CUI Abbottabad

Department of Computer Science

Web Technologies

Lecture -11-[CLO-2]
Javascript-Part5

COMSATS University Islamabad, Abbottabad Campus

Agenda

- Random image generation without Array
- GUI Event Handling
  - Event occurring
  - Event Capturing
  - Event Handling
- Scope Rules
  - Global Scope
  - Local Scope
- Arrays in Javascript
  - Creating and initializing Arrays
  - Passing array to functions
  - Searching in array
  - Use of indexOf(),lastIndexOf() and join()
  - Random Image Generation using Arrays

WHAT YOU HAVE LEARNED SO FAR?

- Until now, all user interactions with scripts have been done so far through:
  - either a prompt dialog (in which the user types an input value for the program)
  - or an alert dialog (in which a message is displayed to the user, and the user can click **OK** to dismiss the dialog).

Limitations :

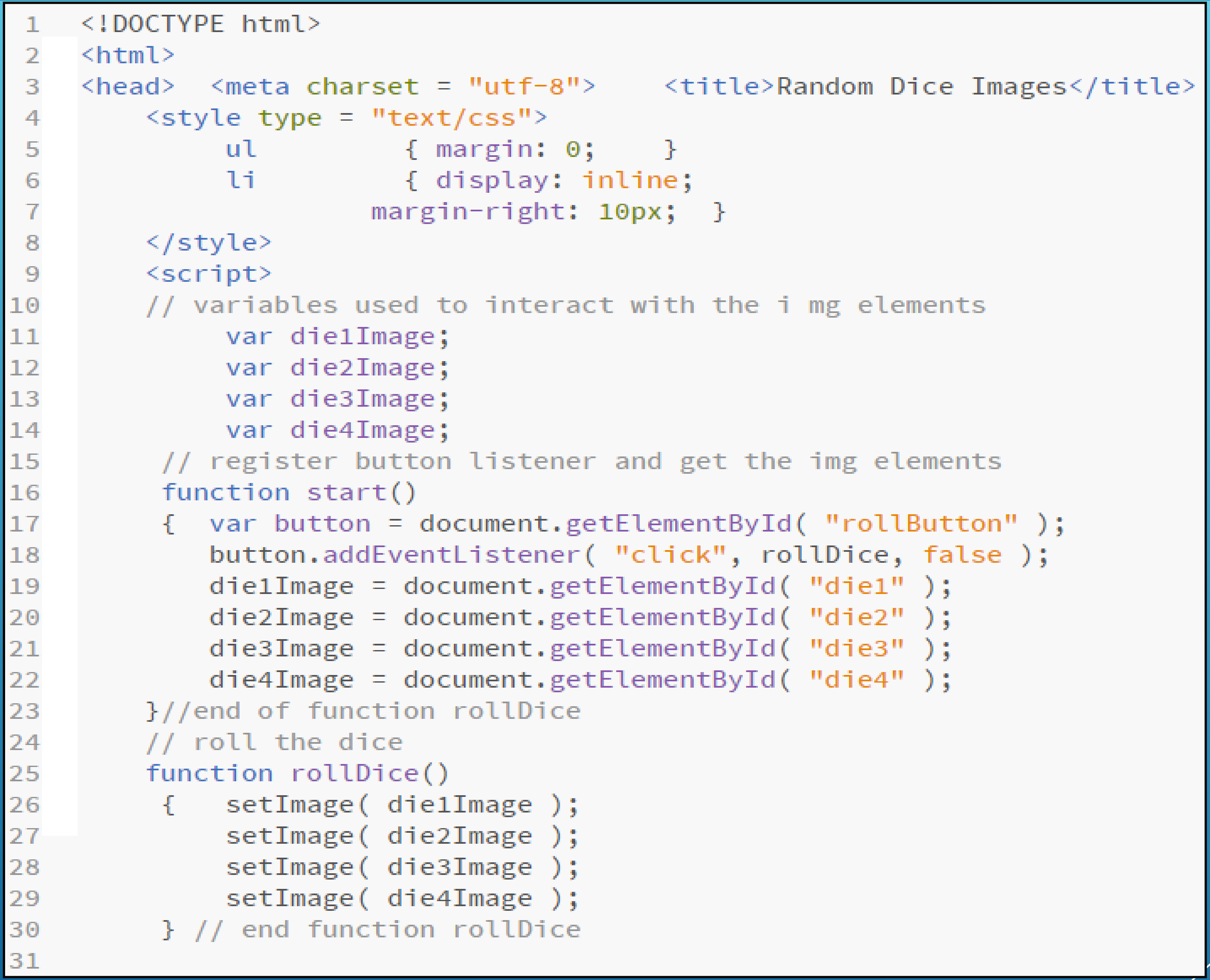
- Although these dialogs are valid ways to receive input from a user and to display messages, they're fairly limited in their capabilities.
  - A prompt dialog can obtain only one value at a time from the user,
  - a message dialog can display only one message.
  - Inputs are typically received from the user via an HTML5 form (for example one in which the user enters name and address information).

EVENT HANDLING IN RESPONSE TO USER INTERACTION WITH FORM ELEMENTS

Example-11.1--------------Random Images Generation

- Now we are going to begin our introduction to more elaborative user interfaces, this program uses:
    - an HTML5 form
    - graphical user interface concept—GUI **event handling**.
- This is our first example in which the JavaScript executes in response to the user's interaction with an element in a form.
- This interaction causes an ***event***. Scripts are often used to respond to user initiated events.
- In the next example Listing-11-1, we build a random image generator—a script that displays four randomly selected die images every time the user clicks a Roll Dice button on the page.
- For our convenience we put source script listing-11-1-RandomeImage.html and related seven image files (blank.png, die1.png, die2.png, die3.png, die4.png,die5.png and die6.png) in the same folder/directory .

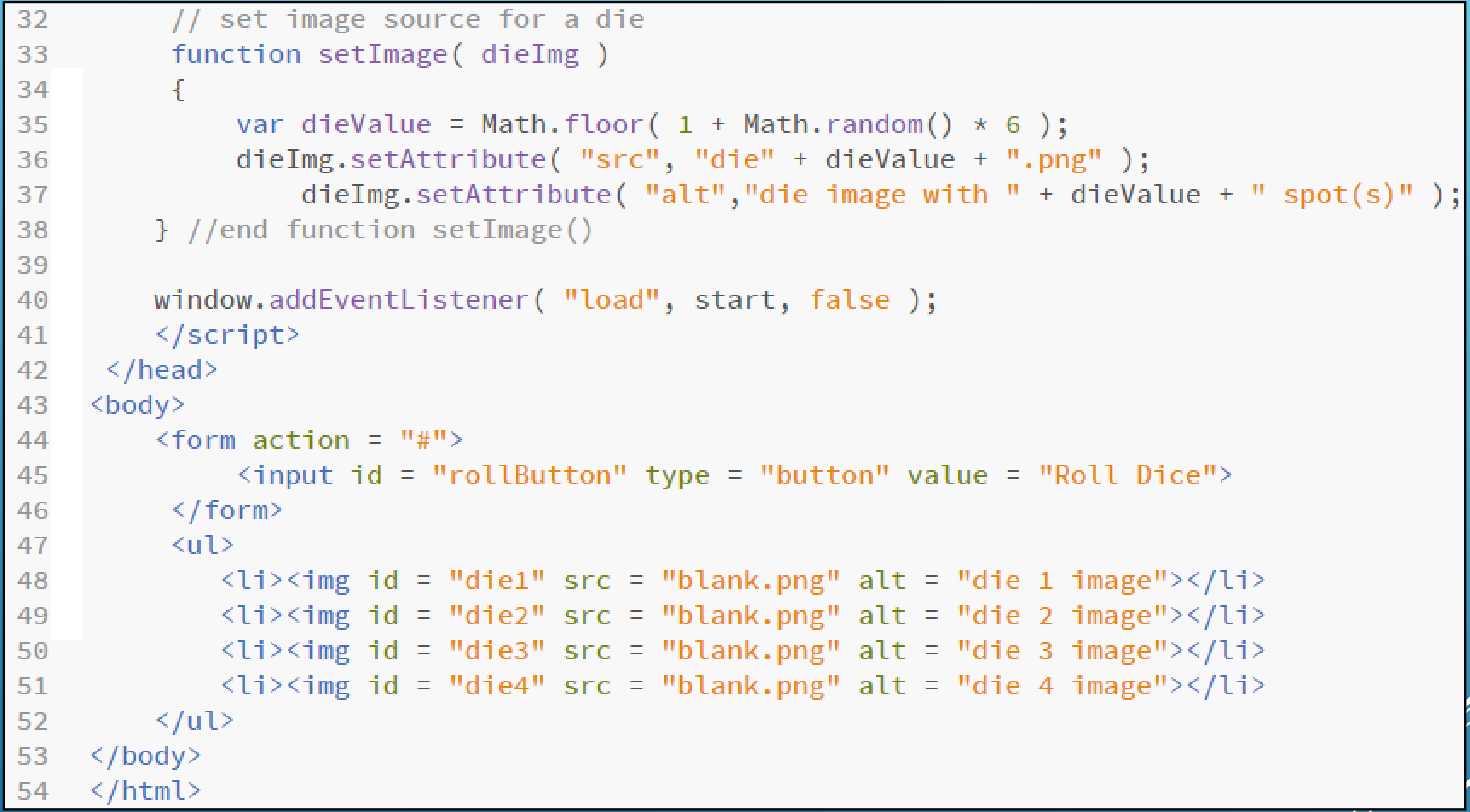
RANDOM IMAGES GENERATION AND EVENT HANDLING



```
<!DOCTYPE html>
<html>
<head>  <meta charset = "utf-8">    <title>Random Dice Images</title>
    <style type = "text/css">
         ul          { margin: 0;    }
         li          { display: inline;
                  margin-right: 10px;   }
    </style>
    <script>
    // variables used to interact with the i mg elements
         var die1Image;
         var die2Image;
         var die3Image;
         var die4Image;
     // register button listener and get the img elements
     function start()
     {   var button = document.getElementById( "rollButton" );
        button.addEventListener( "click", rollDice, false );
        die1Image = document.getElementById( "die1" );
        die2Image = document.getElementById( "die2" );
        die3Image = document.getElementById( "die3" );
        die4Image = document.getElementById( "die4" );
    }//end of function rollDice
    // roll the dice
    function rollDice()
     {    setImage( die1Image );
          setImage( die2Image );
          setImage( die3Image );
          setImage( die4Image );
     } // end function rollDice

```

Listing-11-1
RandomImage.html

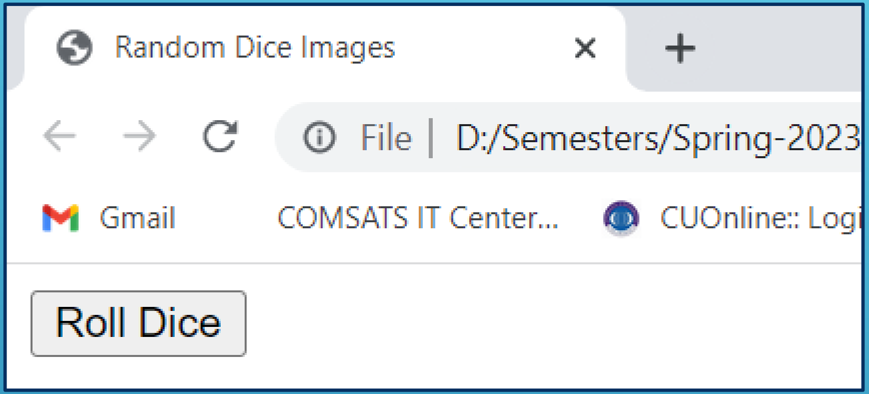
RANDOM IMAGES GENERATION AND EVENT HANDLING



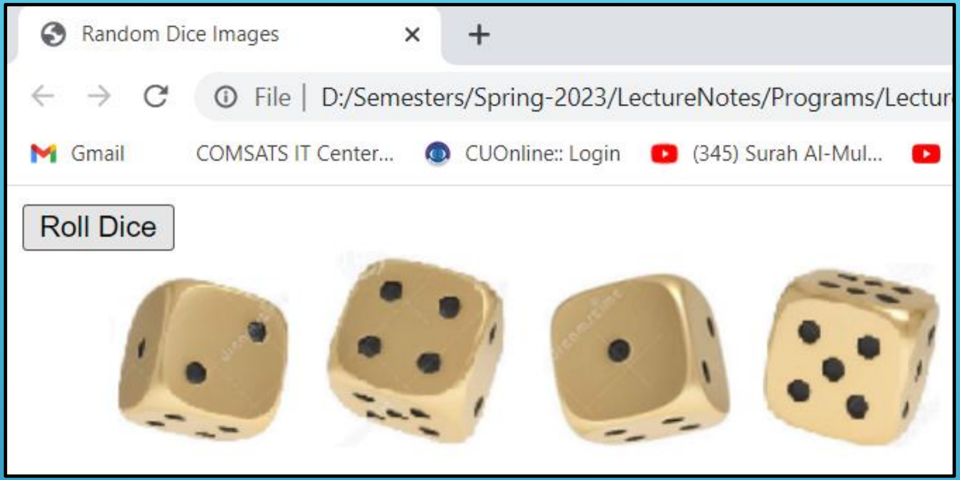
```
     // set image source for a die
     function setImage( dieImg )
     {
         var dieValue = Math.floor( 1 + Math.random() * 6 );
         dieImg.setAttribute( "src", "die" + dieValue + ".png" );
             dieImg.setAttribute( "alt","die image with " + dieValue + " spot(s)" );
    } //end function setImage()

    window.addEventListener( "load", start, false );
    </script>
 </head>
<body>
    <form action = "#">
         <input id = "rollButton" type = "button" value = "Roll Dice">
     </form>
     <ul>
        <li><img id = "die1" src = "blank.png" alt = "die 1 image"></li>
        <li><img id = "die2" src = "blank.png" alt = "die 2 image"></li>
        <li><img id = "die3" src = "blank.png" alt = "die 3 image"></li>
        <li><img id = "die4" src = "blank.png" alt = "die 4 image"></li>
    </ul>
</body>
</html>
```

Listing-11-1-RandomImage.html

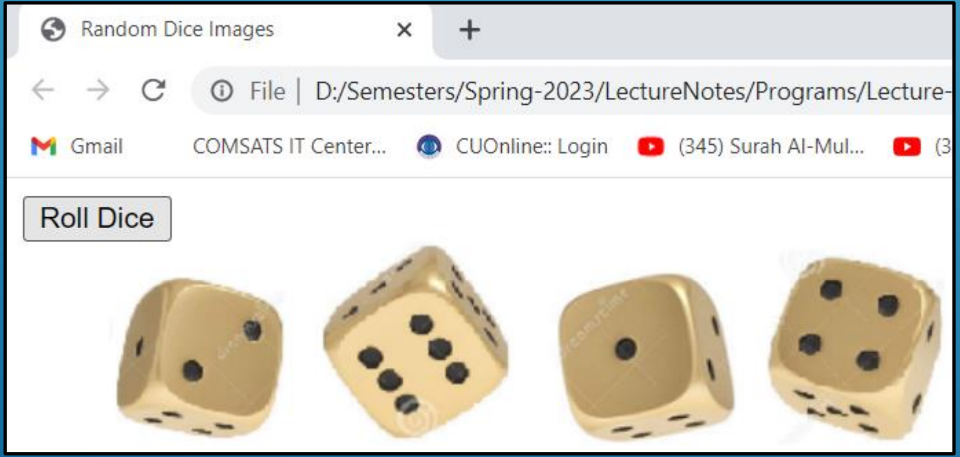
RANDOM IMAGES GENERATION AND EVENT HANDLING



Output-11-1-RandomImage.html(a)



Output-11-1-RandomImage .html(b)



Output-11.1-RandomImage .html(c)

RANDOM IMAGES GENERATION AND EVENT HANDLING

Example-11-1--------------Random Images Generation

- Body Section:
- Line#44--The HTML5 standard requires that every **form** contain an **action** attribute, but because this form does not post its information to a web server, the string **"#"** is used simply to allow this document to validate. The **#** symbol by itself represents the current page.
- A form which, has only one button with id "rollbutton"—line#45-46
- An ordered list which contains 4 images items, each has different id, die1,die2,die3,die4,die5 and die6, when the page open in browser at that time only blank.png will be displayed in all four image places. Line#47-line#52
- Head section:
  - **CSS** (line#4-Line#8) style for ul and li , note **display:inline** property set for li, it means list items will be displayed in one line not in separate lines
  - **JavaScript** (line#9-line#41)
    - Global variables declaration for four images(line#11-line#14)
    - Function start()---line#16-line#23
    - Function rollDice()---line#25-line#30
    - Function setImage()—line#33-line#38
    - Window.addEventListener()---line#40

RANDOM IMAGES GENERATION AND EVENT HANDLING

Program flow Explanation -Example-11-1----Random Images Generation

1. When the page opens in the browser it will display only one button and four image places with blank image
2. As the page loads in browser(load event) first the four global variables declared and then the line#40 executed:

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

The above statement execute before any function call because it is not written inside any function definition. This statement is calling addEventListener() method of Window object. Note that Whenever we want that javascript listen the event we call this function.

- From this point forward, many of our examples will execute a JavaScript function when the document finishes loading in the web browser window.
- This is accomplished by handling the window object’s **load event**.
- To specify the function to call when an event occurs, it means you are **registering an event handler** for that event.
- We register the window’s load event handler at line 40.
- Method **addEventListener ()** is available for every DOM node.

RANDOM IMAGES GENERATION AND EVENT HANDLING

- The method takes three arguments:
  - the first is the name of the event for which we’re registering a handler
  - the second is the function that will be called to handle the event
  - the last argument is optional and by default it is false and used for Event Bubbling propagation—the true is used for Event Capturing Propagation. Here true-value is beyond current scope

  (more detail of Event Bubbling and Event Capture is given at end of lecture)
- Line#40 indicates that function start (lines 16–23) should execute as soon as the page finishes loading.

3. Function start()

- When the window’s load event occurs, function start registers the **Roll Dice** button’s click event handler (lines# 17–18), which instructs the browser to **listen for events** (click events in particular).
- If no event handler is specified for the **Roll Dice** button, the script will not respond when the user presses the button.

- Line# 17 uses the document object’s:
  - **getElementById method**, which, given an HTML5 element’s id as an argument, finds the element with the matching id attribute and returns a JavaScript object representing the element.

RANDOM IMAGES GENERATION AND EVENT HANDLING

- This object allows the script to programmatically interact with the corresponding element in the web page.
- For example, line#18 uses the object representing the button to call function addEventListener—in this case, to indicate that function rollDice should be called when the button's click event occurs.
- Lines#19–22 get the objects representing the four img elements in lines# 48-51 and assign them to the script variables in declared in lines 11–14.

4. Function rollDice()

- The user clicks the **Roll Dice** button to roll the dice. This event invokes function rollDice (lines 25–30) in the script.
- Function rollDice takes no arguments, so it has an empty parameter list.
- Lines 26–29 call function setImage (lines 33–38) to randomly select and setthe image for a specified img element.

5. Function setImage()

- Function setImage (lines 33–38) receives one parameter (dieImg) that represents the specific img element in which to display a randomly selected image.
- Line 35 picks a random integer from 1 to 6.
- Line 36 demonstrates how to access an img element's src attribute programmatically in JavaScript.

RANDOM IMAGES GENERATION AND EVENT HANDLING

5. Function setimage….

- Each JavaScript object that represents an element of the HTML5 document has a **setAttribute** method that allows you to change the values of most of the HTML5 element's attributes.
- In this case, we change the src attribute of the img element referred to by dieImg.
    - The src attribute specifies the location of the image to display. We set the src to a concatenated string containing the word "die", a randomly generated integer from 1 to 6 and the file extension ".png" to complete the image file name. Thus, the script dynamically sets the img element's src attribute to the name of one of the image files in the current directory.

6. Continuing to Roll the Dice

- The program then waits for the user to click the **Roll Dice** button again.
- Each time the user does so, the program calls rollDice, which repeatedly calls setImage to display new die images.

SCOPE RULES

- The **scope** of an identifier(name) for a variable or function is the portion of the program in which the identifier can be referenced.
- Global variable or Script level variables are those variables which do not declare inside any function parameter list or inside the definition block{} of a function

Global Scope:

- Those **Global variables** or **script-level variables** that are declared in the **head element** are accessible in *any* part of a script and are said to have global scope. Thus every function in the page's script(s) can potentially use the variables.

