JavaScript: Arrays

10.1 Introduction

- Arrays
  - Data structures consisting of related data items
- JavaScript arrays
  - “Dynamic” entities that can change size after they are created

10.2 Arrays

- An array is a group of memory locations
  - All have the same name and normally are of the same type (although this attribute is not required in JavaScript)
- Each individual location is called an element
- We may refer to any one of these elements by giving the array's name followed by the position number of the element in square brackets ([])

10.2 Arrays (Cont.)

- The first element in every array is the zeroth element
- The **i**th element of array **c** is referred to as **c[i-1]**
- Array names follow the same conventions as other identifiers
- A subscripted array name
  - Can be used on the left side of an assignment to place a new value into an array element
  - Can be used on the right side of an assignment operation to use its value
- Every array in JavaScript knows its own length, which it stores in its length attribute and can be found with the expression **arrayname.length**
  - **c.length**

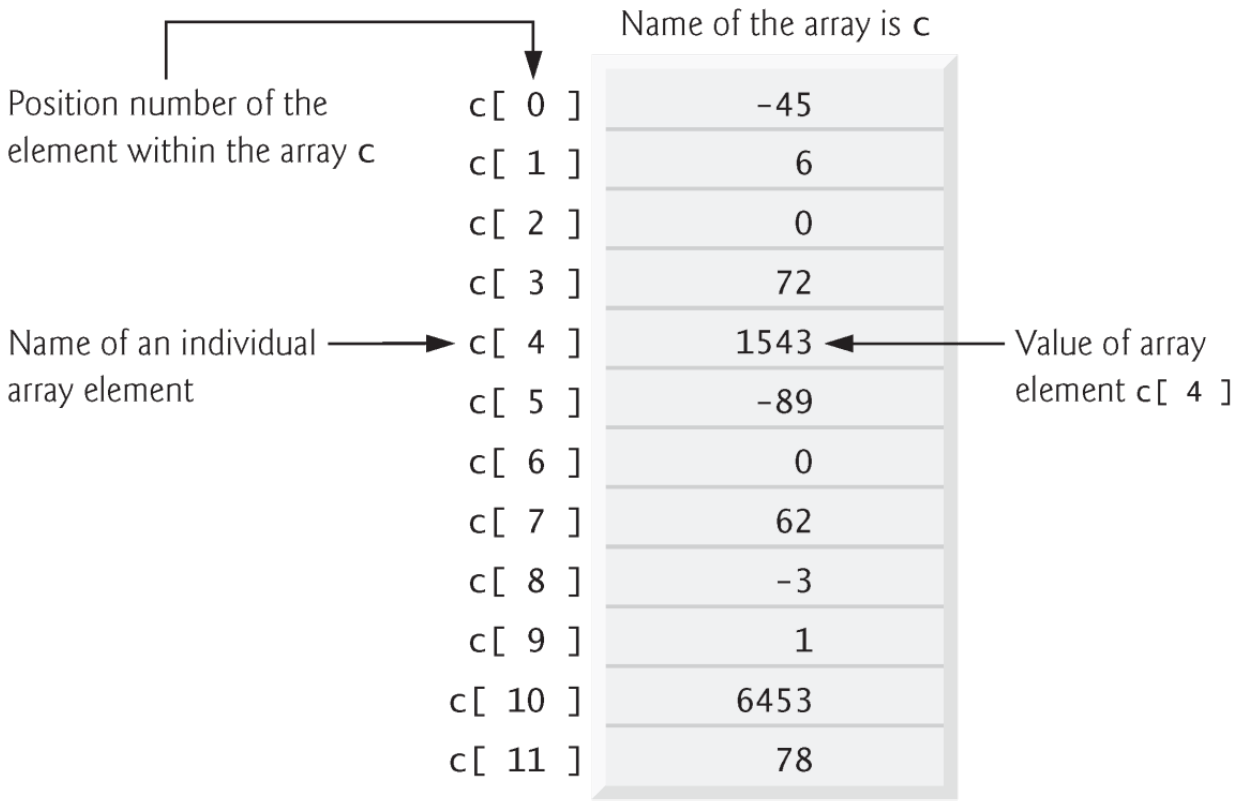


Fig. 10.1 | Array with 12 elements.

10.2 Arrays (Cont.)

- The array's 12 elements are referred to as c[0], c[1], c[2], …, c[11]
- The **value** of c[0] is -45, the value of c[1] is 6, the value of c[2] is 0, the value of c[7] is 62 and the value of c[11] is 78
- The following statement calculates the sum of the values contained in the first three elements of array c and stores the result in variable sum
  - **sum = c[0] + c[1] + c[2];**

Operators	Associativity	Type
`()  []  .`	left to right	highest
`++  --  !`	right to left	unary
`*  /  %`	left to right	multiplicative
`+  -`	left to right	additive
`<  <=  >  >=`	left to right	relational
`==  !=`	left to right	equality
`&&`	left to right	logical AND
`||`	left to right	logical OR
`?:`	right to left	conditional
`=  +=  -=  *=  /=  %=`	right to left	assignment

Fig. 10.2 | Precedence and associativity of the operators discussed so far.

10.3 Declaring and Allocating Arrays

- JavaScript arrays are Array objects
- 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);**

10.3 Declaring and Allocating Arrays (Cont.)

- 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, the elements are *not* initialized
  - They have the value undefined

10.4 Examples Using Arrays

- Zero-based counting is usually used to iterate through arrays
- JavaScript reallocates an Array when a value is assigned to an element that is **outside** the bounds of the original Array

