Lecture Notes 4
JavaScript Functions, Arrays and Objects
Client-side JavaScript

Anand Tripathi

CSci 4131

Internet Programming

Topics

- JavaScript Functions
  - Scope rules, parameter arguments
- JavaScript Array
  - Array operations
  - Array parameters to functions by reference
- JavaScript with HTML Document Object Model
  - Dynamic content generation and modification

Variable Declaration and Scope

1. A variable can be declared at any place in the code.

```
var i ;                 // declares variable i
var m = "Hello" ;       // initializes m to string "Hello"
```

2. A variable can be declared any number of times. A re-declaration with a value initialization acts like an assignment.
3. If a variable is not declared using "var" but used in some statement, that causes creation of that variable in the global scope.
4. There is no block scope: Any variable declared within a block enclosed by curly braces is visible outside of that block.
   For example:

```
{ ....
   { var k ...
   }
 variable k is visible here
}
```

Functions

1. Functions can be nested.
2. JavaScript follows lexical (static) scoping rules for accessing variables.

There are several ways in JavaScript to define a function. Function are first-class data types.

```
function square ( x ) { return x * x ; }
```

- This declares a function named "square" which can be called as square(5) to return value 25.

Functions

- A function definition can be assigned to a variable.

The following creates an anonymous function.

```
var S = function ( x ) { return x * x ; }
```

Now we can call: S(10) to square 10.

- A function literal can be defined and called, as shown below:

```
var FivePlusFive = ( function (x) { return x + x; } ) ( 5 );
```

- A function can also be defined using the Function() object constructor.

```
var f = new Function( "x", "return x*x;" );
```

The Function constructor creates a new function object.

Static Scope Rules

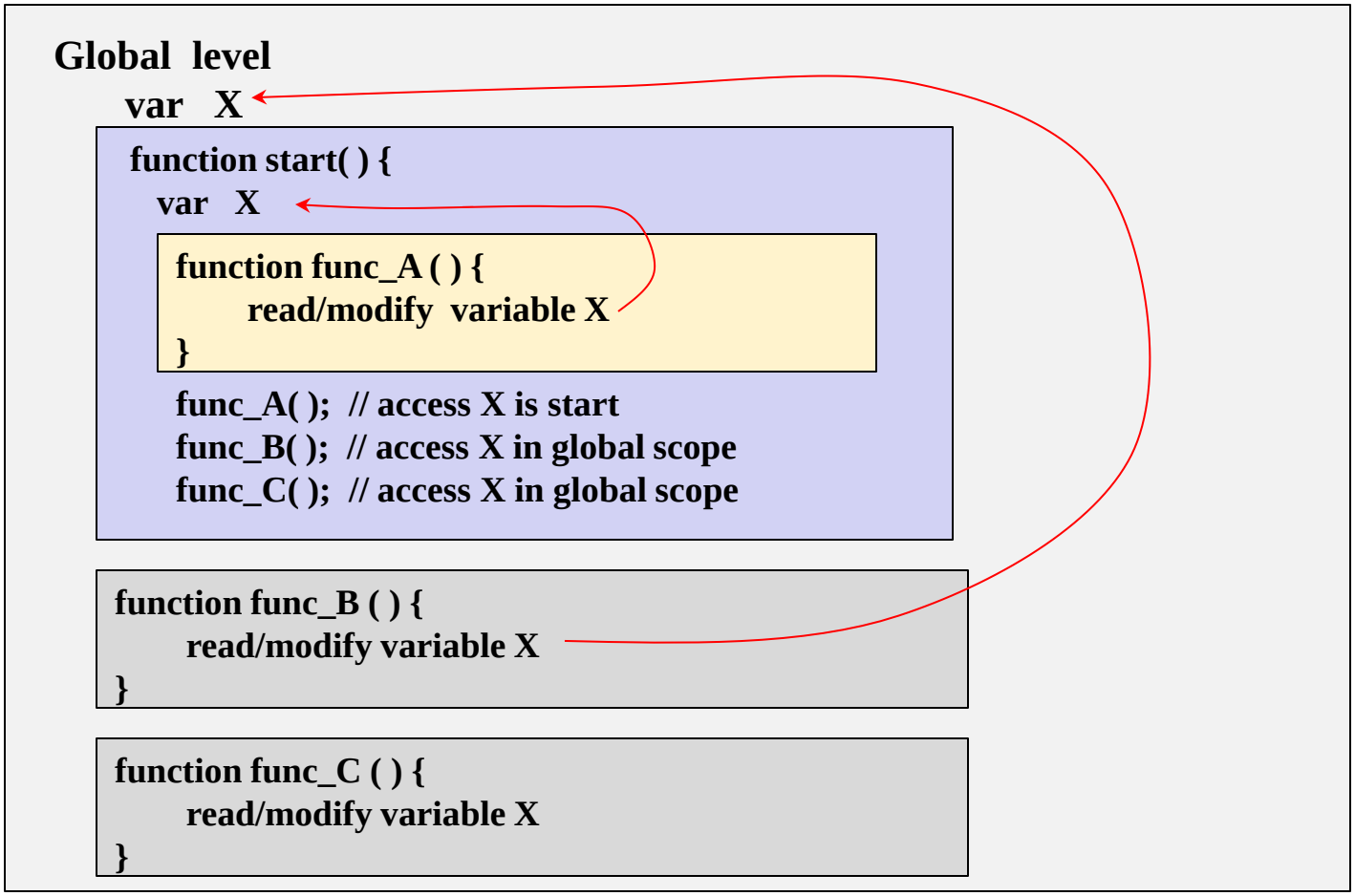
When a variable is accessed within a function, the following static scope search rules apply:

