JavaScript

Handling Events

Introduction of the Event

- JavaScript reacts to events. We have been talking about events since day 1.
- Events make the pages interactive and dynamic
- JavaScript events are asynchronous, meaning that they can happen at any time.
- Event is initiated by a user visiting a Web page; for example,
  - user submits a form
  - moves the mouse over a link or an image…
- Events are blur, click, change, keystroke ...

Event handling

- When an event occurs, your browser has some default actions it takes; for example
  - if user clicks on a link, it opens up the location in the *href*.
- JavaScript can also respond to a user's action and execute code in response.
- JavaScript's response to one of these user initiated events is called **event handling.**

The Inline Model for Handling Events

- Oldest and easiest way to handle events.
- The event handlers are attributes of a HTML tag
- For example the
  - ***onClick*** *handler will handle the click event*
  - ***onSubmit*** *handler* will handle the submit event
- Disadvantage is that JavaScript is not separated from the HTML markup.
- JavaScript's response to one of these user initiated events is called **event handling.**

Inline Example

- JavaScript event handlers are **not enclosed between <*script*></*script*> *tags*.**
- ***Event handlers*** are attributes of HTML tags (specified in the HTML 4 specification).

```
<form>
<input type="button” value="Wake me” onClick="wakeUpCall()"  />
</form>
```

Registering Event

- If the event is associated with a form tag, then it will be an attribute of the *<form> tag*
- *If associated* with a link, it will be an attribute of the *<a href> tag* …
- *Once you have decided* what event you want to handle, you normally assign a function to the event handler. This is called registering the event.
- To register an event, a string representing a command is assigned to the event handler.
- The command, usually a JavaScript function, will be executed when the event is triggered by the user.
- Whereas a property or method might be associated with a single object, events are usually associated with more than one object.
- The *onClick event handler,* for example, may be associated with a form's input tag, but it could also be associated with a link tag, or an image map area, or a simple button. (event bubbling and capturing)
- There is an order in which events are handled.

Example

```
<html><head><title>Wake up call</title>

<script type="text/javascript">
function wakeUpCall()
{
setTimeout('alert("Time to get up!")', 5000);
}
</script>

</head><body>

<form>
<input type="button“ value="Wake me“ onClick="wakeUpCall()“ >
</form>

</body></html>
```

Events

Event Handler	**What It Affects**	**When It Happens**
onAbort	*Images*	*When image loading has been interrupted.*
onBlur	*Windows, frames,*all form objects	When focus moves out of this object except *hidden; e.g., when the cursor leaves a textbox.*
onChange	*Input, select, and text areas*	*When a user changes the value of an element* and it loses the input focus. Used for form Validation.
onClick	*Links, buttons, form objects,* image map areas	When a user clicks on an object. Return false to cancel default action.
onDblClick	*Links, buttons, form objects*	*When a user double-clicks on an object.*

Events cont.

Event Handler	**What It Affects**	**When It Happens**
onDragDrop	*Windows*	*When a user drops an object,* *such as a file,* onto the browser window.
onError	*Script*	*When an error in the script occurs;* e.g., *a* syntax error.
onFocus	*Windows, frames,* all form objects	When a mouse is clicked or moved in a window or frame and it gets focus; except *hidden.*
onKeyDown	*Documents, images, links, forms*	*When a key is pressed.*
onKeyPress	*Documents, images, links, forms*	*When a key is pressed and released.*
onKeyUp	*Documents, images, links, forms*	*When a key is released.*
onLoad Body	*framesets, images*	*After the document or image is loaded.*

Events cont.

Event Handler	**What It Affects**	**When It Happens**
onMouseOut	Links (and images within links)	When the mouse moves away from a link.
onMouseOver	Links (and images within links)	When the mouse moves over a link. Return true to prevent link from showing in the status Bar.
onMove	Windows	When the browser window is moved.
onReset	Forms reset button	When the form's Reset button is clicked. Return false to stop reset.
onResize	Windows	When the browser window changes size.
onSelect	Form elements	When a form element is selected.
onSubmit	Forms	When you want to send a form to the server. Return false to stop submission to the server.
onUnload	Body, framesets	After the document or frameset is closed or reset.

Event handler value

- The event handler value can be
  - a built-in method such as *alert() or confirm(),*
  - *a user-defined function,*
  - *or a* string of JavaScript statements.

- Although the event handler is an attribute of an HTML tag, if a user-defined function is assigned to the event handler, then that function must be defined either in a JavaScript program or as direct script statements (separated by semicolons).

Quotes

- Be careful with quotes!
- The handling function must be enclosed within either double or single quotes.
- If you have double quotes within the function, surround the whole thing in single quotes, and if you have single quotes within the function, either escape the single quote with a backslash, or surround the whole thing in double quotes.
- *built-in method*
  - *onClick*=**"*window.open('myhome.html', 'newWin')*"**
- *user-defined function*
  - *onUnLoad*=**"*timeOver();*"**
- *group of statements*
  - *onChange*=**"*if (!checkVal(this.value, 1, 10))*{ this.focus(); this.select();}"**

Return Values

- Sometimes the event handler's return value is necessary if a certain action is to proceed.
- The browser's default actions can be suppressed by returning a *false value, or a form's submission* can be completed by sending back a *true value*.
- *For example, if the onSubmit handler* returns true value back from a function, then a form may be submitted to the server, and if false not, the form will be stopped to send.