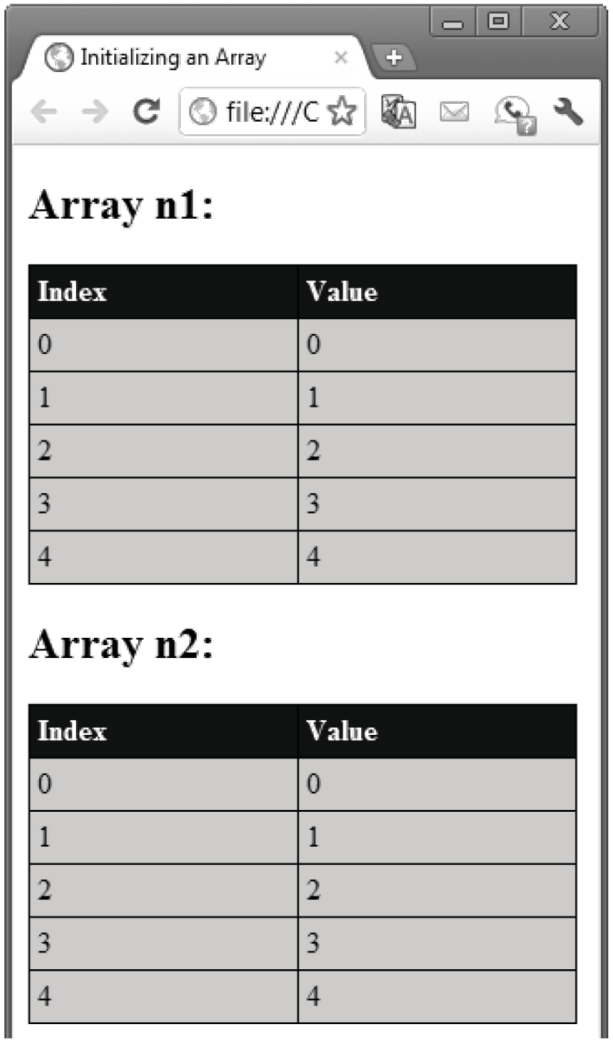
Software Engineering Observation 10.1

It's considered good practice to separate your JavaScript scripts into separate files so that they can be reused in multiple web pages.

```
<!DOCTYPE html>

<!-- Fig. 10.3: InitArray.html -->
<!-- Web page for showing the results of initializing arrays. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Initializing an Array</title>
      <link rel = "stylesheet" type = "text/css" href = "tablestyle.css">
      <script src = "InitArray.js"></script>
   </head>
   <body>
      <div id = "output1"></div>
      <div id = "output2"></div>
   </body>
</html>
```

Fig. 10.3 | Web page for showing the results of initializing arrays. (Part 1 of 2.)


Initializing an Array
file:///C
Array n1:
Index
Value
0
0
1
1
2
2
3
3
4
4
Array n2:
Index
Value
0
0
1
1
2
2
3
3
4
4

```
1  // Fig. 10.4: InitArray.js
2  // Create two arrays, initialize their elements and display them
3  function start()
4  {
5     var n1 = new Array( 5 ); // allocate five-element array
6     var n2 = new Array(); // allocate empty array
7
8     // assign values to each element of array n1
9     var length = n1.length; // get array's length once before the loop
10
11    for ( var i = 0; i < length; ++i )
12    {
13       n1[ i ] = i;
14    } // end for
15
16    // create and initialize five elements in array n2
17    for ( i = 0; i < 5; ++i )
18    {
19       n2[ i ] = i;
20    } // end for
21
22    outputArray( "Array n1:", n1, document.getElementById( "output1" ) );
23    outputArray( "Array n2:", n2, document.getElementById( "output2" ) );
24 } // end function start
```

Fig. 10.4 | Create two arrays, initialize their elements and display them. (Part 1 of 2.)

```
25
26    // output the heading followed by a two-column table
27    // containing indices and elements of "theArray"
28    function outputArray( heading, theArray, output )
29    {
30       var content = "<h2>" + heading + "</h2><table>" +
31          "<thead><th>Index</th><th>Value</th></thead><tbody>";
32
33       // output the index and value of each array element
34       var length = theArray.length; // get array's length once before loop
35
36       for ( var i = 0; i < length; ++i )
37       {
38          content += "<tr><td>" + i + "</td><td>" + theArray[ i ] +
39             "</td></tr>";
40       } // end for
41
42       content += "</tbody></table>";
43       output.innerHTML = content; // place the table in the output element
44    } // end function outputArray
45
46    window.addEventListener( "load", start, false );
```

Fig. 10.4 | Create two arrays, initialize their elements and display them. (Part 2 of 2.)

Software Engineering Observation 10.2

JavaScript automatically reallocates an array when a value is assigned to an element that's outside the bounds of the array. Elements between the last element of the original array and the new element are `undefined`.

Error-Prevention Tip 10.1

When accessing array elements, the index values should never go below 0 and should be less than the number of elements in the array (i.e., one less than the array's size), unless it's your explicit intent to grow the array by assigning a value to a nonexistent element.

10.4 Examples Using Arrays (Cont.)

- Using an Initializer List
- Arrays can be created using a comma-separated initializer list enclosed in square brackets ([])
  - The array's size is determined by the number of values in the initializer list
    - **var n = [10, 20, 30, 40, 50];**
- The initial values of an array can be specified as arguments in the parentheses following new Array
  - The size of the array is determined by the number of values in parentheses
    - **var n = new Array(10, 20, 30, 40, 50);**

10.4.2 Initializing Arrays with Initializer Lists

- 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
  - **var n = [10, 20, , 40, 50];**
- **n** is creates a five-element array in which the third element (**n[2]**) has the value undefined
- The example in Figs. 10.5–10.6 creates three Array objects to demonstrate initializing arrays with initializer lists
- Figure 10.5 is nearly identical to Fig. 10.3 but provides three **divs** in its body element for displaying this example's arrays

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 10.5: InitArray2.html -->
4  <!-- Web page for showing the results of initializing arrays. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Initializing an Array</title>
9        <link rel = "stylesheet" type = "text/css" href = "tablestyle.css">
10       <script src = "InitArray2.js"></script>
11    </head>
12    <body>
13       <div id = "output1"></div>
14       <div id = "output2"></div>
15       <div id = "output3"></div>
16    </body>
17 </html>
```

Fig. 10.5 | Web page for showing the results of initializing arrays. (Part 1 of 2.)

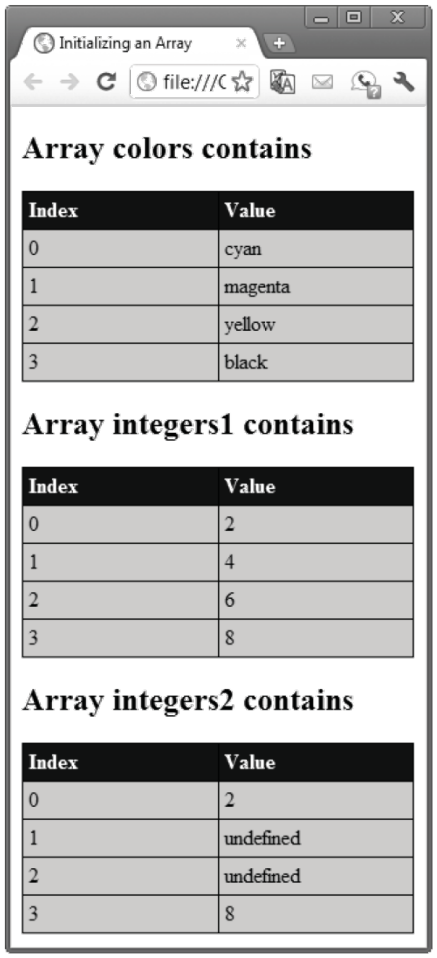


Fig. 10.5 | Web page for showing the results of initializing arrays. (Part 2 of 2.)