1. Look for variable if it is defined in the scope of that function, i.e.
   - Check if it a local variable in the function

2. If not found in step 1, the search the nesting outer function if the variable is declared there.
   - If not found there, then continue the search to the next outer nesting function.

Static Scope Rules

based on static nesting of functions



Example of Scope

```
var x = "global";
function f() {
    alert( x ); // alert will say undefined
    var x = "local";
    alert( x ); // this will display "local"
}
 f( );
```

- When alert is called for the first time in function f, the local variable "x" is not yet assigned a value.
- Hence, it says "x is undefined".

Hoisting of variable declaration

The previous can be viewed as:

```
var x = "global";
function f() {
    var x ;     // declaration of the variable, undefined value
                // declaration is hoisted
    alert( x );   // alert will say undefined
    x = "local";
    alert( x );   // this will display "local"
}
```

Function Objects

```
<html>
  <head>
    <title>Scoping Example</title>
    <script type = "text/javascript">
      function start() {
          function multiplier (x) {
              return function (y)  {     return x * y;   };
          }
    var someFunction  = multiplier(3);
          var result = someFunction (7);
          alert ( result );
      }
    </script>
  </head>
  <body onload = "start()“ ></body>
</html>
```

Click here to see the execution of this code

Example of Scope Rules

- Please carefully study the example in given in Section 97 of Deitels' book.
- The example code is given in Figure 9.9

Arrays

1. An array in JavaScript is a numbered collection of objects.
2. An array can contain data of different types.
3. Array index starts with 0.
4. Objects and arrays are basically the same kind of entities.
5. An array can have any number of elements.

```
var a = new Array ( ); // Creates an empty array

var A = new Array ( 10 ); // creates an array of length 10

var b = new Array ( 5, 2, "some", 1 );   // different types data
```

Arrays

- var colors = new Array("cyan", "magenta","yellow", "black");
- var integers1 = [2, 4, 6, 8];
- var integers2 = [2, , , 8];
- var n1 = new Array(5); // allocate five-element Array

```
// assign values to each element of Array n1
  for ( var i = 0; i < n1.length; ++i )
          n1[ i ] = i;
```

- One can store function references in an array:
- a[0] = square ; a[1] = 20; a[2] = a[0](a[1]);
- // this will compute square of 20 and store in a[2]

Array operations

Several useful methods are defined for arrays in JavaScript:

- join -- a.join() Concatenates all elements

  a.join(separator)

  - If separator character is omitted, then comma is used a the separator.

- sort -- a.sort()

  a.sort (compareFunction)

  – If compareFunction is not given then sort performs “dictionary” sorting, by converting the array elements to strings

- reverse -- a.reverse();
- concat --
  var a = [1, 2, 3];
  a.concat(4, 5); // now a = [1, 2, 3, 4, 5]

- slice -- a.slice(1, 3) // first arg is start index, second is last index.

