SWEN 1005

MOBILE WEB PROGRAMMING

Getting Started With JavaScript

WHY USE JAVASCRIPT; ORIGINS OF JAVASCRIPT; SIMPLE USAGE EXAMPLES; DATA TYPES; CORE OBJECTS

Why JavaScript?

- HTML has one disadvantage—it has a fixed state
- Changes require a trip to the server
  - Uses server-side technology e.g. ColdFusion, ASP, ASP.NET, PHP, or JSP
  - Sends the information to the server via forms
  - Server does the calculating/testing etc.
  - Server writes the HTML document to show results
  - HTML document returned to the browser

Why JavaScript?

- Process is cumbersome and could be slow
  - A small page still needs to reload
- Not very impressive in the “Web 2.0” world
- Immediate feedback to the user is ideal
  - Need something to provide that immediate feedback without page reload, some examples:
  - Displaying errors due to bad input field values
  - Performing simple calculations or data validations

Why JavaScript?

- Reduced trips to the server speeds up the site
  - Each change does not trigger a request
  - Allows the client to perform some of the work in true C/S tradition
- Lightweight environment is ideal
  - Flash, Silverlight etc. are good, but can become large
  - JavaScript stays relatively small in comparison

The Pitfalls of JavaScript

- Gratuitous “special effects”
  - Moving page elements, pop-up windows, scrolling text, etc.
  - Other elements that provide no user benefit
- Need to support a large user agent base
  - User agents == the web browser
  - However, many other user agents exist, e.g. PDAs, set-top boxes, text-only browsers: e.g. Lynx, etc.
  - Supporting these user agents can become very difficult
  - All users won’t have the same experience due to the agent they have

The Pitfalls of JavaScript

- JavaScript can be turned off!
- No JavaScript interpreter, no JavaScript functionality

The Origins of JavaScript?

- Started out as LiveScript
- Netscape changed the name to get on the “Java” gravy train

or

- Because it became a joint-venture with Sun in 1995
- Caused confusion because there is no real connection
  - Java is to JavaScript what Car is to Carpet
    (JavaScript discussion group on Usenet)

The Origins of JavaScript?

- Was included in Netscape Navigator 2.0
  - Used an interpreter to read and execute the JavaScript added to the HTML
- Language has grown in popularity since then
  - Now a standard called ECMAScript (ECMA-262)
  - All modern browsers support it

What Is JavaScript?

- A client-side scripting language
  - JavaScript code is called a script, not a program, and is embedded in the HTML
- It is interpreted
  - The interpreter an integral part of the browser architecture

What Is JavaScript?

- It is prototype-based, not object-oriented or object-based
  - Depends on who you talk to
  - Mimics class-based inheritance
  - Mimics polymorphism
- It is a dynamically-typed language
  - Data can change types depending on usage at runtime

How JavaScript Works

- JavaScript is executed entirely in the browser
- After the script is downloaded to the browser, no other information is sent to the server
  - JavaScript can issue requests to load other pages
- JavaScript does not require Java VM
  - Scripts run quickly as a result

Simple JavaScript Usage

- Using JavaScript is easy
- Older browsers and strict XHTML require the code be commented out

```
<script type="text/javascript">
// Your code here
</script>
```

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

```
<script type="javascript">
<!--//--><![CDATA[//><!-
// Your code here
//--><!]]>
</script>
```

JavaScript Syntax - Basic Elements

Syntax Element	Description
//	Indicates the rest of the current line is a comment
/* */	Indicates that the comment covers multiple lines starting with /* and ending with */.
{ }	Indicate start of a block of code. Primarily used in functions and if..then..else statements etc.
;	Defines the end of a statement. **Note**: A newline also defines the end of a statement, making semicolons an optional end-of-line delimiter. Should be included, however, since it makes the code more readable
Identifiers	Start with a letter or underscore and followed by any number of letters, underscores, and digits. Case sensitive. 25 reserved words

JavaScript Data Types

- JavaScript values have one of the five data types:
  - Number
    - Indicates a number including floating point
    - All numeric values are stored in double-precision floating point
  - String
    - A series of characters, e.g. “here are characters”
    - String literals are delimited by either ' or "
  - Boolean: contains either true or false
  - Undefined: indicates that something has not be defined and given a value. This is very important to note when working with variables
  - Null: indicates that there is no data

JavaScript Data Types

- Number, String, and Boolean
  - Have wrapper objects
- Number and String are special cases
  - Scalar values and objects of these types can be interchanged
  - This means scalar types can be treated like objects and vice-versa
- Remember: JavaScript is dynamically typed
  - Any variable can be used for anything (scalar value or reference to any object)
  - The interpreter determines the type of a particular occurrence of a variable at runtime