```
1  // Fig. 10.6: InitArray2.js
2  // Initializing arrays with initializer lists.
3  function start()
4  {
5     // Initializer list specifies the number of elements and
6     // a value for each element.
7     var colors = new Array( "cyan", "magenta","yellow", "black" );
8     var integers1 = [ 2, 4, 6, 8 ];
9     var integers2 = [ 2, , , 8 ];
10
11    outputArray( "Array colors contains", colors,
12       document.getElementById( "output1" ) );
13    outputArray( "Array integers1 contains", integers1,
14       document.getElementById( "output2" ) );
15    outputArray( "Array integers2 contains", integers2,
16       document.getElementById( "output3" ) );
17 } // end function start
18
```

Fig. 10.6 | Initializing arrays with initializer lists. (Part 1 of 2.)

```
19  // output the heading followed by a two-column table
20  // containing indices and elements of "theArray"
21  function outputArray( heading, theArray, output )
22  {
23     var content = "<h2>" + heading + "</h2><table>" +
24        "<thead><th>Index</th><th>Value</th></thead><tbody>";
25
26     // output the index and value of each array element
27     var length = theArray.length; // get array's length once before loop
28
29     for ( var i = 0; i < length; ++i )
30     {
31        content += "<tr><td>" + i + "</td><td>" + theArray[ i ] +
32           "</td></tr>";
33     } // end for
34
35     content += "</tbody></table>";
36     output.innerHTML = content; // place the table in the output element
37  } // end function outputArray
38
39  window.addEventListener( "load", start, false );
```

Fig. 10.6 | Initializing arrays with initializer lists. (Part 2 of 2.)

10.4.3 Summing the Elements of an Array with for and for...in

- The example in Figs. 10.7–10.8 sums an array's elements and displays the results
- The document in Fig. 10.7 shows the results of the script in Fig. 10.8
- JavaScript's **for...in** Repetition Statement
  - Enables a script to perform a task for each element in an array
  - **for (var element in theArray)**

Error-Prevention Tip 10.2

When iterating over all the elements of an array, use a `for...in` statement to ensure that you manipulate only the existing elements. The `for...in` statement skips any undefined elements in the array.

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 10.7: SumArray.html -->
4  <!-- HTML5 document that displays the sum of an array's elements. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Sum Array Elements</title>
9        <script src = "SumArray.js"></script>
10    </head>
11    <body>
12       <div id = "output"></div>
13    </body>
14 </html>
```

Fig. 10.7 | HTML5 document that displays the sum of an array's elements.

```
1  // Fig. 10.8: SumArray.js
2  // Summing the elements of an array with for and for...in
3  function start()
4  {
5     var theArray = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ];
6     var total1 = 0, total2 = 0;
7
8     // iterates through the elements of the array in order and adds
9     // each element's value to total1
10    var length = theArray.length; // get array's length once before loop
11
12    for ( var i = 0; i < length; ++i )
13    {
14       total1 += theArray[ i ];
15    } // end for
16
17    var results = "<p>Total using indices: " + total1 + "</p>";
18
```

Fig. 10.8 | Summing the elements of an array with `for` and `for...in`. (Part 1 of 2.)

```
19      // iterates through the elements of the array using a for... in
20      // statement to add each element's value to total2
21      for ( var element in theArray )
22      {
23         total2 += theArray[ element ];
24      } // end for
25
26      results += "<p>Total using for...in: " + total2 + "</p>";
27      document.getElementById( "output" ).innerHTML = results;
28   } // end function start
29
30   window.addEventListener( "load", start, false );
```

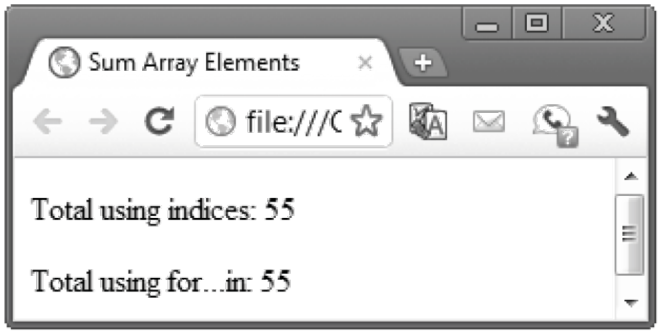


Fig. 10.8 | Summing the elements of an array with `for` and `for...in`. (Part 2 of 2.)

10.4.4 Using the Elements of an Array as Counters

- The example in Section 9.5.3 allowed the user to roll 12 dice at a time and kept statistics showing the number of times and the percentage of the time each face occurred
- An array version of this example is shown in Figs. 10.9–10.10
- We divided the example into three files
  - style.css contains the styles (not shown here)
  - RollDice.html (Fig. 10.9) contains the HTML5 document and
  - RollDice.js (Fig. 10.10) contains the JavaScript

```
<!DOCTYPE html>

<!-- Fig. 10.9: RollDice.html -->
<!-- HTML5 document for the dice-rolling example. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Roll a Six-Sided Die 6000000 Times</title>
      <link rel = "stylesheet" type = "text/css" href = "style.css">
      <script src = "RollDice.js"></script>
   </head>
   <body>
      <p><img id = "die1" src = "blank.png" alt = "die 1 image">
         <img id = "die2" src = "blank.png" alt = "die 2 image">
         <img id = "die3" src = "blank.png" alt = "die 3 image">
         <img id = "die4" src = "blank.png" alt = "die 4 image">
         <img id = "die5" src = "blank.png" alt = "die 5 image">
         <img id = "die6" src = "blank.png" alt = "die 6 image"></p>
```

