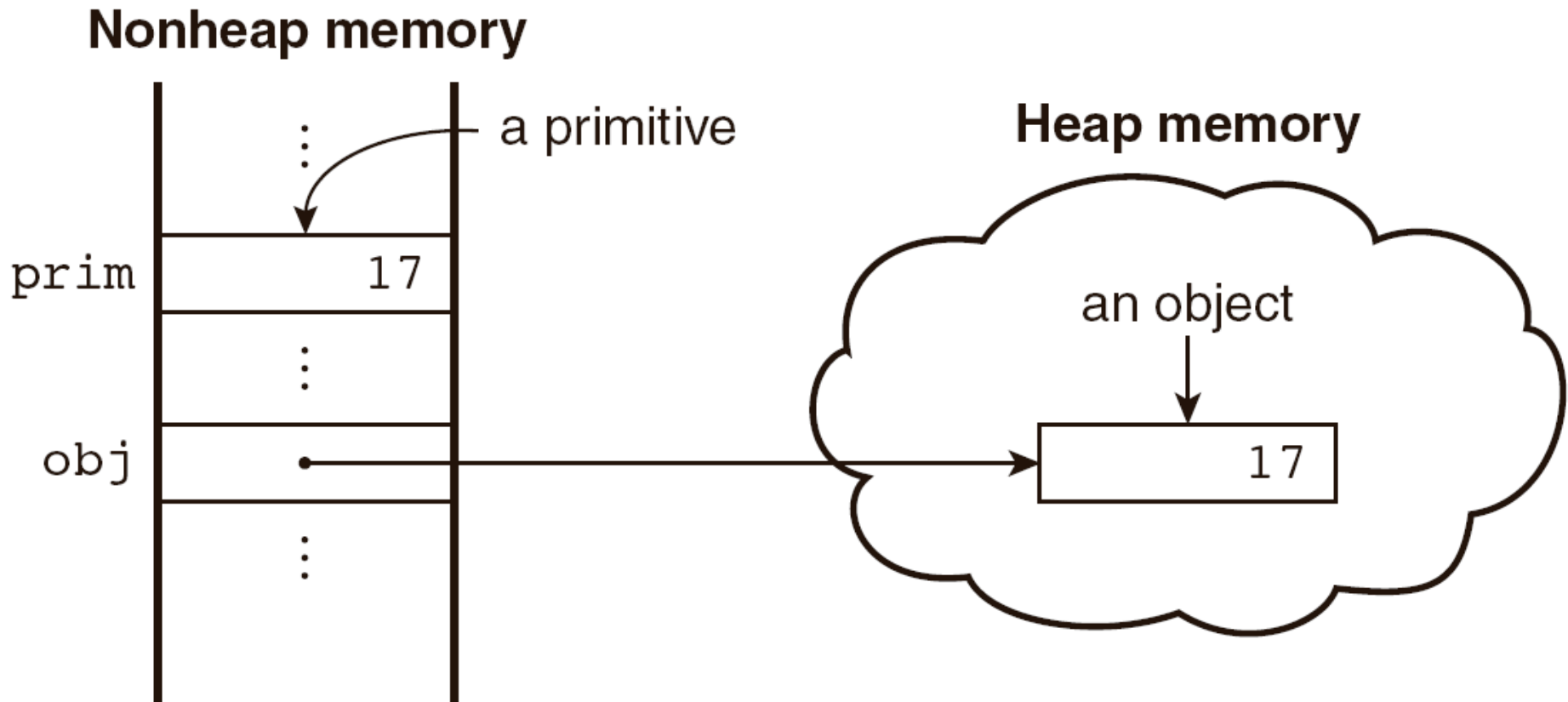
---

- ▶ Five primitive types
  - ▶ Number
  - ▶ String
  - ▶ Boolean
  - ▶ Undefined
  - ▶ Null



# Primitive and Object Storage

---



# Numeric and String Literals

---

- ▶ Number values are represented internally as double-precision floating-point values
  - ▶ Number literals can be either integer or float
  - ▶ Float values may have a decimal and/or and exponent
- ▶ A String literal is delimited by either single or double quotes
  - ▶ There is no difference between single and double quotes
  - ▶ Certain characters may be *escaped* in strings
    - ▶ \' or \" to use a quote in a string delimited by the same quotes
    - ▶ \\ to use a literal backspace
  - ▶ The empty string "" or "" has no characters
    - ▶ "D:\\bookfiles"