- push
  - Add element at the end (highest index position)
- pop
  - removes the last element and returns the element
- shift
  - Returns the first element and returns it
- unshift
  - Adds new element at the front of the array

Passing Arguments to a Function

- Arguments of primitive types such as numbers and strings are passed by value.
- An object or an array argument is passed by reference.
  - See example Figure 10.14 in Deitel's book

Example 10: Array Arguments to Functions

- An array argument is passed by reference.
- An array element as an argument is passed by value.
- See here Example in Figures 13 and 14 from Chapter 10

```
var a = [ 1, 2, 3, 4, 5 ];
document.writeln( "<h2>Effects of passing entire " + "array by reference</h2>" );

outputArray( "Original array: ", a );

modifyArray( a );  // array a passed by reference

outputArray( "Modified array: ", a );

document.writeln( "<h2>Effects of passing array " +  "element by value</h2>" +
        "a[3] before modifyElement: " + a[ 3 ] );

modifyElement( a[ 3 ] ); // array element a[3] passed by value
document.writeln( "<br />a[3] after modifyElement: " + a[ 3 ] );
```

```
// outputs heading followed by the contents of "theArray"
function outputArray( heading, theArray )
{
   document.writeln(
      heading + theArray.join( " " ) + "<br />" );
} // end function outputArray

// function that modifies the elements of an array
function modifyArray( theArray )
{
   for ( var j in theArray )
      theArray[ j ] *= 2;
} // end function modifyArray

// function that modifies the value passed
function modifyElement( e )
{
   e *= 2; // scales element e only for the duration of the
           // function
   document.writeln( "<br />value in modifyElement: " + e );
} // end function modifyElement
```

Example 11: Sorting an array

Click here to See this example

```
var a = [ 10, 1, 9, 2, 8, 3, 7, 4, 6, 5 ];
document.writeln( "<h1>Sorting an Array</h1>" );
outputArray( "Data items in original order: ", a );
a.sort( compareIntegers );  // sort the array
outputArray( "Data items in ascending order: ", a );

// output the heading followed by the contents of theArray
function outputArray( heading, theArray ) {
   document.writeln( "<p>" + heading + theArray.join( " " ) + "</p>" );
} // end function outputArray

// comparison function for use with sort
function compareIntegers( value1, value2 )  {
   return parseInt( value1 ) - parseInt( value2 );
} // end function compareIntegers
```

Multidimensional Array

Click here to See this example

```
var array1 = [ [ 1, 2, 3 ], // first row
               [ 4, 5, 6 ] ]; // second row
var array2 = [ [ 1, 2 ], // first row
               [ 3 ], // second row
               [ 4, 5, 6 ] ]; // third row

outputArray( "Values in array1 by row", array1 );

outputArray( "Values in array2 by row", array2 );
```

Multidimensional Array

Click here to See this example

```
function outputArray( heading, theArray ) {
        document.writeln( "<h2>" + heading + "</h2><pre>" );
          // iterates through the set of one-dimensional arrays

          for ( var i in theArray ) {
             // iterates through the elements of each one-dimensional array

             for ( var j in theArray[ i ] )
                document.write( theArray[ i ][ j ] + " " )

             document.writeln( "<br />" );
          } // end for

          document.writeln( "</pre>" );
       } // end function outputArray
```

OBJECTS

1. An object represents a named collection of data.
2. The names are called the "properties" of the object.
3. Object properties can be accessed using the dot . convention as well as using array indexing like accessor ['property'].
4. The [ ] accessor is necessary when properties are dynamically created.
5. with (object) { } helps in accessing properties of an object without specifying object with "dot" accessor.
6. New properties can be added, or existing ones can be removed. This allows dynamic definition of objects.
7. A function can be stored as an object's property.
8. JavaScript object-oriented programming model is based on "prototype" objects instead of classes.