Fig. 10.9 | HTML5 document for the dice-rolling example. (Part 1 of 3.)

```
19          <p><img id = "die7" src = "blank.png" alt = "die 7 image">
20             <img id = "die8" src = "blank.png" alt = "die 8 image">
21             <img id = "die9" src = "blank.png" alt = "die 9 image">
22             <img id = "die10" src = "blank.png" alt = "die 10 image">
23             <img id = "die11" src = "blank.png" alt = "die 11 image">
24             <img id = "die12" src = "blank.png" alt = "die 12 image"></p>
25          <form action = "#">
26             <input id = "rollButton" type = "button" value = "Roll Dice">
27          </form>
28          <div id = "frequencyTableDiv"></div>
29       </body>
30    </html>
```

Fig. 10.9 | HTML5 document for the dice-rolling example. (Part 2 of 3.)

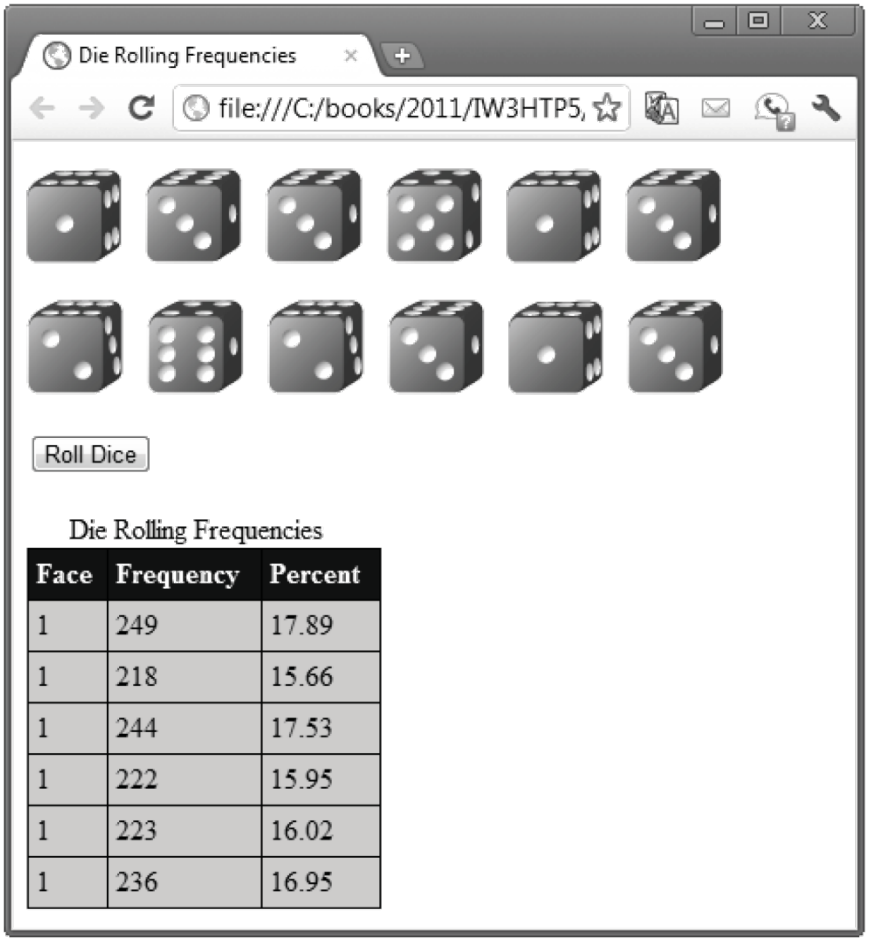


Fig. 10.9 | HTML5 document for the dice-rolling example. (Part 3 of

```
1  // Fig. 10.10: RollDice.js
2  // Summarizing die-rolling frequencies with an array instead of a switch
3  var frequency = [ , 0, 0, 0, 0, 0, 0 ]; // frequency[0] uninitialized
4  var totalDice = 0;
5  var dieImages = new Array(12); // array to store img elements
6
7  // get die img elements
8  function start()
9  {
10    var button = document.getElementById( "rollButton" );
11    button.addEventListener( "click", rollDice, false );
12    var length = dieImages.length; // get array's length once before loop
13
14    for ( var i = 0; i < length; ++i )
15    {
16       dieImages[ i ] = document.getElementById( "die" + (i + 1) );
17    } // end for
18 } // end function start
19
```

Fig. 10.10 | Summarizing die-rolling frequencies with an array instead of a switch. (Part 1 of 5.)

```
20  // roll the dice
21  function rollDice()
22  {
23     var face;  // face rolled
24     var length = dieImages.length;
25
26     for ( var i = 0; i < length; ++i )
27     {
28        face = Math.floor( 1 + Math.random() * 6 );
29        tallyRolls( face ); // increment a frequency counter
30        setImage( i, face ); // display appropriate die image
31        ++totalDice; // increment total
32     } // end for
33
34     updateFrequencyTable();
35  } // end function rollDice
36
37  // increment appropriate frequency counter
38  function tallyRolls( face )
39  {
40     ++frequency[ face ]; // increment appropriate counte
41  } // end function tallyRolls
42
```

Fig. 10.10 | Summarizing die-rolling frequencies with an array instead of a `switch`. (Part 2 of 5.)

```
43  // set image source for a die
44  function setImage( dieImg )
45  {
46     dieImages[ dieNumber ].setAttribute( "src", "die" + face + ".png" );
47     dieImages[ dieNumber ].setAttribute( "alt",
48        "die with " + face + " spot(s)" );
49  } // end function setImage
50
```

Fig. 10.10 | Summarizing die-rolling frequencies with an array instead of a `switch`. (Part 3 of 5.)

```
51  // update frequency table in the page
52  function updateFrequencyTable()
53  {
54     var results = "<table><caption>Die Rolling Frequencies</caption>" +
55        "<thead><th>Face</th><th>Frequency</th>" +
56        "<th>Percent</th></thead><tbody>";
57     var length = frequency.length;
58
59     // create table rows for frequencies
60     for ( var i = 1; i < length; ++i )
61     {
62        results += "<tr><td>1</td><td>" + i + "</td><td>" +
63           formatPercent(frequency[ i ] / totalDice) + "</td></tr>";
64     } // end for
65
66     results += "</tbody></table>";
67     document.getElementById( "frequencyTableDiv" ).innerHTML = results;
68  } // end function updateFrequencyTable
69
```

Fig. 10.10 | Summarizing die-rolling frequencies with an array instead of a `switch`. (Part 4 of 5.)

```
70   // format percentage
71   function formatPercent( value )
72   {
73      value *= 100;
74      return value.toFixed(2);
75   } // end function formatPercent
76
77   window.addEventListener( "load", start, false );
```

Fig. 10.10 | Summarizing die-rolling frequencies with an array instead of a `switch`. (Part 5 of 5.)