# Other Primitive Types

---

## ▶ Null

- ▶ A single value, null
- ▶ `null` is a reserved word
- ▶ A variable that is used but has not been declared nor been assigned a value has a null value
- ▶ Using a null value usually causes an error

## ▶ Undefined

- ▶ A single value, undefined
- ▶ However, undefined is not, itself, a reserved word
- ▶ The value of a variable that is declared but not assigned a value

## ▶ Boolean

- ▶ Two values: true and false



# Declaring Variables

---

- ▶ JavaScript is *dynamically typed*, that is, variables do not have declared types
  - ▶ A variable can hold different types of values at different times during program execution
- ▶ A variable is declared using the keyword `var`

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



# Numeric Operators

---

- ▶ Standard arithmetic

- ▶  $+$   $*$   $-$   $/$   $\%$

- ▶ Increment and decrement

- ▶  $--$   $++$

- ▶ Increment and decrement differ in effect when used before and after a variable

- ▶ Assume that  $a$  has the value 7, initially

- ▶  $(++a) * 3$  has the value 24

- ▶  $(a++) * 3$  has the value 21

- ▶  $a$  has the final value 8 in both cases

# Precedence of Operators

---

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



# Example of Precedence

---

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



# The Math Object

---

- ▶ Provides a collection of properties and methods useful for Number values
- ▶ This includes the trigonometric functions such as `sin` and `cos`
- ▶ When used, the methods must be qualified, as in `Math.sin(x)`





# The Number Object

---

## ▶ Properties

- ▶ `MAX_VALUE`
- ▶ `MIN_VALUE`
- ▶ `NaN`
- ▶ `POSITIVE_INFINITY`
- ▶ `NEGATIVE_INFINITY`
- ▶ `PI`

## ▶ Operations resulting in errors return `NaN`

- ▶ Use `isNaN(a)` to test if `a` is `NaN`

## ▶ **toString** method converts a number to string

```
var price = 427, str_price;  
str_price = price.toString();
```



# String Catenation

---

- ▶ JavaScript strings are not stored or treated as arrays (they are unit scalar values)
- ▶ The operation `+` is the string catenation operation
- ▶ In many cases, other types are automatically converted to string
- ▶ Example: value of `first` is "Freddie"  
`first + "Freeloder" → "Freddie Freeloder"`



# Implicit Type Conversion

---

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



# Explicit Type Conversion

---

- ▶ Explicit conversion of string to number
  - ▶ `Number(aString)`
  - ▶ `aString - 0`
  - ▶ Number must begin the string and be followed by space or end of string
- ▶ `parseInt` and `parseFloat` convert the beginning of a string but do not cause an error if a non-space follows the numeric part



# String Properties and Methods

---

- ▶ One property: length
  - ▶ Note to Java programmers, this is not a method!
- ▶ Character positions in strings begin at index 0
- ▶ Example:

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

# String Methods

---

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



# The `typeof` Operator

---

- ▶ Returns “`number`” or “`string`” or “`boolean`” for primitive types
- ▶ Returns “`object`” for an object or null
- ▶ Two syntactic forms
  - ▶ `typeof x`
  - ▶ `typeof(x)`
- ▶ `typeof` always returns a string



# Assignment Statements

---

- ▶ Plain assignment indicated by =
- ▶ Compound assignment with
  - ▶  $+=$      $-=$      $/=$      $*=$      $\%=$     ...
- ▶  $a += 7$  means the same as
- ▶  $a = a + 7$



# The Date Object

---

- ▶ A Date object represents a *time stamp*, that is, a point in time
- ▶ A Date object is created with the new operator
  - ▶ `var now= new Date();`
  - ▶ This creates a Date object for the time at which it was created



# The Date Object: Methods

---

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



# Window and Document

---

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




# Screen Output and Keyboard Input

---

- ▶ Standard output for JavaScript embedded in a browser is the window displaying the page in which the JavaScript is embedded
- ▶ The write method of the Document object write its parameters to the browser window
- ▶ The output is interpreted as HTML by the browser
- ▶ If a line break is needed in the output, interpolate `<br/>` into the output

```
document.write("The result is: ", result, "<br/>");
```