Example return value

```
<html><head><title>An HTML Form and the onSubmit Event Handler</title>
<script type="text/javascript">
function checkForm(yourinfo){
    if(yourinfo.namestring.value == "" || yourinfo.namestring.value == null){
            alert("Please type in your name");
            return ( false );}
    else{ return(true); }}
</script>
</head><body>
<form name="info" action="process.aspx"  method="post"
                                      onSubmit="return checkForm(document.info)">
Type your name here:
<input type="text" name="namestring" size="50" />
<input type="submit" value="Submit" />
<input type="reset" value="Clear" />
</form></body></html>
```

Handling a Window or Frame Event

Event Handler	***When It Is Triggered***
onBlur	*When the mouse moves away from the window or frame and it loses focus.*
onFocus	*When the mouse is clicked or moved in a window or frame and it gets focus.*
onLoad	*When a document or image has finished loading.*
onMove	*When a window is moved.*
onUnLoad	*When a page is exited or reset.*

The *onLoad and onUnLoad Events*

```
<html><head><title>load and unload Events</title>
<script type="text/javascript">

var sec=0;
function onDocLoad(){
var newdate= new Date();                    var hour=newdate.getHours();
 var minutes=newdate.getMinutes();   var seconds=newdate.getSeconds();
var timestr=hour+":"+minutes+":"+seconds;
window.setInterval("trackTime()", 1000);
alert("Document has finished loading\n"+ "The time: "+timestr);
}

function trackTime(){ sec++; }

function howLong(){ alert("You have been browsing here for "+ sec+" seconds"); }

</script></head><body  onLoad="onDocLoad(); " onUnLoad="howLong();">
When leave or reload this page, an alert dialog box will appear.</body></html>
```

Handling Mouse Events

Event Handler	***When It Is Triggered***
onClick	*When the mouse is clicked on a link and on form objects like button,*submit.
onDblClick	*When the mouse is double-clicked on a link, document, form object,* image.
onMouseDown	*When the mouse is pressed on a link, document.*
onMouseMove	*When the mouse is moved when it is over a link, form object, or most* elements.
onMouseOut	*When a mouse is moved out of a link, imagemap.*
onMouseOver	*When a mouse is moved over or enters a link, imagemap, or most* elements.
onMouseUp	*When the mouse is released from a link, document.*

Usage examples

<body bgColor="CCFF00" **onDblClick**="alertme()";>

<a href="#" **onMouseOver**="alert(‘MouseOver');" />

<a href="#" **onMouseOut**="alert('MouseOut');" />

<input type="button” value=“ok“
onMouseMove="alert('MouseMoving');" />

Each time a user moves the mouse one pixel, a **mousemove** event occurs. It takes system resources to process all mousemove events.

Use this event carefully!

Handling Link Events

- When the user clicked or moved the mouse over a link, a link event was triggered.

Event Handler	***When It Is Triggered***
onClick	*When the mouse is clicked on a link*
onMouseOut	*When a mouse is moved out of a link*
onMouseOver	*When a mouse is moved over a link*

JavaScript URLs

Discarding the link

- <a href="#“ **onClick='alert(“Sorry!");** return false;’>go to…</a>

- or by using the *JavaScript: protocol followed by the void operator to guarantee that any* return value from the function will be discarded:

- <a href="JavaScript:void(0);" onMouseOver="return changeSeason();"

Handling a Form Event

Object	***Event Handler***
button	*onClick, onBlur, onFocus*
checkbox	*onClick, onBlur, onFocus*
FileUpLoad	*onClick, onBlur, onFocus*
hidden	*none*
password	*onBlur, onFocus, onSelect*
radio	*onClick, onBlur, onFocus*
reset	*onReset*
select	*onFocus, onBlur, onChange*
submit	*onSubmit*
text	*onClick, onBlur, onFocus, onChange*
textarea	*onClick, onBlur, onFocus, onChange*

Buttons

- One of the most common GUI form elements is the button.
- The button object has no default action and is normally used to trigger an event such as the ***onClick*** *event.*
- *HTML* 4 allows you to create a *<button> tag without the <input> tag.*
- *There are several buttons* associated with a form; the buttons are called:
  - *submit*
  - *reset*
  - *button*

Form Event Handlers

Event Handler	***When It Is Triggered***
onBlur	*When a form's select, text, or textarea field loses focus.*
onChange	*When a select, text, or textarea field loses focus and its value has been* changed.
onClick	*When an object on a form is clicked.*
onFocus	*When a field receives input focus by tabbing with the keyboard or clicking* with the mouse in the field.
onReset	*When the user resets the form.*
onSelect	*When a user selects some of the text within a text or textarea field.*
onSubmit	*When a user submits a form.*

onFocus and onBlur Event Handlers

```
<html><head><title>Using the onFocus Event Handler</title>
<script type="text/javascript">
function handler(message){
window.status = message; // Watch the status bar
}
</script></head><body>
<form name="form1">
<p>Type your name: <input type="text” name="namestring”
    onFocus="handler('Don\'t forget to enter your name')">
<br />About you:<br />
<textarea name="comments” onFocus="handler('Did you add comments?')”
    >I was born...
</textarea>
<p><input type="button” value="submit“> <input type="reset”
    value="clear“> </form></body></html>
```

onChange Event Handler

- The *onChange event handler is triggered after the user modifies the value or contents of* an HTML input, select, or text area element in a form, and then releases the mouse.
- This is another event handler that can be useful in checking or validating user input.

```
<html><head><title>onChange Event Handler</title></head>
<body><form> Please enter your grade:

<input type="text"
onChange=“ grade=parseInt(this.value);
alert(grade >0? ‘incorrect':‘correct’)" />m
</form></body></html>
```

The *onError Event*

- The error event fires when a JavaScript error has occurred (window) or when an image cannot be found (image elements).