JavaScript Objects

- There are seven core objects
  - Array: stores multiple values in logical contiguous storage and accessed by a single variable
  - Boolean: represents true and false values
  - Date: manipulates dates and times
  - Function: Every JavaScript function is a Function object
  - String: manipulates text
  - Math: properties and methods to perform mathematical tasks
  - RegExp: provides properties and methods to use regular expressions for pattern matching

The JavaScript String Object

- The number of characters in a string is stored in the **`length`** property

```
var newStringOjbect = new String("George");
var len = newStringObject.length;
```

The JavaScript String Object

- Some String Object methods:

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

The JavaScript String Object

- Example using indexOf() method

```
<html>
 <body>
  <script type="text/javascript">
    var userEmail= prompt("Please enter your email address ", "" );
    document.write( userEmail.indexOf( "@" ) );
  </script>
 </body>
</html>
```

Delimiters

- JavaScript expects strings to be enclosed
  - Enclosing is done using either single- or double-quotes
  - These enclosing marks are called delimiters
- Either method of delimiting strings is acceptable
  - Note: strings must be closed the way they are opened
  - “some string” (right) “some string’ (wrong)

Delimiters

- Including one type of quote in a string is possible
  - When including a single quote, delimit with a double quote
  - When including a double-quote, delimit with a single quote
- Including both types requires escape sequences
  - \’ or \”

Escape Sequences

- Non-keyboard characters via escape sequences

Escape Sequences	Character Represented
\b	Backspace.
\f	Form feed.
\n	Newline.
\r	Carriage return.
\t	Tab.
\'	Single quote.
\"	Double quote.
\\	Backslash.
\xNN	*NN* is a hexadecimal number that identifies a character in the Latin-1 character set (the Latin-1 character is the norm for English-speaking countries).
\uDDDD	*DDDD* is a hexadecimal number identifying a Unicode character.

The JavaScript Date Object

- Create a Date object with the new keyword

  var today = new Date(); // no parameters required

- Local time methods of Date:
  - toLocaleString – returns a string of the date
  - getDate – returns the day of the month
  - getMonth – returns the month of the year (0 – 11)

The JavaScript Date Object

- Local time methods of Date:
  - getDay – returns the day of the week (0 – 6)
  - getFullYear – returns the year
  - getTime – returns the # of milliseconds since January 1, 1970
  - getHours – returns the hour (0 – 23)
  - getMinutes – returns the minutes (0 – 59)
  - getMilliseconds – returns the millisecond (0 – 999)

The JavaScript Date Object

- The date object has many methods
  - Can get and set local date/time, i.e. time on the user's computer
    - Beware when using local time since users can set their time zone incorrectly
  - Can get/set UTC (Coordinated Universal Time)
- Date takes a number of parameters
  - Date(aYear, aMonth, aDate, anHour, aMinute, aSecond, aMilliSecond)
  - You can leave off a parameter, but cannot skip a parameter
    - e.g. new Date(2009, 1, , 14) is not allowed

JavaScript Operators

Operator	What It Does
+	Adds two numbers together or concatenates two strings.
-	Subtracts the second number from the first.
*	Multiplies two numbers.
/	Divides the first number by the second.
%	Finds the modulus—the reminder of a division. For example, 98 % 10 = 8.
--	Decreases the number by 1: only useful with variables, which we'll see at work later.
++	Increases the number by 1: only useful with variables, which we'll see at work later.

Creating JavaScript Variables

- Most programming languages require variables be declared before being used
- In JavaScript, simply using a variable creates it
  - Best practice is to always declare variables before use
- Variables can be declared using the `var` keyword:
  - `var` declares variables with local scope in functions

Creating JavaScript Variables

- Variable names must be a sequence of letters and digits and cannot:
  - Begin with a digit
  - Contain spaces
  - Contain mathematical operators
  - Contain special characters like & % ^ etc.