Object Operations

```
var person = new Object();
person.name = "Tipsy";
person.lastname = "Bee";
person.email = "bee@honeycomb.com";
person.age = 2;
person.occupation = "beezee";
person.hobby = square ; // square is a function
```

We can access a property as:

person["name"] or person.name

At a later point we can delete a property:

```
delete  person.email;
```

It is possible to iterate over an object's properties:

```
function ListProperties ( obj ) {
var names = ""; // empty string
for ( var i in obj ) {
    // iterator over object properties
    names +=  i + "=" + obj[i]+  "\n";
}
alert ( names );
```

Click here to see this example execution

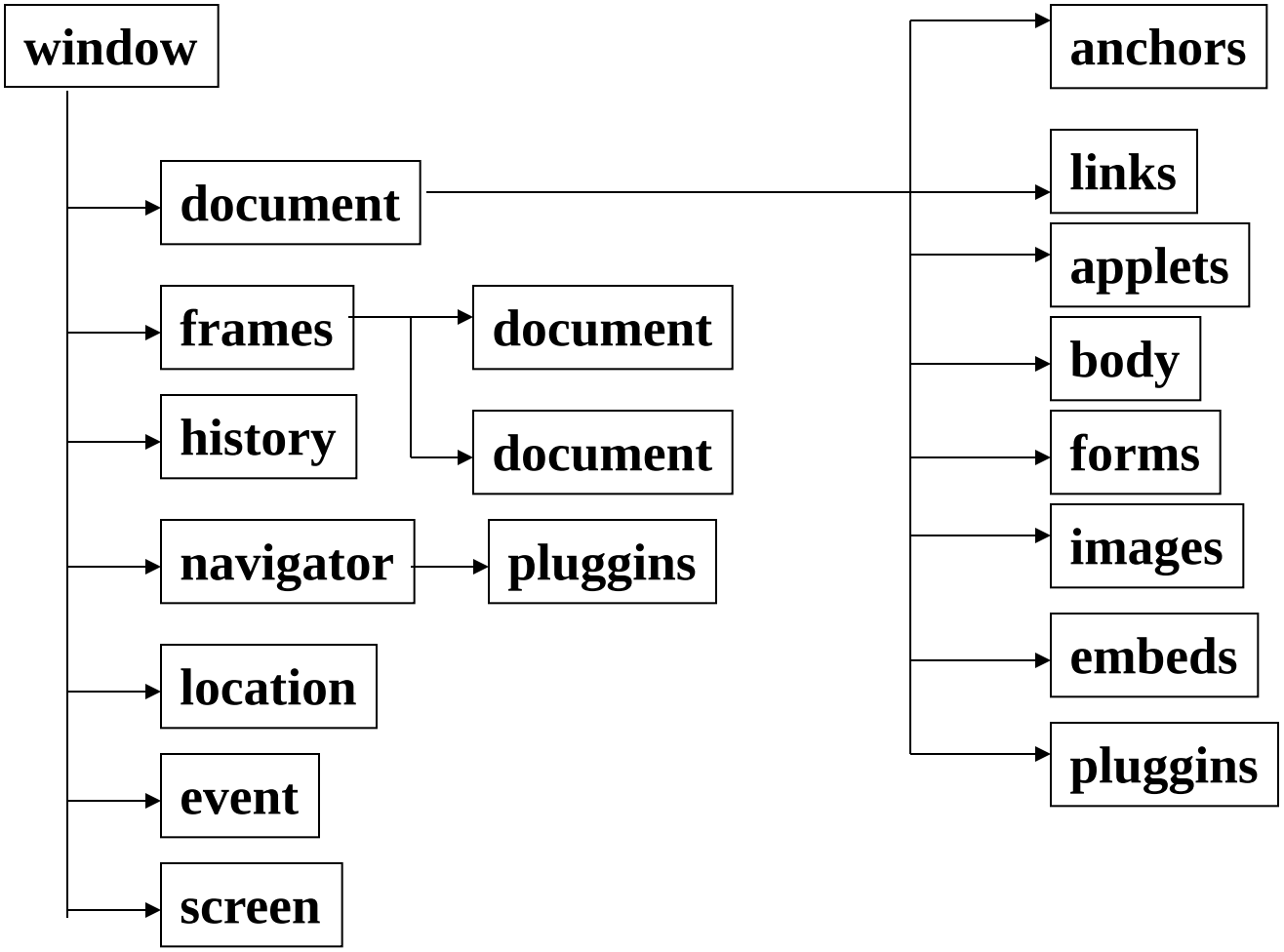
CLIENT-SIDE JAVASCRIPT

- Use of JavaScript to control dynamic content generation and modification of an HTML document displayed in a browser window.
  - Based on user interactions and events
  - Based on timer-based events

CLIENT-SIDE JAVASCRIPT

- This allows you to control your browser's behavior.
- One can control its various objects such as
  - window frames
  - elements of the document displayed in a frame
    - Forms , Buttons, Input fields, Images
  - PAGE 251 of Flanagan's Book
- Load a new document in a window/frame.
- A naming/referencing mechanism that allows a JavaScript code running in one frame to refer to other frame/window objects or the documents displayed in those frames.
- It also supports an event based model for taking actions on user's actions with the mouse or on timer-based events.