Local Scope:

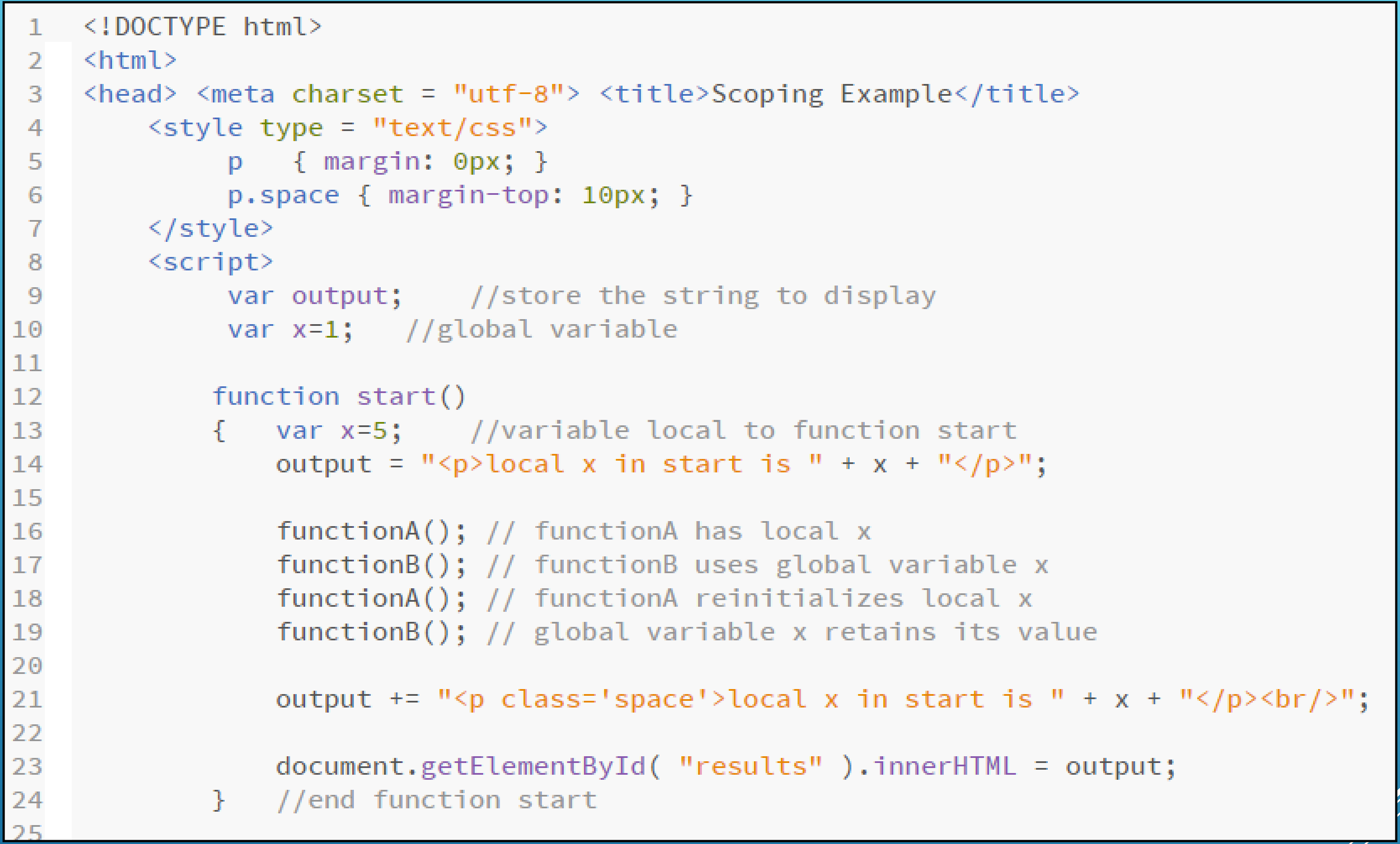
- Identifiers declared inside a function have **function** (or **local**) **scope** and can be used only in that function.
- Function scope begins with the opening left brace ({) of the function in which the identifier is declared and ends at the function's terminating right brace (}).
- Local variables of a function and function parameters have function scope.

Note: If a local variable in a function has the same name as a global variable, the global variable is "hidden" from the body of the function.

SCOPE RULE-EXAMPLE PROGRAM

- The script in Listing 11-2 demonstrates the **scope rules** that resolve conflicts between global variables and local variables of the same name.
- Once again, we use the window’s load event (line#46), which calls the function start when the HTML5 document is completely loaded into the browser window.
- In this example, we build an output string (declared at line# 9) that is displayed at the end of function start’s execution.

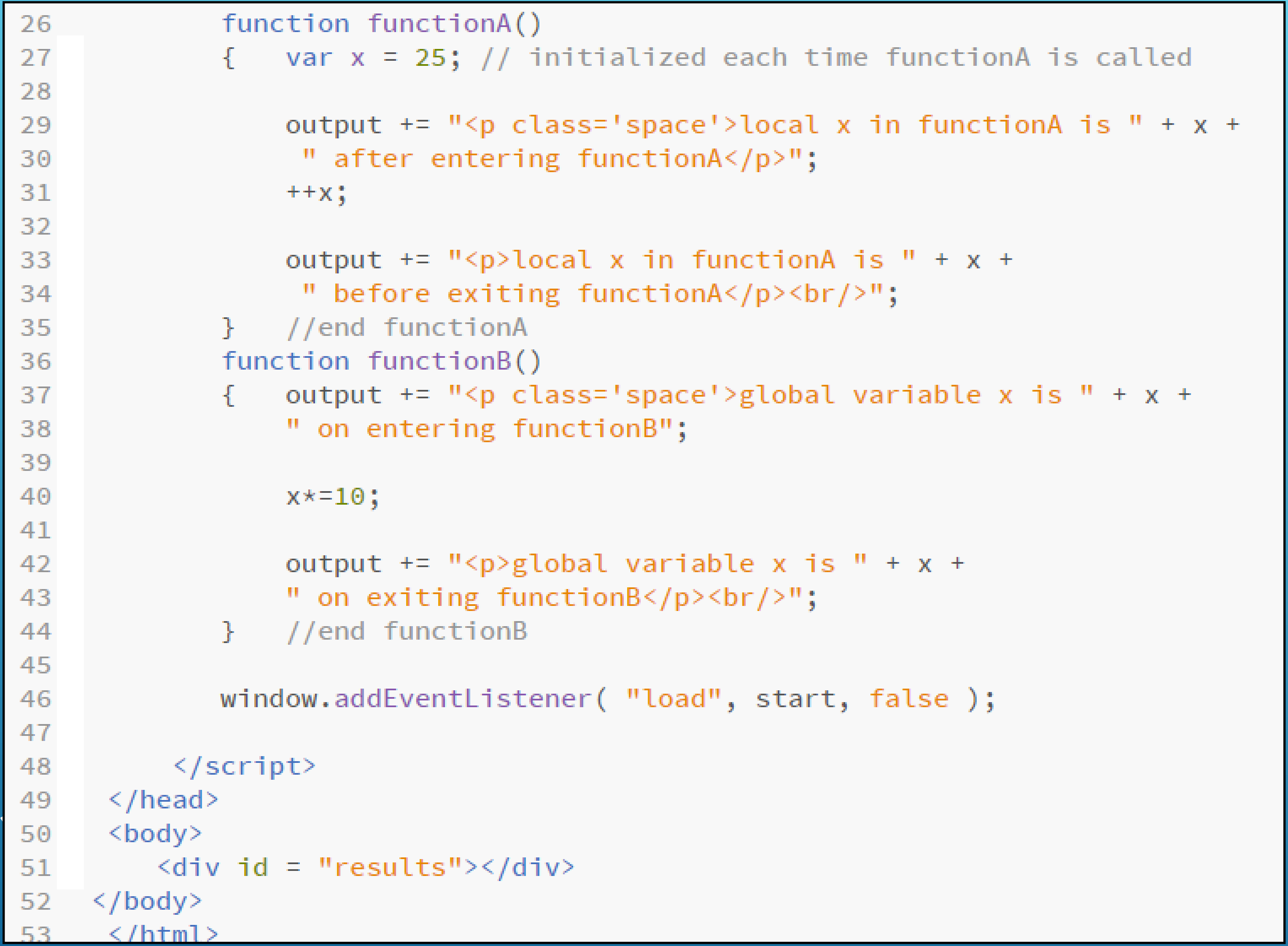
SCOPE RULE-EXAMPLE PROGRAM



```
<!DOCTYPE html>
<html>
<head> <meta charset = "utf-8"> <title>Scoping Example</title>
    <style type = "text/css">
         p   { margin: 0px; }
         p.space { margin-top: 10px; }
    </style>
    <script>
         var output;    //store the string to display
         var x=1;   //global variable

        function start()
        {   var x=5;    //variable local to function start
            output = "<p>local x in start is " + x + "</p>";

            functionA(); // functionA has local x
            functionB(); // functionB uses global variable x
            functionA(); // functionA reinitializes local x
            functionB(); // global variable x retains its value

            output += "<p class='space'>local x in start is " + x + "</p><br/>";

            document.getElementById( "results" ).innerHTML = output;
        }   //end function start
```

Listing-11-2-ScopeRules.html

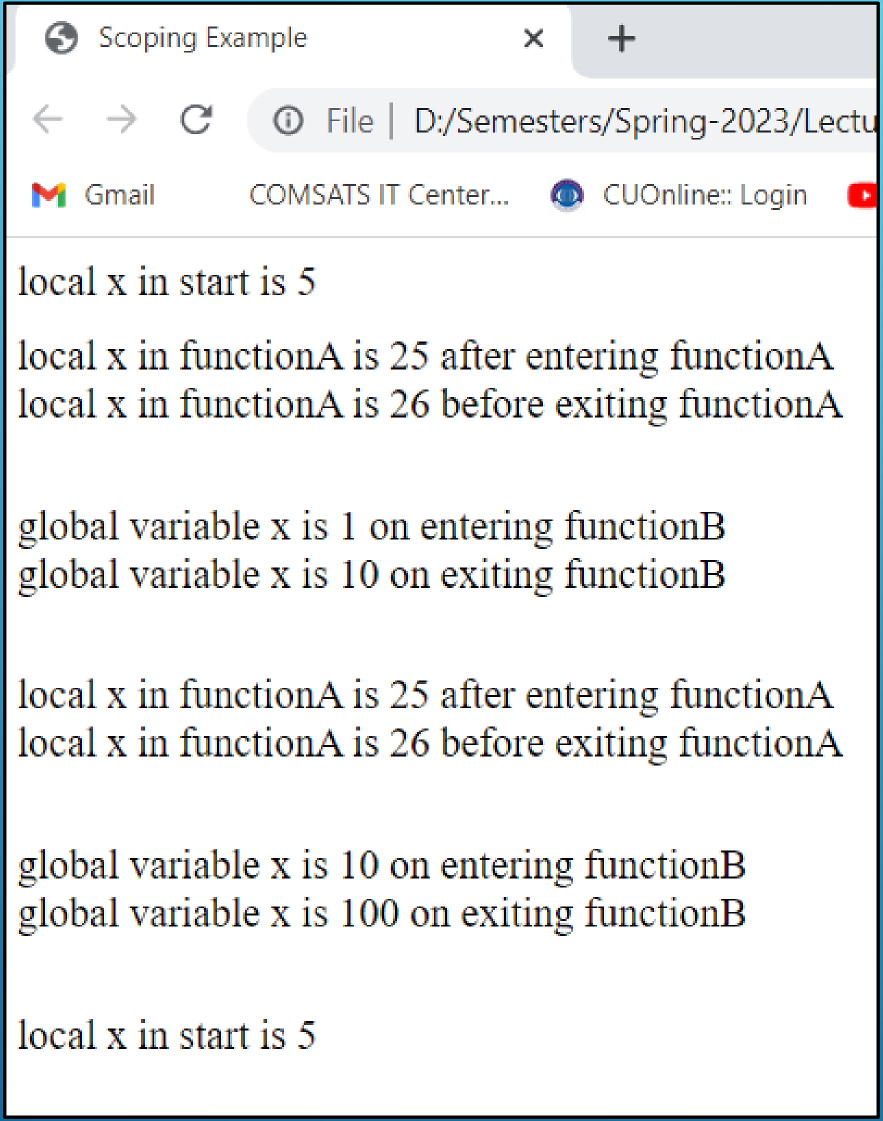
SCOPE RULE-EXAMPLE PROGRAM



```
        function functionA()
        {   var x = 25; // initialized each time functionA is called

            output += "<p class='space'>local x in functionA is " + x +
             " after entering functionA</p>";
            ++x;

            output += "<p>local x in functionA is " + x +
             " before exiting functionA</p><br/>";
        }   //end functionA
        function functionB()
        {   output += "<p class='space'>global variable x is " + x +
            " on entering functionB";

            x*=10;

            output += "<p>global variable x is " + x +
            " on exiting functionB</p><br/>";
        }   //end functionB

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

     </script>
  </head>
  <body>
     <div id = "results"></div>
 </body>
  </html>
```

Listing-11-2

SCOPE RULE-EXAMPLE PROGRAM



```
local x in start is 5

local x in functionA is 25 after entering functionA
local x in functionA is 26 before exiting functionA


global variable x is 1 on entering functionB
global variable x is 10 on exiting functionB


local x in functionA is 25 after entering functionA
local x in functionA is 26 before exiting functionA


global variable x is 10 on entering functionB
global variable x is 100 on exiting functionB


local x in start is 5
```

SCOPE RULE-EXAMPLE PROGRAM

Explanation-Listing-11-2(line#8-line#48)

- The script has declared one global variable x (line#10) and initialized to 1. Note if any function has variable with same name then inside that function this global variable will be hidden.
- The script has declared one global string variable output(line#9)
- we use the window's load event (line#46), which calls the function **start** when the HTML5 document is completely loaded into the browser window.
- **Function start---lines(12-24):**
  - declares a local variable x (line 13) and initializes it to 5.
  - Line 14 creates a paragraph element containing x's value as a string and assigns the string to the global variable output.
  - From inside start function:
    - 2 calls to functionA line#16 and line#18
    - 2 calls to functionB line#17 and line#19
  - This output string is updated at line#21 and then Finally the output displayed at line#23.
  - In the sample output, this shows that the global variable x is *hidden* in start.

SCOPE RULE-EXAMPLE PROGRAM

Function functionA() --lines(26-35):

- Defines local variable x (line 27) and initializes it to 25.
- When functionA() is called, the variable's value is placed in a paragraph element and appended to variable output to show that the global variable x is *hidden* in functionA();
- then the variable is incremented(line 31)
- and appended to output (line 33) again before the function exits.
- **Each time this function is called, local variable x is re-created and initialized to 25.**

Function functionB() --lines (36-44):

- Does not declare any variables. Therefore, when it refers to variable x, the global variable x is used.
- When functionB() is called, the global variable's value is placed in a paragraph element and appended to variable output (line#37)
- then it's multiplied by 10 (line#40)
- and appended to variable output (line#42) again before the function exits.

- The next time function functionB() is called, the global variable has its modified value, 10, which again gets multiplied by 10 and 100 is output.

EVENT BUBBLING AND EVENT CAPTURING

- There are two ways of event propagation in the HTML DOM, bubbling and capturing.
- Event propagation is a way of defining the element order when an event occurs. If you have a <p> element inside a <div> element, and the user clicks on the <p> element, which element's "click" event should be handled first?
- In **bubbling** the inner most element's event is handled first and then the outer: the <p> element's click event is handled first, then the <div> element's click event.
- In **capturing** the outer most element's event is handled first and then the inner: the <div> element's click event will be handled first, then the <p> element's click event.
- With the addEventListener() method you can specify the propagation type by using the "useCapture" parameter:

addEventListener(*event*, *function*, *useCapture*);

- The default value is false, which will use the bubbling propagation, when the value is set to true, the event uses the capturing propagation.

https://javascript.info/bubbling-and-capturing

ARRAYS- INTRODUCTION AND DECLARATION

- In JavaScript arrays:
  - An array is a group of memory locations that all have the same name and normally are of the same type **(although this property is not required in JavaScript)**
  - Each individual location is called an index
  - “dynamic” entities that can change size after they are created
- Arrays occupies space in memory, Truly speaking, an array in JavaScript is an **Array object**.
- You use the **new operator** to create an array and to specify the number of elements in an array.
- The new operator creates an object as the script executes by obtaining enough memory to store an object of the type specified to the right of new .
- To allocate 12 elements for integer array c, use a new expression like:

```
var c = new Array( 12 );//an object of Array class
```

- The preceding statement can also be performed in two steps, as follows:

```
var c; // declares a variable that will hold the array
c = new Array( 12 ); // allocates the array
```

- When arrays are created as above mentioned way(only declaration and allocating size), in this way the elements are *not* initialized yet—they have undefined value.

EXAMPLE- DECLARATION AND ALLOCATING DATA TO ARRAYS

Our next example consists of three Listings. 11-3, Listing-11-4 and Listing-11-5).

Listing-11-3:

- It is CSS file named tableStyle.css for the creation of table

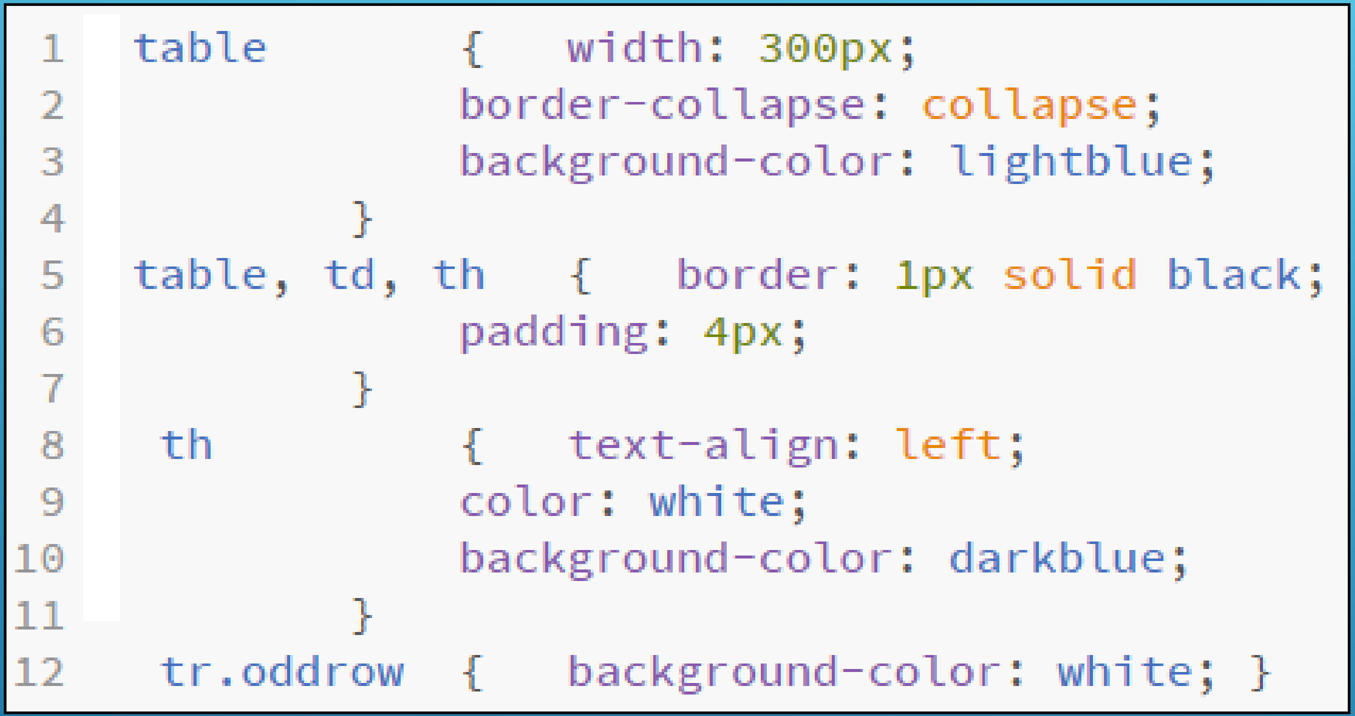
Listing-11-4:

- It is an HTML5 document in which we embedded
  - External CSS file named tableStyle.css file in <link> element—line#4
  - external javaScript file named myArray1.js(Listing-11-5) in <script>-line#6
    - The external JavaScript file will return two arrays in the form of tables and will be displayed in two <div> tags with id “output1” and “output2”---line#9 and line#10

Listing-11-5(Javascript file):

- uses operator new to allocate an array n1 with size 5--line#2
- uses operator new to allocate an empty array n2--line#3
- The script demonstrates initializing an array of existing elements line#7 to line#8 and also shows that an array can grow dynamically to accommodate new elements line#10 and line#11.