- **<img src=“hi.gif“ onError="alert(‘error loading!')">**

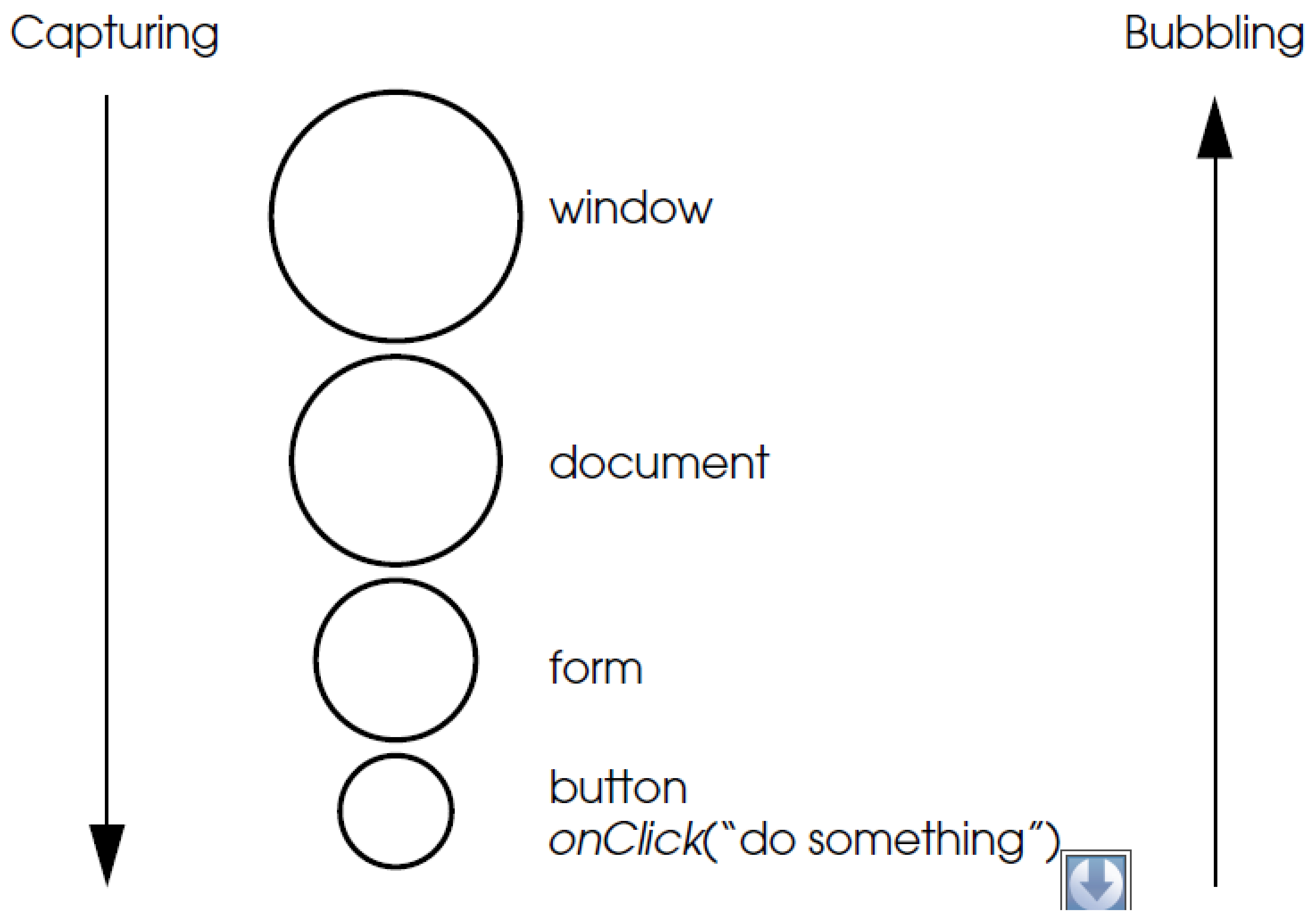
The *event Object*

- Event objects are sent to an event handler with each event that occurs within a document; for example, when the user clicks on the left mouse button, JavaScript registers the event,
  - what key was pressed
  - its coordinates (pixel positions of where it was pressed on the screen),
  - and so on.

Capturing and Bubbling (Trickle Down and Bubble Up)

- The way that the events are handled differs by the browser and is based on how Netscape and Internet Explorer dealt with events back in the 1990s.
- Suppose you have used an *onClick event handler in the button of a form.*
- Netscape said that the event is captured on window level and trickles down the document to the form object until it finally reaches the button, its target:
- When it reaches its target, the button, the event is fired.
- Internet Explorer says the event occurs on the target (i.e., the button) and then sends information about the event back up from the button to the form to the document, then window, like the bubbling up effect of soda water in a glass

Capturing and Bubbling (Trickle Down and Bubble Up)



Event Object Properties

- Many of the examples listed on different Web sites might not work when you try them because they were written for a specific client.
- You might be using Opera or Safari and think, "This doesn't work at all. I give up."
- But if you bring up Internet Explorer, the program seems to work fine.
- To see what does or doesn't apply to your browser go to "Events Compatibility Table" at *http://www.quirksmode.org/dom/events/index.html*.
- *Although the DOM provides a standard* model, not all browsers are compliant.

```
<html><head><title></title>
<script type="text/javascript">
function OnBodyClick(eventObj)
{
alert(eventObj.button)
}
</script>
</head>
<body onClick="OnBodyClick(event)">
…
</body>
</html>
```

Properties of the *event Object: Internet Explorer*

Property	***What It Describes***
altKey, ctrlKey, shiftKey	*Set to true or false to test if Alt, Shift, or Control keys were pressed when* the event occurred.
button	*An integer indicating which mouse button was pressed or released, (0001) 1*=left, (0010)2 = right, (0100) 4 =middle. If multiple buttons are pressed, the value is the sum of both buttons, such as 3 (1+2) for left and right.(Logical OR)
cancelBubble	*Set to true or false to cancel or enable event bubbling. To cancel across* browsers, the *stopPropagation() method is supported.*
clientX and clientY	*The cursor's horizontal and vertical position in pixels, relative to the* upper-left corner Web page in which the event occurred. Also good for the Mozilla/Firefox and W3C event model.
fromElement, toElement	*Used to indicate the elements where a mouse is (mouseout and* mouseover) leaving from or moving into. See *relatedTarget for* W3C/Firefox.