Dynamic HTML: Object Model



Two important objects

- window object
  - How to create a new window (pop-up).
  - How to access or change window object properties.
- document object
  - How to access different elements of a document object.
- Flanagan Book Examples can be downloaded from O'Reilly

Properties of the Window Object

The window object represents the browser's display window.

The following are some of the properties defined for a window object:

- **closed** boolean which is true if the window has been closed
- **document** reference to the document object displayed in the window Through this reference the properties of a document object can be accessed.
- **frames[ ]** array of frames contained within the window
- **history** browsing history
- **location** represents the URL of the document displayed in the window. Its "href" property can be set to load a new document.
- **location-bar, scrollbars, statusbar, toolbar, personbar, menubar**
- Various bar objects.
- **name** window's string name; can be used with TARGET attribute in HTML
- **opener** window object that opened this one.
  Status line text at the bottom of the browser.
- **pageXOffset, pageYOffset**

- parent if the current window is a frame, this contains a reference to the frame of the window that contains this one.
- self self reference
- top If the current window is a frame, this contains a reference to the window object of the top-level window.
- alert(), confirm(), and prompt() methods
- alert() prints a message in a box.
- confirm() asks for "yes" or "no" kind of response and the JavaScript execution is suspended until that response is received.
- prompt() asks for some user input. Execution is suspended until a response is received.
- focus() and blur() to acquire or release keyboard focus for the window.
- moveBy() or moveTo() Used for moving a window.
- setInterval() and clearInterval() Schedule or cancel a function for periodic execution.
- setTimeout() and clearTimeout() Set or cancel execution of a function to be executed once after a specified timeout in milliseconds.
- close Close a window

Document Object Model (DOM) and JavaScript

- The elements of a document in the browser window can be accessed using JavaScript
  - getElementById
- Attributes of an element can be changed
- The text part enclosed by an element can be changed
  - Notion of “innerHTML”

   Some text of the paragraph.
- New elements can be added

Accessing and Modifying an HTML Element

Two concepts:

- Get reference to an element using:

  document.getElementById(“uniqueID”)

  returns a reference to the HTML element in the document whose id=“uniqueID”

- After getting reference to the element, one can call “setAttribute” to change the attributes of the element, or by changing the properties of the object.

innerHTML

For any HTML element the text enclosed between its start tag and end tag is the string that innerHTML on that node would return.

Example: innerHTML of div tag is in the box.

```
<div>
    <p> This is an example of innerHTML  </p>
    <a href=http://www.cs.umn.edu>
        Department Homepage
     </a>
</div>
```

Example: Set or Clear Image Border

See this example This will be helpful for your assignment

```
<head>
  <script  type="text/javascript">
    function setBorder () {
     document.getElementById("gopher").style.borderStyle = "solid";
     document.getElementById("gopher").style.borderColor = "red";
    }
  </script>
</head>
<body>
  <img id="gopher"  src="Gopher.jpg"  width="200"  onClick = "setBorder()"  />
 </form>
</body>
</html>
```

Example: Use of innerHTML

See this example

```
<head>
  <script  type="text/javascript">
    function addText () {
     now =  new Date();
     document.getElementById("para1").innerHTML = now.toString();
    }
  </script>
</head>
<body>
  <input  type="button" value="Current date/time will appear here"
onClick="addText()" />
  <div id="para1"> This is first paragraph. </div>
</body>
</html>
```

setTimeout and clearTimeout

- setTimeout
  - It is used to schedule execution of some JavaScript code after the specified number of milliseconds.

  timerEvent = setTimeout("code or function ", milliseconds)

  It returns reference to a timer event which can be used to cancel a scheduled execution using clearTimeout
- clearTimeout (timerEvent)
  - It cancels a scheduled execution.

setInterval and clearInterval

- setInterval method will automatically reschedule the execution each time the specified code is executed. Returns an id.

  setInterval (code, milliseconds)

  setInterval(func, milliseconds, args…)

Both these functions return an identifier which can be used to cancel the scheduled execution.