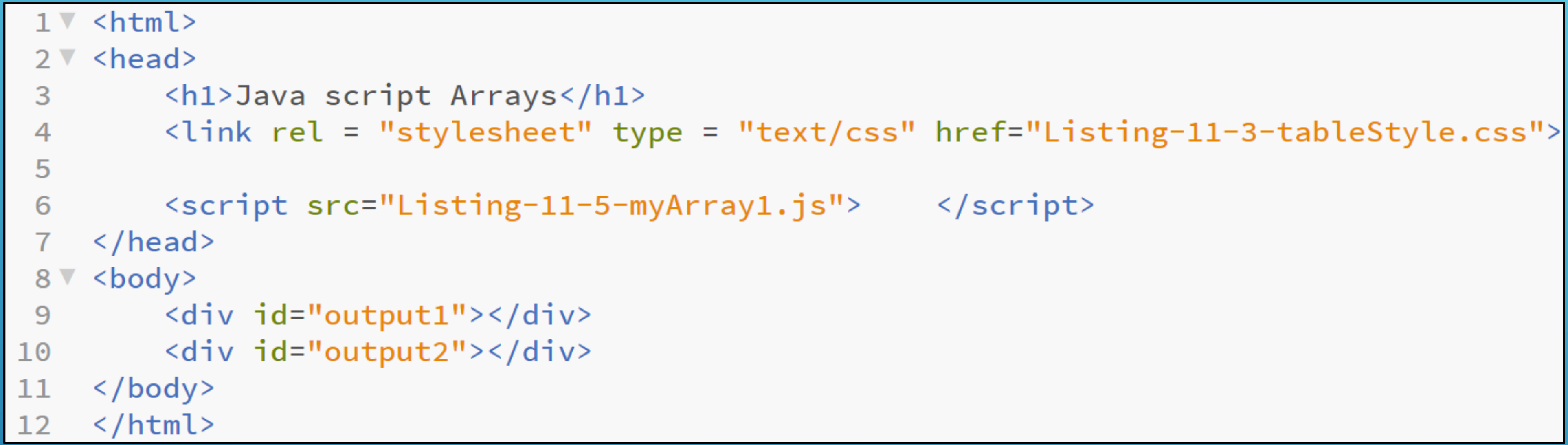
THE TABLESTYLE.CSS FILE



```css
table         {   width: 300px;
                  border-collapse: collapse;
                  background-color: lightblue;
              }
table, td, th     {   border: 1px solid black;
                  padding: 4px;
              }
 th           {   text-align: left;
                  color: white;
                  background-color: darkblue;
              }
 tr.oddrow    {   background-color: white; }
```

Listing-11-3-tableStyle.css

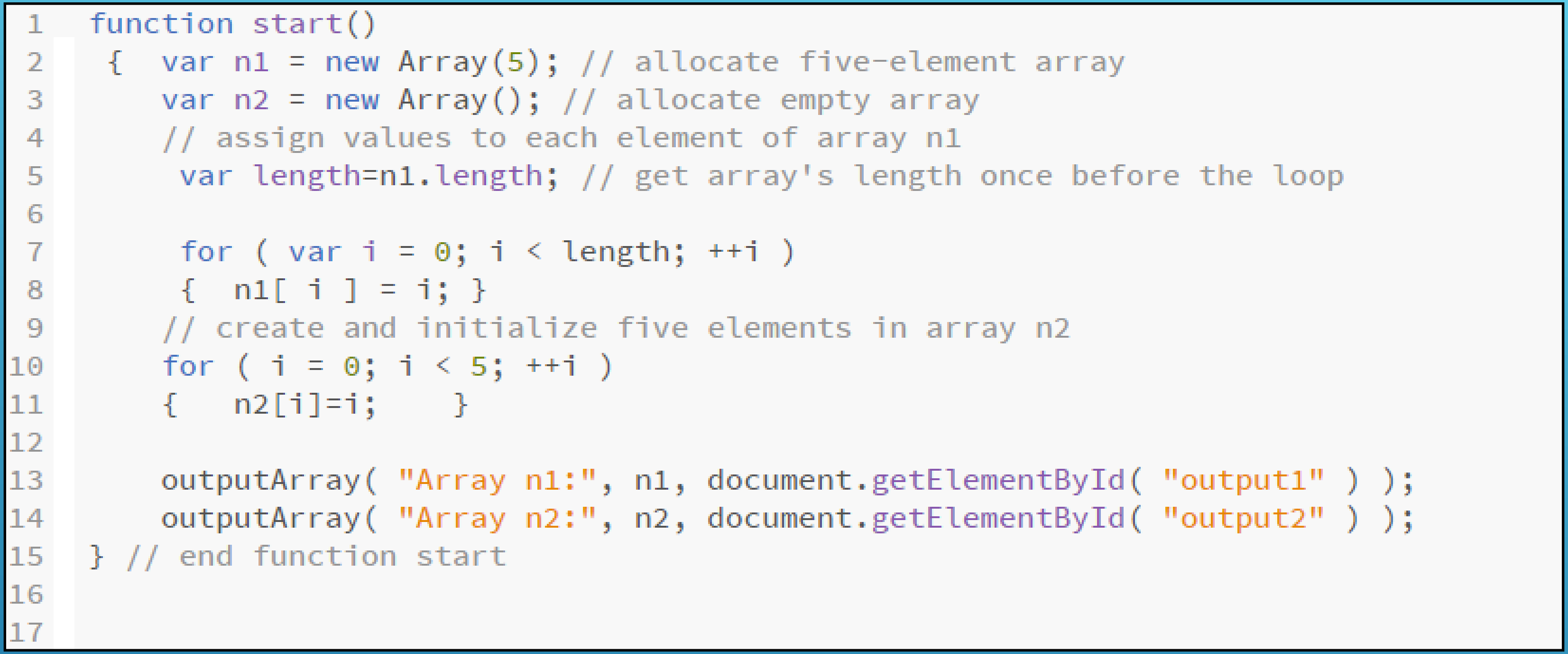
THE HTML FILE



```
<html>
<head>
    <h1>Java script Arrays</h1>
    <link rel = "stylesheet" type = "text/css" href="Listing-11-3-tableStyle.css">

    <script src="Listing-11-5-myArray1.js">    </script>
</head>
<body>
    <div id="output1"></div>
    <div id="output2"></div>
</body>
</html>
```

Listing-11-4.html

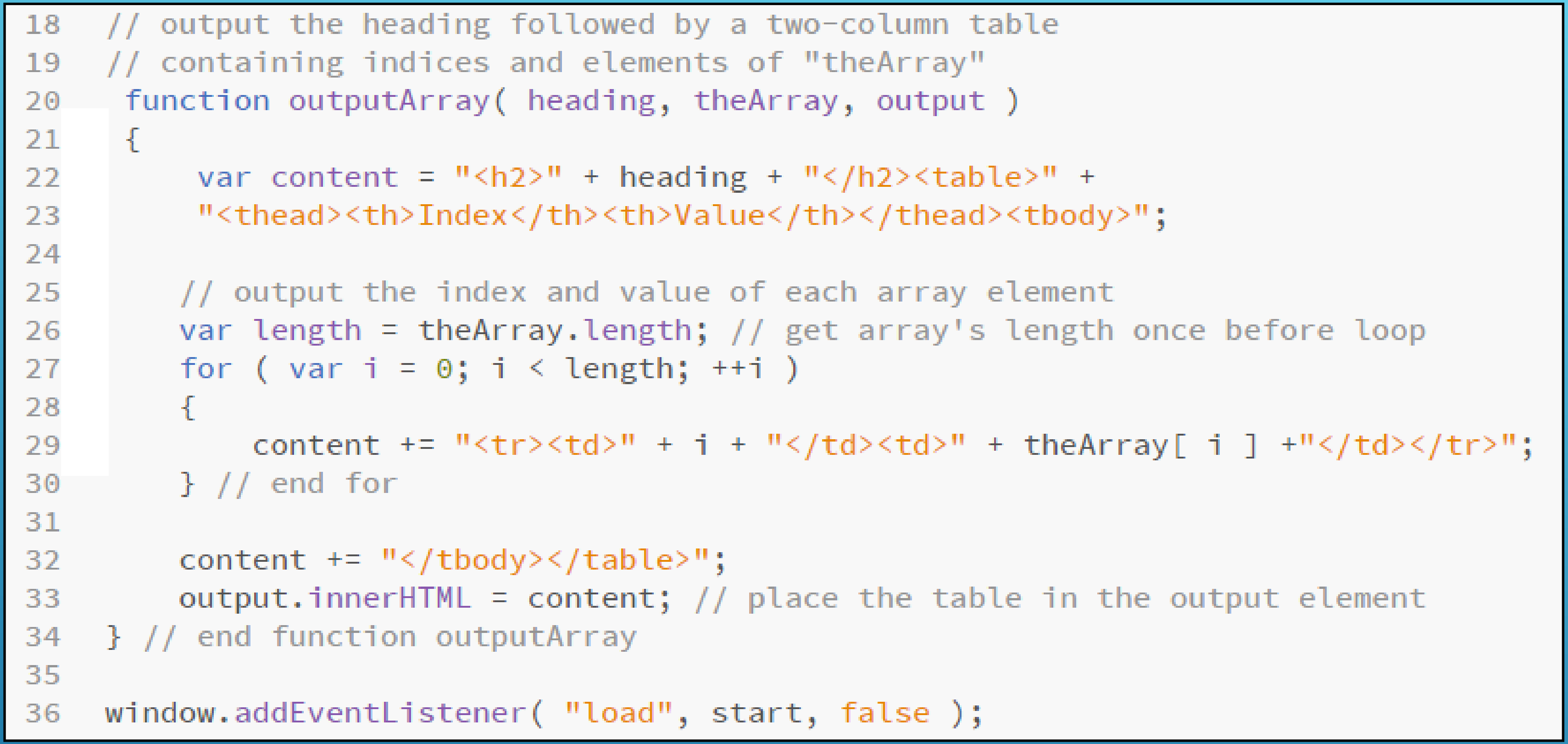
THE JAVA SCRIPT FILE



```javascript
function start()
 {  var n1 = new Array(5); // allocate five-element array
    var n2 = new Array(); // allocate empty array
    // assign values to each element of array n1
     var length=n1.length; // get array's length once before the loop

     for ( var i = 0; i < length; ++i )
     {  n1[ i ] = i; }
    // create and initialize five elements in array n2
    for ( i = 0; i < 5; ++i )
    {   n2[i]=i;    }

    outputArray( "Array n1:", n1, document.getElementById( "output1" ) );
    outputArray( "Array n2:", n2, document.getElementById( "output2" ) );
} // end function start
```

Listing-11-5-myArray1.js

THE JAVA SCRIPT FILE

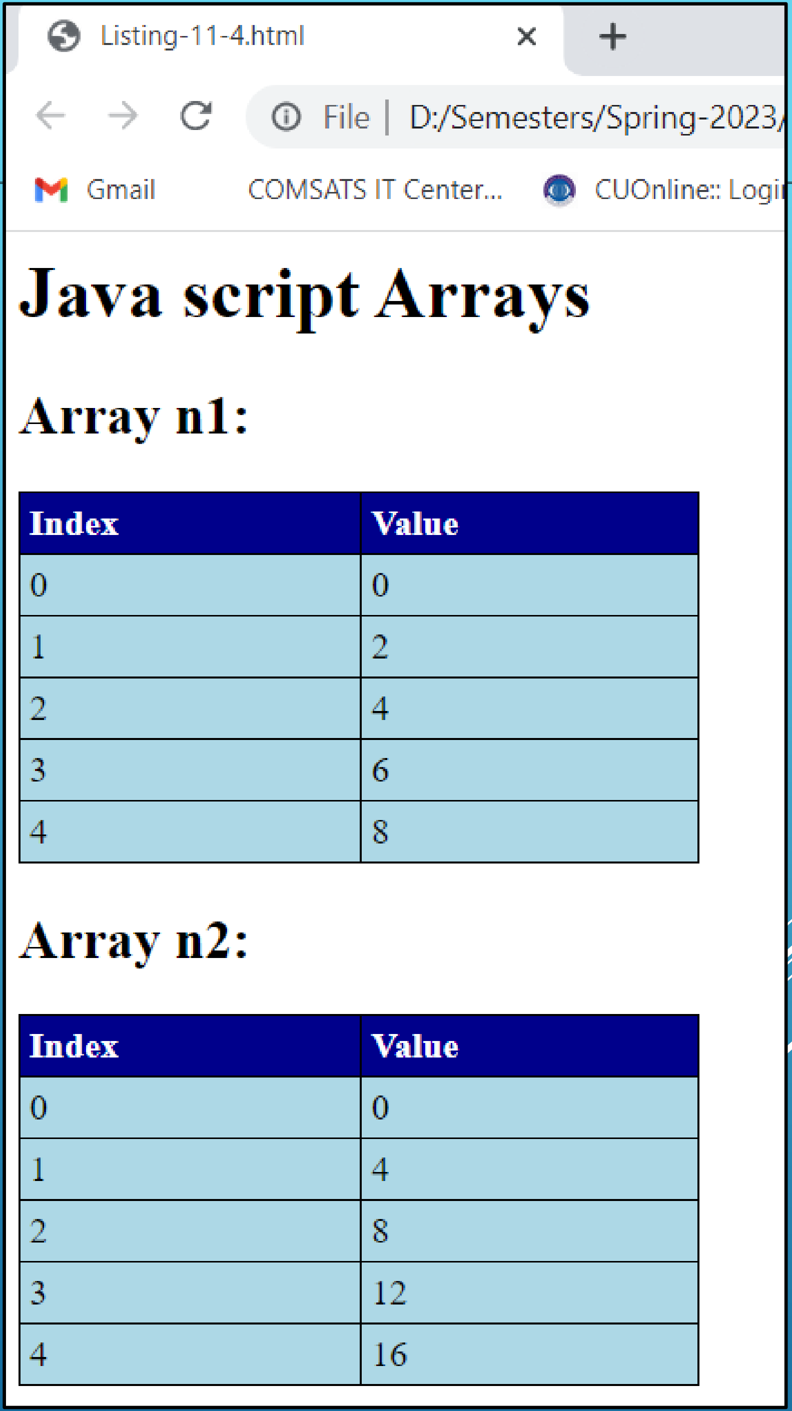


```
// output the heading followed by a two-column table
// containing indices and elements of "theArray"
 function outputArray( heading, theArray, output )
 {
     var content = "<h2>" + heading + "</h2><table>" +
     "<thead><th>Index</th><th>Value</th></thead><tbody>";

    // output the index and value of each array element
    var length = theArray.length; // get array's length once before loop
    for ( var i = 0; i < length; ++i )
    {
        content += "<tr><td>" + i + "</td><td>" + theArray[ i ] +"</td></tr>";
    } // end for

    content += "</tbody></table>";
    output.innerHTML = content; // place the table in the output element
} // end function outputArray

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

Listing-11-5-myArray1.js

THE OUTPUT LISTING-11-4

Output-Listing-11-4.html



Java script Arrays

Array n1:

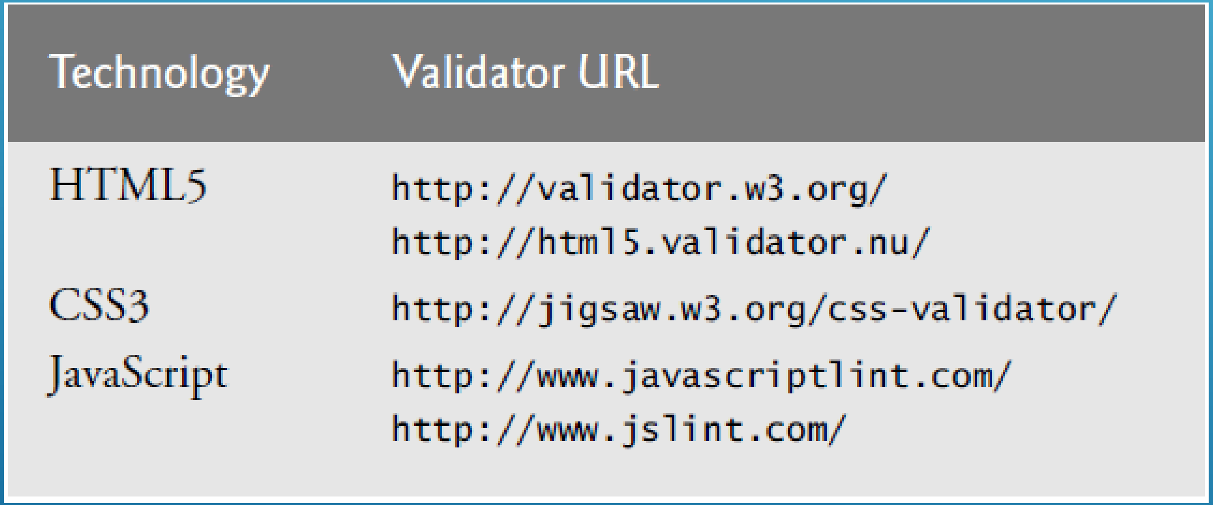
Index	Value
0	0
1	2
2	4
3	6
4	8

Array n2:

Index	Value
0	0
1	4
2	8
3	12
4	16

VALIDATION OF HTML, CSS AND JAVASCRIPT CODE

- As you’ll see, JavaScript programs typically have HTML5 and CSS3 portions as well.
- You must use proper HTML5, CSS3 and JavaScript syntax to ensure that browsers process your documents properly.
- Following figure lists the validators we used to validate the code



Technology	Validator URL
HTML5	`http://validator.w3.org/` `http://html5.validator.nu/`
CSS3	`http://jigsaw.w3.org/css-validator/`
JavaScript	`http://www.javascriptlint.com/` `http://www.jslint.com/`

EXPLANATION OF OUTPUT CREATED BY LISTING-11-5 AND SENT TO LISTING-11-4

- As the html document opened in browser, it called myArray1.js file(Listing-11-5)
- Inside myArray1.js file
    - when the window's load event occurs at line#36 then it called start() function
    - Inside start()
        - Line#2 creates array n1 with five elements.
        - Line#3 creates array n2 as an *empty* array.
        - Lines# 7–8 use a for statement to initialize the elements of n1 to their index values (0 to 4). With arrays, we use zero-based counting so that the loop can access *every* array element.
        - Line#5 uses the expression n1.length to determine the array's length. JavaScript's arrays are dynamically resizable, so it's important to get an array's length once before a loop that processes the array—in case the script changes the array's length. In this example, the array's length is 5, so the loop continues executing as long as the value of control variable i is less than 5 .
        - This process is known as **iterating through the array's elements**. For a five element array, the index values are 0 through 4, so using the less-than operator, <, guarantees that the loop does not attempt to access an element beyond the end of the array.
        - Zero based counting is usually used to iterate through arrays.

EXPLANATION OF OUTPUT CREATED BY LISTING-11-5 AND SENT TO LISTING-11-4

Growing an Array Dynamically

  - Line# 10–11 use a for statement to add five elements to the array n2 and initialize each element to its index value (0 to 4).
  - The array grows dynamically to accommodate each value as it's assigned to each element of the array.
- Line# 13–14 invoke function outputArray (defined in line# 20–34) to display the contents of each array in an HTML5 table in a corresponding div.
- Function outputArray()
  - **receives three arguments:**
    - a string to be output as an h2 element before the HTML5 table that displays the contents of the array,
    - the array to output and
    - the div in which to place the table.
  - Line# 27-30 use a for statement to define each row of the table.

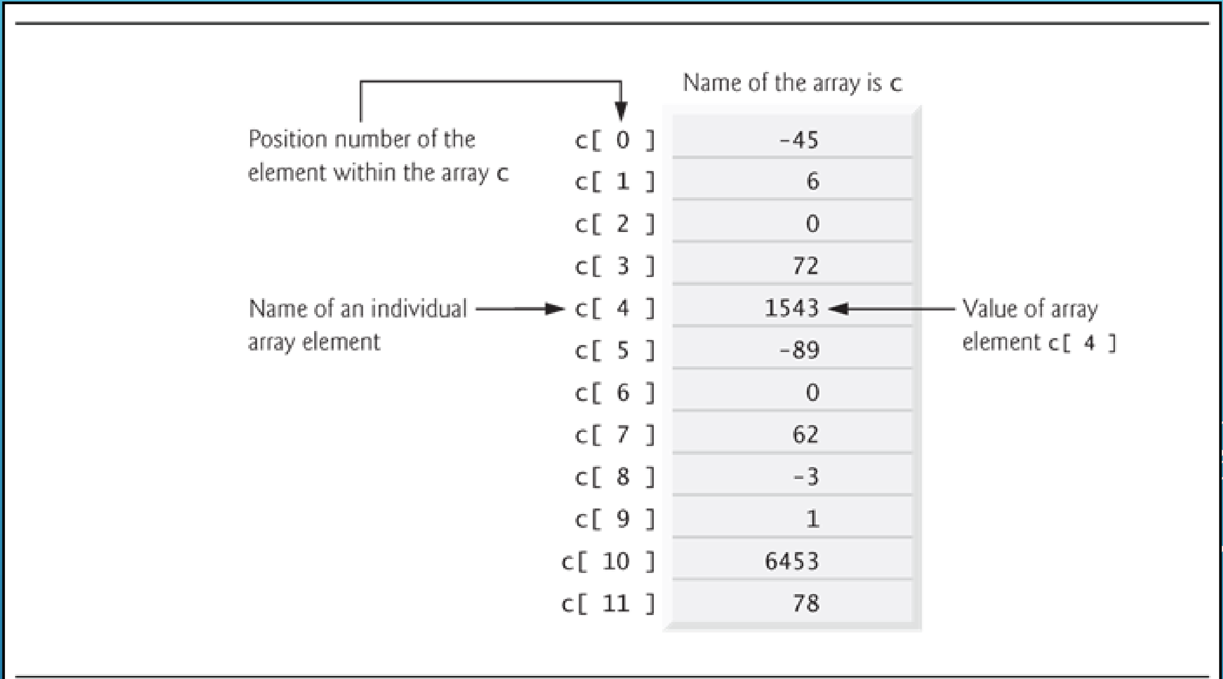
ARRAY INITIALIZATION

Initialization of array

- Syntax

  var *array_name* = [*item1*, *item2*, ...];

  - array_name[0] will refer to item1,
  - array_name[1] will refer to item2



Array with 12 elements.

- Figure shows an array of integer values names c.

  var c=[-45, 6, 0, 72,1543, -89, 0, 62, -3, 1, 6453, 78];

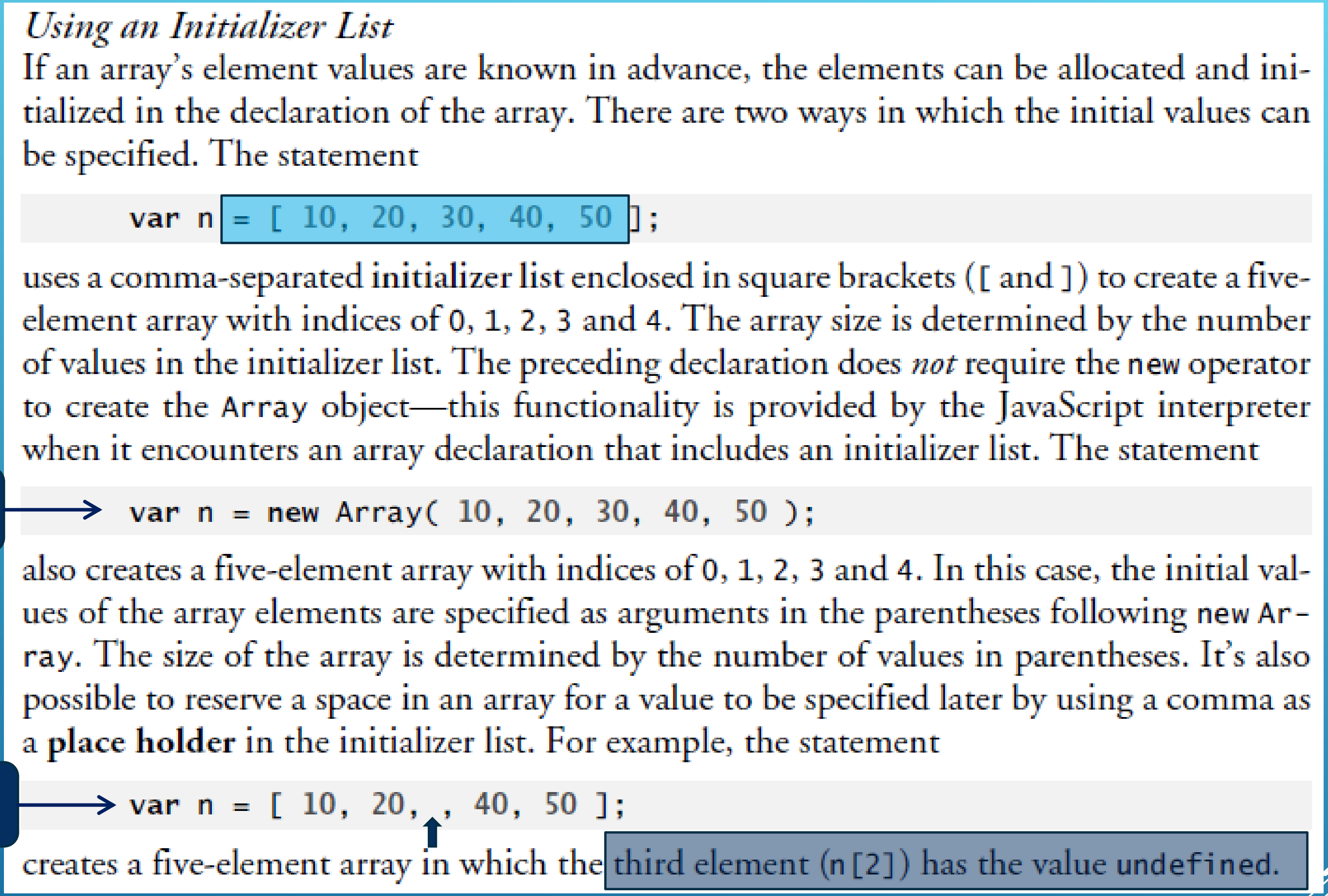
ARRAY INITIALIZATION

- Every array in JavaScript knows its own length, which it stores in its **length** attribute and can be found with the expression **arrayname.length**
- **For example:**

document.writeln(“Total elements in array c are: “+c.length);

Will display :Total elements in array c are 12

ARRAY INITIALIZATION-SUMMARY



Using an Initializer List

If an array's element values are known in advance, the elements can be allocated and initialized in the declaration of the array. There are two ways in which the initial values can be specified. The statement