Converting Data Types

- The interpreter generally works out the data type to be used

```
<html>
  <body>
    <script type="text/javascript">
      var myCalc = 1 + 2;
      document.write("The calculated number is " + myCalc );
    </script>
  </body>
</html>
```

Converting Data Types

- However, with user input
  - The code below works out an answer of 12 if you input 2. Why?

```
<html>
  <body>
    <script type="text/javascript">
      var userEnteredNumber = prompt( "Please enter a number”, "" );
      var myCalc = 1 + userEnteredNumber;
      var myResponse = "The number you entered + 1 = " + myCalc;
      document.write( myResponse );
    </script>
  </body>
</html>
```

Converting Data Types

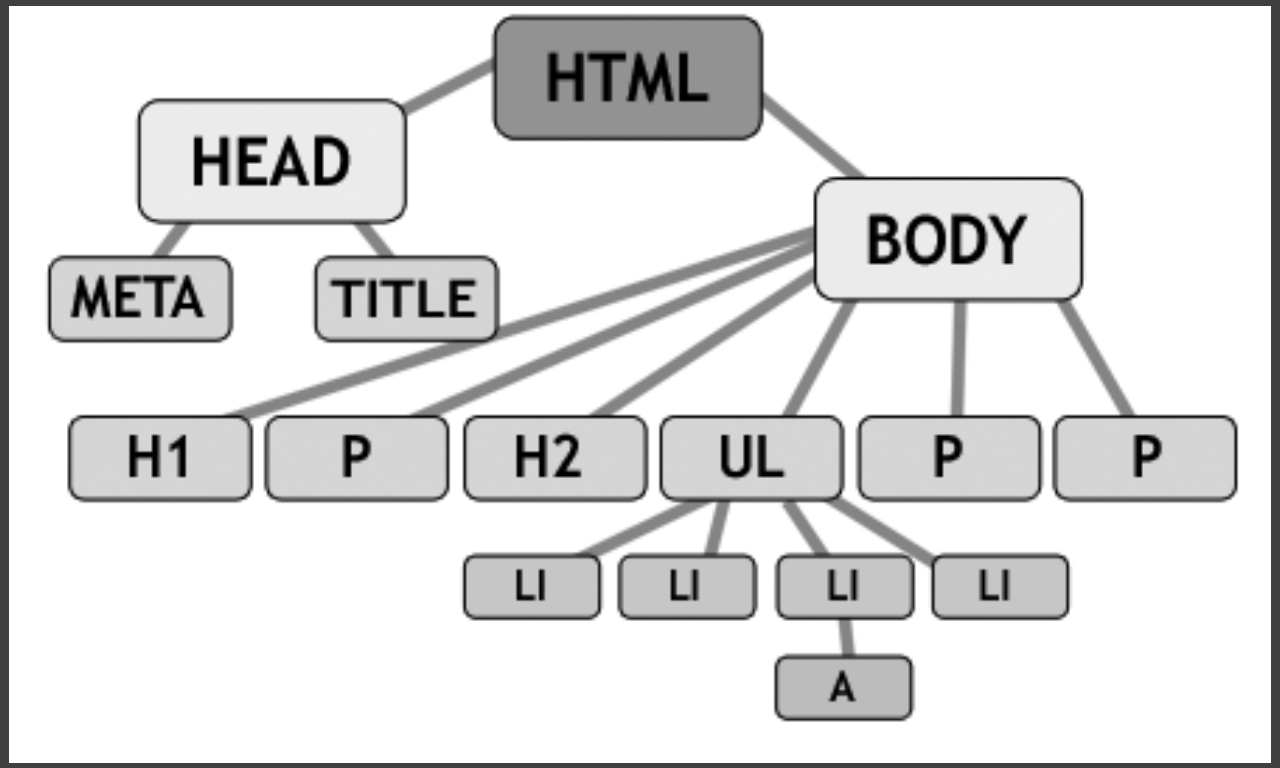
- When dealing with numerical input from the user, explicitly declare the data as a number
  - Number(): tries to convert the value of the variable inside the parentheses into a number
  - parseFloat(): tries to convert the value of a floating point
    - If the value can't be converted, it issues NaN
  - parseInt() Converts the value to an integer by removing any fractional part without rounding up or down

HTML & JavaScript

ACCESSING THE DOCUMENT OBJECT MODEL (DOM); NAVIGATING USING NODES; UNDERSTANDING CHILDREN, SIBLINGS AND PARENTS; CHANGING ATTRIBUTES; MANIPULATING ELEMENTS

Anatomy of an HTML Document

- It is very important to recognize HTML for what it is:
- HTML is structured content and not a visual construct like an image with elements placed at different coordinates.



