JavaScript: Arrays
With sobs and tears he sorted out
Those of the largest size . . .  
—Lewis Carroll

Attempt the end, and never stand to doubt;
Nothing’s so hard, but search will find it out.
—Robert Herrick

Now go, write it before them in a table,
and note it in a book.  
—Isaiah 30:8

’Tis in my memory lock’d,
And you yourself shall keep the key of it.
—William Shakespeare
OBJECTIVES

In this chapter you will learn:

- To use arrays to store lists and tables of values.
- To declare an array, initialize an array and refer to individual elements of an array.
- To pass arrays to functions.
- To search and sort an array.
- To declare and manipulate multidimensional arrays.
10.1 Introduction

• Arrays are data structures consisting of related data items (sometimes called collections of data items).

• JavaScript arrays are “dynamic” entities in that they can change size after they are created.
10.2 Arrays

• An array is a group of memory locations that 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. Any one of these elements may be referred to by giving the name of the array followed by the position number (an integer normally referred to as the subscript or index) of the element in square brackets ([ ] ).
10.2 Arrays (Cont.)

- The first element in every array is the zeroth element. In general, 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 is a left-hand-side expression—it can be used on the left side of an assignment to place a new value into an array element. It can also be used on the right side of an assignment operation to assign its value to another left-hand-side expression.

- Every array in JavaScript knows its own length, which it stores in its \( \text{length} \) attribute and can be found with the expression \( arrayname.length \).
Fig. 10.1 | Array with 12 elements.
Common Programming Error 10.1

It is important to note the difference between the “seventh element of the array” and “array element seven.” Because array subscripts begin at 0, the seventh element of the array has a subscript of 6, while array element seven has a subscript of 7 and is actually the eighth element of the array. This confusion is a source of “off-by-one” errors.
<table>
<thead>
<tr>
<th>Operators</th>
<th>Associativity</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>() [ ] .</td>
<td>left to right</td>
<td>highest</td>
</tr>
<tr>
<td>++ -- !</td>
<td>right to left</td>
<td>unary</td>
</tr>
<tr>
<td>* / %</td>
<td>left to right</td>
<td>multiplicative</td>
</tr>
<tr>
<td>+ -</td>
<td>left to right</td>
<td>additive</td>
</tr>
<tr>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>left to right</td>
<td>relational</td>
</tr>
<tr>
<td>== != ! =</td>
<td>left to right</td>
<td>equality</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>left to right</td>
<td>logical AND</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>? :</td>
<td>right to left</td>
<td>conditional</td>
</tr>
<tr>
<td>= += -= *= /= %=</td>
<td>right to left</td>
<td>assignment</td>
</tr>
</tbody>
</table>
10.3 Declaring and Allocating Arrays

• JavaScript arrays are represented by `Array` objects.

• The process of creating new objects using the `new` operator is known as creating an instance or instantiating an object, and operator `new` is known as the dynamic memory allocation operator.
Common Programming Error 10.2

Assuming that the elements of an array are initialized when the array is allocated may result in logic errors.
10.4 Examples Using Arrays

- Zero-based counting is usually used to iterate through arrays.
- JavaScript automatically reallocates an `Array` when a value is assigned to an element that is outside the bounds of the original `Array`. Elements between the last element of the original `Array` and the new element have undefined values.
Fig. 10.3 | Initializing the elements of an array (Part 1 of 3).

Operator `new` allocates an Array called `n1` with five elements

Operator `new` allocates an empty Array called `n2`

Zero-based counting used in for loop to set each element’s value equal to its subscript

Five elements added and initialized in `n2`, which dynamically expands
Fig. 10.3
Initializing the elements of an array (Part 2 of 3).

```javascript
// output the heading followed by a two-column table
// containing subscripts and elements of "theArray"
function outputArray( heading, theArray )
{
    document.writeln( "<h2>" + heading + "</h2>" );
document.writeln( "<table border = "1"" );
document.writeln( "<thead><th>Subscript</th><th>Value</th></thead><tbody>" );

// output the subscript and value of each array element
for ( var i = 0; i < theArray.length; i++ )
    document.writeln( "<tr><td>" + i + "</td><td>" + theArray[ i ] + "</td></tr>" );
document.writeln( "</tbody></table>" );
// output the subscript and value of each array element
for ( var i = 0; i < theArray.length; i++ )
    document.writeln( "<tr><td>" + i + "</td><td>" + theArray[ i ] + "</td></tr>" );
document.writeln( "</tbody></table>" );
} // end function outputArray

// -->
<script>
<head><body></head>
</html>
```