```
var n = [ 10, 20, 30, 40, 50 ];
```

uses a comma-separated **initializer list** enclosed in square brackets ([and]) to create a five-element array with indices of 0, 1, 2, 3 and 4. The array size is determined by the number of values in the initializer list. The preceding declaration does *not* require the new operator to create the Array object—this functionality is provided by the JavaScript interpreter when it encounters an array declaration that includes an initializer list. The statement

Way-1

```
var n = new Array( 10, 20, 30, 40, 50 );
```

also creates a five-element array with indices of 0, 1, 2, 3 and 4. In this case, the initial values of the array elements are specified as arguments in the parentheses following new Array. The size of the array is determined by the number of values in parentheses. It's also possible to reserve a space in an array for a value to be specified later by using a comma as a **place holder** in the initializer list. For example, the statement

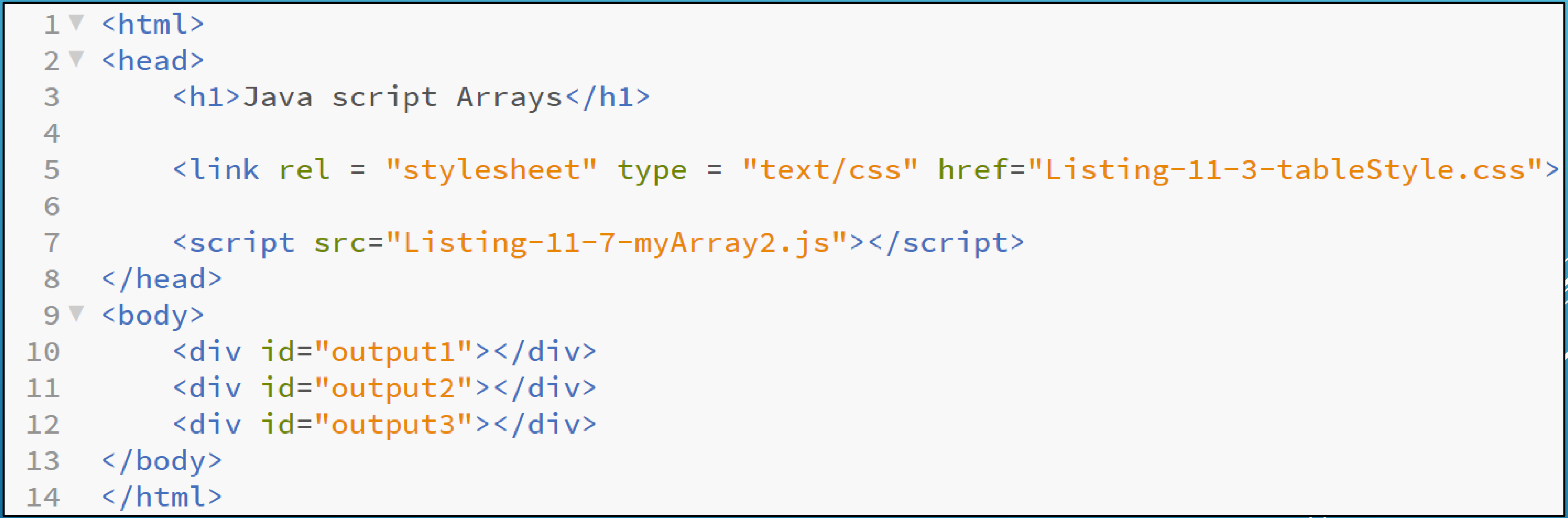
Way-2

```
var n = [ 10, 20, , 40, 50 ];
```

creates a five-element array in which the third element (n[2]) has the value undefined.

THE HTML FILE

- The example in Listing-11-6 and Listing-11-7 are creating three Array objects to demonstrate initializing arrays with initializer lists.
- Output-Listing-11-6 is nearly identical to Output-Listing-11-4 but provides three div elements in its body element for displaying this example's arrays.

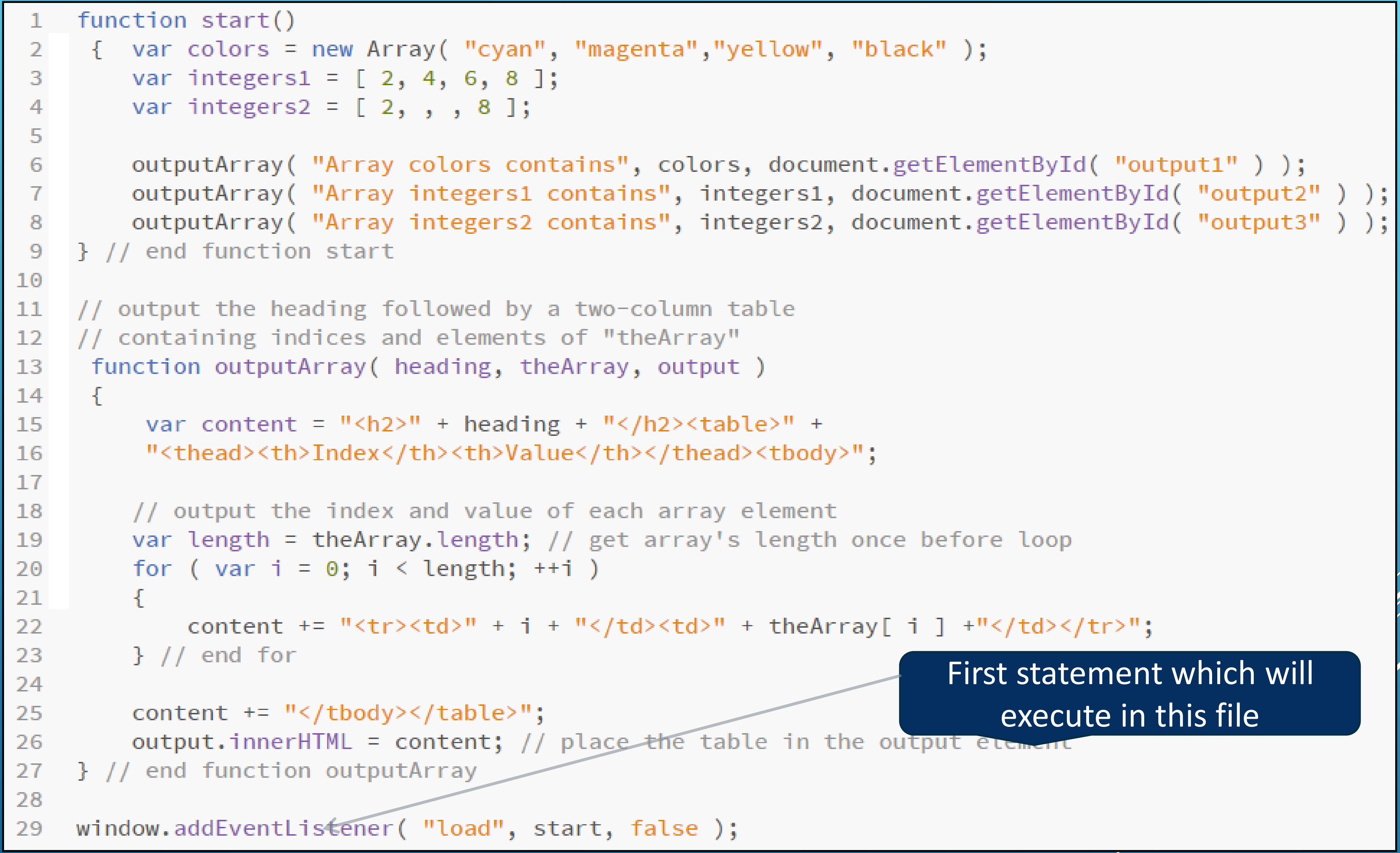


```html
<html>
<head>
    <h1>Java script Arrays</h1>

    <link rel = "stylesheet" type = "text/css" href="Listing-11-3-tableStyle.css">

    <script src="Listing-11-7-myArray2.js"></script>
</head>
<body>
    <div id="output1"></div>
    <div id="output2"></div>
    <div id="output3"></div>
</body>
</html>
```

Listing-11-6

THE JAVASCRIPT FILE

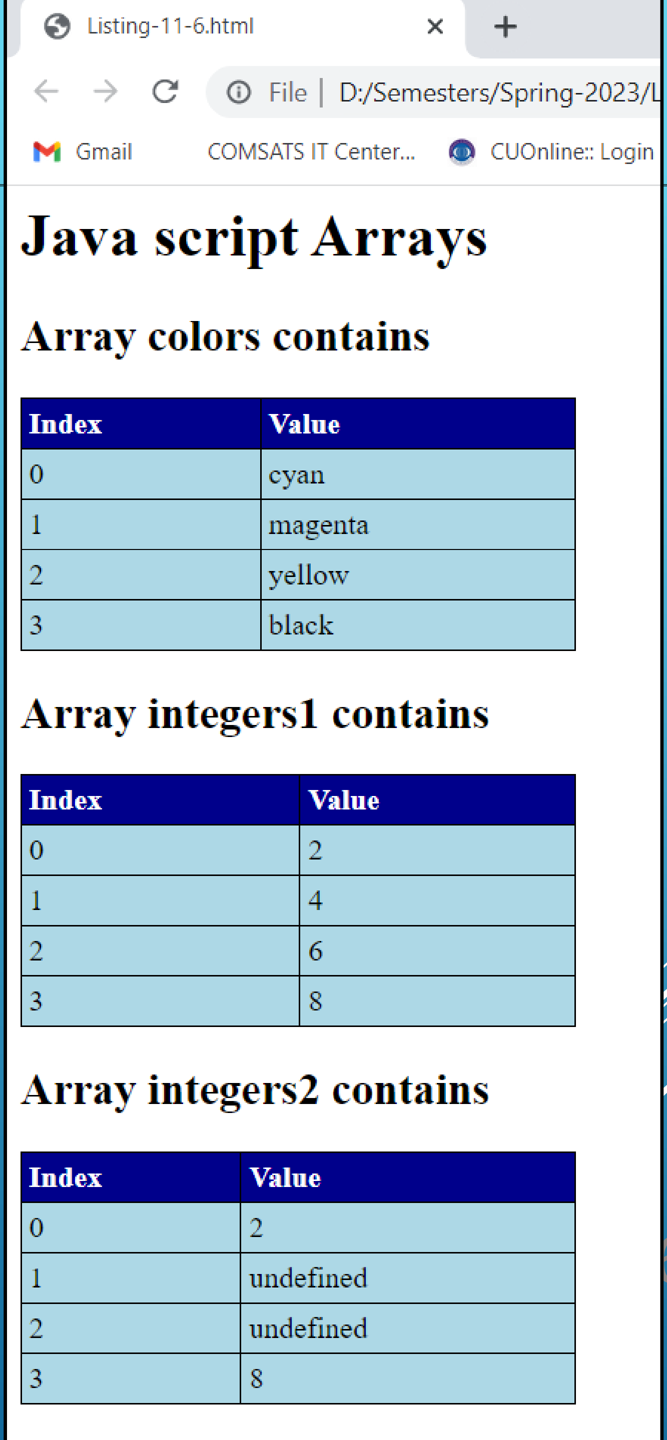
Listing-11-7-myArray2.js



```
function start()
 {  var colors = new Array( "cyan", "magenta","yellow", "black" );
    var integers1 = [ 2, 4, 6, 8 ];
    var integers2 = [ 2, , , 8 ];

    outputArray( "Array colors contains", colors, document.getElementById( "output1" ) );
    outputArray( "Array integers1 contains", integers1, document.getElementById( "output2" ) );
    outputArray( "Array integers2 contains", integers2, document.getElementById( "output3" ) );
} // end function start

// output the heading followed by a two-column table
// containing indices and elements of "theArray"
 function outputArray( heading, theArray, output )
 {
     var content = "<h2>" + heading + "</h2><table>" +
     "<thead><th>Index</th><th>Value</th></thead><tbody>";

    // output the index and value of each array element
    var length = theArray.length; // get array's length once before loop
    for ( var i = 0; i < length; ++i )
    {
        content += "<tr><td>" + i + "</td><td>" + theArray[ i ] +"</td></tr>";
    } // end for

    content += "</tbody></table>";
    output.innerHTML = content; // place the table in the output element
} // end function outputArray

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

First statement which will execute in this file

THE OUTPUT LISTING-11-6.HTML



Java script Arrays

Array colors contains

Index	Value
0	cyan
1	magenta
2	yellow
3	black

Array integers1 contains

Index	Value
0	2
1	4
2	6
3	8

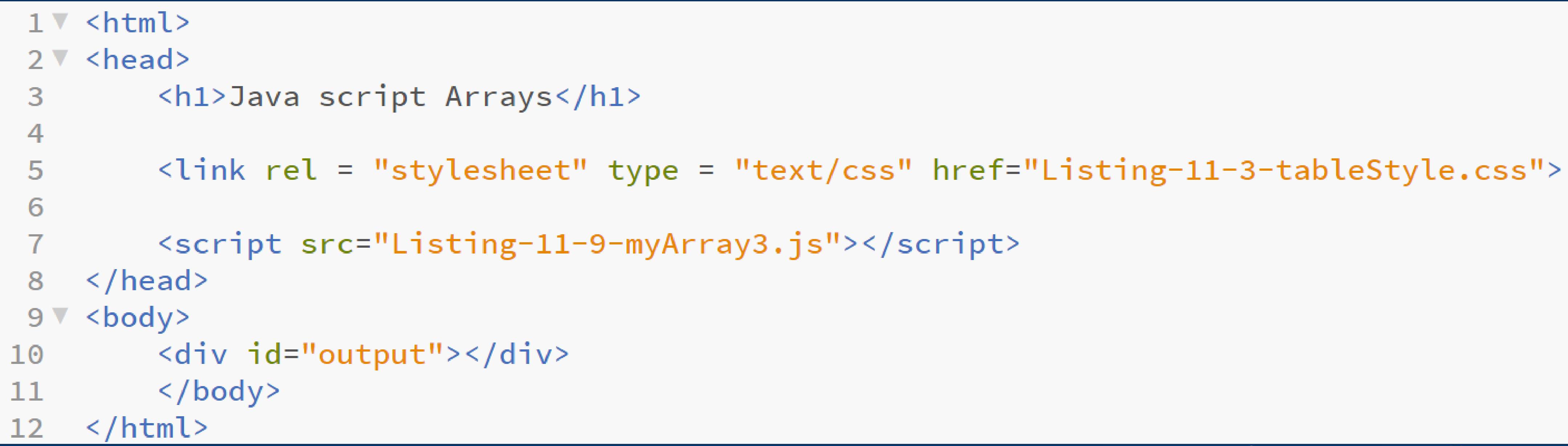
Array integers2 contains

Index	Value
0	2
1	undefined
2	undefined
3	8

Output-Listing-11-6

SUMMING THE ELEMENTS OF AN ARRAY WITH FOR AND FOR…IN

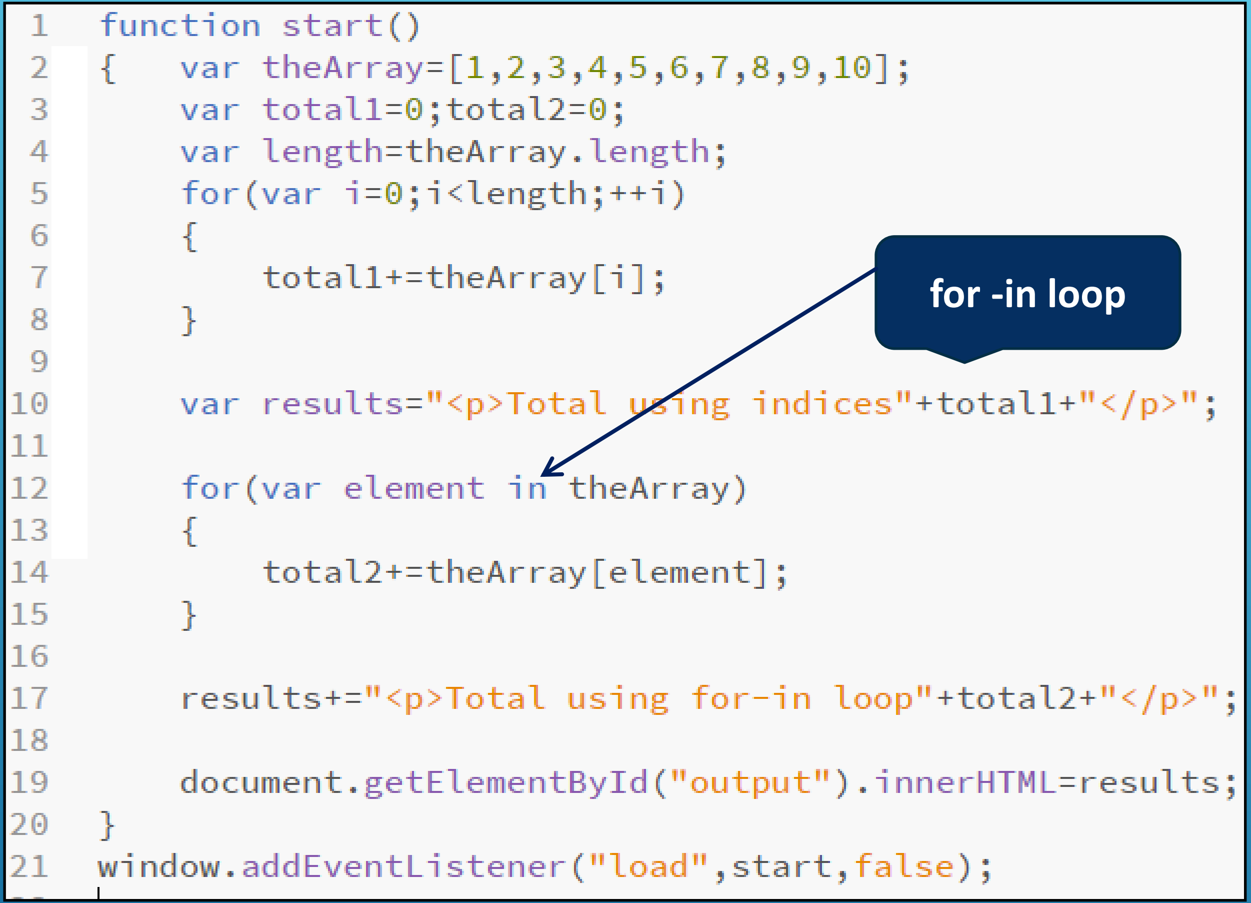
- The example in Listing:11-8.html and Listing-11-9-myArray3.js sums an array’s elements and displays the results.
- The html document in Listing. 22.6 shows the results of the JavaScript file in Listing-11-9-myArray3.js.
- JavaScript’s for…in Repetition Statement
  - Enables a script to perform a task for each element in an array



```html
<html>
<head>
    <h1>Java script Arrays</h1>

    <link rel = "stylesheet" type = "text/css" href="Listing-11-3-tableStyle.css">

    <script src="Listing-11-9-myArray3.js"></script>
</head>
<body>
    <div id="output"></div>
    </body>
</html>
```