- clearInterval cancel it. Id is passed to cancel it.

  clearInterval(identifier)

IMAGE ANIMATION

Click here to see this program in action

```
<html>
<head>
<script type="text/javascript">
var   num = 0;
var up = 1;
var snapshots = new Array(6) ;
Var timerEvent;

function StartShow () {
  document.flower.src = snapshots[num].src ;
  num  = (num + 1) % 6;
  timerEvent = setTimeout( "StartShow()",  3000 );
}
```

IMAGE ANIMATION

```
function Init() {
   for ( var i = 0; i<6; i++ ) {
       snapshots[i] = new Image();
    }
  snapshots[0].src = "lotus-1.jpg";
  snapshots[1].src = "lotus-2.jpg";
  snapshots[2].src = "lotus-3.jpg";
  snapshots[3].src = "lotus-4.jpg";
  snapshots[4].src = "lotus-5.jpg";
  snapshots[5].src = "lotus-6.jpg";
  StartShow();
}
<body  onload=“init()” >
     <img src=“lotus-1.jpg" name=“flower" width=“100” height=“100”  />
</body> </html>
```

Example of setInterval

See Figure 13.1

```
<html xmlns = "http://www.w3.org/1999/xhtml">
   <head>
      <title>onload Event</title>
      <script type = "text/javascript">
         var seconds = 0;
         // called when the page loads to begin the timer
         function startTimer()
         {
            // 1000 milliseconds = 1 second
            window.setInterval( "updateTime()", 1000 );
         } // end function startTimer
```

Example

```
// called every 1000 ms to update the timer
        function updateTime()
        {
           ++seconds;
           document.getElementById( "soFar" ).innerHTML = seconds;
        } // end function updateTime
</script>
   </head>
   <body onload = "startTimer()">
      <p>Seconds you have spent viewing this page so far:
      <strong id = "soFar">0</strong></p>
   </body>
</html>
```

Script file: RandomPicture.js

Example Fig 10_11-12 from the book

Click on the picture - it will pick a new image randomly.

```
<html>
  <head>
    <meta charset = "utf-8">
    <title>Random Image Generator</title>
    <script src = "RandomPicture.js"></script>
  </head>

 <body>
    <img id = "image" src = "CPE.png" alt = "Common
    Programming Error">
 </body>
</html>
```

Script file: RandomPicture.js

```
var iconImg;
var pictures = [ "CPE", "EPT", "GPP", "GUI", "PERF", "PORT", "SEO" ];
var descriptions = [ "Common Programming Error",
   "Error-Prevention Tip", "Good Programming Practice",
   "Look-and-Feel Observation", "Performance Tip", "Portability Tip",
   "Software Engineering Observation" ];


// pick a random image and corresponding description then modify
// the img element in the document's body
function pickImage()
{
   var index = Math.floor( Math.random() * 7 );
   iconImg.setAttribute( "src", pictures[ index ] + ".png" );
   iconImg.setAttribute( "alt", descriptions[ index ] );
} // end function pickImage
```

Script file: RandomPicture.js

```
// registers iconImg's click event handler

function start()
{
   iconImg = document.getElementById( "image" );
   iconImg.addEventListener( "click", pickImage, false );
} // end function start

window.addEventListener( "load", start, false );
```

How to find a selected item in drop-down menu?

Using JavaScript we can find what value is selected by a user in a drop-down selection menu.

- See this example

```
<head>
  <script  type="text/javascript">
    function showSelection() {
      var selectionElement = document.getElementById("subjectSelection");
      var selectedOptionVal =
              selectionElement.options[selectionElement.selectedIndex].value;
      alert(selectedOptionVal);
    }
  </script>
</head>
```

```
<body>
  <form id = "myForm" method="GET" action="">
  Please indicate your favorite topics in computer science:   <br/> <br/>
   <select id= "subjectSelection"  name="subjects"  size="4" >
        <option  value="os"> Operating System </option>
        <option  value="internet"> Internet Programming </option>
        <option  value="compls"> Compilers </option>
        <option  value="arch"> Architecture </option>
   </select>
   <br/>
   <input type="button" value="Click to see selection " onClick =
                                                          "showSelection()"  />
 </form>
</body>
```