Outputs the subscript and value of every array element in a table
Array n1:

<table>
<thead>
<tr>
<th>Subscript</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Array n2:

<table>
<thead>
<tr>
<th>Subscript</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Fig. 10.3 | Initializing the elements of an array (Part 3 of 3).
Software Engineering Observation 10.1

JavaScript automatically reallocates an *Array* when a value is assigned to an element that is outside the bounds of the original *Array*. Elements between the last element of the original *Array* and the new element have undefined values.
Common Programming Error 10.3

Referring to an element outside the *Array* bounds is normally a logic error.
Error-Prevention Tip 10.1

When using subscripts to loop through an *Array*, the subscript should never go below 0 and should always be less than the number of elements in the *Array* (i.e., one less than the size of the *Array*). Make sure that the loop-terminating condition prevents the access of elements outside this range.
10.4 Examples Using Arrays (Cont.)

• Arrays can be created using a comma-separated initializer list enclosed in square brackets ([ and ]). The array’s size is determined by the number of values in the initializer list.

• The initial values of an array can also be specified as arguments in the parentheses following new Array. The size of the array is determined by the number of values in parentheses.
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Fig. 10.4: InitArray2.html -->
<!-- Declaring and initializing arrays. -->
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Initializing an Array with a Declaration</title>
<style type="text/css">
    table { width: 15em }
    th { text-align: left }
</style>
<script type="text/javascript">
    // Initializer list specifies the number of elements and a value for each element.
    var colors = new Array( "cyan", "magenta", "yellow", "black" );
    var integers1 = [ 2, 4, 6, 8 ];
    var integers2 = [ 2, , , 8 ];

    outputArray( "Array colors contains", colors );
    outputArray( "Array integers1 contains", integers1 );
    outputArray( "Array integers2 contains", integers2 );

    // output the heading followed by a two-column table containing the subscripts and elements of theArray function outputArray( heading, theArray ) {

</script>
</head>
<body>
</body>
</html>
Declaring and initializing arrays (Part 2 of 3).
### Array colors contains

<table>
<thead>
<tr>
<th>Subscript</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>cyan</td>
</tr>
<tr>
<td>1</td>
<td>magenta</td>
</tr>
<tr>
<td>2</td>
<td>yellow</td>
</tr>
<tr>
<td>3</td>
<td>black</td>
</tr>
</tbody>
</table>

### Array integers1 contains

<table>
<thead>
<tr>
<th>Subscript</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

### Array integers2 contains

<table>
<thead>
<tr>
<th>Subscript</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>undefined</td>
</tr>
<tr>
<td>2</td>
<td>undefined</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>
• JavaScript’s `for...in` statement enables a script to perform a task for each element in an array. This process is known as iterating over the elements of an array.
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 10.5: SumArray.html -->
<!-- Summing elements of an array. -->
<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
<title>Sum the Elements of an Array</title>
<script type = "text/javascript">
  <!--
  var theArray = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ];
  var total1 = 0, total2 = 0;
  
  // iterates through the elements of the array in order and adds
  // each element's value to total1
  for ( var i = 0; i < theArray.length; i++ )
  total1 += theArray[ i ];
  
  document.writeln( "Total using subscripts: " + total1 );
  
  // iterates through the elements of the array using a for... in
  // statement to add each element's value to total2
  for ( var element in theArray )
  total2 += theArray[ element ];

</script>
</head>
</html>
Fig. 10.5
Summing elements of an array (Part 2 of 2).
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 of the Array. Note that a for...in statement skips any undefined elements in the array.
Fig. 10.6 | Dice-rolling program using an array instead of a switch (Part 1 of 2).

- Creates a frequency array with each element’s index corresponding to a face value (we leave index 0 uninitialized because the lowest face value is 1).
- Randomly picks a face of the die and increments the value of the element with the corresponding index in the frequency array.
// generate entire table of frequencies for each face
for ( face = 1; face < frequency.length; ++face )
    document.writeln( "<tr><td>" + face + "</td><td>" +
    frequency[ face ] + "</td></tr>" );

// -->
</script>
</head>
<body>
<p>Click Refresh (or Reload) to run the script again</p>
</body>
</html>

Fig. 10.6 Dice-rolling program using an array instead of a switch (Part 2 of 2).
10.5 Random Image Generator Using Arrays