Properties of the *event Object: Internet Explorer*

- *keyCode* *The Unicode key code associated with a keypress event. Use String.fromCharCode(keyCode) to convert keycode code to a string.*
- *offsetX and offsetY* *The cursor's horizontal and vertical position in pixels, relative to the* container in which the event occurred, or if outside the container returns the upper left corner of the document.
- *returnValue* *The return value of the event handler, either true or false. For* W3C/Firefox use *preventDefault() method.*
- *srcElement* *The element from where the event originated, not necessarily the element* where it was assigned due to bubbling. Use *target for Firefox.*
- *srcFilter* *Specifies the filter object that caused an onfilterchange event.*
- *x and y* *The cursor's horizonal and vertical position in pixels, relative to the* document in which the event occurred.
- *reason* *Used to indicate the status of a data transfer for data source objects.*

Properties of the *event Object: Mozilla Firefox*

- ***altKey, ctrlKey, metaKey, shiftKey***

  Set to true or false to test if Alt, Shift, Control, or Meta keys were pressed when the event occurred (Internet Explorer doesn't support metaKey).
- ***pageX and pageY***

  Horizontal and vertical cursor position within a Web page, relative to the document, not supported by Internet Explorer.
- ***bubbles***
  Set to Boolean value indicating whether or not the event bubbles.
- ***button***
  An integer indicating which mouse button was pressed or released, 0 =left, 2 = right, 1 = middle. Slightly different in Internet Explorer, as described earlier.
- ***cancelable***
  A Boolean value indicating whether or not the event can be canceled.
- ***charCode***
  Indicates the Unicode for the key pressed. Use String.fromCharCode(which) to convert code to string.
- ***clientX, clientY***
  Returns the mouse coordinates at the time of the event relative to upperleft corner of the window.

Properties of the *event Object: Mozilla Firefox*

- ***currentTarget***
  The node that this event handler is currently being run on.

- ***eventPhase***
  An integer value indicating which phase of the event flow this event is being processed in. One of CAPTURING_PHASE (1), AT_TARGET (2) or BUBBLING_PHASE (3).

- ***layerX and layerY***
  Returns the horizontal and vertical cursor position within a layer, not standard.

- ***relatedTarget***
  On a mouseover event it indicates the node that the mouse has left. On a mouseout event it indicates the node the mouse has moved onto.

- ***screenX and screenY***
  Returns the coordinates of the mouse relative to the screen when the event fired.

Properties of the *event Object: Mozilla Firefox*

- ***target***
  The node from which the event originated.
- ***timestamp***
  Returns the time (in milliseconds since the epoch) the event was created. Not all events return a timestamp.
- ***type***
  A string indicating the type of event, such as "mouseover", "click", and so on.
- ***which***
  Netscape legacy property indicating the Unicode for the key pressed. Identical to *charCode.*
- ***modifiers***
  The bitmask representing modifier keys such as Alt, Shift, Meta, and so on.
- ***data***
  Array of URLs for dragged and dropped.
- ***height and width***
  Height and width of the window.

Using Event Object Properties

The *srcElement/target and type Properties.*

- The ***srcElement*** *(Internet Explorer)* and the ***target*** *properties (Firefox) return the element that fired the event.*
- The ***srcElement*** also has a *tagName property:* **event.srcElement.tagName** *will return "IMG" if you* click on an image object.
- Read styles, so if the image has a style height of 100px, then ***event.srcElement.style.height*** *will return "100px".*
- The ***type*** *property contains the event name (e.g., click, mouseover, keypress, and so* on).
- This is the same for nonphysical events. If we capture and handle an *onload event,* type will be "load", and so on.

Passing Events to a JavaScript Function

```
<html><head><title>Event Object and Incompatible Browsers</title><script>
function whichEvent(e) {                    // passing e is for Firefox
if(!e)
{ var e=window.event;}                      // Microsoft IE

if(e.target)
    { targ=e.currentTarget; targ=targ.id; }  // Firefox
else if(e.srcElement)
    { targ=e.srcElement.id; }               // Microsoft IE

alert(targ +" has recieved a "+e.type);
}
</script></head>
<body id="main" onload="whichEvent(event);“> <br /><div style="background-color:lightgreen">
<form id="form1" onclick="whichEvent(event);“><br />
<input type="text" id="textbox1” onkeypress="whichEvent(event);" /><br /><br />
<input type="button" value="rollover me" id="button1” onmouseout="whichEvent(event);" />
<br /></form></div></body></html>
```

Mouse Positions

```
<html><head><title>Mouse Coordinates</title>
<script type="text/javascript">
function getCoords(e) {
var x = 0; // x and y positions
var y = 0;
if (!e) var e = window.event;                    // Internet Explorer
if (e.pageX || e.pageY){                // Firefox
x = e.pageX;              y = e.pageY;}
else if (e.clientX || e.clientY) {
   x = e.clientX + document.body.scrollLeft + document.documentElement.scrollLeft;
   y = e.clientY + document.body.scrollTop+ document.documentElement.scrollTop;
}
// x and y contain the mouse position relative to the document
alert(x +", "+ y);
}
</script><head><body>
<div style="background-color:aqua;position:absolute;top:50px“ onMouseover="return
   getCoords(event);">
<h1>Mouse positions are relative to the document, not the &lt;div&gt; container</h1>
</div></body></html>
```

How JavaScript and Style Sheets

How JavaScript views style sheets

- Sometimes a page consists of more that one style sheet.
- When the page is loaded, all external and embedded style sheets are stored in an array in the order in which they are placed within the document, just as are images, forms, and links.
- The array is called *styleSheets, a property of the document object, so document.styleSheets[0] is the first* style sheet, normally the general style sheet for the entire page.
- The next style sheet, maybe one that uses the *@import rule, would be document.styleSheets[1], and if you* embedded another style within the document, that would be *document.styleSheets[2]*.
- Because each of the elements of the *styleSheets array has a disabled property that can be* turned on and off, you can dynamically toggle between styles, allowing a user to select a style suited to his or her own taste.
- It is also possible to access a specific rule within a style sheet by using the W3C *cssRules array or the Microsoft rules array.*
- *Both arrays* work with index numbers. The first rule is *cssRules[0], the second one cssRulse[1], and* so on. Example contains two style sheets, one that is imported and one that is embedded.
- By using JavaScript the user can toggle between two styles. When reverting to the first style, one of the rules is changed; that is, the rule for the *h1 selector is* changed to purple.

