5.1 JavaScript Execution Environment

- The JavaScript `window` object represents the window in which the browser displays documents.

- The `window` object provides the largest enclosing referencing environment for scripts.

- Its properties are visible to all scripts in the document (they are the globals).

- Other `window` properties:

  - `document` - a reference to the `Document` object that the window displays.

  - `frames` - an array of references to the frames of the document.

  - `forms` - an array of references to the forms of the document.

  - Each `Form` object has an `elements` array, which has references to the form’s elements.

  - Form elements are usually referenced by name, but this is a problem for radio buttons.
5.2 The Document Object Model

- Under development by W3C since the mid-90s
  - DOM 0 is supported by all JavaScript browsers
  - DOM 2 is the latest approved standard
    - Nearly completely supported by NS6
    - IE6’s support is lacking some important things
- The DOM is an abstract model that defines the interface between HTML documents and application programs
- It is an OO model - document elements are objects
- A language that supports the DOM must have a binding to the DOM constructs
- In the JavaScript binding, HTML elements are represented as objects and element attributes are represented as properties
  
  e.g., `<input type = "text" name = "address">`
  
  would be represented as an object with two properties, `type` and `name`, with the values "text" and "address"

→ SHOW document & DOM tree
5.3 Element Access in JavaScript

- There are several ways to do it

- Example (a document with just one form):

  <form action = "" >
    <input type = "button"  name = "pushMe" >
  </form>

1. DOM address

   document.forms[0].element[0]

   - Problem: A change in the document could invalidate this address

2. Element names – requires the element and all of its ancestors (except body) to have name attributes

   - Example:

     <form name = "myForm"  action = "" >
       <input type = "button"  name = "pushMe" >
     </form>

     document.myForm.pushMe

   - Problem: Strict standard does not allow form elements to have names
5.3 Element Access in JavaScript  
(continued)

3. `getElementById` Method

- Example:

```html
<form action = "">
    <input type = "button" id = "pushMe">
</form>

document.getElementById("pushMe")
```

5.4 Events and Event Handling

- We look at the DOM 0 event model first

- In event-driven programming, code is executed as a result of a user or browser action

- An event is a notification that something specific has occurred, either with the browser or an action of the browser user

- An event handler is a script that is implicitly executed in response to the appearance of an event
5.4 Events and Event Handling
(continued)

- Because events are JavaScript objects, their names are case sensitive - all are in lowercase only

- The process of connecting an event handler to an event is called *registration*

- Don’t use `document.write` in an event handler, because the output may go on top of the display

- Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Tag Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>abort</td>
<td>onAbort</td>
</tr>
<tr>
<td>blur</td>
<td>onBlur</td>
</tr>
<tr>
<td>change</td>
<td>onChange</td>
</tr>
<tr>
<td>click</td>
<td>onClick</td>
</tr>
<tr>
<td>error</td>
<td>onError</td>
</tr>
<tr>
<td>focus</td>
<td>onFocus</td>
</tr>
<tr>
<td>load</td>
<td>onLoad</td>
</tr>
<tr>
<td>mouseout</td>
<td>onMouseOut</td>
</tr>
<tr>
<td>mouseover</td>
<td>onMouseOver</td>
</tr>
<tr>
<td>reset</td>
<td>onReset</td>
</tr>
<tr>
<td>resize</td>
<td>onResize</td>
</tr>
<tr>
<td>select</td>
<td>onSelect</td>
</tr>
<tr>
<td>submit</td>
<td>onSubmit</td>
</tr>
<tr>
<td>unload</td>
<td>onUnload</td>
</tr>
</tbody>
</table>
5.4 Events and Event Handling
(continued)

- The same attribute can appear in several different tags
  
e.g., The `onClick` attribute can be in `<a>` and `<input>`

- 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

→ SHOW Table 5.2

- Event handlers can be specified in two ways:
  
  1. By assigning the event handler script to an event tag attribute

     ```javascript
     onClick = "alert('Mouse click!');"
     onClick = "myHandler();"
     ```
5.4 Events and Event Handling
(continued)

- Example: the load event - triggered when the loading of a document is completed

```html
<! -- load.html
   An example to illustrate the load events -->
<html>
<head>
<title> The onLoad event handler </title>
<script type = "text/javascript">
<! --
// The onload event handler

function load_greeting () {
    alert("You are visiting the home page of 
    + "Pete's Pickled Peppers 
    + "WELCOME!!!");
}
// -->
</script>
</head>

<body onload="load_greeting();">
</body>
</html>
```
5.4 Events and Event Handling

(continued)

- **Radio buttons**

  ```html
  <input type = "radio" name = "button_group"
  value = "blue" onClick = "handler()">
  ```

  - The `checked` property of a radio button object is true if the button is pressed