10.5 Random Image Generator Using Arrays

- In Chapter 9, the random image generator required image files to be named with the word die followed by a number from 1 to 6 and the file extension .png (e.g, die1.png)
- The example in Figs. 10.11–10.12 does not require the image filenames to contain integers in sequence

10.5 Random Image Generator Using Arrays

- It uses an array pictures to store the names of the image files as strings
  - Each time you click the image in the document, the script generates a random integer and uses it as an index into the pictures array
  - The script updates the **img** element's src attribute with the image filename at the randomly selected position in the pictures array
  - We update the alt attribute with an appropriate description of the image from the descriptions array

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 10.11: RandomPicture.html -->
4  <!-- HTML5 document that displays randomly selected images. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Random Image Generator</title>
9        <script src = "RandomPicture.js"></script>
10    </head>
11    <body>
12       <img id = "image" src = "CPE.png" alt = "Common Programming Error">
13    </body>
14 </html>
```

Fig. 10.11 | HTML5 document that displays randomly selected images. (Part 1 of 2.)

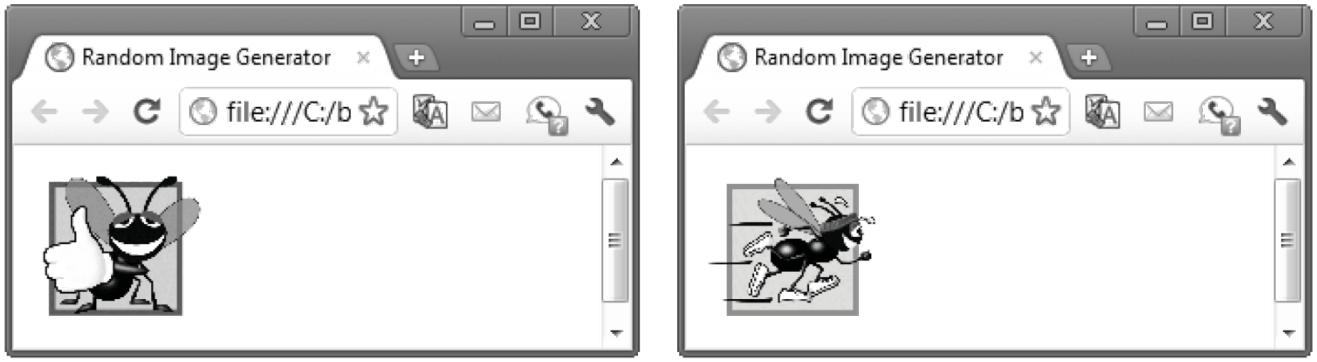


Fig. 10.11 | HTML5 document that displays randomly selected images. (Part 2 of 2.)

```
15  // Fig. 10.12: RandomPicture2.js
16  // Random image selection using arrays
17  var iconImg;
18  var pictures = [ "CPE", "EPT", "GPP", "GUI", "PERF", "PORT", "SEO" ];
19  var descriptions = [ "Common Programming Error",
20     "Error-Prevention Tip", "Good Programming Practice",
21     "Look-and-Feel Observation", "Performance Tip", "Portability Tip",
22     "Software Engineering Observation" ];
23
24  // pick a random image and corresponding description, then modify
25  // the img element in the document's body
26  function pickImage()
27  {
28     var index = Math.floor( Math.random() * 7 );
29     iconImg.setAttribute( "src", pictures[ index ] + ".png" );
30     iconImg.setAttribute( "alt", descriptions[ index ] );
31  } // end function pickImage
32
33  // registers iconImg's click event handler
34  function start()
35  {
36     iconImg = document.getElementById( "iconImg" );
37     iconImg.addEventListener( "click", pickImage, false );
38  } // end function start
39
40  window.addEventListener( "load", start, false );
```

Fig. 10.12 | Random image selection using arrays.

10.6 References and Reference Parameters

- Two ways to pass arguments to functions (or methods)
  - **pass-by-value**
  - **pass-by-reference**
- **Pass-by-value**
  - A copy of the argument's value is made and is passed to the called function
  - In JavaScript, **numbers**, **boolean** values and **strings** are passed to functions by value

10.6 References and Reference Parameters

- **Pass-by-reference**
  - The caller gives the called function access to the caller's data and allows the called function to modify the data if it so chooses
  - Can improve performance because it can eliminate the overhead of copying large amounts of data, but it can weaken security because the called function can access the caller's data
  - All **objects** are passed to functions by reference

Error-Prevention Tip 10.3

With pass-by-value, changes to the copy of the value received by the called function do not affect the original variable's value in the calling function. This prevents the accidental side effects that hinder the development of correct and reliable software systems.

Software Engineering Observation 10.3

When information is returned from a function via a `return` statement, numbers and boolean values are returned by value (i.e., a copy is returned), and objects are returned by reference (i.e., a reference to the object is returned). When an object is passed-by-reference, it's not necessary to return the object, because the function operates on the original object in memory.

10.7 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
  - **var hourlyTemperatures = new Array(24);**
  - **modifyArray(hourlyTemperatures);**
  - We pass an array object into a function!
- Although entire arrays are passed by reference, **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**

10.7 Passing Arrays to Functions (Cont.)

- To pass an array element to a function, use the indexed name of the element as an argument in the function call
- join method of an Array
  - Return 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