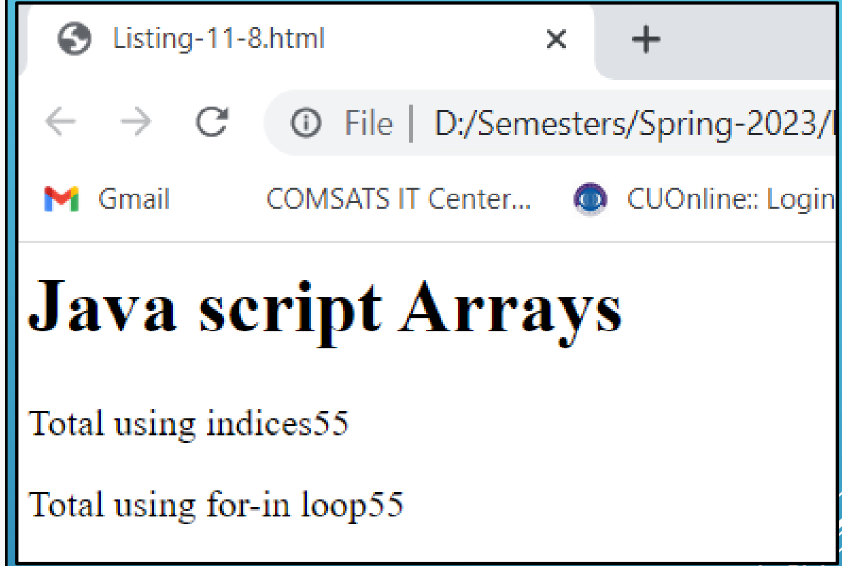
Listing-11-8.html

JAVASCRIPT FILE AND OUTPUT LISTING-11-8.HTML



```
function start()
{   var theArray=[1,2,3,4,5,6,7,8,9,10];
    var total1=0;total2=0;
    var length=theArray.length;
    for(var i=0;i<length;++i)
    {
        total1+=theArray[i];
    }

    var results="<p>Total using indices"+total1+"</p>";

    for(var element in theArray)
    {
        total2+=theArray[element];
    }

    results+="<p>Total using for-in loop"+total2+"</p>";

    document.getElementById("output").innerHTML=results;
}
window.addEventListener("load",start,false);
```

Listing-11-9-myArray3.js



```
Java script Arrays

Total using indices55

Total using for-in loop55
```

OUTPUT CREATED BY LISTING-11-9-MYARRAY3.JS AND SENT TO LISTING 11-8.HTML

- The script in listing-11-9 sums the values contained in the Array, the 10-element integer array declared, allocated and initialized in line#2.
- The statement in line#14 in the body of the first for statement does the totaling.

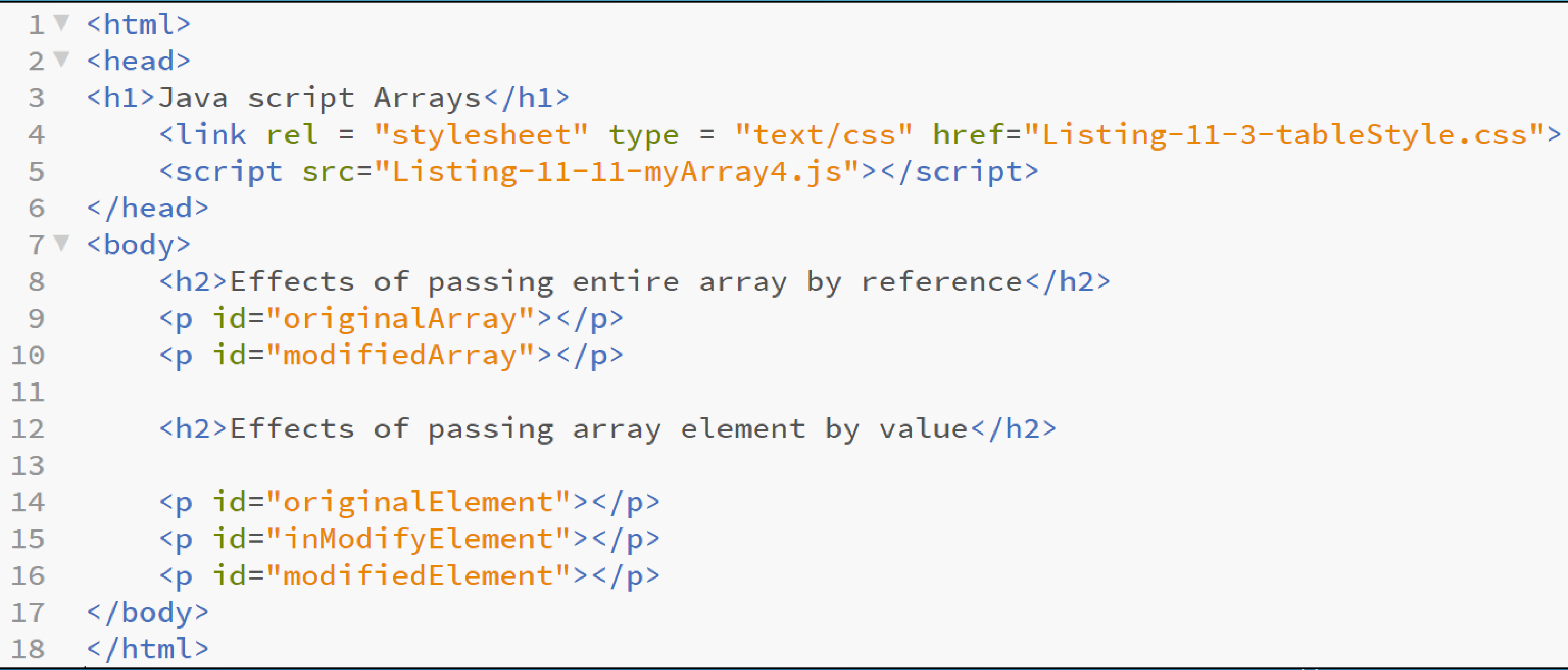
The for…in Repetition Statement

- In this example, we introduce JavaScript's **for…in statement**, which enables a script to perform a task for each element in an array.
- Line# 12–15 show the syntax of a for…in statement. Inside the parentheses, we declare the **element** variable used to select each element in the object to the right of keyword in (**theArray** in this case).
- When you use for…in, JavaScript automatically determines the number of elements in the array.
- As the JavaScript interpreter iterates over the Array's elements, variable element is assigned a value that can be used as an index for the Array. In the case of an array, the value assigned is an index in the range from 0 up to, but not including, the Array.length.
- Each value is added to total2 to produce the sum of the elements in the array.
- ***Note: for-in loop skips any undefined element in the array***

PASSING ARRAYS TO FUNCTIONS

- Pass an array as an argument to a function
  - Specify the array's name (a reference to the array) without brackets as argument
- Although entire arrays are **passed by reference**, but in case of individual numeric and boolean array elements are **passed by value** exactly as simple numeric and boolean variables are passed
  - Such simple single pieces of data are called scalars, or scalar quantities
  - To pass an array element to a function(known as pass by value), use the indexed name of the element as an argument in the function call
- **join** method of an Array
  - Returns a string that contains all of the elements of an array, separated by the string supplied in the function's argument
  - If an argument is not specified, the empty string is used as the separator
  - We use **join** method in listing-11-11-myArray4.js at line#12

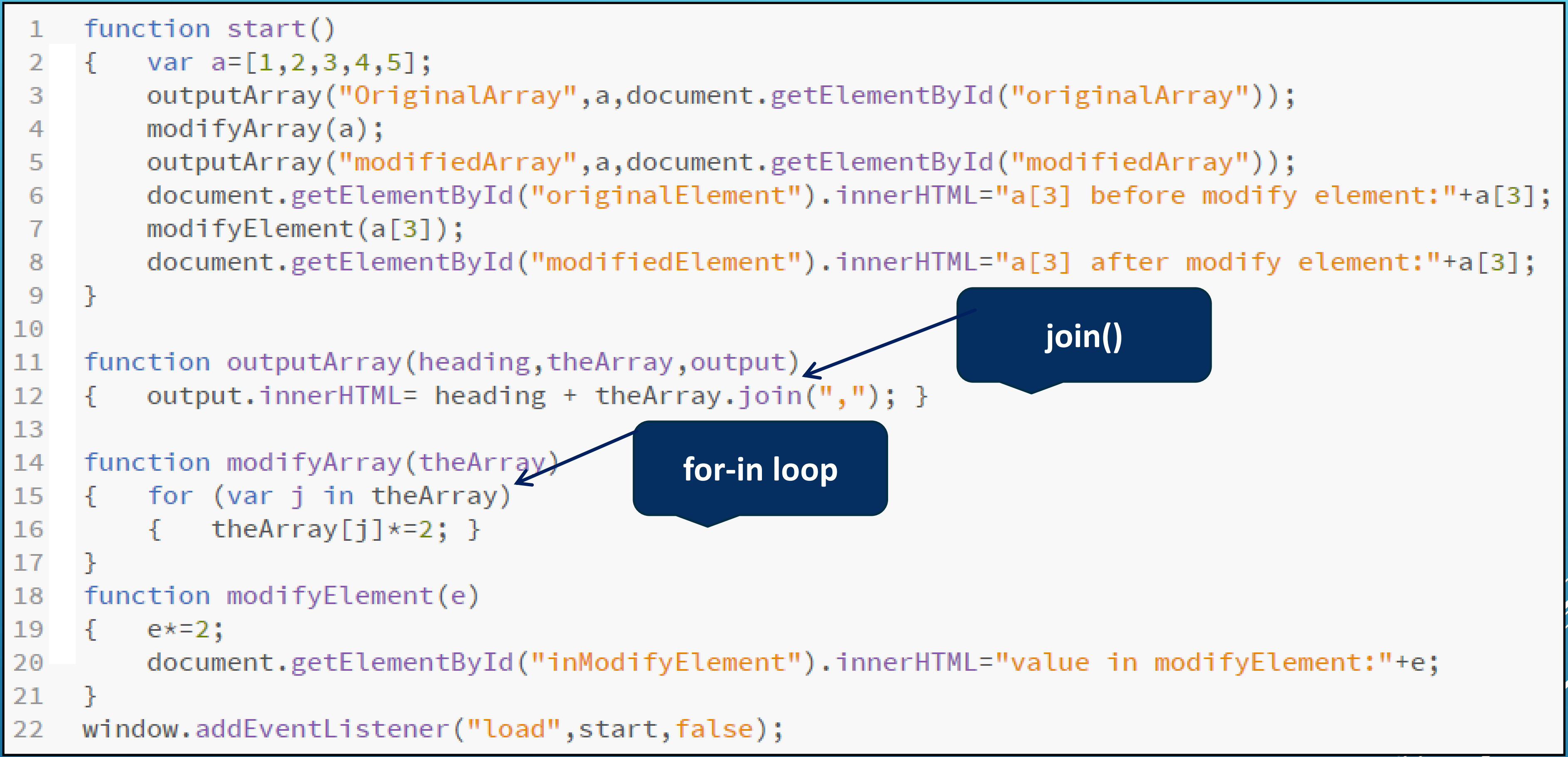
THE HTML FILE



```html
<html>
<head>
<h1>Java script Arrays</h1>
    <link rel = "stylesheet" type = "text/css" href="Listing-11-3-tableStyle.css">
    <script src="Listing-11-11-myArray4.js"></script>
</head>
<body>
    <h2>Effects of passing entire array by reference</h2>
    <p id="originalArray"></p>
    <p id="modifiedArray"></p>

    <h2>Effects of passing array element by value</h2>

    <p id="originalElement"></p>
    <p id="inModifyElement"></p>
    <p id="modifiedElement"></p>
</body>
</html>
```

Listing-11-10.html

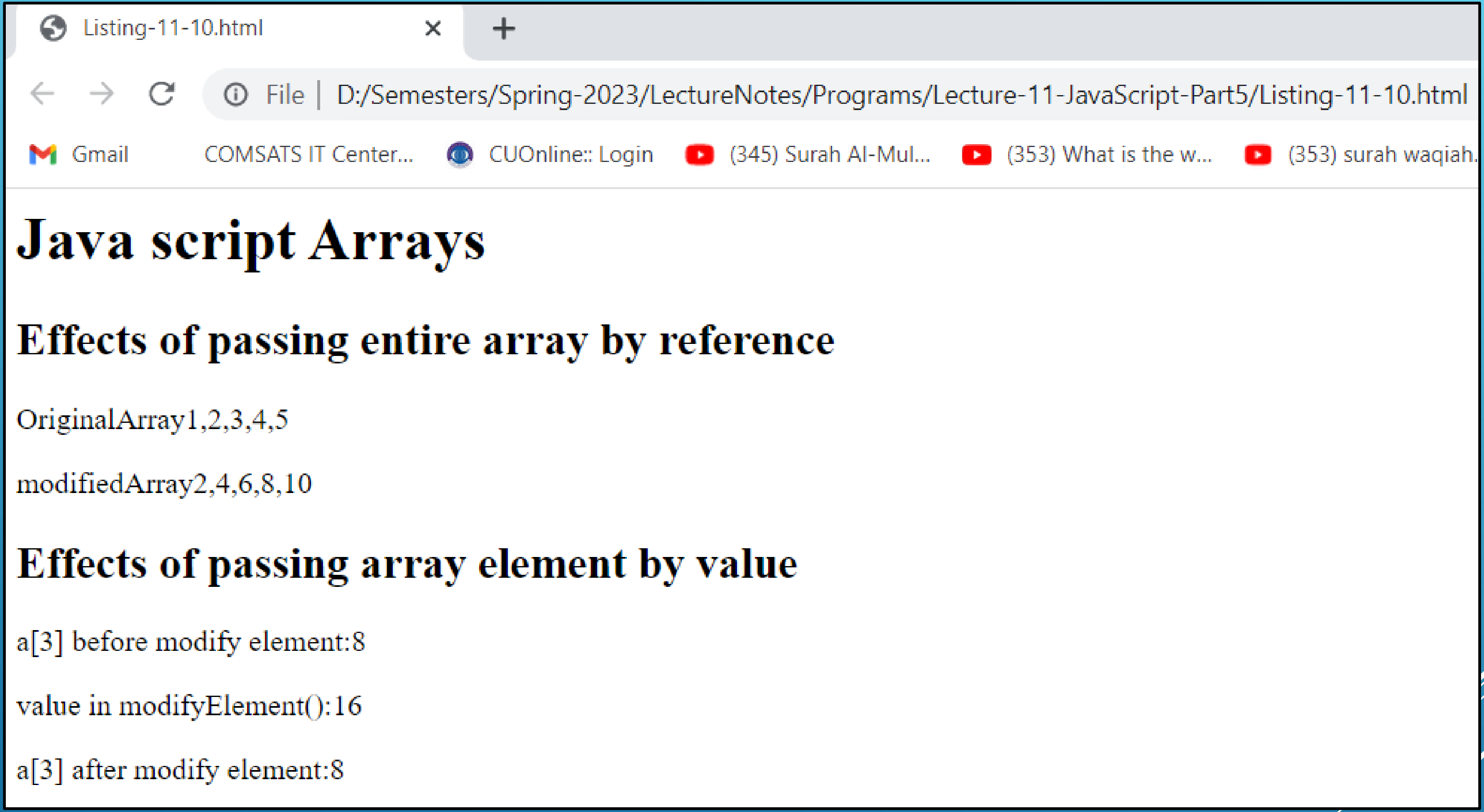
THE JAVASCRIPT FILE



```
function start()
{   var a=[1,2,3,4,5];
    outputArray("OriginalArray",a,document.getElementById("originalArray"));
    modifyArray(a);
    outputArray("modifiedArray",a,document.getElementById("modifiedArray"));
    document.getElementById("originalElement").innerHTML="a[3] before modify element:"+a[3];
    modifyElement(a[3]);
    document.getElementById("modifiedElement").innerHTML="a[3] after modify element:"+a[3];
}

function outputArray(heading,theArray,output)
{   output.innerHTML= heading + theArray.join(","); }

function modifyArray(theArray)
{   for (var j in theArray)
    {   theArray[j]*=2; }
}
function modifyElement(e)
{   e*=2;
    document.getElementById("inModifyElement").innerHTML="value in modifyElement:"+e;
}
window.addEventListener("load",start,false);
```

Listing-11-11-myArray4.js

OUTPUT LISTING-11-10.HTML CREATED BY LISTING-11-11-MYARRAY4.JS



```
Java script Arrays

Effects of passing entire array by reference

OriginalArray1,2,3,4,5

modifiedArray2,4,6,8,10

Effects of passing array element by value

a[3] before modify element:8

value in modifyElement():16

a[3] after modify element:8
```

Output-Listing-11-10.html

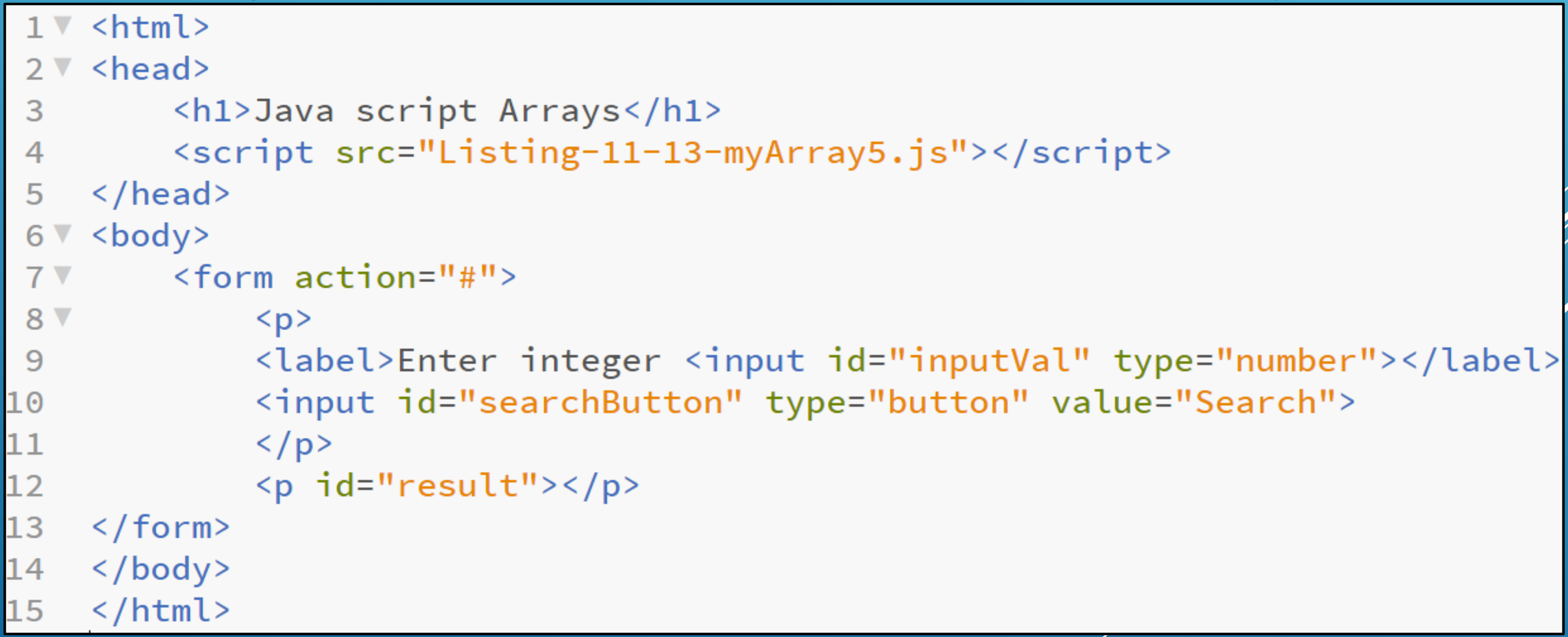
SEARCHING ARRAYS WITH ARRAY METHOD INDEXOF

- It's often necessary to determine whether an array contains a value that matches a certain *key value*.
- The process of locating a particular element value in an array is called *searching*.
- The Array object in JavaScript has built-in methods **indexOf** and **lastIndexOf** for searching arrays.
  - Method indexOf searches for the first occurrence of the specified key value
  - Method lastIndexOf searches for the last occurrence of the specified key value.
- If the key value is found in the array, each method returns the index of that value; otherwise, -1 is returned.
- Every input element has a **value** property that can be used to get or set the element's value.