Example

```
<html><head><title>Stylesheets</title>
<style type="text/css">
@import url("pstyle.css");
</style><script type="text/javascript">
 function changeStyle(){
document.styleSheets[0].disabled=false; // now visible
document.styleSheets[1].disabled=true;
}
function enableOldStyle(){
document.styleSheets[0].disabled=true;
if(document.styleSheets[1].cssRules){                    // W3C
      document.styleSheets[1].cssRules[1].style.color="purple";
else{                                                    //Microsoft Internet Explorer
      document.styleSheets[1].rules[1].style.color="purple";
      document.styleSheets[1].disabled=false;
      }
document.styleSheets[1].disabled=false;                  // now visible
}
</script><style type="text/css">
p { background-color:darkblue; font-weight:bold ; font-size: 12pt; font-family:arial; color:yellow; }
h1{color:red;}
</style></head><body><form><input type="radio" onclick="JavaScript:changeStyle()">new style<br />
<input type="radio" onclick="JavaScript:enableOldStyle()">old style<br /> </form></body></html>
```

The *style Object*

- The style object contains a set of properties corresponding to the CSS attributes supported by your browser.
- Each HTML object has a *style property used to access the CSS* style attributes assigned to it; for example, an *h1 element might have been defined with* a CSS *font-style, color, and padding.*
- *The style object has properties to reflect each of the* CSS attributes.

The *style Object*

- CSS style attributes contain hyphens (-) in their names
  - *background-color,*
  - *font-size*
- In JavaScript the name would not contain a hyphen
- And multiple words after the first word are usually capitalized.
  - on**M**ouse**O**ver
- Therefore, the CSS naming convention is different with the properties of the *style object.*
- *The hyphen is removed and the* first letter of each word after the hyphen is capitalized.
- For example, the CSS attribute,
  - *background**-c**olor is background**C**olor*
  - *font**-s**ize is font**S**ize*
  - *border**-r**ight**-w**idth is border**R**ight**W**idth.*

style object’s Format

- elementname.style.property="value";
- div2.**style.fontFamily** = "arial";

Example

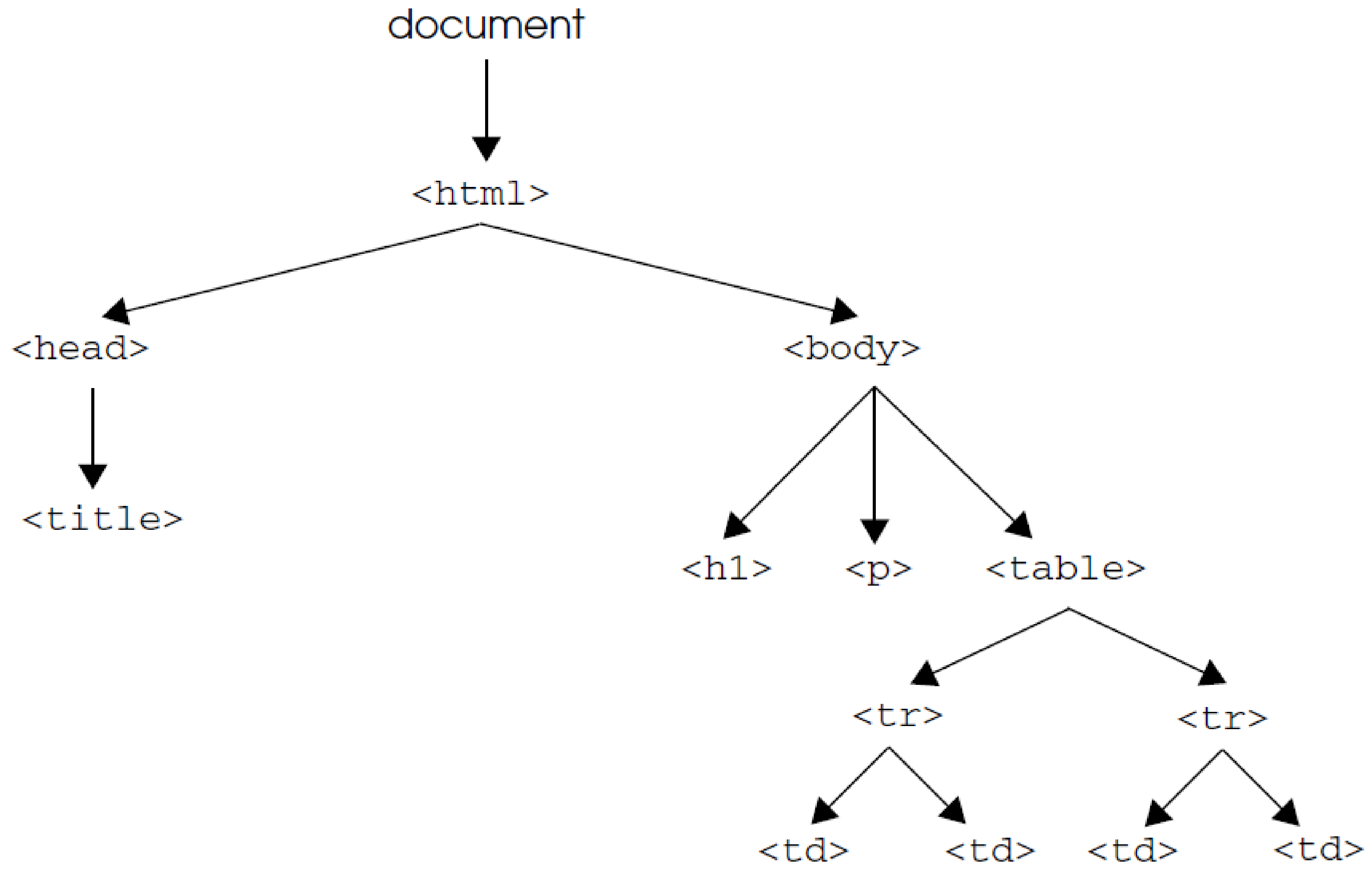
```
<html><head><title>Changing Background Color Dynamically</title>
<script type="text/javascript">

function bodyColor(){
var i = document.form1.body.selectedIndex;
bodycolor = document.form1.body.options[i].value;
document.getElementById("bdy").style.backgroundColor=bodycolor; }

</script>
</head><body id="bdy">
<p>Pick a background color for this page.</p>
<form name="form1“> <b> Color </b>
    <select name="body" onChange="bodyColor();">
            <option value="pink">pink</option>
            <option value="lightblue">blue</option>
    </select>
</form><p>This is a test.</p></body></html>
```