• We create a more elegant random image generator than the one in the previous chapter that does not require the image filenames to be integers by using a pictures array to store the names of the image files as strings and accessing the array using a randomized index.
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 10.7: RandomPicture2.html -->
<!-- Random image generation using arrays. -->
<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
<title>Random Image Generator</title>
<style type = "text/css">
table { width: 15em }
th    { text-align: left }
</style>
<script type = "text/javascript">
<!--
var pictures =
[ "CPE", "EPT", "GPP", "GUI", "PERF", "PORT", "SEO" ];

// pick a random image from the pictures array and displays by
// creating an img tag and appending the src attribute to the
// filename

document.write ("<img src = "" +
pictures[ Math.floor( Math.random() * 7 ) ] + ".gif\" />" );
// -->
</script>
</head>
<body>
<p>Click Refresh (or Reload) to run the script again</p>
</body>
</html>

Fig. 10.7 | Random image generation using arrays (Part 1 of 2).

Creates an array with the names of the images to choose from

Randomly selects an element from the array and appends its value to " .gif " to create the src attribute’s value
Fig. 10.7  Random image generation using arrays (Part 2 of 2).
10.6 References and Reference Parameters

- Two ways to pass arguments to functions (or methods) in many programming languages are pass-by-value and pass-by-reference.
- When an argument is passed to a function 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.
- With pass-by-reference, the caller gives the called function direct access to the caller’s data and allows it to modify the data if it so chooses. Pass-by-reference 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.
10.6 References and Reference Parameters (Cont.)

- In JavaScript, all objects (and thus all \texttt{Arrays}) are passed to functions by reference.
- Arrays are objects in JavaScript, so \texttt{Arrays} are passed to a function by reference—a called function can access the elements of the caller’s original \texttt{Arrays}. The name of an array is actually a reference to an object that contains the array elements and the \texttt{length} variable, which indicates the number of elements in the array.
Error-Prevention Tip 10.3

With pass-by-value, changes to the copy of the called function do not affect the original variable’s value in the calling function. This prevents the accidental side effects that so greatly hinder the development of correct and reliable software systems.
Unlike some other languages, JavaScript does not allow the programmer to choose whether to pass each argument by value or by reference. Numbers, boolean values and strings are passed by value. Objects are passed to functions by reference. When a function receives a reference to an object, the function can manipulate the object directly.
Software Engineering Observation 10.3

When returning information from a function via a `return` statement, numbers and boolean values are always returned by value (i.e., a copy is returned), and objects are always returned by reference (i.e., a reference to the object is returned). Note that, in the pass-by-reference case, it is not necessary to return the new value, since the object is already modified.
10.7 Passing Arrays to Functions

• To pass an array argument to a function, specify the name of the array (a reference to the array) without brackets.

• 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. To pass an array element to a function, use the subscripted name of the element as an argument in the function call.

• The `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.
Fig. 10.8 | Passing arrays and individual array elements to functions (Part 1 of 3).

Passes array `a` to function `modifyArray` by reference

Passes array element `a[3]` to function `modifyElement` by value
Fig. 10.8  Passing arrays and individual array elements to functions (Part 2 of 3).

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

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

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

// -->
</script>
</head></body></body>
</html>

Creates a string containing all the elements in theArray, separated by " "

Multiplies each element in theArray by 2, which persists after the function has finished

Multiplies the array element by 2, but only for the duration of the function
**Fig. 10.8** Passing arrays and individual array elements to functions (Part 3 of 3).

**Effects of passing entire array by reference**

Original array: 1 2 3 4 5  
Modified array: 2 4 6 8 10

**Effects of passing array element by value**

a[3] before modifyElement: 8  
value in modifyElement: 16  
a[3] after modifyElement: 8
Software Engineering Observation 10.4

JavaScript does not check the number of arguments or types of arguments that are passed to a function. It is possible to pass any number of values to a function. JavaScript will attempt to perform conversions when the values are used.
10.8 Sorting Arrays

• Sorting data (putting data in a particular order, such as ascending or descending) is one of the most important computing functions.
• The `Array` object in JavaScript has a built-in method `sort` for sorting arrays.
• By default, `Array` method `sort` (with no arguments) uses string comparisons to determine the sorting order of the Array elements.
• Method `sort` takes as its optional argument the name of a function (called the comparator function) that compares its two arguments and returns a negative value, zero, or a positive value, if the first argument is less than, equal to, or greater than the second, respectively.
• 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.
Fig. 10.9 | Sorting an array with `sort` (Part 1 of 2).