```
<!DOCTYPE html>

<!-- Fig. 10.13: PassArray.html -->
<!-- HTML document that demonstrates passing arrays and -->
<!-- individual array elements to functions. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Arrays as Arguments</title>
      <link rel = "stylesheet" type = "text/css" href = "style.css">
      <script src = "PassArray.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>
```

Fig. 10.13 | HTML document that demonstrates passing arrays and individual array elements to functions. (Part 1 of 2.)

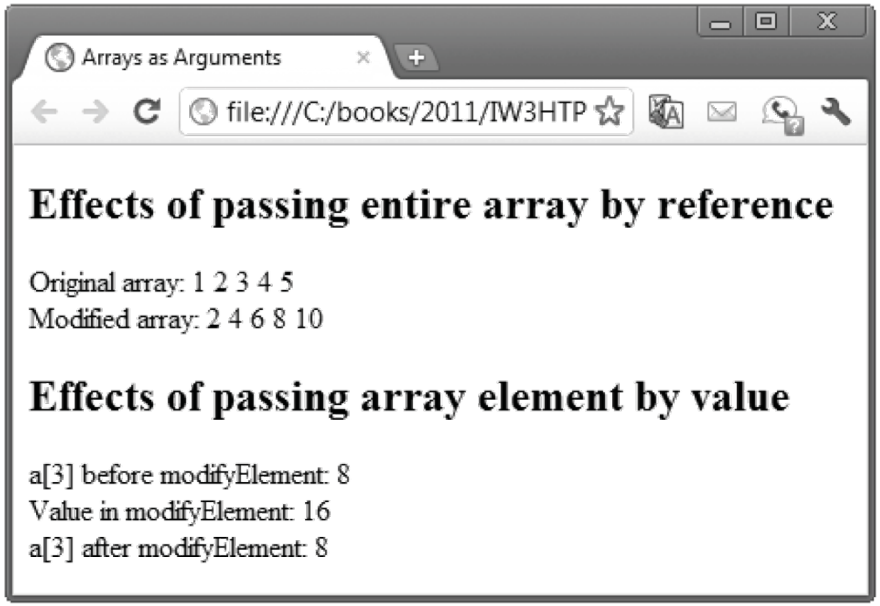


Fig. 10.13 | HTML document that demonstrates passing arrays and individual array elements to functions. (Part 2 of 2.)

Software Engineering Observation 10.4

JavaScript does not check the number of arguments or types of arguments that are passed to a function. It's possible to pass any number of values to a function.

```
1  // Fig. 10.14: PassArray.js
2  // Passing arrays and individual array elements to functions.
3  function start()
4  {
5     var a = [ 1, 2, 3, 4, 5 ];
6
7     // passing entire array
8     outputArray( "Original array: ", a,
9        document.getElementById( "originalArray" ) );
10    modifyArray( a );  // array a passed by reference
11    outputArray( "Modified array: ", a,
12       document.getElementById( "modifiedArray" ) );
13
14    // passing individual array element
15    document.getElementById( "originalElement" ).innerHTML =
16       "a[3] before modifyElement: " + a[ 3 ];
17    modifyElement( a[ 3 ] ); // array element a[3] passed by value
18    document.getElementById( "modifiedElement" ).innerHTML =
19       "a[3] after modifyElement: " + a[ 3 ];
20 } // end function start()
21
```

Fig. 10.14 | Passing arrays and individual array elements to functions. (Part 1 of 2.)

```
22  // outputs heading followed by the contents of "theArray"
23  function outputArray( heading, theArray, output )
24  {
25     output.innerHTML = heading + theArray.join( " " );
26  } // end function outputArray
27
28  // function that modifies the elements of an array
29  function modifyArray( theArray )
30  {
31     for ( var j in theArray )
32     {
33        theArray[ j ] *= 2;
34     } // end for
35  } // end function modifyArray
36
37  // function that modifies the value passed
38  function modifyElement( e )
39  {
40     e *= 2; // scales element e only for the duration of the function
41     document.getElementById( "inModifyElement" ).innerHTML =
42        "Value in modifyElement: " + e;
43  } // end function modifyElement
44
45  window.addEventListener( "load", start, false );
```

Fig. 10.14 | Passing arrays and individual array elements to functions. (Part 2 of 2.)

10.8 Sorting Arrays with Array Method Sort

- Sorting data
  - Putting data in a particular order, such as ascending or descending
  - One of the most important computing functions
- Array object in JavaScript has a built-in method sort
  - With no arguments, the method uses string comparisons to determine the sorting order of the array elements
  - Method sort takes as its argument the name of a function that compares its two arguments and returns
    - A negative value if the first argument is less than the second argument,
    - Zero if the arguments are equal, or a positive value if the first argument is greater than the second
    - http://www.w3schools.com/jsref/jsref_sort.asp

```
<!DOCTYPE html>

<!-- Fig. 10.15: Sort.html -->
<!-- HTML5 document that displays the results of sorting an array. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Array Method sort</title>
      <link rel = "stylesheet" type = "text/css" href = "style.css">
      <script src = "Sort.js"></script>
   </head>
   <body>
      <h1>Sorting an Array</h1>
      <p id = "originalArray"></p>
      <p id = "sortedArray"></p>
   </body>
</html>
```

Fig. 10.15 | HTML5 document that displays the results of sorting an array. (Part 1 of 2.)

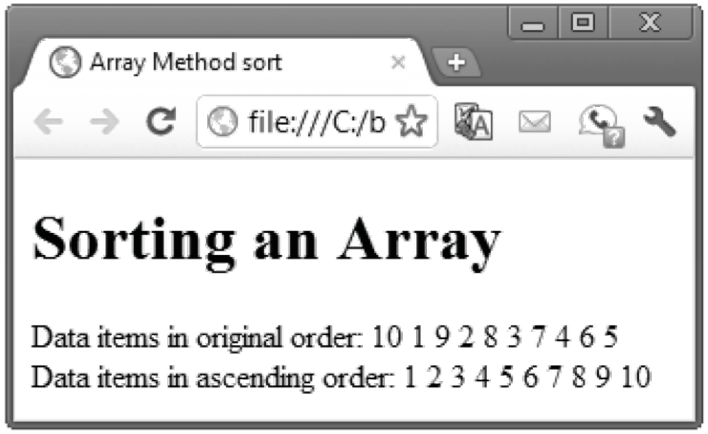


Fig. 10.15 | HTML5 document that displays the results of sorting an array. (Part 2 of 2.)