Accessing the DOM

- You can access a web document via the DOM
  - Already done using the `document.write()` method
- The document object is what we want to stop using
  - `document.write()` only adds a string to the document
  - Does not a manipulate the nodes and attributes in the document
  - Cannot place the JavaScript out into a separate file
  - `document.write()` only works if it is in the HTML

Accessing the DOM

- User agents read a document as a set of nodes and attributes
- The DOM provides tools to grab these nodes and attributes via two methods
  - document.getElementsByTagName('*tag_name*')
    - Returns a list of all the elements with the name x
  - document.getElementById('*id*')
    - Returns the element with the ID as an object

Accessing the DOM

- Wait!
  - DOM access methods do not work until the document is fully loaded and rendered
  - Call access functions when the window has finished loading
  - Document has finished loading when the `onload` event of the window object is fired

Accessing the DOM

- You can also access elements like you access an array

```
var firstpara = document.getElementsByTagName('p')[0];
```

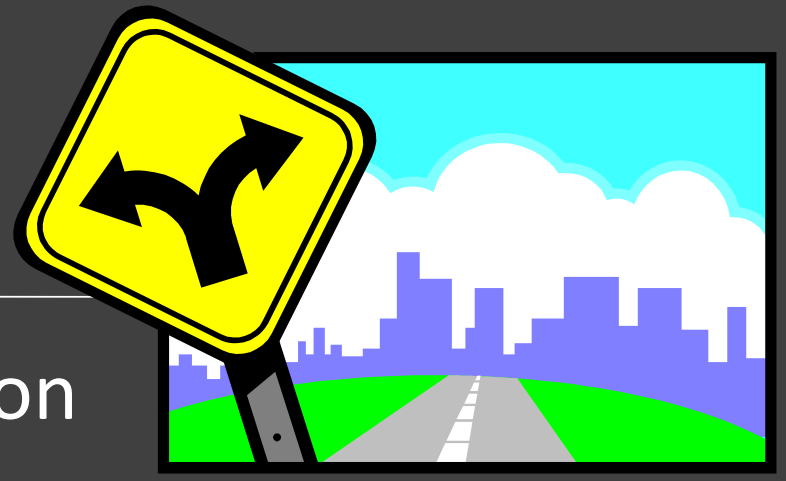
- getElementsByTagName() method can be combined
  - To reach the first hyperlink inside the third list item

```
var targetLink= \ document.getElementsByTagName('li')[2]. \
getElementsByTagName ('a')[0];
```

  - To reach the last element, use the length property of the array

```
var lastListElement = \
listElements[listElements.length - 1];
```

Accessing the DOM



- You can mix access methods to cut down on the elements to loop through
  - Every element or a specific element in the document can be reached
- These access methods can be cumbersome
  - Also you have to know the exact HTML to access and change
- There are other ways to access, navigate and modify the document

Children, Parents, Siblings

- There is a more generic way to traverse the HTML document
- The concept of children, parents, and siblings is available
  - These relationships describe where the current element is in the tree and whether it contains other elements or not

Children, Parents, Siblings

- Simple HTML example focusing on the document body

```
<body>
  <h1>Heading</h1>
  <p>Paragraph</p>
  <h2>Subheading</h2>
  <ul id="eventsList">
    <li>List 1</li>
    <li>List 2</li>
    <li><a href="http://www.foo.com">Linked List Item</a></li>
    <li>List 4</li>
  </ul>
  <p>Paragraph</p>
  <p>Paragraph</p>
</body>
```

Children, Parents, Siblings

- All the indented elements are children of BODY
- H1, H2, UL, and P are siblings
- LI elements are children of the UL element—and siblings to another
- The link is a child of the third LI element
- Text inside the paragraphs, headings, list elements, and links also consists of nodes

Children, Parents, Siblings

- Every node in the document has several valuable properties
- **nodeType**: describes what the node is:
  - Element, attribute, a comment, text, and several more types (12 in all)

Element type	NodeType
Element	1
Attribute	2
Text	3
Comment	8
Document	9

Children, Parents, Siblings

- **nodeName:** the name of the element or #text if it is a text node
  - Can be either upper- or lower case depending on the user agent
    - Best to convert to lowercase before testing for a name

```
if(obj.nodeName.toLowerCase()=='li'){}
```

  - For element nodes, you can use the tagName property.

Children, Parents, Siblings

- **nodeValue**: the value of the node
  - null if it is an element
  - The text content if it is a text node
- With text nodes, nodeValue can be read and set
  - This allows you to alter the text content of the element

```
document.getElementsByTagName('p')[0].\
firstChild.nodeValue='Hello World';
```

From the Parents to the Children

- The firstChild property shown in the previous slide is a shortcut
  - Every element can have any number of children listed in a property called **`childNodes`**
- **`childNodes`** is a list of all the first-level child nodes of the element
  - It does not cascade down into deeper levels
  - Can access a child element of the current element via the array counter