SEARCHING ARRAYS WITH ARRAY METHOD INDEXOF…

Optional Second Argument to indexOf and lastIndexOf

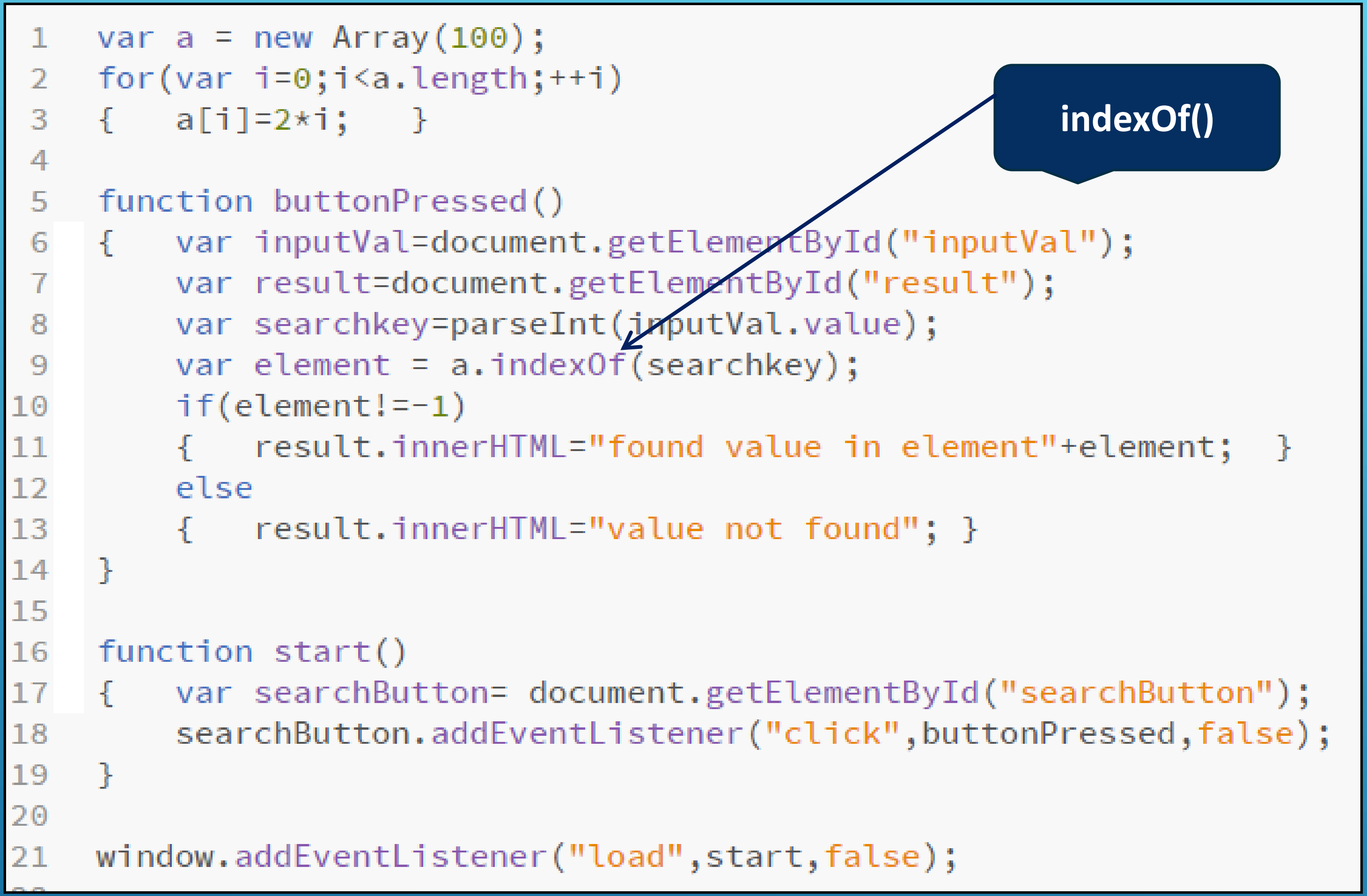
- You can pass an optional second argument to methods indexOf and lastIndexOf that represents the index from which to start the search.
- By default, this argument's value is 0 hence the method search the entire array from index 0 to index n-1, where n is the total length of array.
- If the argument is greater than or equal to the array's length, the methods simply return -1.
- If the argument's value is negative, it's used as an offset from the end of the array.



```html
<html>
<head>
    <h1>Java script Arrays</h1>
    <script src="Listing-11-13-myArray5.js"></script>
</head>
<body>
    <form action="#">
        <p>
        <label>Enter integer <input id="inputVal" type="number"></label>
        <input id="searchButton" type="button" value="Search">
        </p>
        <p id="result"></p>
</form>
</body>
</html>
```

Listing-11.12.html

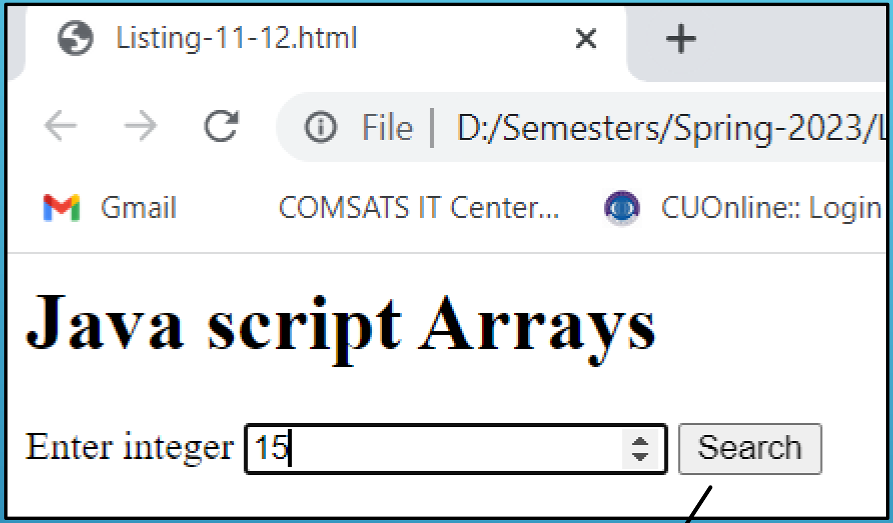
THE JAVASCRIPT FILE



```
var a = new Array(100);
for(var i=0;i<a.length;++i)
{   a[i]=2*i;   }

function buttonPressed()
{   var inputVal=document.getElementById("inputVal");
    var result=document.getElementById("result");
    var searchkey=parseInt(inputVal.value);
    var element = a.indexOf(searchkey);
    if(element!=-1)
    {   result.innerHTML="found value in element"+element;  }
    else
    {   result.innerHTML="value not found"; }
}

function start()
{   var searchButton= document.getElementById("searchButton");
    searchButton.addEventListener("click",buttonPressed,false);
}

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

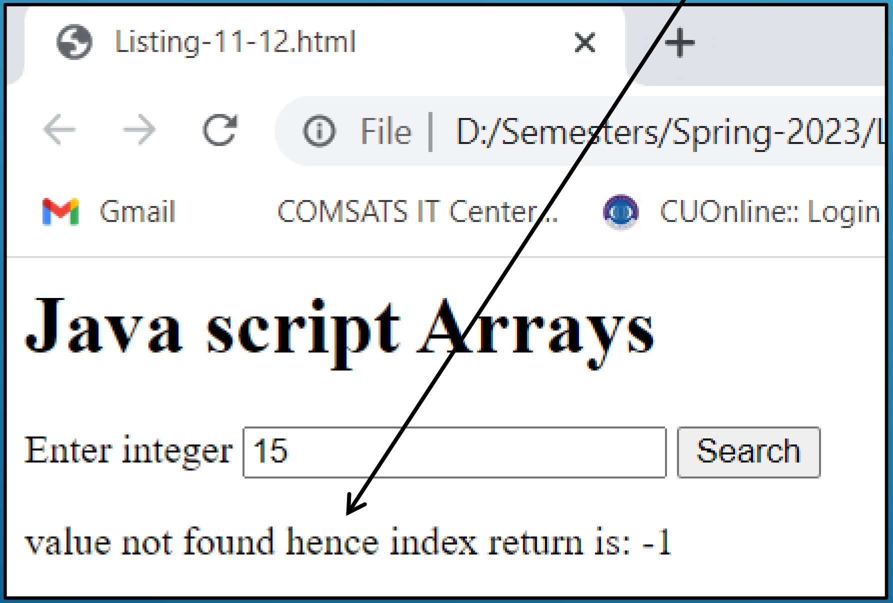
Listing-11-13-myArray5.js

OUTPUT LISTING-11-12.HTML CREATED BY LISTING-11-13-MYARRAY5.JS



Java script Arrays

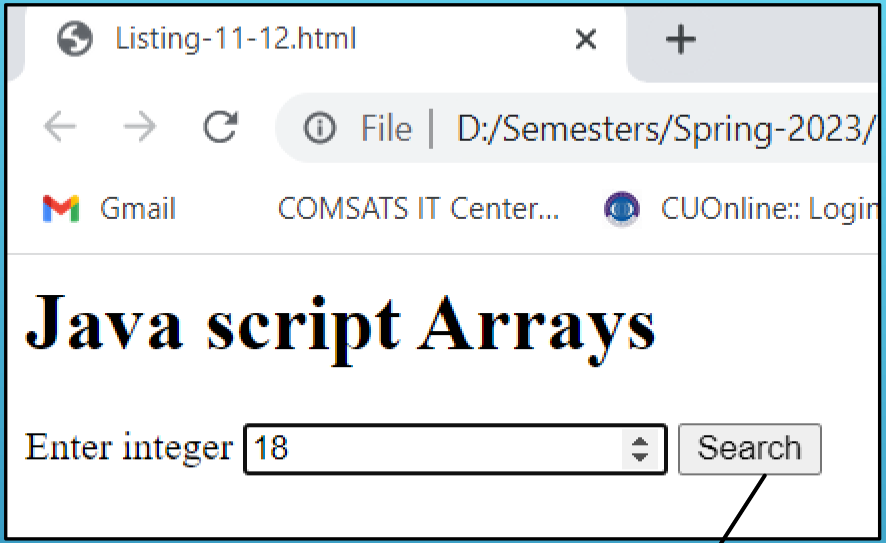
Enter integer 15 Search



Java script Arrays

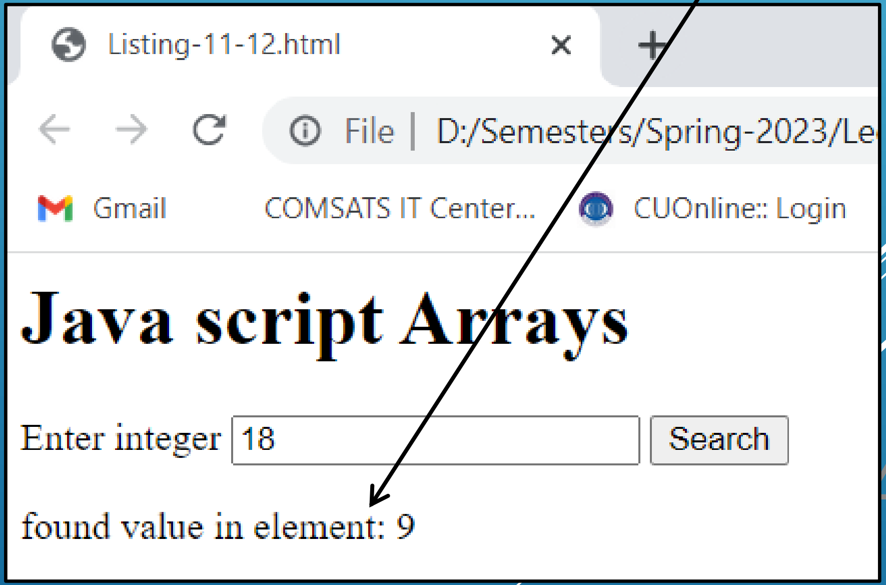
Enter integer 15 Search

value not found hence index return is: -1



Java script Arrays

Enter integer 18 Search

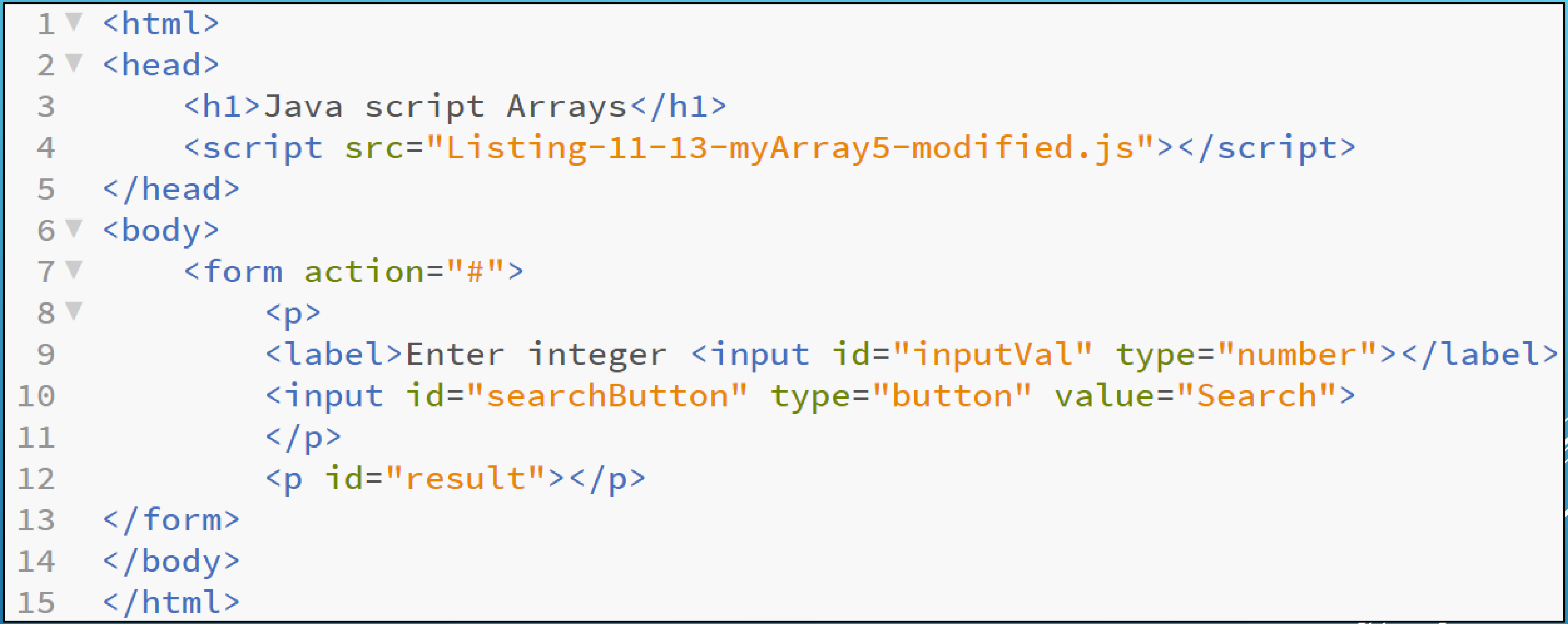


Java script Arrays

Enter integer 18 Search

found value in element: 9

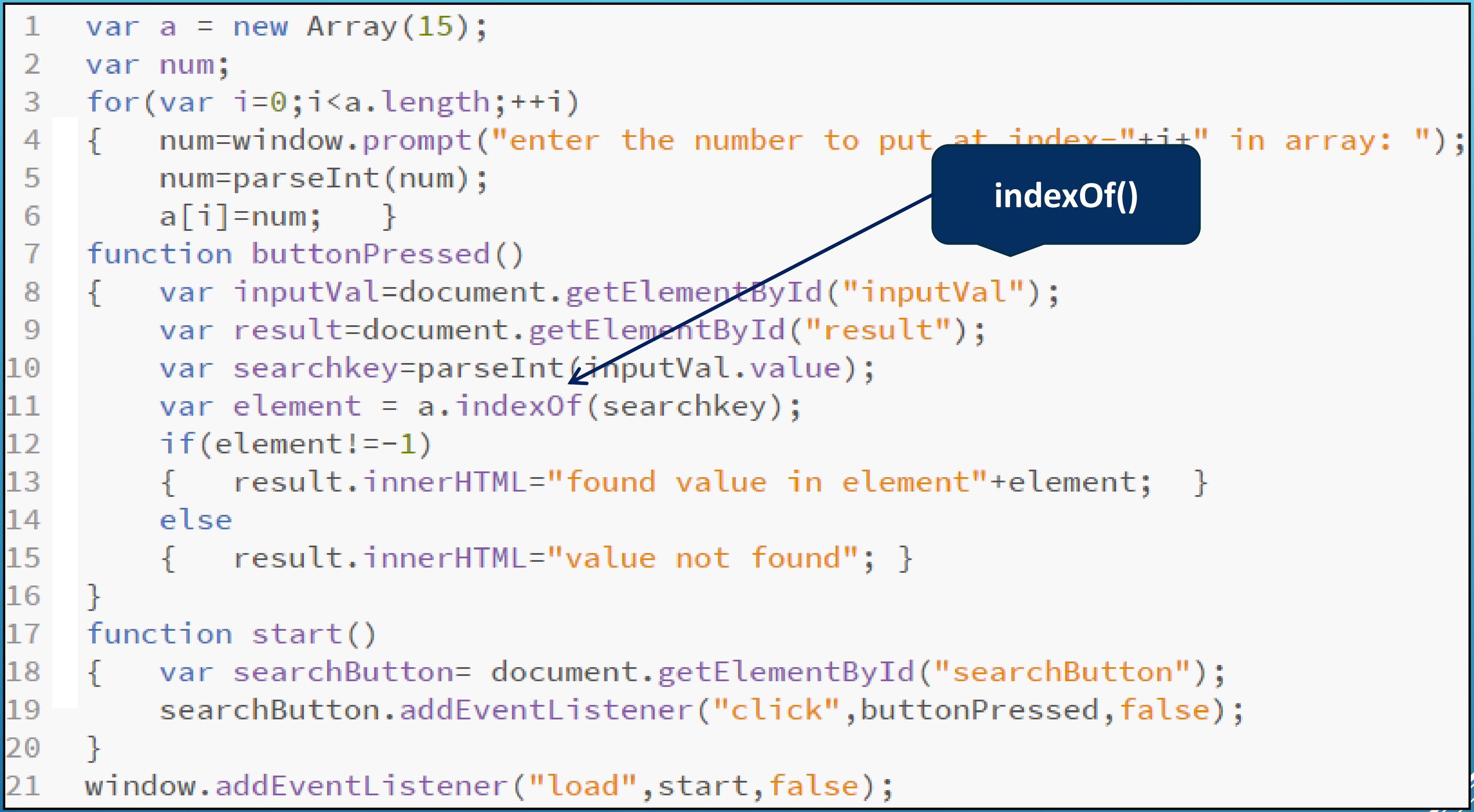
THE HTML FILE