The result is: 42

# The alert Method

---

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

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



# The confirm Method

---

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

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



# The prompt Method

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

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



# Example of Input and Output

---

▶ roots.html





# Control Expressions

---

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

# Selection Statements

---

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

# switch Statement Syntax

---

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



# switch Statement Semantics

---

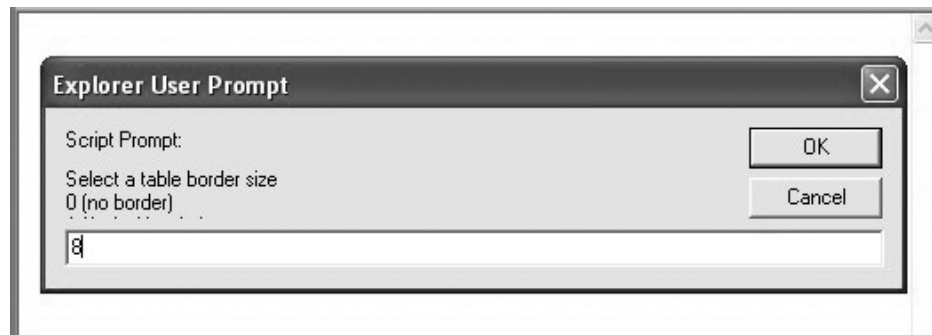
- ▶ The expression is evaluated
- ▶ The value of the expressions is compared to the value in each case in turn
- ▶ If no case matches, execution begins at the default case
- ▶ Otherwise, execution continues with the statement following the case
- ▶ Execution continues until either the end of the switch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



# Example borders2.js

---

## User Input Prompt



## Results



A screenshot of a web browser window showing the results of the script. The page title is '2006 NFL Divisional Winners'. The content is a table with a border, displaying the winners of the 2006 NFL Divisional games. The table has three columns: 'Conference', 'American Conference', and 'National Conference'. The rows represent the four divisions: East, North, West, and South.

	American Conference	National Conference
East	New England Patriots	Philadelphia Eagles
North	Baltimore Ravens	Chicago Bears
West	San Diego Chargers	Seattle Seahawks
South	Indianapolis Colts	New Orleans Saints

# Loop Statements

---

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

- ▶ While

`while` (*control expression*)  
*statement or compound statement*

- ▶ For

`for` (*initial expression; control expression; increment expression*)  
*statement or compound statement*

- ▶ do/while

`do` *statement or compound statement*  
`while` (*control expression*)



# date.js Example

---

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



# while Statement Semantics

---

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





## for Statement Semantics

---

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



# do/while Statement Semantics

---

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



# Object Creation and Modification

---

- ▶ The `new` expression is used to create an object
  - ▶ This includes a call to a *constructor*
  - ▶ The `new` operator creates a blank object, the constructor creates and initializes all properties of the object
- ▶ Properties of an object are accessed using a dot notation: *object.property*
- ▶ Properties are not variables, so they are not declared
  - ▶ An object may be thought of as a Map/Dictionary/Associative-Storage
- ▶ The number of properties of an object may vary dynamically in JavaScript



# Dynamic Properties

---

- ▶ Create my\_car and add some properties

```
// Create an Object object
```

```
var my_car = new Object();
```

```
// Create and initialize the make property
```

```
my_car.make = "Ford";
```

```
// Create and initialize model
```

```
my_car.model = "Contour SVT";
```

- ▶ The delete operator can be used to delete a property from an object

```
▶ delete my_car.model
```



# The for-in Loop

---

- ▶ Syntax

```
for (identifier in object)  
statement or compound statement
```

- ▶ The loop lets the identifier take on each property in turn in the object

- ▶ **Printing the properties in my\_car:**

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

- ▶ **Result:**

```
Name: make; Value: Ford
```

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



# Events and Event Handling

---

- ▶ *Event-driven programming* is a style of programming in which pieces of code, *event handlers*, are written to be activated when certain *events* occur
- ▶ Events represent activity in the environment including, especially, user actions such as moving the mouse or typing on the keyboard
- ▶ An *event handler* is a program segment designed to execute when a certain event occurs
- ▶ Events are represented by JavaScript objects
- ▶ *Registration* is the activity of connecting a script to a type of event
  - ▶ Assign an event attribute an event handler
  - ▶ Assign a DOM node an event handler