From the Parents to the Children

- Easier methods of accessing children
  - `anElement.firstChild` and `anElement.lastChild`
  - Easier versions of anElement.childNodes[0] and anElement.childNodes [anElement.childNodes.length-1]
- Can check if an element has any children
  - Uses the method hasChildNodes() which returns a Boolean value
- Be aware of browser-specific implementations
  - Older MSIE versions do not return line-breaks
  - Other browsers return line breaks as #text nodes

From the Children to the Parents

- You can navigate from child elements back to their parents via the **parentNode** property
- Example: Looping using parent nodes to find a class name called ‘dynamic’

```
var myLinkItem = document.getElementById('linkedItem');
var parentElm = myLinkItem.parentNode;
while(!parentElm.className != 'dynamic' && parentElm != 'document.body')
{
  parentElm=parentElm.parentNode;
}
alert(parentElm);
```

Among Siblings

- Siblings are elements on the same level
  - Can reach a different child node on the same level via the previousSibling and nextSibling properties of a node
  - Note: because of the difference of browser implementations, the next and previous siblings **are not the LI elements** on modern browsers, but text nodes with the line break as content
- If the current object is the last child of the parent element, nextSibling will be undefined
  - Will cause an error if you don't test for it properly
  - There are no shortcut properties for the first and last siblings

```
<!DOCTYPE html>
<html>
<head>
   <title>JS Example</title>
</head>
<body>
  <h1>The ultimate image directory</h1>
  <a href="search.html">Search Directory</a>
  <div id="nav">
     <img alt="the image" src="oldimage.gif" title="old title">
  </div>
</body>
</html>
```

Changing Attributes of Elements

- Once you have the element you want to change, you can read/change its attributes in two ways:
  - Older method: using object properties to get/set attributes

```
var firstLink=document.getElementsByTagName('a')[0];
if(firstLink.href=='search.html')
{
  firstLink.href='http://www.google.com';
}
var mainImage = \
    document.getElementById('nav').getElementsByTagName('img')[0];
mainImage.src='dynamiclogo.gif';
mainImage.alt='Generico Corporation - We do generic stuff!';
mainImage.title='Go back to Home';
```

Changing Attributes of Elements

- DOM specifications provide two methods to read and set attributes
  - The getAttribute() method: has one parameter - the attribute name
  - The setAttribute() method: takes two parameters - the attribute name and the new value

Changing Attributes of Elements

```
var firstLink=document.getElementsByTagName('a')[0];
if(firstLink.getAttribute('href') =='search.html')
{
  firstLink.setAttribute('href') ='http://www.google.com';
}
var mainImage= \
        document.getElementById('nav').getElementsByTagName('img')[0];
mainImage.setAttribute('src') ='dynamiclogo.gif';
mainImage.setAttribute('alt') = \
       'Generico Corporation - We do generic stuff';
mainImage.setAttribute('title') ='Go back to Home';
```

Changing Attributes of Elements

- All the attributes defined in the HTML specifications are available and can be accessed.
- Some are read-only for security reasons, but most can be set and read.
- Storing a value in an attribute of an element can save a lot of testing and looping

Changing Attributes of Elements

- Beware of attributes that have the same name as JavaScript commands—for example, `for`.
  - Trying to set element.for='something‘ will give an error.
  - Workarounds exist in such cases
  - Example: for (an attribute of the label element) the property name is htmlFor.
  - The class attribute is a JavaScript reserved word so `className` is used instead

Manipulating Elements

- DOM also provides methods for changing the structure of the document
  - Not limited to changing existing elements
  - Can create new ones and replace or remove old ones as well

Manipulating Elements

- Methods for element manipulation
  - document.createElement('element'):
    - Creates a new element node with the tag name element.
  - document.createTextNode('string'):
    - Creates a new text node with the node value of string.
  - node.appendChild(newNode):
    - Adds newNode as a new child node to node, following any existing children of node.

Manipulating Elements

- Methods for Element Manipulation
  - newNode=node.cloneNode(bool):
    - Creates newNode as a copy (clone) of node. If bool is true, the clone includes clones of all the child nodes and attributes of the original.
  - node.insertBefore(newNode,oldNode):
    - Inserts newNode as a new child node of node before oldNode.
  - node.removeChild(oldNode):
    - Remove the child oldNode from node.

Using InnerHTML

- Microsoft implemented the nonstandard property
- It is supported by most browsers
- What it allows is the definition of a string containing HTML and assign it to an object.
- The user agent then does the rest
  - Node generation and adding of the child nodes.