  - Can’t use the element’s name to identify it, because all buttons in the group have the same name

  - Must use the DOM address of the element, e.g.,

  ```javascript
  var radioElement = document.myForm.elements;
  ```

  - Now we have the name of the array of elements of the form

  ```javascript
  for (var index = 0;
       index < radioElement.length; index++) {
    if (radioElement[index].checked) {
      element = radioElement[index].value;
      break;
    }
  }
  ```
5.4 Events and Event Handling
   (continued)

SHOW radio_click.html & Figures 5.3 & 5.4

2. Event handlers can be specified by assigning them to properties of the JavaScript objects associated with the HTML elements

- The property names are lowercase versions of the attribute names

- If the event handler is a function, just assign its name to the property, as in

  document.myForm.elements[0].onclick = myHandler;

- This sets the handler for the first element in the form

- This would need to follow both the handler function and the HTML form

- If this is done for a radio button group, each element of the array must be assigned

SHOW radio_click2.html
5.4 Events and Event Handling  
(continued)

- The disadvantage of specifying handlers by assigning them to event properties is that there is no way to use parameters

- The advantage of specifying handlers by assigning them to event properties are:

  1. It is good to keep HTML and JavaScript separate
  2. The handler could be changed during use

- *Checking Form Input*

- A good use of JavaScript, because it finds errors in form input before it is sent to the server for processing

- *Things that must be done:*

  1. Detect the error and produce an alert message
  2. Put the element in focus (the focus function)
  3. Select the element (the select function)
5.4 Events and Event Handling
(continued)

- The `focus` function puts the element in focus, which puts the cursor in the element

  ```javascript
document.getElementById("phone").focus();
```

- The `select` function highlights the text in the element
- Neither `select` nor `focus` work with NS 6.2

- To keep the form active after the event handler is finished, have it return `false`

- *Example* – comparing passwords

  - If a password will be used later, the user is asked to type it in twice

  - The program must verify that the second typing of the password is the same as the first

  - The form just has two password input boxes to get the passwords and Reset and Submit buttons

  - The event handler is triggered by the Submit button
5.4 Events and Event Handling
(continued)

- **Handler actions:**
  1. If no password has been typed in the first box, focus on that box and return `false`
  2. If the two passwords are not the same, focus and select the first box and return `false` if they are the same, return `true`

--> SHOW `pswd_chk.html` & Figures 5.5 & 5.6

- **Another Example** – Checking the format of a name and phone number

  - The event handler will be triggered by the `change` event of the text boxes for the name and phone number

  - If an error is found in either, an `alert` message is produced and both focus and select are called on the text box element

  - Another event handler is used to produce a thank you `alert` message when the input is ok

→ SHOW `validator.html` & Figures 5.7 & 5.8
5.5 The DOM 2 Event Model

- Does not include DOM 0 features, but they are still supported

- Much more powerful than the DOM 0 model

- Microsoft does not support it, yet

- Event propagation

  - The node of the document tree where the event is created is called the target node

  - The first phase is called the capturing phase

  - Events begin at the root and move toward the target node

    - If there are registered event handlers at nodes along the way (before the target node is reached), if one is enabled, it is run

  - The second phase is at the target node

    - If there are registered handlers there for the event, they are run

  - The third phase is the bubbling phase
    - Event goes back to the root; all encountered registered handlers are run
5.5 The DOM 2 Event Model
(continued)

- Not all events bubble

- Any handler can stop further propagation by calling the `stopPropagation` method of the `Event` object

- DOM2 model uses the `Event` object method, `preventDefault` to stop default operations, such as submission of a form, even though an error has been detected

- Event handler registration is done with the `addEventListener` method

  - Three parameters:
    1. Name of the event, as a string literal
    2. The handler function
    3. A Boolean value that specifies whether the event is enabled during the capturing phase

    `node.addEventListener("change", chkName, false);`
5.5 The DOM 2 Event Model

(continued)

- A temporary handler can be created by registering it and then unregistering it with removeEventLister

- The currentTarget property of Event always references the object on which the handler is being executed

- The MouseEvent object (a subobject of Event) has two properties, clientX and clientY, that have the x and y coordinates of the mouse cursor, relative to the upper left corner of the browser window

- An example: A revision of validator, using the DOM 2 event model

SHOW validator2.html

- Note: DOM 0 and DOM 2 event handling can be mixed in a document
5.6 The `navigator` object

- Indicates which browser is being used

- Two useful properties
  1. The `appName` property has the browser’s name
  2. The `appVersion` property has the version #

- Microsoft has chosen to set the `appVersion` of IE6 to 4 (?)

- Netscape has chosen to set the `appVersion` of NS6 to 5.0 (?)

→ SHOW `navigator.html` & Figures 5.9 & 5.10