# Events, Attributes and Tags

---

## *Event*

blur

change

click

focus

load

mousedown

mousemove

mouseout

mouseover

mouseup

select

submit

unload

## *Tag Attribute*

onblur

onchange

onclick

onfocus

onload

onmousedown

onmousemove

onmouseout

onmouseover

onmouseup

onselect

onsubmit

onunload



# Events, Attributes and Tags

---

- ▶ Particular events are associated to certain attributes
- ▶ The attribute for one kind of event may appear on different tags allowing the program to react to events affecting different components
- A text element gets focus in three ways:
  1. When the user puts the mouse cursor over it and presses the left button
  2. When the user tabs to the element
  3. By executing the `focus` method
- ▶ Losing the focus is *blurring*





# Setting a Handler

---

- ▶ Using a an attribute, a JavaScript command can be specified:

```
<input type="button" name="myButton"  
  onclick=  
    "alert('You clicked the button!')"/>
```

- ▶ A function call can be used if the handler is longer than a single statement

```
<input type="button" name="myButton"  
  onclick="myHandler()" />
```



# Handling Events from Body Elements

---

- ▶ See the [load.html](#) example
- ▶ This example illustrates a script that is run when the page first loads



# Handling Events from Button Elements

---

- ▶ An event can be registered for this tag in two ways

```
<input type="button" name="freeOffer"
      id="freeButton"/>
```

- ▶ Using an event attribute

```
<input type="button" name="freeOffer"
      id="freeButton"
      onclick="freebuttonHandler()"/>
```

- ▶ Assigning to a property of the element node

```
document.getElementById("freeButton").onclick =
  freeButtonHandler
```

- ▶ Note that the function name, a reference to the function, is assigned
- ▶ Writing `freeButtonHandler()` would assign the return value of the function call as the handler (possible, but unlikely)



# Checkboxes and Radio Buttons

---

- ▶ Example [radio\\_click.html](#) illustrates a script that displays an alert when a radio button is clicked
  - ▶ Note that a parameter is passed to the handler function
- ▶ In example [radio\\_click2.html](#), a reference to the handler function is assigned to the appropriate property of each element node
  - ▶ Note that no parameters are passed to the function when called by the JavaScript system
  - ▶ The handler code must identify the element that caused the call
- ▶ The handler call can be enclosed in an anonymous function
  - ▶ 

```
dom.elements[0].onclick = function()  
{planeChoice(152)};
```



# Handling Events from Text Box and Password Elements

---

- ▶ By manipulating the focus event the user can be prevented from changing the amount in a text input field
  - ▶ Example [nochange.html](#) illustrates 'blurring' a field whenever it gains focus



# Validating Form Input

---

- ▶ Validating data using JavaScript provides **quicker interaction** for the user
- ▶ Validity checking on the **server requires a round-trip** for the server to check the data and then to respond with an appropriate error page
- ▶ Handling a data validity error
  - ▶ Put the focus in the field in question
  - ▶ Highlight the text for easier editing
- ▶ If an event handler returns false, default actions are not taken by the browser
  - ▶ This can be used in a Submit button event handler to check validity and not submit if there are problems
- ▶ Example **pswd\_chk.html** illustrates validity checking
  - ▶ May not work properly in FireFox



# Validating Input

---

- ▶ The validator.html example demonstrates using regular expressions to validate text input
- ▶ The name is Last, First, Middle-Initial, each part capitalized
  - ▶ `/^[A-Z][a-z]+, ?[A-Z][a-z]+, ?[A-Z]\.?$`
- ▶ The phone is ddd-ddd-dddd where d is a digit
  - ▶ `/^\d{3}-\d{3}-\d{4}$`
- ▶ Each pattern uses the `^` and `$` anchors to make sure the entire string matches



# The navigator Object

- ▶ Properties of the `navigator` object allow the script to determine characteristics of the browser in which the script is executing
- ▶ The `appName` property gives the name of the browser
- ▶ The `appVersion` gives the browser version