Using InnerHTML

- In general innerHTML is faster than normal DOM methods
  - The HTML parser is always faster than the DOM engine
  - For complicated changes, use innerHTML
  - For simple changes it does not really matter which method you use
    - innerHTML remains theoretically faster

DOM Cheat Sheets

Reaching Elements in a Document
document.getElementById('id'): Retrieves the element with the given id as an object
document.getElementsByTagName('tagname'): Retrieves all elements with the tag name tagname and stores them in an array-like list

Navigating Between Nodes
node.previousSibling: Retrieves the previous sibling node and stores it as an object.
node.nextSibling: Retrieves the next sibling node and stores it as an object.
node.childNodes: Retrieves all child nodes of the object and stores them in an list. There are shortcuts for the first and last child node, named node.firstChild and node.lastChild.
node.parentNode: Retrieves the node containing the current node.

DOM Cheat Sheets

Reading Element Attributes, Node Values, and Other Node Data
node.getAttribute('attribute'): Retrieves the value of the attribute with the name attribute
node.setAttribute('attribute', 'value'): Sets the value of the attribute with the name attribute to value
node.nodeType: Reads the type of the node (1 = element, 3 = text node)
node.nodeName: Reads the name of the node (either element name or #textNode)
node.nodeValue: Reads or sets the value of the node (the text content in the case of text nodes)

DOM Cheat Sheets

Creating New Nodes
document.createElement(element): Creates a new element node with the name element. You provide the element name as a string.
document.createTextNode(string): Creates a new text node with the node value of string.
newNode =**node.cloneNode(bool)**: Creates newNode as clone of node. If bool is true, the clone includes clones of all the child nodes of the original.
node.appendChild(newNode): Adds newNode as a new (last) child node to node.
node.insertBefore(newNode,oldNode): Inserts newNode as a new child node of node before oldNode.
node.removeChild(oldNode): Removes the child oldNode from node.
node.replaceChild(newNode, oldNode): Replaces the child node oldNode of node with newNode.
element.innerHTML: Reads/writes the HTML content of the given element as a string— including all child nodes with their attributes and text content

JavaScript Event Handling

CREATING INTERACTIVE WEB-BASED APPLICATIONS USING BASIC EVENT HANDLING

The Event-Driven Model

- JavaScript programs use an event-driven programming model
- An event
  - A notification generated by the web browser in response to a user or browser action
  - Different occurrences generate different types of events
  - Moving a mouse over a hyperlink, is a different type of event than clicking the mouse on the hyperlink
  - Same occurrence may generate different types of events based on context, example clicking a Submit vs. a Reset button

Event-Driven Models

- Original Flavour
  - The simple event-handling scheme commonly seen in simple JavaScript examples
  - Not thoroughly documented
  - Somewhat codified by the HTML 4
  - Informally considered part of the DOM Level 0 API.
  - Possess limited features
  - Supported by all JavaScript-enabled web browsers which makes it portable.

Event-Driven Models

- New Improved
  - Standardized by DOM Level 2
  - Supported by all modern browsers except Internet Explorer
  - Powerful and full-featured

Event-Driven Models

- The Renegade
  - Originated in IE 4 and extended in IE 5
  - Has some, but not all, of the advanced features of the standard event model
  - Microsoft participated in the creation of the DOM Level 2 event model
  - Stayed with their proprietary model

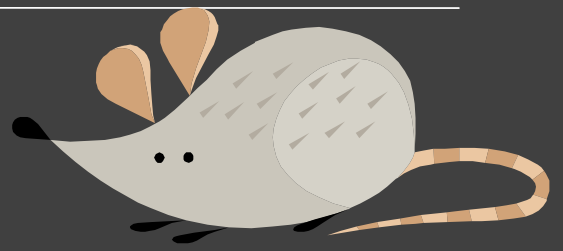
Event Types

- Event Types in the Original Model
  - An event is an abstraction internal to the web browser
  - JavaScript code cannot manipulate an event directly
  - An ***event type*** in the original event model, really means the ***name of the event handler*** that is invoked in response to the event
  - Dealing with different event types required using the attributes of HTML elements (and the corresponding properties of the associated JavaScript objects) to specify the event handling code
  - Hence onmouseover: user moves the mouse over a link; onclick: user clicks a button etc.