DOM

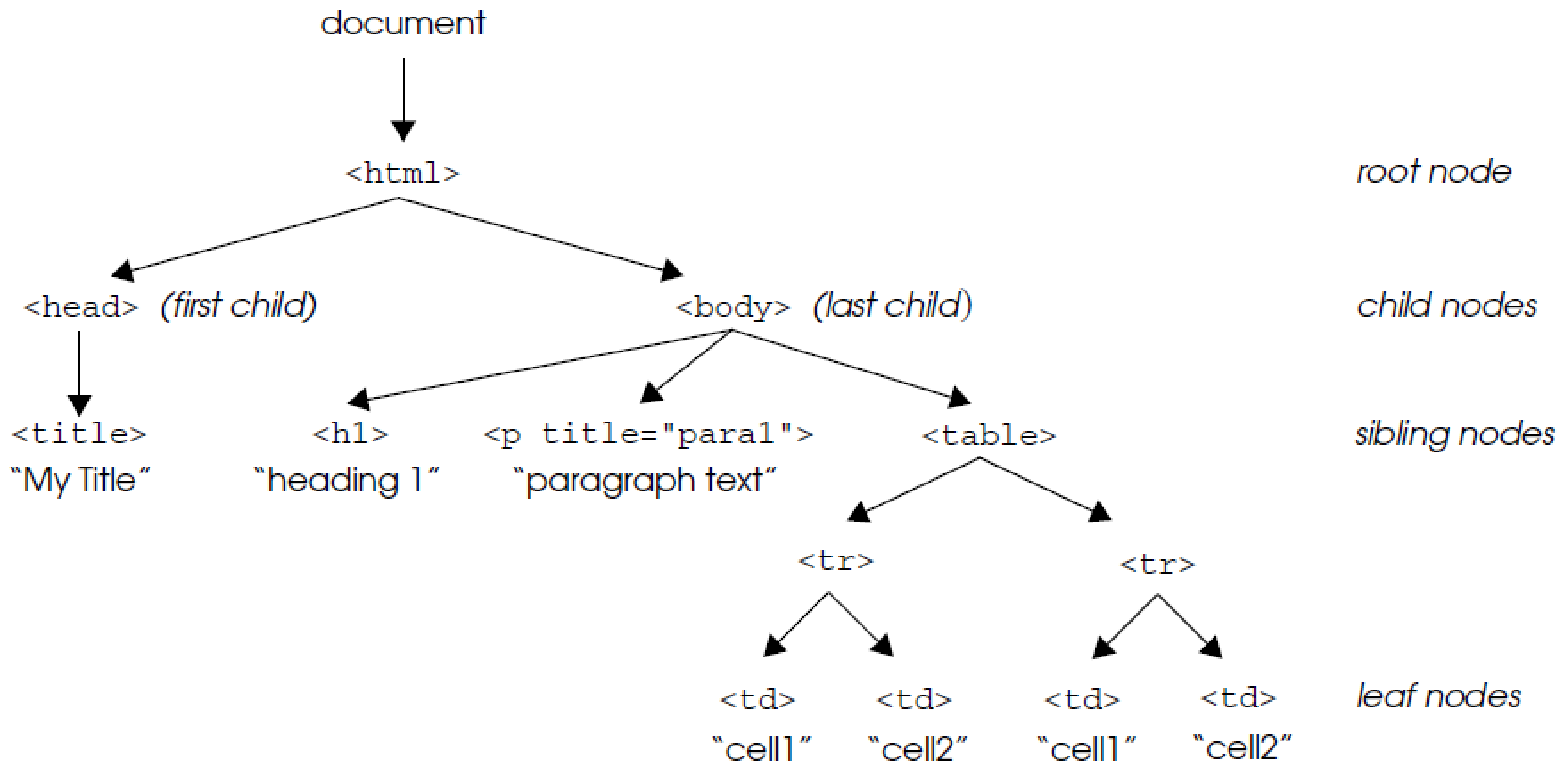
HTML document as a tree.



Some DOM Objects

Object	*Definition*
Node	The primary data type that represents an HTML element.
Document	The root of the document tree.
Element	An element within an HTML document.
Attribute	Attributes of an HTML tag.
Text	The text between markup tags, such as the text between <*h1*> and </*h1*>.

A tree of nodes



Node Properties

Property	***What It Does***
firstChild	*Returns the first child node of an element.*
lastChild	*Returns the last child node of an element.*
nextSibling	*Returns the next child node of an element at the same level as the current child node.*
nodeName	*Returns the name of the node.*
nodeType	*Returns the type of the node as a number: 1 =element, 2 = attribute, 3 = text.*
nodeValue	*Sets the value of the node in plain text.*
ownerDocument	*Returns the root node of the document that contains the node.*
parentNode	*Returns the element that contains the current node.*
previousSibling	*Returns the previous child node of an element at the same level as the current child* node.

```
<p id="para1">This is the paragraph.</p>
```

- **p_element = document.getElementById("para1");**
- **alert(p_element.childNodes[0].nodeValue);**
- **alert(p_element.nodeName)**

Node Methods

- ***appendChild(new node)***

  Appends a new node onto the end of the list of child nodes.