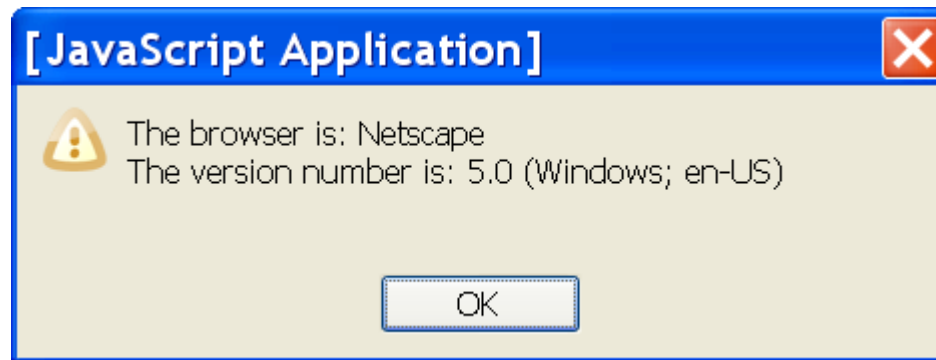




# Output From `navigate.html`

---

- ▶ Note that the browser is actually FireFox and the version is 2.0.0.4





# Dynamic Webpage

# What is a dynamic web page?

---

- ▶ Informally it is a page (XHTML document) that in some way can be changed while it is being displayed by a browser
  - ▶ JavaScript is a common approach (client-side) to implement a dynamic web page
  - ▶ Changes to the documents can be caused by user interaction, time intervals, et.



# Introduction

---

- ▶ Using DOM, JavaScript can change the document in which it is embedded
- ▶ Elements can be move
- ▶ Style can be changed
- ▶ Visibility can be changed



# Moving Elements

---

- ▶ JavaScript code can move elements by changing the top and left properties
  - ▶ Note that the position mode has to be relative or absolute for this to work
- ▶ Example mover.html illustrates dynamically placing elements
  - ▶ Text input fields are provided to enter the x and y coordinates desired for the displayed image
  - ▶ An image element has an id attribute and style to specify it as absolute position
  - ▶ An event handler on a button gets values from the text fields and uses those as parameters to a JavaScript function
  - ▶ The function gets a style node from the image element (variable dom)
  - ▶ The top and left properties of the style element are changed (note the px appended as a unit)



# Element Visibility

---

- ▶ Example [showHide.html](#) illustrates hiding and showing an element by manipulating the visibility property
- ▶ The JavaScript code accesses the style node for the image element (variable dom)
- ▶ The visibility property of the style node is altered to change the visibility of the element



# Changing Colors and Fonts

---

- ▶ Colors and font properties can be manipulated through the style property of an element

# Changing Colors

---

- ▶ Example [dynColors.html](#) illustrates setting background and foreground colors
- ▶ The change event is used which triggers a change depending on which text box was used
- ▶ Note `this` is used to refer to the input tag triggering the event





# Changing Fonts

---

- ▶ The [dynLink.html](#) example illustrates changing font properties using JavaScript
- ▶ A [mouseover event](#) on a link causes the font to change
- ▶ A [mouseout event](#) on the same link causes the font to change back to the original



# Dynamic Content

---

- ▶ By manipulating the DOM tree representing the document, the document content can be changed
  - ▶ Content of an element is accessed through the value property
- ▶ The [dynValue.html](#) example illustrates dynamic content by changing the content of a text area when the mouse moves over other components
- ▶ The `onmouseover` and `onmouseout` events are used to control this
- ▶ The value property of the 'help' box is used to change the content



# Stacking Elements

---

- ▶ `top` and `left` properties allow the placement of an element in 2D
- ▶ The `z-index` style property can be used to govern the layering of elements in the display
  - ▶ If two elements both cover a part of the window, the element with the *higher z-index value will cover the other one*
  - ▶ Think of a artist painting the document content on the screen. Elements with lower `z-index` are painted before those with higher `z-index`
- ▶ The `stacking.html` example illustrates manipulating the `z-index` property dynamically



# Reacting to a Mouse Click

---

- ▶ The [anywhere.html](#) example is another example using mouse location information
- ▶ The example uses `mousedown` and `mouseup` events to drive the action
- ▶ The position and visibility of an element are manipulated by the event handler