Passes function `compareIntegers` to method `a.sort` to arrange the elements of `a` in ascending numerical order.
Fig. 10.9  Sorting an array with `sort` (Part 2 of 2).

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

Defines a function comparing integers to be passed to method `sort` (to replace the default string comparison function).
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: Linear Search and Binary Search

- The linear search algorithm iterates through the elements of an array until it finds an element that matches a search key, and returns the subscript of the element. If the key is not found, the function returns \(-1\).
- If the array being searched is not in any particular order, it is just as likely that the value will be found in the first element as the last. On average, therefore, the program will have to compare the search key with half the elements of the array.
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Fig. 10.10: LinearSearch.html -->
<!-- Linear search of an array. -->
<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
<title>Linear Search of an Array</title>
<script type = "text/javascript">
<!--
var a = new Array( 100 );  // create an Array

// fill Array with even integer values from 0 to 198
for ( var i = 0; i < a.length; ++i )
a[ i ] = 2 * i;

// function called when "Search" button is pressed
function buttonPressed()
{
   // get the input text field
   var inputVal = document.getElementById( "inputVal" );

   // get the result text field
   var result = document.getElementById( "result" );

   // get the search key from the input text field
   var searchKey = inputVal.value;

   // output search result
   result.innerHTML = "searchKey exists in position " + a.indexOf( searchKey );
}
//-->
</script>
</head>
</html>
// Array a is passed to linearSearch even though it
// is a global variable. Normally an array will
// be passed to a method for searching.

var element = linearSearch( a, parseInt( searchKey ) );

if ( element != -1 )
    result.value = "Found value in element " + element;
else
    result.value = "Value not found";
} // end function buttonPressed

// Search "theArray" for the specified "key" value

function linearSearch( theArray, key )
{
    // iterates through each element of the array in order
    for ( var n = 0; n < theArray.length; ++n )
        if ( theArray[ n ] == key )
            return n;

    return -1;
} // end function linearSearch

<!-- -->
</script>
</head>
When the Search button is pressed, calls function buttonPressed.
10.9 Searching Arrays: Linear Search and Binary Search (Cont.)

• The binary search algorithm is more efficient than the linear search algorithm, but it requires that the array be sorted.
• The binary search algorithm tests the middle element in the array and returns the index if it matches the search key. If not, it cuts the list in half, depending on whether the key is greater than or less than the middle element, and repeats the process on the remaining half of the sorted list. The algorithm ends by either finding an element that matches the search key or reducing the subarray to zero size.
• When searching a sorted array, the binary search provides a tremendous increase in performance over the linear search. For a one-billion-element array, this is the difference between an average of 500 million comparisons and a maximum of 30 comparisons.
• The maximum number of comparisons needed for the binary search of any sorted array is the exponent of the first power of 2 greater than the number of elements in the array.
<xml version="1.0" encoding="utf-8"/>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Fig. 10.11: BinarySearch.html -->
<!-- Binary search of an array. -->
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Binary Search</title>
<script type="text/javascript">
<!--
var a = new Array( 15 );

for ( var i = 0; i < a.length; ++i )
a[ i ] = 2 * i;

// function called when "Search" button is pressed
function buttonPressed()
{
  var inputVal = document.getElementById( "inputVal" );
  var result = document.getElementById( "result" );
  var searchKey = inputVal.value;

  result.value = "Portions of array searched\n";
"-->
</script>
</head>
</html>
Array a is passed to binarySearch even though it is a global variable. This is done because normally an array is passed to a method for searching.

```javascript
var element =
    binarySearch( a, parseInt( searchKey ) );

if ( element != -1 )
    result.value += "\nFound value in element " + element;
else
    result.value += "\nValue not found";
```

Calls function binarySearch with arguments a and the key specified by the user

While the search has not checked all values, find the midpoint of the unchecked region

Displays the portion of the array currently being examined
if (key == theArray[middle]) // match
    return middle;
else if (key < theArray[middle])
    high = middle - 1; // search low end of array
else
    low = middle + 1; // search high end of array
} // end while

return -1; // searchKey not found
} // end function binarySearch

// Build one row of output showing the current
// part of the array being processed.
function buildOutput( theArray, low, mid, high )
{
    var result = document.getElementById( "result" );

    for ( var i = 0; i < theArray.length; i++ )
    {
        if ( i < low || i > high )
            result.value += "    ";
        else if ( i == mid ) // mark middle element in output
            result.value += theArray[i] +
                (theArray[i] < 10 ? " * " : " ");
        else
            result.value += theArray[i] +
                (theArray[i] < 10 ? "   " : "  ");
    } // end for

