ICT286
Web and Mobile Computing

Topic 4
JavaScript Programming in the Web Browser Environment

Objectives

- Understand relationship between JavaScript core and its execution environment.
- Understand what built-in properties are available from the JavaScript web browser environment and how to access them.
- Understand and be able to debug JavaScript code in web browsers such as Chrome, Firefox and IE.
- Understand and be able to use HTML forms.
- Understand the relationship between `Document` object and the HTML document in the in browser window.
- Understand Document Object Model (DOM) and be able to access HTML elements using the form array, the elements array, name attributes and id attributes.

Objectives

- Understand event-driven programming. Understand and be able to use common events. Be able to define event handlers and be able to register event handlers with the relevant events in HTML elements.
- Be able to perform validations on form inputs using JavaScript.

JavaScript Execution Environment

- JavaScript defines a global object that contains the language build-ins and host-specific properties and methods are defined.
- This global object has different name in different host environment.
- For Node.JS, the object reference of this global object is `global`. We discussed JavaScript under Node.js in Topic 3.
- For a web browser, this global object is `Window`, and its object reference is `window`.
- JavaScript has introduced a standard reference for the global object: `globalThis`.
- In this topic, we will use JavaScript in the browser environment.

JavaScript Execution Environment

- The `Window` object represents the window in which the browser displays the document that contains/references the JavaScript code. This object contains the properties and methods defined in JavaScript core, such as `NaN`, `parseInt`, `Object`, `Array`, `String`, `Math`, `RegExp`, `JSON`, as well as properties and methods for the browser environment (`document`, `navigator`, `location`, etc).
- Properties and methods defined in `Window` are automatically available everywhere in your scripts, with or without the object reference. Eg,

```
console.log(window.parseInt("3.14xradius");
console.log(parseInt("3.14xradius");
```

JavaScript Execution Environment

- The following are some of the properties inside `Window` object representing the host environment:
  - `window` - which references itself
  - `navigator` - the browser
  - `screen` - the computer screen
  - `location` - the url of the current page
  - `history` - browsing history
  - `alert`, `confirm`, `prompt` methods
  - `setTimeout` and `setInterval` methods
  - and many more . . .
- All global variables and functions you declare are also part of `Window`. However this feature exits for backward compatibility. You should not reply on it.

JavaScript Execution Environment

- One of the most important objects in `Window` is the `Document` object. Its object reference is `document`.
- The `Document` object represents the HTML document displayed in the current window. It consists of a tree of objects, each representing one element in the HTML page.
- This tree of objects is known as the DOM tree for the HTML page. It is created by parsing the HTML page.
- The DOM tree resides in the browser's memory.
- By manipulating the DOM tree using DOM interface, we can dynamically change the look and behavior of the web page.

Embedding JavaScript in HTML

- There are a number of ways to embed JavaScript code in HTML.
- Firstly, JavaScript code can be placed inside a `<script>` element:

```
<script>
   document.write("Hello, world!");
</script>
```

- JavaScript functions are usually placed inside the head element in your HTML document.

Embedding JavaScript in HTML

- The second method is to put your JavaScript code in a file and include the url of the file in `<script>` element. Eg.

```
<script src="myscript.js"> </script>
```

- This is the preferred way of embedding JavaScript to HTML.
- The JavaScript code doesn't have to be stored on the same server. Eg

```
<head>
<script src="https://ajax.aspnetcdn.com/aja
x/jQuery/jquery-3.3.1.min.js"></script>
</head>
```

Embedding JavaScript in HTML

- The last method is by expressing it as an event handler within an HTML tag (mostly form elements). Eg.

```
<input type="button"
    name="Button1"
    value="Open Sesame!"
    onclick="window.open('mydoc.html',
                         'newWin')” />
```

Debugging JavaScript in Web Browsers

- Most web browsers provide a console to support for code debugging.
  - Crome:
    View=> Developer=>JavaScript Console
  - Firefox:
    Tools => Web Developer => Debugger
  - Safari:
    Develop => Show JavaScript Console
  - IE:
    Settings and more => Developer Tools
- You can display debugging messages and error messages inside the console using `console.log()` method. These messages are for the developer, *not for the user of the web application*.

Some Window Methods

Method	Description
`open(`*`url, name, options`*`)`	Creates a new window with the URL of the window set to *url*, the name set to *name*, and the features set by the string *options*.
`close()`	Closes this window.
`focus()`	Gives the focus to this window.
`blur()`	Takes the focus away from this window
`print()`	Print the contents of this window.