```html
<html>
<head>
    <h1>Java script Arrays</h1>
    <script src="Listing-11-13-myArray5-modified.js"></script>
</head>
<body>
    <form action="#">
        <p>
        <label>Enter integer <input id="inputVal" type="number"></label>
        <input id="searchButton" type="button" value="Search">
        </p>
        <p id="result"></p>
</form>
</body>
</html>
```

Listing-11-12-modified.html

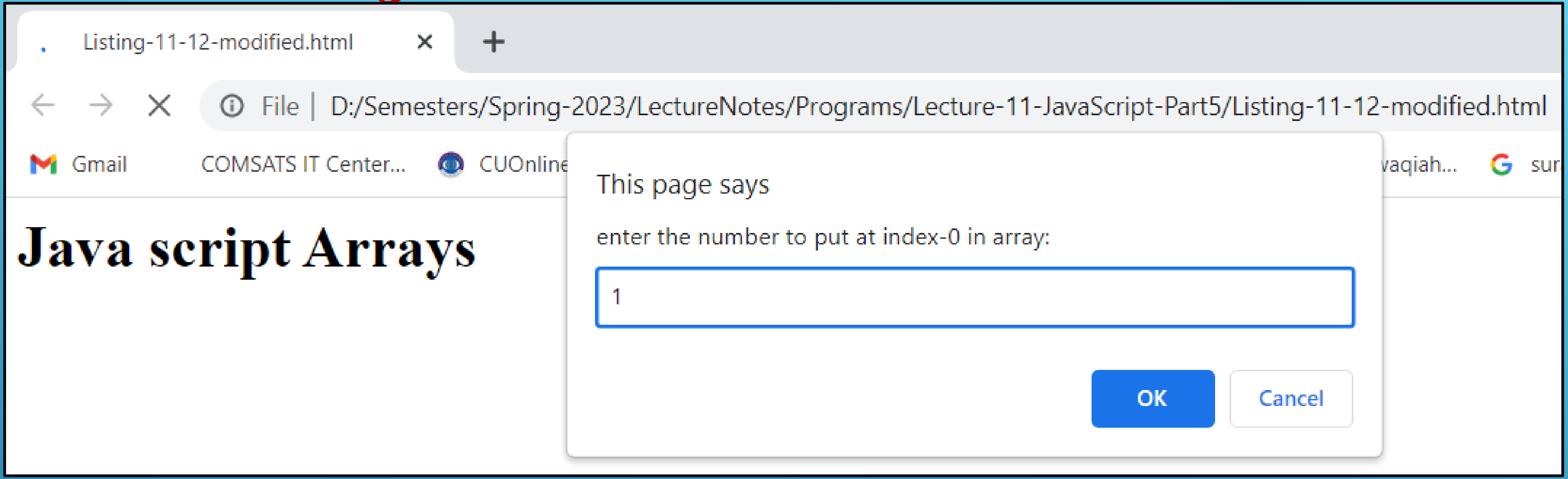
THE JAVASCRIPT FILE



```
var a = new Array(15);
var num;
for(var i=0;i<a.length;++i)
{   num=window.prompt("enter the number to put at index="+i+" in array: ");
    num=parseInt(num);
    a[i]=num;   }
function buttonPressed()
{   var inputVal=document.getElementById("inputVal");
    var result=document.getElementById("result");
    var searchkey=parseInt(inputVal.value);
    var element = a.indexOf(searchkey);
    if(element!=-1)
    {   result.innerHTML="found value in element"+element;  }
    else
    {   result.innerHTML="value not found"; }
}
function start()
{   var searchButton= document.getElementById("searchButton");
    searchButton.addEventListener("click",buttonPressed,false);
}
window.addEventListener("load",start,false);
```

Listing-11-13-myArray5_modified.js

OUTPUT LISTING-11-12-MODIFIED.HTML CREATED BY LISTING-11-13-MODIFIED.JS


Listing-11-12-modified.html

File | D:/Semesters/Spring-2023/LectureNotes/Programs/Lecture-11-JavaScript-Part5/Listing-11-12-modified.html

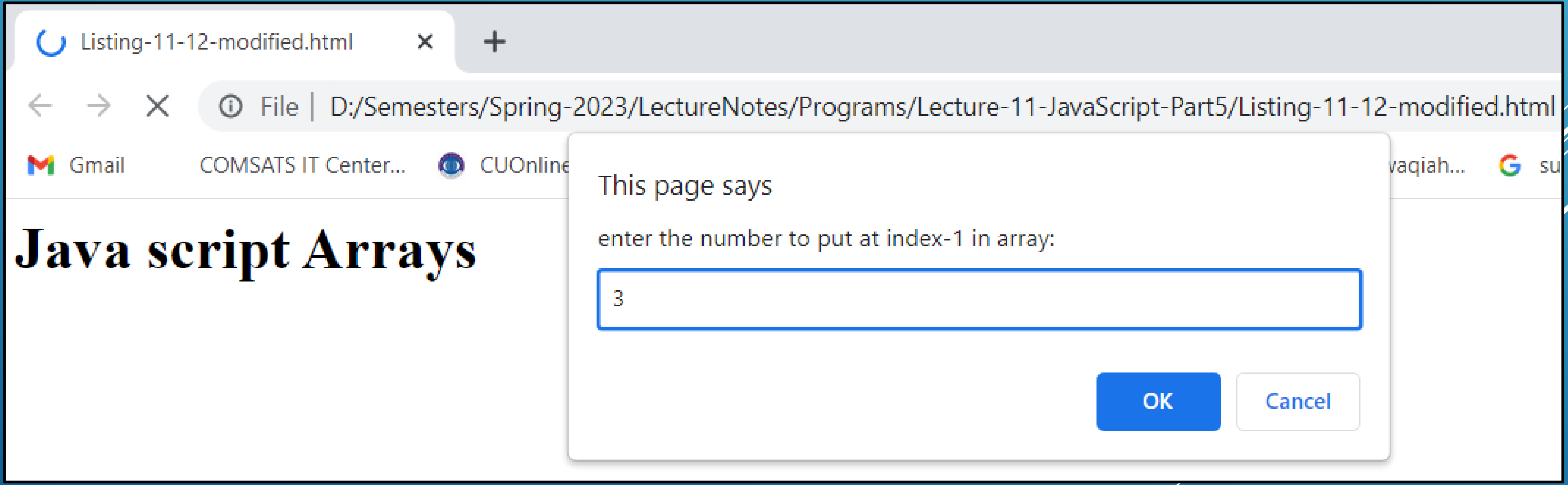
Gmail COMSATS IT Center... CUOnline vaqiah... sur

Java script Arrays

This page says

enter the number to put at index-0 in array:

1


Listing-11-12-modified.html

File | D:/Semesters/Spring-2023/LectureNotes/Programs/Lecture-11-JavaScript-Part5/Listing-11-12-modified.html

Gmail COMSATS IT Center... CUOnline vaqiah... su

Java script Arrays

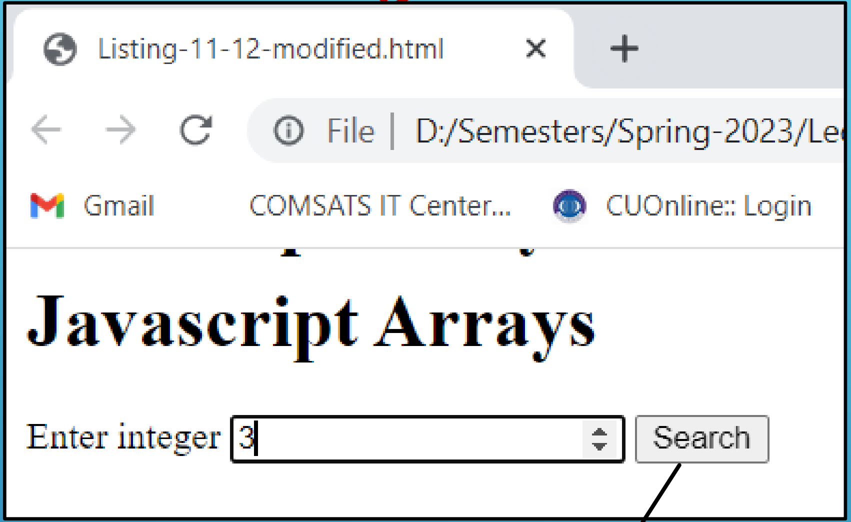
This page says

enter the number to put at index-1 in array:

3

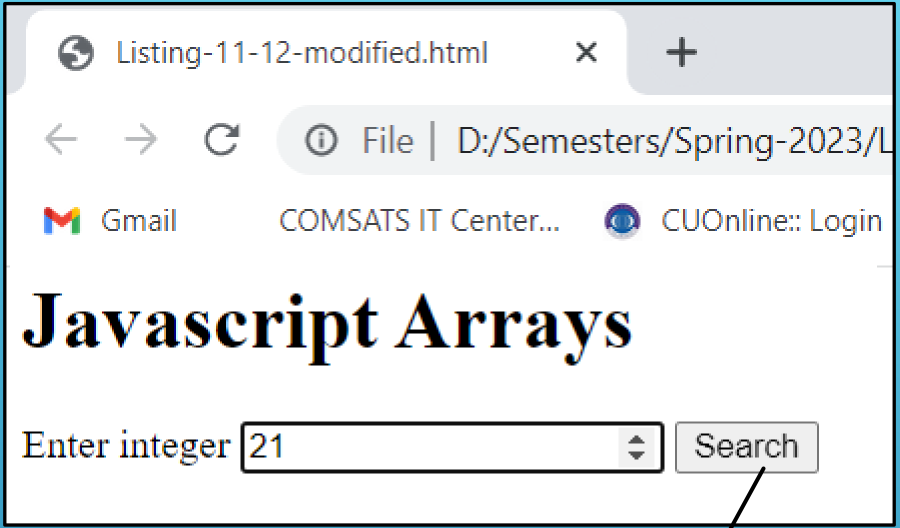
Output-Listing-11-12-modified.html

OUTPUT LISTING-11-12-MODIFIED.HTML CREATED BY LISTING-11-13-MODIFIED.JS


Listing-11-12-modified.html
File | D:/Semesters/Spring-2023/Le
Gmail
COMSATS IT Center...
CUOnline:: Login
Search

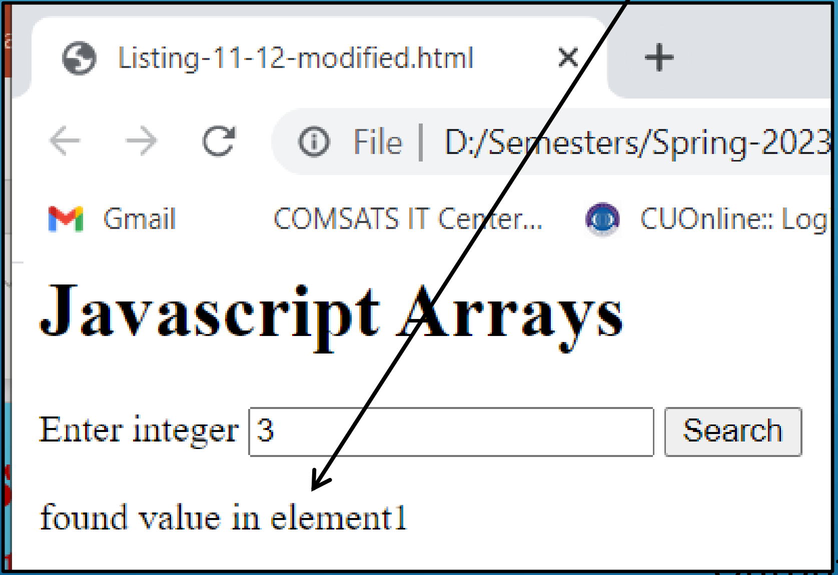
Javascript Arrays

Enter integer 3


Listing-11-12-modified.html
File | D:/Semesters/Spring-2023/L
Gmail
COMSATS IT Center...
CUOnline:: Login
Search

Javascript Arrays

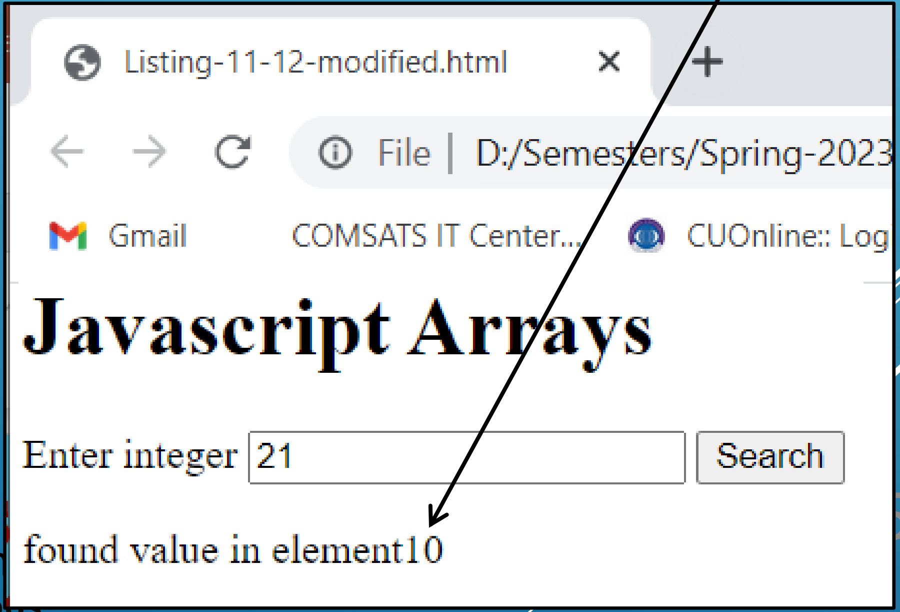
Enter integer 21



Javascript Arrays

Enter integer 3

found value in element1



Javascript Arrays

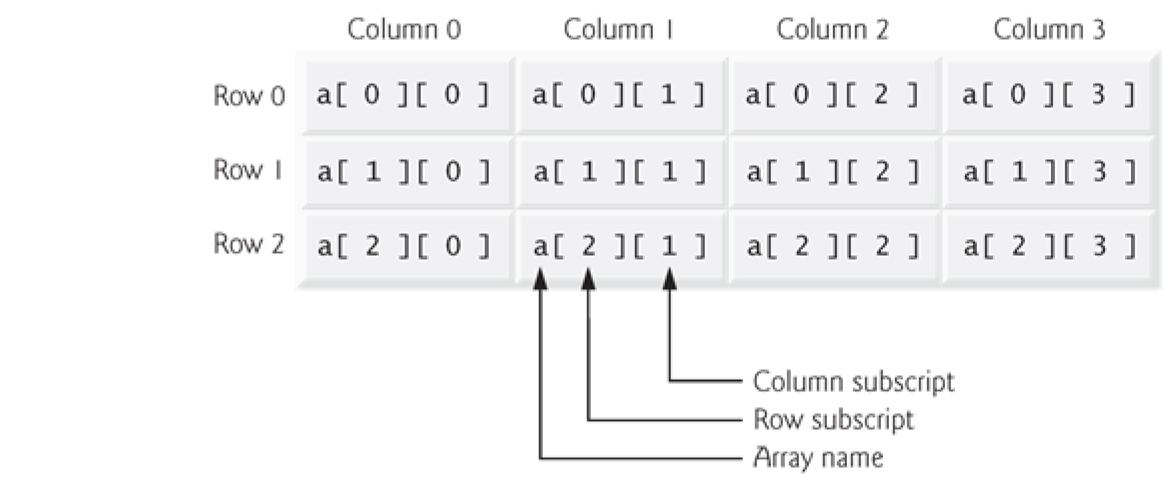
Enter integer 21

found value in element10

MULTIDIMENSIONAL ARRAYS

- To identify a particular two-dimensional multidimensional array element
  - Specify the two indices
  - By convention, the first identifies the element’s row, and the second identifies the element’s column
- In general, an array with *m* rows and *n* columns is called an *m*-by-*n* array
- Two-dimensional array element accessed using an element name of the form a[row][column]
  - a is the name of the array
  - row and column are the indices that uniquely identify the row and column
- Multidimensional arrays can be initialized in declarations like a one-dimensional array, with values grouped by row in square brackets
  - The interpreter determines the number of rows by counting the number of sub initializers
  - The interpreter determines the number of columns in each row by counting the number of values in the sub-array that initializes the row
- The rows of a two-dimensional array can vary in length
- A multidimensional array in which each row has a different number of columns can be allocated dynamically with operator new

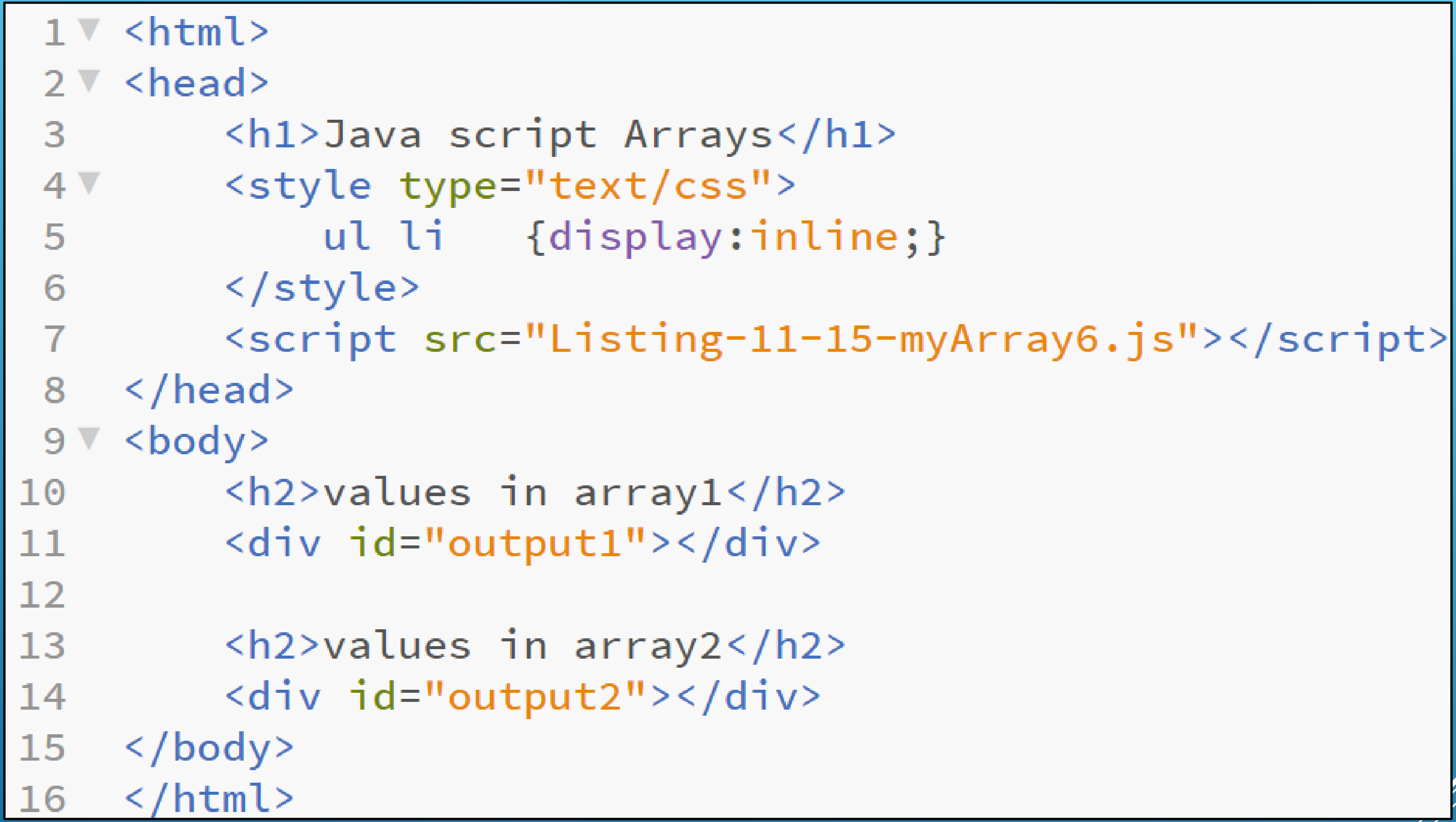
MULTIDIMENSIONAL ARRAYS



	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Two-dimensional array with three rows and four columns.

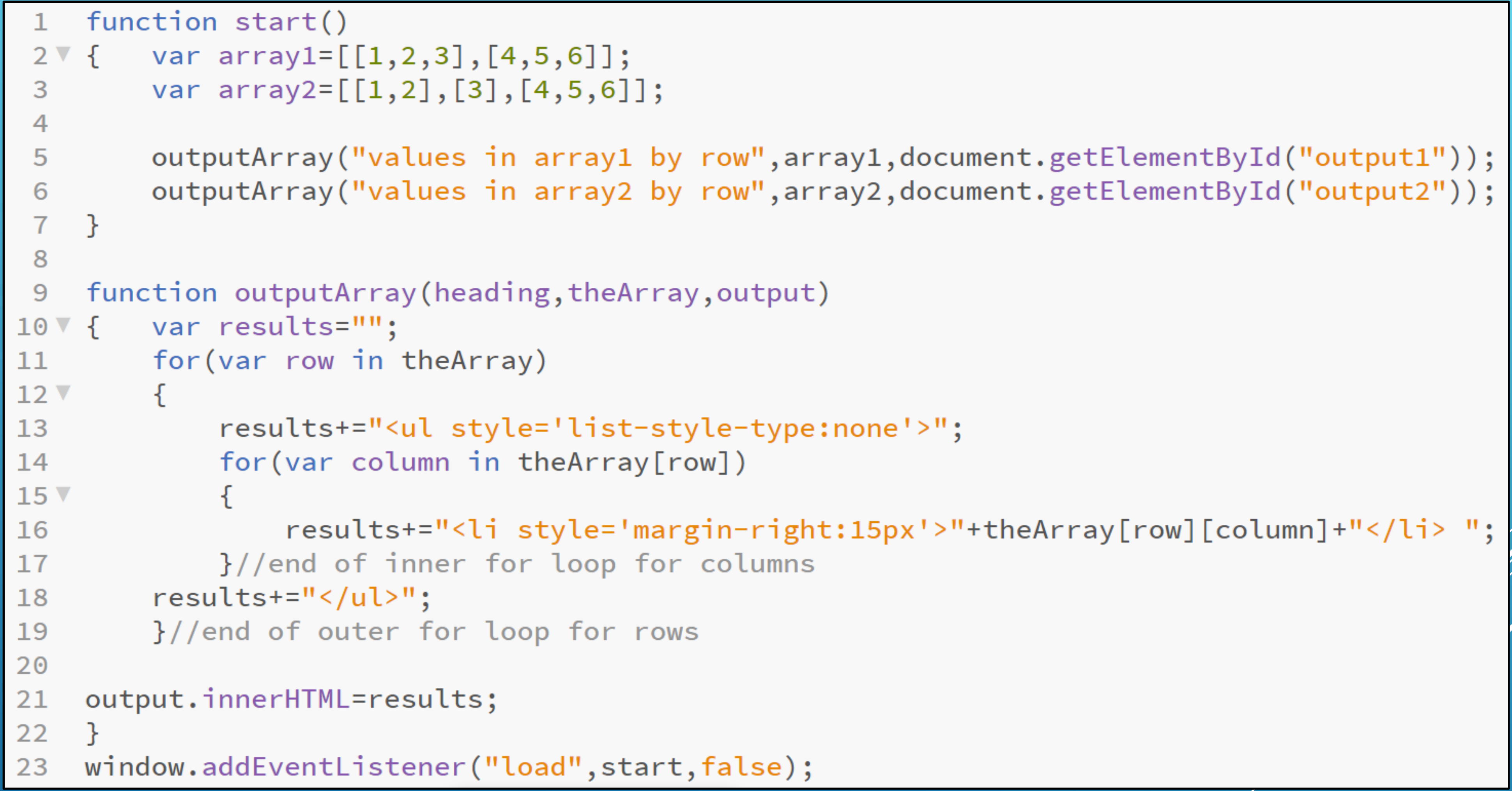
THE HTML FILE