```
1  // Fig. 10.16: Sort.js
2  // Sorting an array with sort.
3  function start()
4  {
5     var a = [ 10, 1, 9, 2, 8, 3, 7, 4, 6, 5 ];
6
7     outputArray( "Data items in original order: ", a,
8        document.getElementById( "originalArray" ) );
9     a.sort( compareIntegers );  // sort the array
10    outputArray( "Data items in ascending order: ", a,
11       document.getElementById( "sortedArray" ) );
12 } // end function start
13
14 // output the heading followed by the contents of theArray
15 function outputArray( heading, theArray, output )
16 {
17    output.innerHTML = heading + theArray.join( " " );
18 } // end function outputArray
19
```

Fig. 10.16 | Sorting an array with `sort`. (Part 1 of 2.)

```
20   // comparison function for use with sort
21   function compareIntegers( value1, value2 )
22   {
23      return parseInt( value1 ) - parseInt( value2 );
24   } // end function compareIntegers
25
26   window.addEventListener( "load", start, false );
```

Fig. 10.16 | Sorting an array with `sort`. (Part 2 of 2.)

Software Engineering Observation 10.5

Functions in JavaScript are considered to be data. Therefore, functions can be assigned to variables, stored in arrays and passed to functions just like other data types.

10.9 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
  - http://www.w3schools.com/jsref/jsref_indexof_array.asp

10.9 Searching Arrays with Array Method indexOf (Cont.)

- Every input element has a value property that can be used to get or set the element's value
- 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 and the methods search the entire 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

```
<!DOCTYPE html>

<!-- Fig. 10.17: search.html -->
<!-- HTML5 document for searching an array with indexOf. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Search an Array</title>
      <script src = "search.js"></script>
   </head>
   <body>
      <form action  = "#">
         <p><label>Enter integer search key:
            <input id = "inputVal" type = "number"></label>
            <input id = "searchButton" type = "button" value = "Search">
         </p>
         <p id = "result"></p>
      </form>
   </body>
</html>
```

Fig. 10.17 | HTML5 document for searching an array with `indexOf`. (Part 1 of 2.)

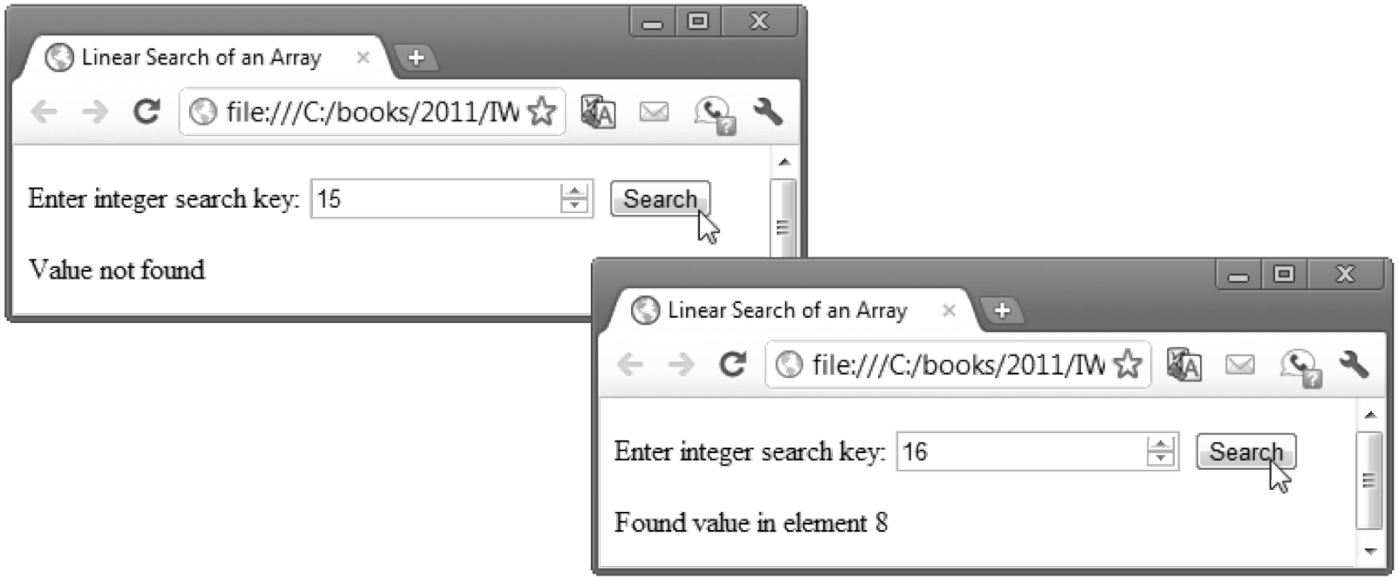


Fig. 10.17 | HTML5 document for searching an array with `indexOf`. (Part 2 of 2.)

```
1  // Fig. 10.18: search.js
2  // Search an array with indexOf.
3  var a = new Array( 100 );  // create an array
4
5  // fill array with even integer values from 0 to 198
6  for ( var i = 0; i < a.length; ++i )
7  {
8     a[ i ] = 2 * i;
9  } // end for
10
11 // function called when "Search" button is pressed
12 function buttonPressed()
13 {
14    // get the input text field
15    var inputVal = document.getElementById( "inputVal" );
16
17    // get the result paragraph
18    var result = document.getElementById( "result" );
19
20    // get the search key from the input text field then perform the search
21    var searchKey = parseInt( inputVal.value );
22    var element = a.indexOf( searchKey );
23
```

Fig. 10.18 | Search an array with `indexOf`.