Window Methods (Dialog Boxes)

Method	Description
`alert(`*`string`*`)`	Display a dialog box with the given *string* and an *OK* button. The method returns when the user clicks the *OK* button.
`confirm(`*`string`*`)`	Display a dialog box with the given *string* and an *OK* and a *Cancel* button. The method returns true if the user clicks *OK*, or false if the user clicks *Cancel*.
`prompt(`*`prompt`*`,` *`default`*`)`	Display a dialog box with the given string *prompt*, a textbox containing string *default* and an *OK* and *Cancel* button. Return the string the user entered if the user clicks the OK button, or null if the user clicks Cancel button.

The `alert` Method

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

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

The `confirm` Method

- The `confirm` method displays a message provided as a parameter. The 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?", "");
```

Some Window Properties

Property	Description
`status`	Set the message in the status bar of the current window
`name`	Get or set the name of the current window
`document`	Object reference to obeject `Document`
`closed`	A Boolean value that is set to true if the window is closed and false if it is not.
`location`	The URL of the web page currently displayed in the window.
`history`	A list of the URLs that have been stored in window.location.

The Navigator Object

- The `navigator` object holds information about the browser being used. The name navigator arose because the first browser from Netscape was named Navigator. This object refers to any browser that is being used, not just Navigator.
- There are some methods for this object, but you will mainly use its properties.

Some Navigator Properties

Property	Description
`appCodeName`	The code name of the browser, eg Mozilla.
`appName`	The name of the browser, eg, Netscape
`userAgent`	The value of the user-agent header sent by the client to the server
`appVersion`	The version of the browser.
`product`	The browser engine, eg Gecko
`platform`	The platform on which the browser is running, eg, MacIntel.
`onLine`	Is the browser online?
`cookieEnabled`	is the cookie enabled in the browser?

The Document Object

- The `document` object represents the document being displayed in the current browser window.
- It is the root of the DOM tree which is actually the internal representation of the HTML document that the browser displays.
- One of the useful methods from `document` object is `write` and `writeln`.
- JavaScript program can access the HTML document using the DOM interface via the `document`.
- More commonly we use JQuery to manipulate the DOM tree. JQuery is a JavaScript library.
- We will cover the manipulation of DOM in the next topic.

Generate Output

- The 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 writes its arguments to the browser window.
- The output is interpreted as HTML by the browser.
- Therefore, if a line break is needed in the output, you must insert `<br/>` into the output, as newline characters will be ignored.

HTML Forms

- HTML forms are a way of getting input from the user, which can then be processed by a back-end program. There are three parts that are coded with forms:
  1. The input screen. This is the part that the user sees, with prompts, textboxes, buttons, etc. It is usually coded in HTML. We will be examining these in this topic.
  2. Form validation. This checks that the values the user has entered are valid and complete, before sending the user input to the back-end program. This is written in JavaScript, usually as an event handler, which runs on the client machine in response to an event. We will discuss event handling in this topic.
  3. Form processing. This is the back-end program that accepts the user input and does something with it, for example sending some information back to the user, or creates an order. This can be coded in many different languages. We will be using PHP which will be covered in Topic 7.

HTML Forms

- The `<form>` tag defines what is in the form.
- Only one attribute, `action`, is required for any form. This attribute specified the url of the program to be executed when the user clicks the *Submit* button of the form.
- When the *Submit* button is clicked, the form data are encoded and sent to the server and the server-side script, such as a PHP script, is executed and the execution result is sent back to the client.
- If no action should be taken when the user clicks *Submit* button, you should set `action=""`.
- You can use name or `id` attribute to identify the form.

Form Elements

- Inside a form element, you may use the following elements to define form components:
  - The `<input>` element – it is used to create various components of the form. The `<input>` element requires the attribute `type`.
    - The `type` attribute of `<input>` element indicates the type of the `<input>` tag: *text*, *password*, *search, number, range, color, checkbox*, *radio*, *reset*, *submit, button, time, date, week, month, datetime, datetime-local, email, tel, and url*.
    - It is important to use `name` attributes to identify the input element so that you can access it from JavaScript and PHP.
  - The `<select>` element – it creates a drop-down list allowing the user to select one or several items from the list. Use `<option>` element to define the list items.
  - The `<textarea>` element – it allows the user to input multi-line text. You must use rows and cols attributes to define the size of text area.

Textboxes

- The most common element in a form is a textbox. Textboxes are used by users to enter text, for example their names.
- The textbox is coded using an `<input>` tag with type="text".
- Example:

```
<form  id="myForm" action="">
   <input type="text"
      name="UserName" id="UserName" />
</form>
```

Textboxes

- You can (optionally) give a textbox a default size in characters, and/or a string that will be placed in the textbox.
- The following example displays a textbox for 30 characters, with an initial text “Type your user name here:” in the textbox. If you type more than 30 characters, the textbox will be scrolled:

```
<form id="myForm" action="">
   <input type="text"
       name="UserName" id="UserName"  size="30"
       value="Type your user name here:" />
</form>
```

Password Textboxes

- You can change a textbox so that when the user types in it, asterisks (***) appear, but the actual input is preserved. You have probably seen these used for passwords. You do this by making the type attribute password.
- Example:

```
<form id="myForm" action="">
   <input type="password"
       name="Password" id="Password"
       size="10" />

</form>
```

Text Around Textboxes

- You will probably want to put some prompts around your textboxes, so that the user knows what to type. This is done by using the label element.
- Example 1: place the prompt and the form element inside a label element:

```
<label> Name:
  <input type="text" name="UserName" size="15" />
</label>
```

- Example 2: bind the prompt and the form element with the `for` attribute:

```
<label for="UserName" > Name:</label>
<input type="text" id="UserName" size="15" />
```

Radio Buttons

- Radio buttons give the user an opportunity to choose from a number of options. A user may only chose ***one*** option from a group of radio buttons. In order for radio buttons to work, you must give all the buttons in a radio button group the same name.
- In the following example, note that the two input elements share the same name:

```
<form id="myForm" action="">
    <label> Female:
        <input type="radio" name="Genders" />
        </label> <br />
    <label> Male:
        <input type="radio" name="Genders" />
        </label> <br />
</form>
```

Checkboxes

- Checkboxes are similar to radio buttons except that they give the user an opportunity to choose more than one option. In order for checkboxes to work, you should give all the checkboxes in a checkbox group the ***same*** name. Eg:

```
<form id="myForm" action="">
<p>
  <label>Cat: <input type="checkbox" name="Animals"/> </label>
  <label>Dog: <input type="checkbox" name="Animals"/> </label>
  <label>Rat: <input type="checkbox" name="Animals"/> </label>
  <label>Bird:<input type="checkbox" name="Animals"/> </label>
</p>
</form>
```

The Checked Attribute

- With radio button groups and checkbox groups, you can make one button in the group appear checked by using the checked attribute. Eg:

```
<form id="myForm" action="">
  <p>
  <label>Cat: <input type="checkbox" name="Animals"/>
         </label>
  <label>Dog: <input type="checkbox" name="Animals"
         checked="checked"/> </label>
   <label>Rat: <input type="checkbox" name="Animals"/>
         </label>
  <label>Bird:<input type="checkbox" name="Animals"/>
         </label>
  </p>
</form>
```

File Selection Boxes

- You may want to get the name (and path) of a file from your web page. This is very simple to do. The creation of the form element, browse button and processing on the client side is taken care of by simply setting the type to “file”. What happens to the file needs to be taken care of by the back-end program. Eg:

```
<input type="file" name="MyFileName" />
```

Drop-down List

- The `<select>` element is also known as drop-down list and it allows you to place many choices in a small area. This is especially useful when screen area is at a premium.
- The user can select one or several items (use attribute `multiple="multiple"`) from the list.
- We create select boxes using the `<select>` and `<option>` tags. Eg:

```
<form id="animal" action="">
   <p>Choose your favourite animal:
   <select name="sbAnimal">
        <option>Cat</option>
        <option>Dog</option>
        <option>Rat</option>
   </select>
   </p>
</form>
```

Text Area

- The `<textarea>` element allows the user to input text consisting of multiple lines.
- You use attributes `rows` and `cols` to define the box size. The attribute `rows` specifies the number of lines and `cols` specifies the number of characters.

```
<form id="feedback" action="">
   <p>Type your feedback:
      <textarea name="feedfback"
            rows="3" cols="25">
        (be brief)
      </textarea>
   </p>
</form>
```

Action Buttons

- You would be familiar with buttons in the real world and on web pages. We use buttons to initiate or stop an action. There are some buttons that have predefined meaning, for example *Reset* and *Submit*. If you use any of these buttons on your web pages, make sure that you give them the same meaning as they normally have.

Reset Button

- The *Reset* button is used to clear all the fields in the form, and start all over again. If you create an input of type `reset`, a reset button will be created and all the processing taken care of by the browser. Eg:

```
<input type="reset" name="ResetButton" />
```

Submit Button

- The *Submit* button is used to send the values from the form to the back-end program. You need to specify two things: the button and the action to be taken (in the `<form>` tag). Eg:

```
<form id="BookOrder"
    action="ProcessForm.php">
    <input type="submit"
        name="SubmitButton"
        value="Submit Order" />
</form>
```

The Plain Button

- You will probably want to create buttons to perform other actions. In this topic, we learn how to create the button in the form and in the next few topics we will see how to code the required processing for the action you want to achieve when clicking the button. Eg:

```
<input type="button"
    name="HelpButton" value="Help" />
```

Access HTML Elements in JavaScript

- Each element in the HTML document has a corresponding object in the DOM tree.
- Objects can be addressed in several ways:
  - `forms` (and `elements` array inside each forms array element) defined in DOM Level 0
    - Individual elements are specified by index
    - The index may change when the form changes
  - Using the name attributes for form and form elements
    - Names are required on elements in a form that provides data to the server
  - Using `getElementById` with id attributes
    - id attribute value must be unique for an element

Using forms Array

- The `document` object contains the `form` array. Each `form` object contains an `elements` array of form elements.
- Example: assume that a document has only one form (hence index 0):

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

This button element can be referenced as

`document.forms[0].elements[0]`

Using name Attributes

- If the element has a name attribute, the name can be used to reference the element.
- To use this method, all ancestors of the element must also have the name attribute.
- Example

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

- Referencing the input element:

```
document.myForm.pushMe
```

Using getElementById

- DOM Level 1 provides `getElementById` method from `document` object.
- Using this method, the element must have a unique id. Eg:

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

- The object reference for the above `input` element is:

```
document.getElementById("turnItOn")
```

Using Id Directly

- In HTML5, you can access the DOM object for an element with an ID using the ID directly in JavaScript.
- Assuming the following element:

```
<input type="text"  id="Greeting">
```

- In JavaScript, you can access the DOM object using ID Greeting directly:

```
Greeting.value = "Type a message here";
```

Using the Arrays in CheckBoxes and Radio Buttons

- Each checkbox and radio button group has an implicit array whose name is the same as the name attribute of the group. This array is available in the object for the form in which the check boxes or radio buttons are defined. Eg,

```
<form id = "topGroup">
  <input type="checkbox"  name="toppings" value="olives" />
  ...
  <input type="checkbox"  name="toppings" value="tomatoes" />
 </form>
```

- JavaScript:

```
var numChecked = 0;
var dom = document.getElementById("topGroup");
for i = 0; i < dom.toppings.length; i++)
   if (dom.toppings[i].checked)
      numChecked++;
```

Event-driven Programming

- Event-driven programming is a style of programming in which pieces of code, called event handlers, are 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 fragment designed to execute when a certain event occurs
- The web browser constantly monitors the events. When an event, such as a mouse click, occurs on an HTML element, the web browser automatically calls the relevant event handler registered for that event in that HTML element.

Events and Event Handling

- An *event* is a notification that something specific has occurred, either with the browser or due to an action of the browser user
- An *event handler* is a script that is automatically executed in response to the occurrence of an event
- The process of connecting an event handler to an event is called *registration*

Events and Event Attributes

More information are available from Table 5.1 and Table 5.2 of the textbook (pages 203 to 205)

Event name (event attribute)	What triggers It
click (`onclick`)	The user clicks the mouse button on the object.
dbclick (`ondblclick`)	The user double clicks the mouse button on the object.
focus (`onfocus`)	The user moves to the object by clicking the object or tabbing into it.
blur (`onblur`)	The user moves off the object by clicking a different place, or tabbing away from it.
mouseover (`onmouseover`)	The user moves the mouse cursor onto the object.
load (`onload`)	The browser finishes loading a window.
unload (`onunload`)	The user exits the document.
change (`onchange`)	The user alters the content of an object.
submit (`onsubmit`)	The submit button in the form is pressed.

Registration of Event Handlers

- An event handler can be registered for an HTML element in two ways. Assume SendOrder is a Javascript function.
- Method 1: Assume the following form element:

```
<input type="button" value="Order Now"
        name="orderButton" />
```

Define the event attribute *onclick*:

```
<input type="button" value="Order Now"
        name="orderButton"
        onclick="SendOrder();"  />
```

Registration of Event Handlers

- Method 2: assigning to a property of the DOM object for the element using `getElementById`. With this method, the element must have an id attribute:

```
<input type="button" value="Order Now"
        name="orderButton" id="OrderButton" />
```

Assign the event handler reference to the event property:

```
document.getElementById("OrderButton").onclick =
    SendOrder;
```

  - Note that the function name (which acts as the reference to the function) is assigned, ie, no parentheses.
    - Writing `SendOrder()` would assign the return value of the function call as the handler instead!
  - The above JavaScript code must be executed *after* the form has been parsed. We usually place the JavaScript code at the end of the HTML code.

Focus and Blur Events

- 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 clicks the left button
  2. When the user tabs to the element
  3. By executing the `focus` method
- Losing the focus is *blur*ring

Form Validation

- Now we can pull together everything we have learned about JavaScript to perform form validation.
- Each of the form elements we considered in HTML can be validated using JavaScript. For example, to check if mandatory fields are present/been checked, or that fields contain valid values.

Form Validation

- Assume that we have a form called MyForm, containing three input elements and one select element:

```
<form id="MyForm" action="">
  <p><label>Type your name: <input type="text"
      name="name" /> </label> </p>

  <p><label>Female: <input type="radio" name="genders" value="F"/>
       </label> <br />
     <label>Male: <input type="radio" name="genders"  value="M"/>
       </label> </p>

  <p<label>Cat: <input type="checkbox" name="animals" value="cat"/>
       </label> <br />
    <label>Dog: <input type="checkbox" name="animals" value="dog"/>
       </label> </p>

  <p><select name="colour">
         <option> Red </option>
         <option> Blue </option>
         <option> Yellow </option>
      </select> </p>
</form>
```

Form Validation Using Form Name

- We can access each of these elements using the name attribute. Eg:

Form:

```
<form name="MyForm" action="" >
   <input type="text" name="name" />
    ...
</form>
```

JavaScript:

```
var name = document.MyForm.name.value;
```

Form Validation Using Ids

- Alternatively we can access each of these elements using method `getElementById`:

  Form:

```
<form action="">
     <input type="text" id = "name" />
     ...
</form>
```

  JavaScript:

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

- Note the id must be unique in the document.

Form Validation – check boxes

Form:

```
<form name="MyForm" action="" >
   <p><label>Cat: <input type="checkbox" value="cat"
          name="animals" /> </label> <br />
   <label>Dog: <input type="checkbox" value="dog"
          name="animals" /> </label> </p>
</form>
```

JavaScript:

```
var animal;
var animals= document.MyForm.animals;
for (var i=0; i<animals.length; i++) {
   if (animals[i].checked) {
      animal = animals[i].value;
      document.write("Animal ", animal, "is checked<br />");
   }
}
```

Form Validation – radio button

Form:

```
<form id="MyForm" action="">
  <p><label>Female: <input type="radio" value="F"
      name="genders" /> </label> <br />
     <label>Male: <input type="radio" value="M"
      name="genders" /> </label></p>
</form>
```

JavaScript:

```
var genderSelected;
var genders = document.getElementById("MyForm").genders;
for (var i=0;i<genders.length; i++) {
    if (genders[i].checked) {
        genderSelected = genders[i].value;
        break;
    }
}
```

Form Validation – list boxes

Form:

```
<form name="MyForm" action="">
   <p><select name="colour" />
       <option>Red </option>
       <option>Blue </option>
       <option>Yellow </option>
    </select></p>
</form>
```

JavaScript:

```
var colour= document.MyForm.colour;
var index=colour.selectedIndex;
var option = colour.options[index];
document.write("Colour selected is ", option.text);
```

- `selectedIndex` contains the index of the selected item.

`document.write` Revisited

- The behaviour of `document.write` (including `document.writeln`) depends on whether the existing HTML document is fully parsed:
  - If the existing document is still being parsed, the output from `document.write` will be inserted into the existing document.
  - However if the parsing of the existing document is completed, the first call to `document.write` will triger a call to `document.open()` *which will clear the existing document!*
    - This means that if you call `document.write` inside an event handler, the existing document will be replaced by the output from the `document.write`.

document.write Revisited

- Compare the following two examples:
- Example 1

```
<!DOCTYPE html>
<html>
<body>
<p> The following is generated by Javascript. </p>
<script>
    var red="<p style=\"color:red\">Javascript</p>";
    var blue="<p style=\"color:blue\">Javascript</p>";
    document.write (red);
    document.write (blue);
</script>
</body>
</html>
```

document.write Revisited (cont'd)

- Example 2

```
<!DOCTYPE html>
<html>
<body>
<script type="text/javascript">
  var red="<p style=\"color:red\">Javascript</p>";
  var blue="<p style=\"color:blue\">Javascript</p>";
</script>
<p> Click the button to execute a Javascript code. </p>
<button onclick="document.write(red);document.write(blue);" >
Click me to run Javascript
</button>
</body>
</html>
```

Readings

Textbook:

Sebesta: Ch 5

Sebesta: Ch 2.9

W3Schools:

http://www.w3schools.com/js/default.asp

Kindle book:

Mark Myers: A Smart Way to Learn JavaScript