- ***cloneNode(child option)***
  Makes a clone of a node.

- ***hasChildNodes()***

  Returns true if the node has children.

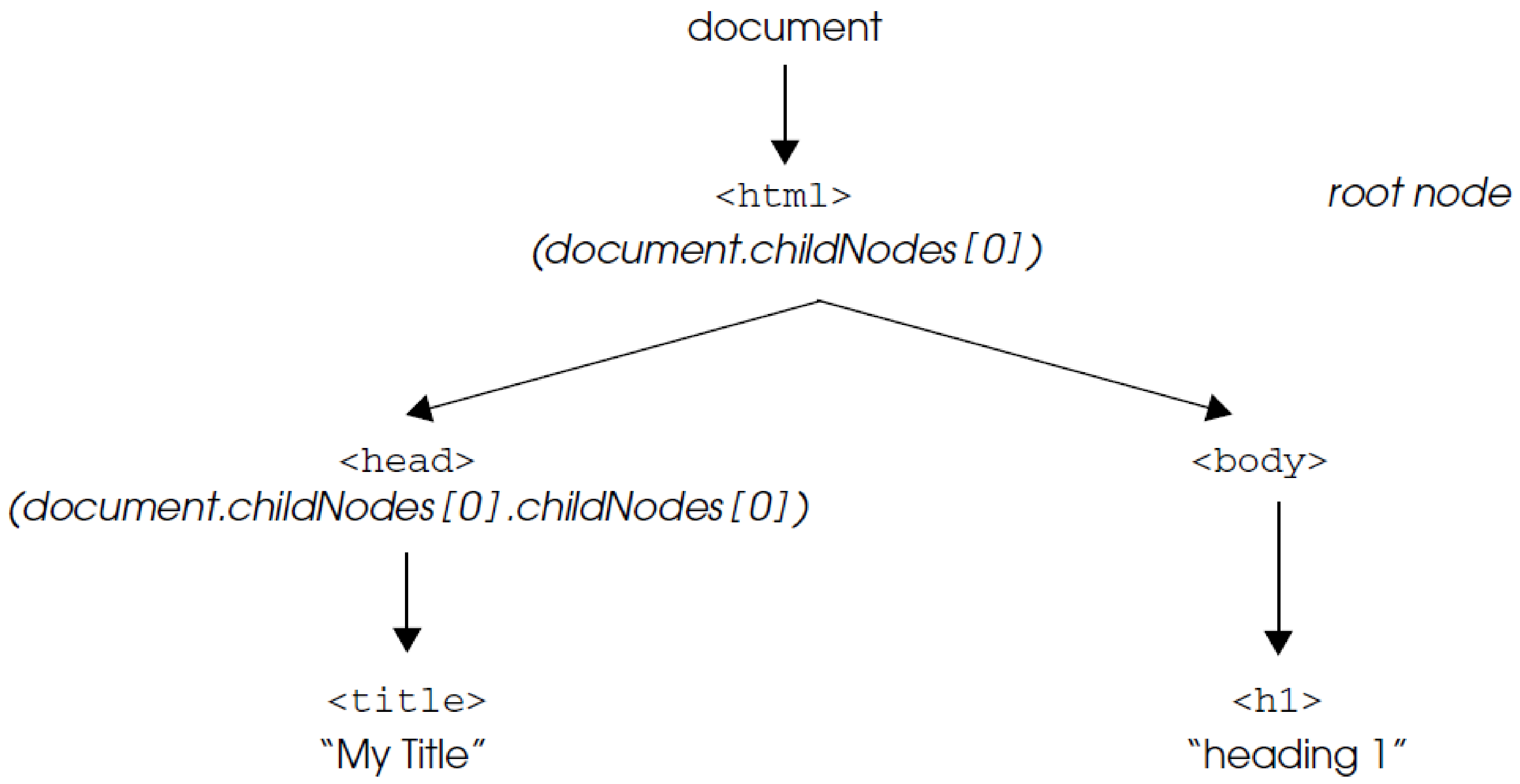
- ***insertBefore(new node, current node)***
  Inserts a new node in the list of children.

- ***removeChild(child node)***
  Removes a child node from a list of children.

- ***replaceChild(new child, old child)***
  Replaces an old child node with a new one.

Parents and Children

document.childNodes[0].childNodes[0].childNodes[0].childNodes[0]



nodeName and nodeType Properties

Node	***nodeName Property***	***nodeType Property***
Element	Name of the element (*h1, p*)	*1*
Attribute	Name of the attribute (*id, href*)	2
Text	#text	3

The Whitespace Bug

- When testing a document's nodes, be careful of the whitespace bug!
- If you break lines, for example, between the <*head*> *and* <*title*> *tags, the whitespace is considered a text* node and your output might not be consistent.

Methods to Shorten the DOM Walk

- In the previous example, walking with the nodes was like walking through a maze in a palace garden. Although the DOM is represented as a hierarchal tree of elements, where each element is represented as a node, walking with the nodes can be daunting, so the W3C provides additional methods and properties to help make the walk easier.