Categories of Event Types

- Device-Dependent Events
  - Tied specifically to the mouse or to the keyboard
  - Events with the word "mouse" or "key" in them are device-dependent events
  - May want to use these events in pairs so that you provide handlers for both a mouse gesture and a keyboard alternative

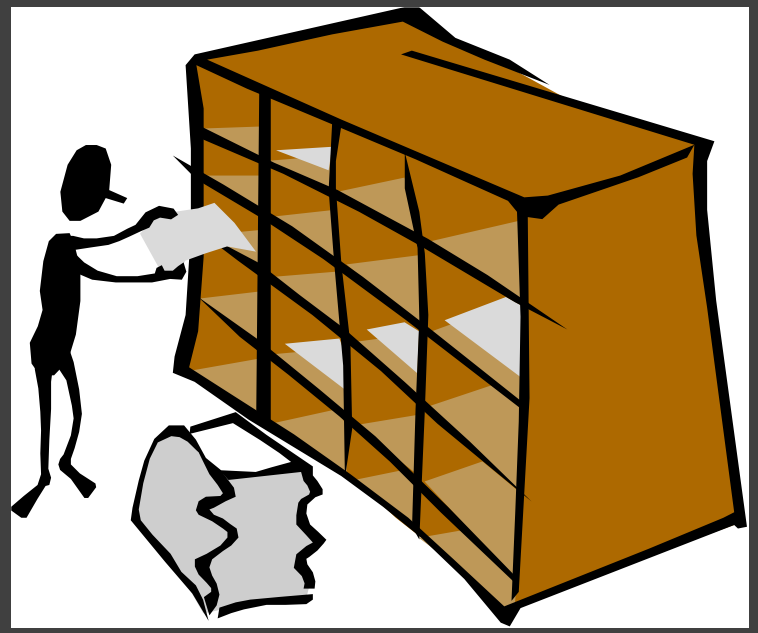
Categories of Event Types



- Device-Independent Events
  - Can be triggered in more than one way
  - Semantic events, e.g. onsubmit and onchange, are almost always device-independent events
  - onclick can be considered a device-independent event since it's not mouse or keyboard dependent

Categories of Event Types

- Raw or Input Events
  - These are the events that are generated when the user moves or clicks the mouse or presses a key on the keyboard
  - Low-level events
  - Describe a user's gesture and have no other meaning



Categories of Event Types

- Semantic Events
  - Higher-level events
  - Have a more complex meaning
  - Typically occur only in specific contexts, e.g. form submission
  - Occurs as a side effect of one or more lower-level events
    - e.g. onsubmit occurs after onmousedown, onmouseup, onclick

Event Handlers and HTML Attributes

Handler	Triggered when	Supported by
onclick	Mouse press and release; follows mouseup event. Return false to cancel default action (i.e., follow link, reset, submit).	Most elements
onblur	Element loses input focus.	<button>, <input>, <label>, <select>, <textarea>, <body>
onchange	Selection in a <select> element or other form element loses focus, and its value has changed since it gained focus.	<input>, <select>, <textarea>
onkeydown	Key pressed down. Return false to cancel.	Form elements and <body>
onkeypress	Key pressed; follows keydown. Return false to cancel.	Form elements and <body>
onload	Document load complete.	<body>, <frameset>, <img>
onmousedown	Mouse button pressed.	Most elements
onmousemove	Mouse moved.	Most elements
onmouseout	Mouse moves off element.	Most elements
onmouseover	Mouse moves over element.	Most elements
onmouseup	Mouse button released.	Most elements

Event Handlers

- An event handler
  - A JavaScript function or snippet of code that performs actions in response to an event that occurred on the element of interest
- JavaScript Event handlers
  - Typically written as strings of JavaScript code that are used as the values of certain HTML attributes, such as onclick
  - This is key to the original event model
  - But additional details need to be understood

Event Handlers as Attributes

- Examples
  - To execute JavaScript code when the user clicks a button, specify that code as the value of the **onclick** attribute of the <input> (or <button>) tag:

```
<input type="button"
       value="Press Me" onclick="alert('thanks');">
```

  - Multiple statements are semi-colon separated

```
<input type="button"
       value="Click Here" onclick="if (window.numclicks) numclicks++; else
       numclicks=1; this.value='Click # ' + numclicks;">
```

  - With multiple statements it's best to use functions instead

Event Handlers as Attributes

- Remember
  - **HTML** is **case-insensitive**
  - Event-handler attributes can be capitalized any way you choose
  - A common convention is to use mixed-case capitalization, with the initial "on" prefix in lowercase: onClick, onLoad, onMouseOut, etc.

BUT

- Use lowercase for compatibility with **XHTML**, which is **case-sensitive**
  - onclick, onload, onmouseout, etc

Event Handlers as Properties

- Recall
  - Each HTML element in a document has a corresponding DOM element in the document tree
  - This includes attributes of the HTML element
  - Remember the getAttribute() method?