```
<html>
<head>
    <h1>Java script Arrays</h1>
    <style type="text/css">
        ul li   {display:inline;}
    </style>
    <script src="Listing-11-15-myArray6.js"></script>
</head>
<body>
    <h2>values in array1</h2>
    <div id="output1"></div>

    <h2>values in array2</h2>
    <div id="output2"></div>
</body>
</html>
```

Listing-11-14.html

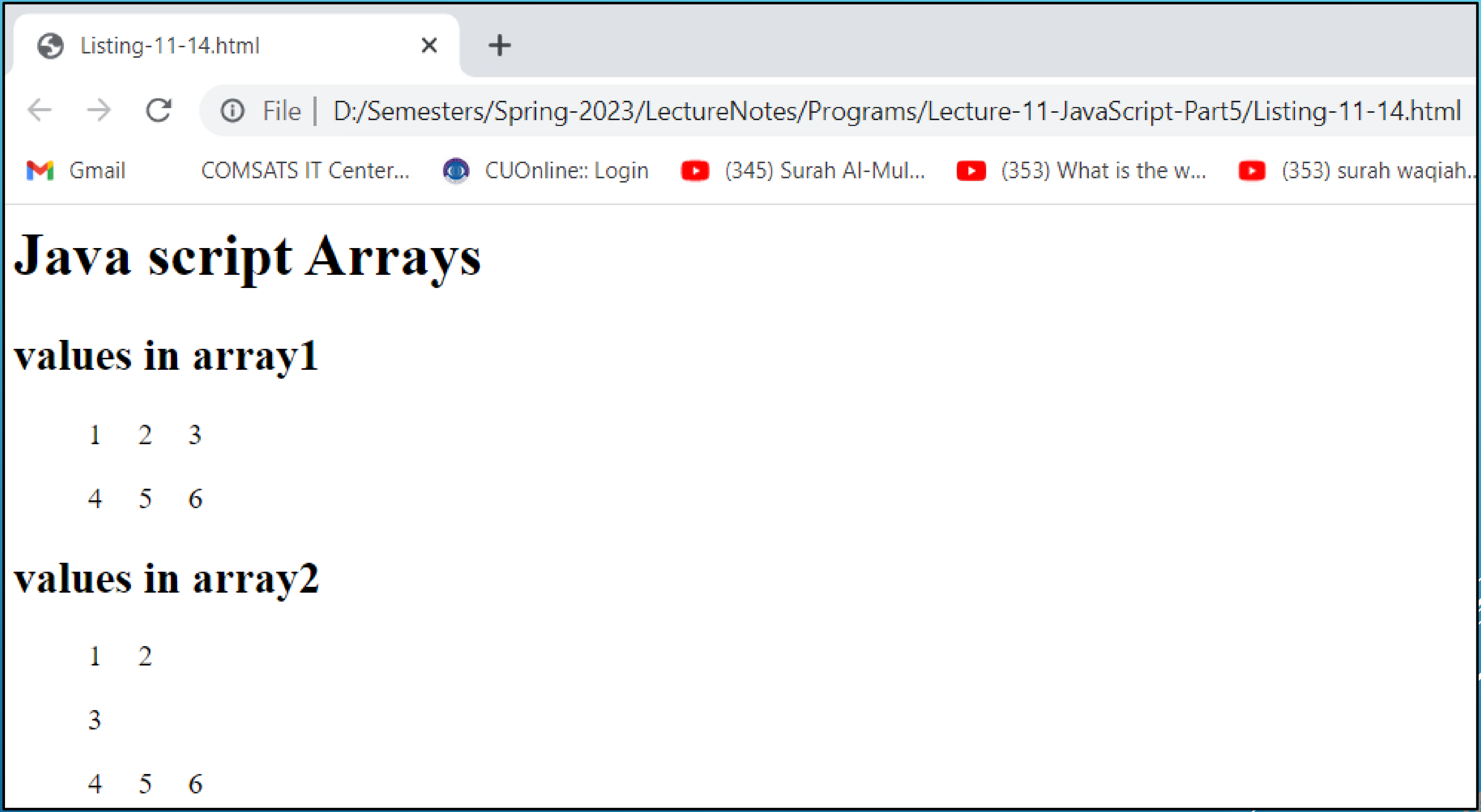
THE JAVASCRIPT FILE



```javascript
function start()
{   var array1=[[1,2,3],[4,5,6]];
    var array2=[[1,2],[3],[4,5,6]];

    outputArray("values in array1 by row",array1,document.getElementById("output1"));
    outputArray("values in array2 by row",array2,document.getElementById("output2"));
}

function outputArray(heading,theArray,output)
{   var results="";
    for(var row in theArray)
    {
        results+="<ul style='list-style-type:none'>";
        for(var column in theArray[row])
        {
            results+="<li style='margin-right:15px'>"+theArray[row][column]+"</li> ";
        }//end of inner for loop for columns
    results+="</ul>";
    }//end of outer for loop for rows

output.innerHTML=results;
}
window.addEventListener("load",start,false);
```

Listing-11-15-myArray6.js

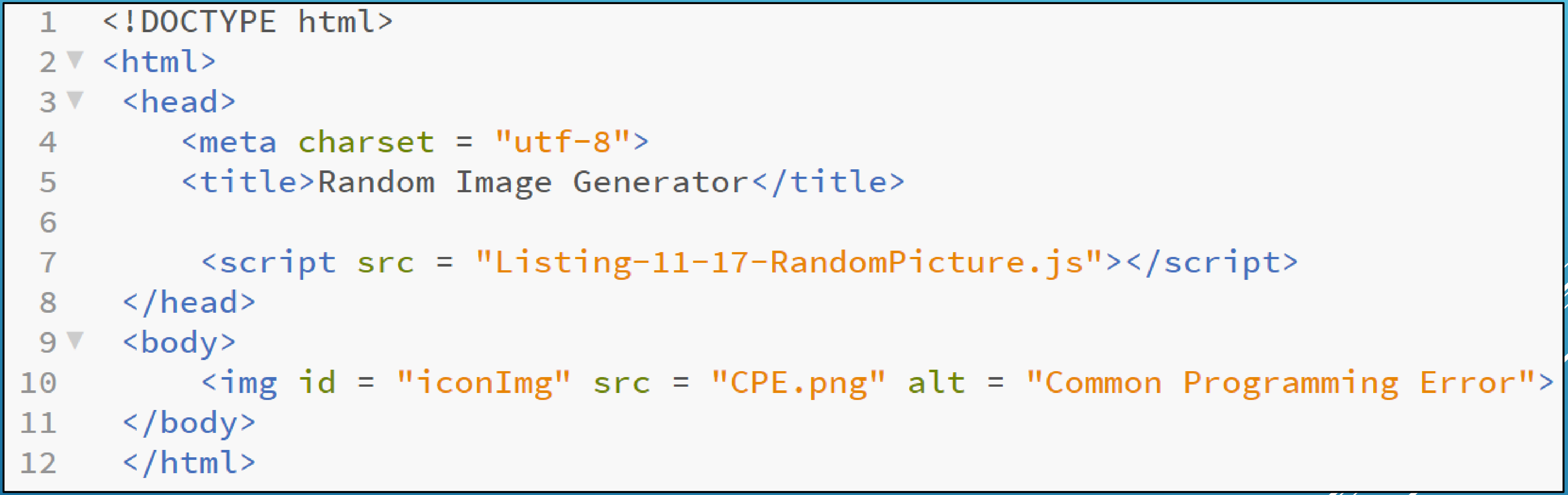
OUTPUT LISTING-11-14.HTML CREATED BY LISTING-11-15-MYARRAY6.JS



```
Java script Arrays

values in array1

    1  2  3

    4  5  6

values in array2

    1  2

    3

    4  5  6
```

RANDOM IMAGE GENERATION USING ARRAYS

- Now we are going to create a more elegant random image generator that does not require the image filenames to contain integers in sequence.
- Each time you click the image button in the web document(Listing-11-16.html), the javascript generates a random integer and uses it as an index into the pictures array.



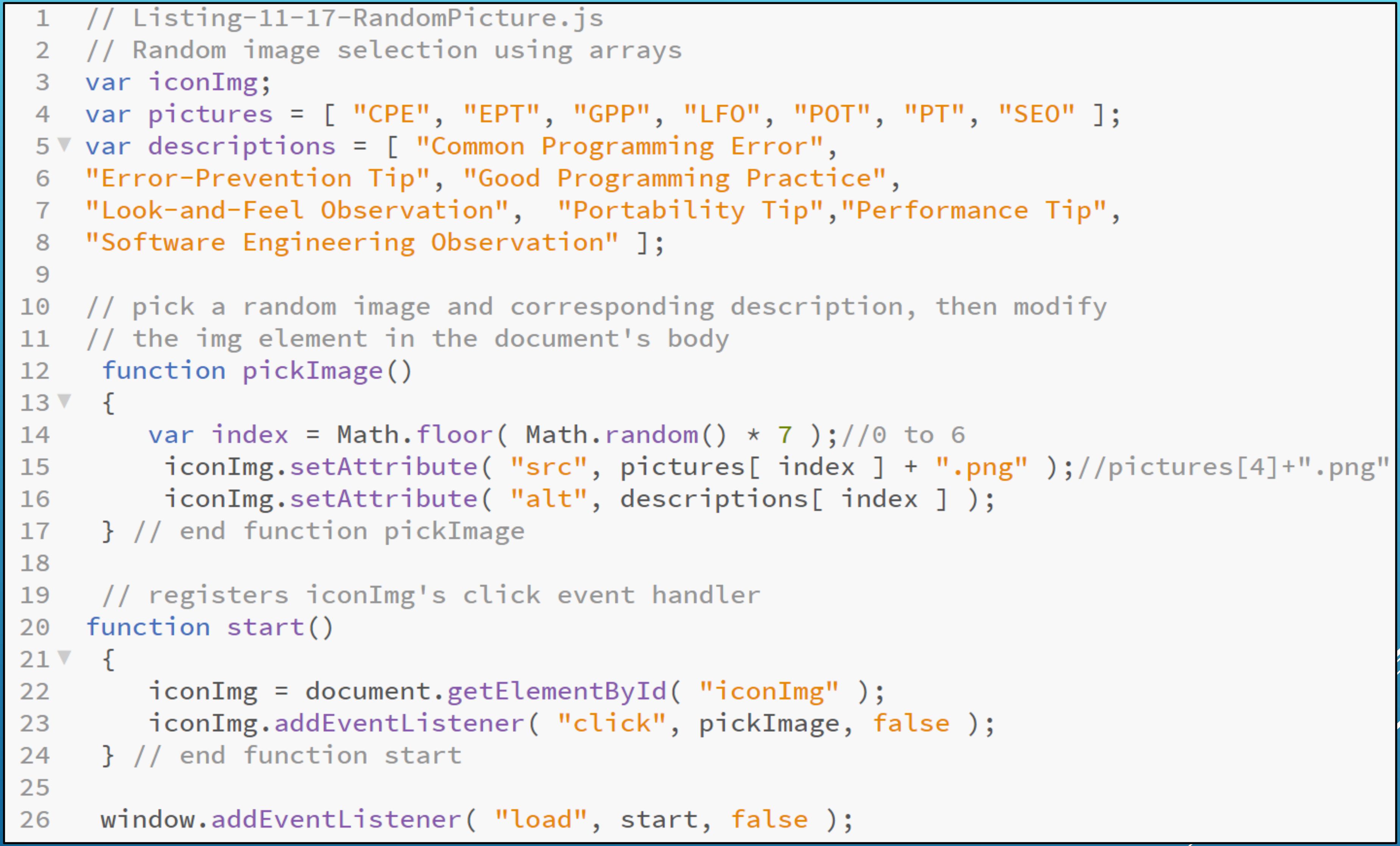
```html
<!DOCTYPE html>
<html>
 <head>
    <meta charset = "utf-8">
    <title>Random Image Generator</title>

     <script src = "Listing-11-17-RandomPicture.js"></script>
 </head>
 <body>
     <img id = "iconImg" src = "CPE.png" alt = "Common Programming Error">
 </body>
 </html>
```

Listing-11-16.html

RANDOM IMAGE GENERATION USING ARRAYS

- This version uses an array of pictures to store the names of the image files as strings.
- The script updates the img element's src attribute with the image filename at the randomly selected position in the pictures array.
- In addition, we update the alt attribute with an appropriate description of the image from the descriptions array.
- The javascript file Listing-11-17-RandomPicture.js declares the array pictures in line 4 and initializes it with thenames of seven image files—the files contain our bug icons that we associate with our programming tips.
- Lines 5–8 create a separate array descriptions that contains the alt text for the corresponding images in the pictures array.
- When the user clicks the img element in the document, function pickImage (lines 12–17) is called to pick a random integer index from 0 to 6 and display the associated image.
- Line 15 uses that index to get a value from the pictures array, appends ".png" to it, then sets the img element's src attribute to the new image file name.
- Similarly, line 16 uses the index to get the corresponding text from the descriptions array and assigns that text to the img element's alt attribute.

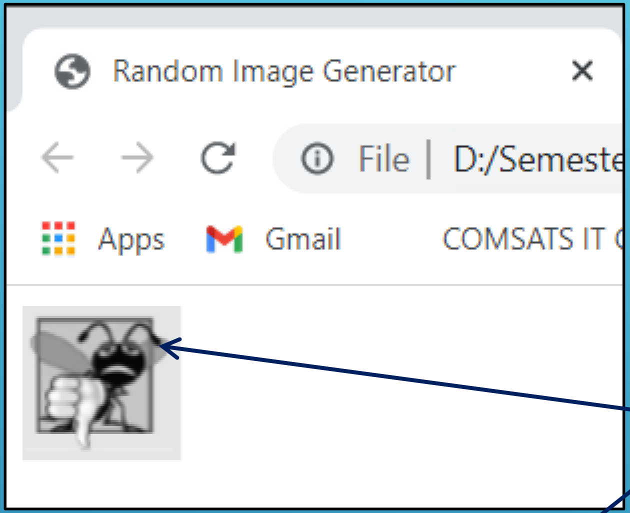
RANDOM IMAGE GENERATION USING ARRAYS

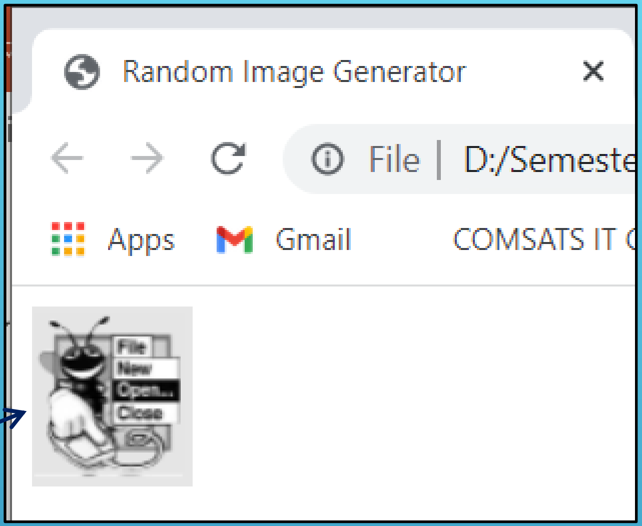


```javascript
// Listing-11-17-RandomPicture.js
// Random image selection using arrays
var iconImg;
var pictures = [ "CPE", "EPT", "GPP", "LFO", "POT", "PT", "SEO" ];
var descriptions = [ "Common Programming Error",
"Error-Prevention Tip", "Good Programming Practice",
"Look-and-Feel Observation",  "Portability Tip","Performance 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 );//0 to 6
     iconImg.setAttribute( "src", pictures[ index ] + ".png" );//pictures[4]+".png"
     iconImg.setAttribute( "alt", descriptions[ index ] );
 } // end function pickImage

 // registers iconImg's click event handler
function start()
 {
    iconImg = document.getElementById( "iconImg" );
    iconImg.addEventListener( "click", pickImage, false );
 } // end function start

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

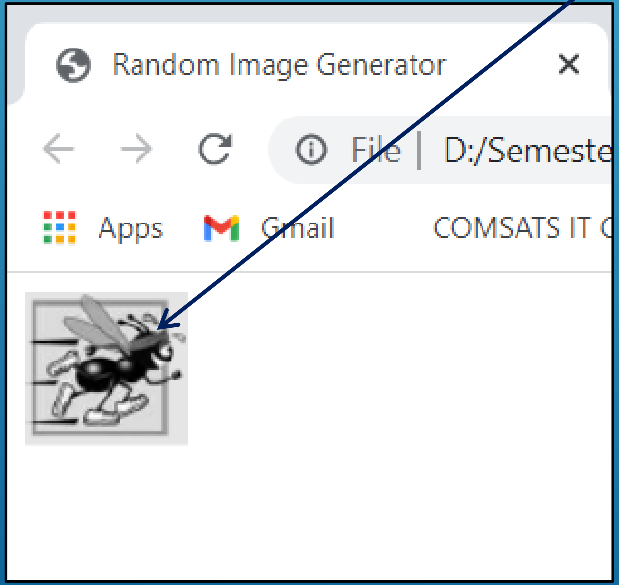
Listing-11-17-RandomPicture.js

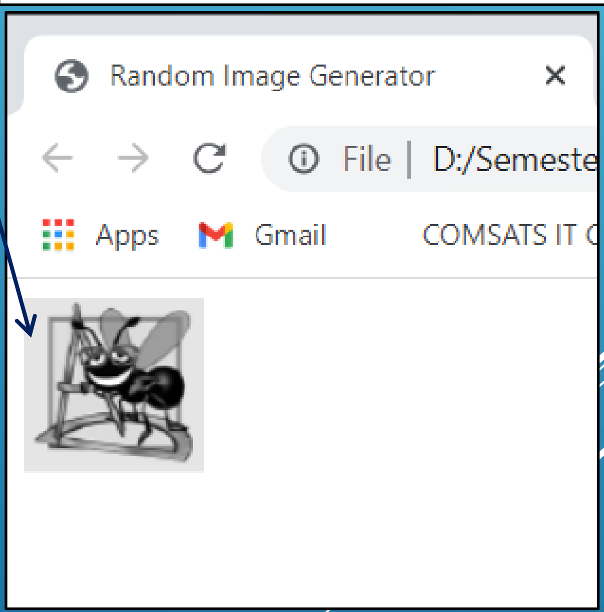
RANDOM IMAGE GENERATION USING ARRAYS





When user clicks on image the random image will be generated





Output-Listing-22.14