- The DOM provides two methods, ***getElementById()*** *and* ***getElementsByTagName()****, to* directly access the target element you are trying to reach.

```
<html><head><title>Working with Tags</title>
</head><body><h1> First</h1><h1> Second</h1><h1> Third</h1>

<script type="text/javascript">
var heading1=document.getElementsByTagName("h1");
document.write(heading1 + "<br />");

document.write("There are "+heading1.length+ " H1  tags.<br />");

for(i=0; i< heading1.length; i++){
    document.write( heading1[i].nodeName+": <em>"+
                                    heading1[i].childNodes[0].nodeValue+"</em><br
    />");
}
</script></body></html>
```

Modifying the DOM (Appending, Copying,and Removing Nodes)

- ***appendChild(new node)*** *Appends a new node onto the end of the list of child nodes.*
- ***cloneNode(child option)*** *Makes a clone of a node.*
- ***hasChildNodes()*** *Returns true if the node has children.*
- ***getAttribute(attributeName)*** *Returns the value of the attribute of the current node.*
- ***hasAtrributes()*** *Returns a Boolean value if the node has defined attributes.*
- ***hasChildNodes()*** *Returns a Boolean value if the node has defined child nodes.*
- ***insertBefore(new node, current node)*** *Inserts a new node in the list of children.*
- ***removeChild(child node)*** *Removes a child node from a list of children.*
- ***setAttributeNode(attributereference)*** *Sets or creates an attribute for the current node.*
- ***replaceChild(new child, old child)*** *Replaces an old child node with a new one.*

The *innerHTML Property and the Element's Content*

- The easiest way to get or modify the content of an element is by using the ***innerHTML*** property.
- Although the *innerHTML is not a part of the W3C DOM specification, it is supported* by all major browsers.
- The *innerHTML property is useful for inserting or replacing* the content of HTML elements; that is, the code and text between the element's opening and closing tag.

```
<html><head><title>innerHTML</title></head>
<body>
<p id="para1"><strong>Hello</strong></p>
<script type="text/javascript">
ptxt1=document.getElementById("para1").innerHTML;
document.write("<p>"+ptxt1.toUpperCase()+ "</p>");
</script>
</body>
</html>
```

Modifying the Content of an Element

```
<html><head><title>Modify Text</title>
   <script type="text/javascript">
   window.onload=function(){
   var divObj = document.getElementById("divtest");
   divObj.innerHTML=“New text";}
   </script>
   </head>
   <body>
         <div id="divtest“>Original text.</div>
   </body>
</html>
```

Creating New Elements with the DOM

- To create new elements on the fly
  - *createElement() method,*
  - *createTextNode() method for text node.*
- *Once you get a reference to the new element,* you must insert or append it to the document.

- If, for example, you have created a *p element* as follows:

  ```
  var pref = document.body.createElement("p");
  ```

  you can now append the new element to the body as follows:

  ```
  document.body.appendChild(pref);
  ```

- add text to the new paragraph by using the *createTextNode() method and* the *appendChild() method as follows:*

  ```
  txt = document.createTextNode("Hello, new paragraph!");
  new_pref.appendChild(txt);
  ```

Inserting Before a Node

insertBefore(newElement, targetElement)

```
<html><head><title>Inserting Before</title>
<script type="text/javascript">
function insertMessage() {
    var newPara = document.createElement("p");
    var newText = document.createTextNode("I am inserting myself above you!");
    newPara.appendChild(newText);
    var firstPara = document.getElementById("firstp");

    document.body.insertBefore(newPara, firstPara);
}
</script></head>
<body onload="insertMessage()">
<p id=”firstp”>I was supposed to be first.</p>
</body></html>
```

Creating Attributes for Nodes

- element.setAttribute(attributeName, value);
- element.setAttribute(attributeName, value, boolean);

  boolean 0 turns off case-sensitivity,

  1 is on the default Internet Explorer only

- var headings = document.getElementsByTagName("h1");
- headings[0].setAttribute("id", "firsth1");

Creating a Table with the DOM

- Creating a table with the DOM can be a little confusing.
- The DOM requires two steps:
  - first, to create an empty element with the *createElement()*
  - *then to insert the element into the document tree* with the *appendChild() or insertChild() method.*
  - *That means for the table element, the* table heading, captions, rows and table data, and so on.
- you must create an empty element and then place it in the tree.

Creating a Table with the DOM

```
<script type="text/javascript">
window.onload = function(){
var Table = document.createElement("table");

Table.setAttribute("id","myTable");

var THead = document.createElement("thead");

var TBody = document.createElement("tbody");

var Row, Cell, i, j;

var header=new Array( "State","Capital","Abbr");

var state =[ ["Arizona","Phoenix","AZ"], ["California","Sacramento","CA"]];

Table.appendChild(THead);

Table.appendChild(TBody);
```

Creating a Table with the DOM

```
Row = document.createElement("tr");
THead.appendChild(Row);
for (i=0; i<header.length; i++){
    Cell = document.createElement("th");
    Cell.innerHTML = header[i];
    Row.appendChild(Cell);}

for (i=0; i<state.length; i++){
    Row = document.createElement("tr");
    TBody.appendChild(Row);

    for (j=0; j<state[i].length; j++){
            Cell = document.createElement("td");
            Cell.innerHTML = state[i][j];
            Row.appendChild(Cell);}
}
Tcontainer=document.getElementById("TableContainer");
Tcontainer.appendChild(Table); }</script>
```

```
</head>
<body>
<div id="TableContainer"></div>
</body>
</html>
```

Cloning Nodes

- cloneNode(boolean)
  - *true: recursive or deep copy;*
  - *false: copy* current node and attributes, not child nodes.

EXAMPLE

- newPara=oldPara.cloneNode(true);

Removing node

FORMAT

- removeChild(referenceToChild)

EXAMPLE

- parentDiv1.removeChild(div2);

Event Listeners with the W3C Model

Adding an Event

- target.addEventListener(type, listener, useCapture);
  - type to listen for (mouseover, click, etc.).
  - A function to be executed when the event is fired.
  - A Boolean (called *useCapture) of true or false to specify the event propagation* type: *true to turn on event capturing and false to turn on event bubbling (the* most cross-browser-compliant way is false).

EXAMPLE

```
var div1 = document.getElementById("mydiv");
div1.addEventListener('click', sayWelcome, false);
div1.addEventListener('mouseover', highLight, false);
```

Removing an *EventListener*

- element.removeEventListener(eventType, function, useCapture);

- element.addEventListener("mouseover", highlight, false);
- element.removeEventListener("mouseover", highlight, false);