Event Handlers as Properties

- JavaScript can access event handlers as properties, but there are rules
  - The JavaScript property must be all lowercase since JavaScript is case-sensitive
  - The value of an HTML event handler attribute is a **string** of JavaScript code, but event-handler properties are **functions**

Event Handlers as Properties

- Example

```
<form name="f1">
  <input name="b1" type="button" value="Press Me">
</form>

document.f1.b1.onclick=function( ) { alert('Thanks!'); };

function plead( ) {
  document.f1.b1.value += ", please!";
}
document.f1.b1.onmouseover = plead;
```

- Note: no parentheses after the name of the function when defining an event handler as a property

Event Handlers as Properties

- Advantages
  - Reduces the mixing of HTML and JavaScript
  - Promotes modularity and cleaner, more maintainable code
  - Makes event handler functions more
  - JavaScript properties can be changed at any time by the JavaScript code even if they are attributes, but more elegant/natural if they are defined as properties

Event Handlers as Properties

- Disadvantages
  - Separates the handler from the element to which it belongs
  - Since you can interact with a document element before the document is fully loaded the event handler for the element may not yet be available

Invoking Event Handlers

- Event handler functions can be invoked directly

```
document.myform.onsubmit( );
```

- Note:
  - Invoking an event handler is not a way to simulate what happens when the event actually occurs
    - Invoking **`onclick`** on an anchor element does not make the browser follow the link and load a new document
  - Explicitly invoke an event-handler if you want to augment an event handler that may already exist in the HTML code

Return Values for Event Handlers

- Return values are used to indicate the state of an event handler
  - This can be done for event handlers defined as HTML attributes or JavaScript properties
  - Example

```
<form action=“addMember.php"
   onsubmit="if (this.elements[0].value.length == 0) return false;">
   <input type="text">
</form>
```

- Return `false` to prevent the browser from performing its default action in response to an event

`this` Keyword

- `this`? What's that?
  - Defining an event handler means that you are assigning a function to a property of a document element, i.e. a new document method.
  - An event handler is invoked as a method of the element on which the event occurred
  - The `this` keyword refers to the target element and the element **only**

```
// this refers to the button object
button.onclick= o.mymethod;

// this refers to the o object
button.onclick = function( ) { o.mymethod( ); }
```

Event Handler Scope

- JavaScript functions are lexically scoped
  - They run in the scope in which they were defined, not in the scope from which they are called
- Event Handler functions are different
  - Defining an event handler by setting the value of an HTML attribute to a string of JavaScript code, is implicitly defining a JavaScript function
  - The scope of such an event-handler function is different from the scope of other normally defined global JavaScript functions
  - Event handlers defined as attributes operate in a different scope than functions

Event Handler Scope

- Head of the scope chain is the call object
  - The code that implements the event handler
  - Arguments passed to the event handler are defined here
- Next object in the scope chain is the object that triggered the event handler
- Scope proceeds up the containment hierarchy
- Final object in scope is the window object
  - This is because it's always in the client-side JavaScript

Event Handler Scope

```
<form>
  <!-- "this" refers to the target element of the event -->
  <!-- So we can refer to a sibling element in the form like this -->
  <input id="b1" type="button" value="Button 1"
         onclick="alert(form.b2.value);">
  <!-- The target element is also in the scope chain, so we can omit "this" -->
  <input id="b2" type="button" value="Button 2"
         onclick="alert(form.b1.value);">
  <!-- And the <form> is in the scope chain, so we can omit "form". -->
  <input id="b3" type="button" value="Button 3"
         onclick="alert(b4.value);">
  <!-- The Document object is on the scope chain, so we can use its methods -->
  <!-- without prefixing them with "document". This is bad style, though. -->
  <input id="b4" type="button" value="Button 4"
         onclick="alert(getElementById('b3').value);">
</form>
```

The Moral of Event Handler Scope

- So what have we learned about event handler scope?
  - Having the target object in the scope chain of an event handler is a useful shortcut
  - Having an extended scope chain that includes other document elements can be a nuisance,
    - e.g. the open() method could refer to window.open() or document.open()
    - Adding a property called window to a Form object would make it impossible to call window.open later in an event handler

The Moral of Event Handler Scope

- Be careful when defining event handlers as HTML attributes
- Keep event handlers very simple
  - Just call a global function defined elsewhere and perhaps return the result
- Simple event handlers are still executed using an unusual scope chain
  - Keep code short to minimize the likelihood that the long scope chain will trip you up

The Moral of Event Handler Scope

- Assume that the scope chain contains only the target element and the global Window object
  - e.g. do not assume that properties like action are available, use form.action instead
- Event-handler scope applies only to event handlers defined as HTML attributes
  - No special scope chain involved with event handler function assigned event-handler property