```
24      if ( element != -1 )
25      {
26         result.innerHTML = "Found value in element " + element;
27      } // end if
28      else
29      {
30         result.innerHTML = "Value not found";
31      } // end else
32   } // end function buttonPressed
33
34   // register searchButton's click event handler
35   function start()
36   {
37      var searchButton = document.getElementById( "searchButton" );
38      searchButton.addEventListener( "click", buttonPressed, false );
39   } // end function start
40
41   window.addEventListener( "load", start, false );
```

Fig. 10.18 | Search an array with `indexOf`.

10.10 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

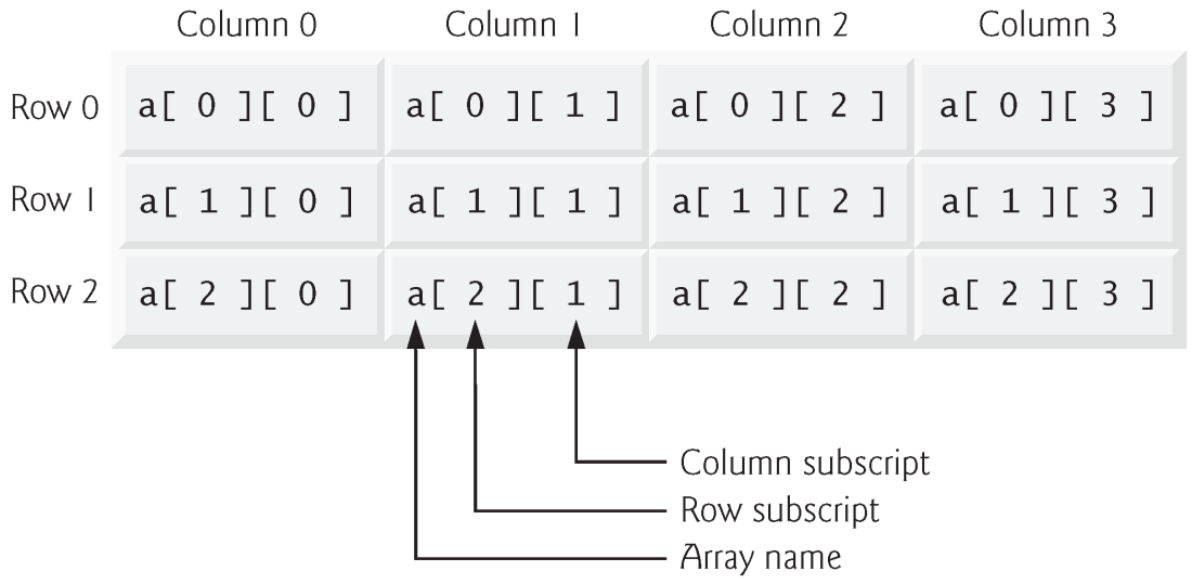


Fig. 10.19 | Two-dimensional array with three rows and four columns.

10.10 Multidimensional Arrays (Cont.)

- 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 initializer
  - 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
  - **var b = [[1, 2], [3, 4, 5]];**

10.10 Multidimensional Arrays (Cont.)

- A multidimensional array in which each row has a different number of columns can be allocated dynamically with operator **new**
  - **var b;**

    b = new Array(2); // allocate two rows

    b[0] = new Array(5); // allocate columns for row 0

    b[1] = new Array(3); // allocate columns for row 1

```
<!DOCTYPE html>

<!-- Fig. 10.20: InitArray3.html -->
<!-- HTML5 document showing multidimensional array initialization. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Multidimensional Arrays</title>
      <link rel = "stylesheet" type = "text/css" href = "style.css">
      <script src = "InitArray3.js"></script>
   </head>
   <body>
      <h2>Values in array1 by row</h2>
      <div id = "output1"></div>
      <h2>Values in array2 by row</h2>
      <div id = "output2"></div>
   </body>
</html>
```

Fig. 10.20 | HTML5 document showing multidimensional array initialization. (Part 1 of 2.)

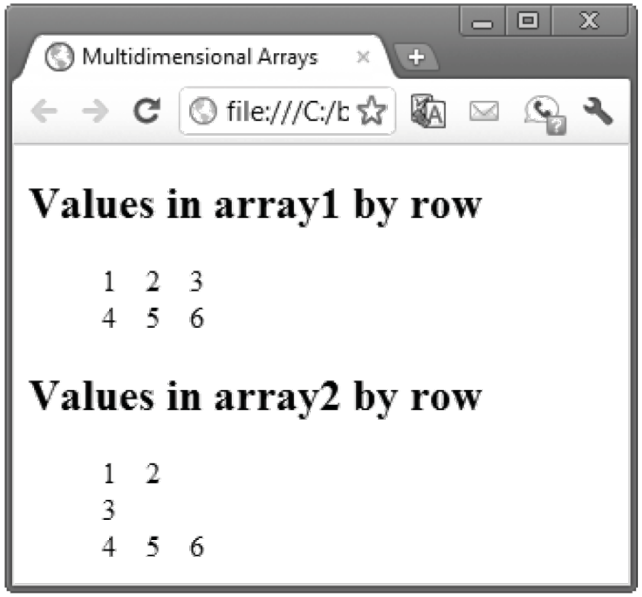


Fig. 10.20 | HTML5 document showing multidimensional array initialization. (Part 2 of 2.)

```
1  // Fig. 10.21: InitArray3.js
2  // Initializing multidimensional arrays.
3  function start()
4  {
5     var array1 = [ [ 1, 2, 3 ], // row 0
6                    [ 4, 5, 6 ] ]; // row 1
7     var array2 = [ [ 1, 2 ], // row 0
8                    [ 3 ], // row 1
9                    [ 4, 5, 6 ] ]; // row 2
10
11    outputArray( "Values in array1 by row", array1,
12       document.getElementById( "output1" ) );
13    outputArray( "Values in array2 by row", array2,
14       document.getElementById( "output2" ) );
15 } // end function start
16
```

Fig. 10.21 | Initializing multidimensional arrays. (Part 1 of 2.)

```
17  // display array contents
18  function outputArray( heading, theArray, output )
19  {
20     var results = "";
21
22     // iterates through the set of one-dimensional arrays
23     for ( var row in theArray )
24     {
25        results += "<ol>"; // start ordered list
26
27        // iterates through the elements of each one-dimensional array
28        for ( var column in theArray[ row ] )
29        {
30           results += "<li>" + theArray[ row ][ column ] + "</li>";
31        } // end inner for
32
33        results += "</ol>"; // end ordered list
34     } // end outer for
35
36     output.innerHTML = results;
37  } // end function outputArray
38
39  window.addEventListener( "load", start, false );
```

Fig. 10.21 | Initializing multidimensional arrays. (Part 2 of 2.)