If the middle element’s value is the key, return its subscript

**Binary search of an array (Part 3)**

Otherwise, if the middle element’s value is higher than the key, we only need to search the bottom half of the array

Otherwise, if the middle element’s value is lower than the key, we only need to search the higher half of the array

If we’ve covered the whole array without encountering the key, return -1
```html
result.value += "\n";
} // end function buildOutput
// -->
</script>
</head>
<body>
<form action="">
<p>Enter integer search key<br />
<input id="inputVal" type="text" />
<input type="button" value="Search" onclick="buttonPressed()" />
</p>
<p>Result<br />
<textarea id="result" rows="7" cols="60">
</textarea></p>
</form>
</body>
</html>
```

**Fig. 10.11**
Binary search of an array (Part 4 of 5).
Fig. 10.11 | Binary search of an array (Part 5 of 5).
10.10 Multidimensional Arrays

• To identify a particular two-dimensional multidimensional array element, we must specify the two subscripts; 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.

• Every element in a two-dimensional array is accessed using an element name of the form $a[i][j]$; $a$ is the name of the array, and $i$ and $j$ are the subscripts that uniquely identify the row and column, respectively, of each element in $a$.

• Multidimensional arrays are maintained as arrays of arrays.
**Fig. 10.12** | 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 lists—arrays nested within the outermost array. 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.
<xml version="1.0" encoding="utf-8"/>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Fig. 10.13: InitArray3.html -->
<!-- Initializing multidimensional arrays. -->
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Initializing Multidimensional Arrays</title>
<script type="text/javascript">

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

outputArray( "Values in array1 by row", array1 );
outputArray( "Values in array2 by row", array2 );

function outputArray( heading, theArray )
{
  document.writeln( "<h2>" + heading + "</h2><pre>" );
  // iterates through the set of one-dimensional arrays
  for ( var i in theArray )
  {
    // iterates through the elements of each one-dimensional
    // array
    for ( var j in theArray[ i ] )
      document.write( theArray[ i ][ j ] + " " );
  }
}
</script>
</head>
</html>

--- Fig. 10.13 | Initializing multidimensional arrays (Part 1 of 2).

Initializes array1 with an initializer list of sub initializer lists

Initializes array2 with rows of different lengths

Nested for...in statements traverse the arrays by iterating through the sets of one-dimensional arrays, then through the elements of each of those one-dimensional arrays
Fig. 10.13
Initializing multidimensional arrays (Part 2 of 2).

Values in array1 by row
1 2 3
4 5 6

Values in array2 by row
1 2
3
4 5 6
10.11 Building an Online Quiz

• An XHTML form's elements can be accessed individually using `getElementById` or through the `elements` property of the containing form object. The `elements` property contains an array of all the form's controls.

• Property `checked` of a radio button is `true` when the radio button is selected, and it is `false` when the radio button is not selected.
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 10.14: quiz.html -->
<!-- Online quiz graded with JavaScript. -->
<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
<title>Online Quiz</title>
<script type = "text/JavaScript">
//
function checkAnswers()
{
  var myQuiz = document.getElementById("myQuiz");

  // determine whether the answer is correct
  if (myQuiz.elements[1].checked)
    alert( "Congratulations, your answer is correct" );
  else // if the answer is incorrect
    alert( "Your answer is incorrect. Please try again" );
} // end function checkAnswers
</script>
</head>
<body>
<form id = "myQuiz" onsubmit = "checkAnswers()" action = "">
<p>Select the name of the tip that goes with the image shown:<br/>
<img src="EPT.gif" alt="mystery tip"/>
<br/>
</p>
</form>
</body>
</html>

Checks to see if the second radio button of the myQuiz form is selected.
Fig. 10.14 | Online quiz graded with JavaScript (Part 2 of 3).

```html
<input type="radio" name="radiobutton" value="CPE" />
<label>Common Programming Error</label>

<input type="radio" name="radiobutton" value="EPT" />
<label>Error-Prevention Tip</label>

<input type="radio" name="radiobutton" value="PERF" />
<label>Performance Tip</label>

<input type="radio" name="radiobutton" value="PORT" />
<label>Portability Tip</label>
<br />

<input type="submit" name="submit" value="Submit" />
<input type="reset" name="reset" value="Reset" />

</form>
```
Fig. 10.14 | Online quiz graded with JavaScript (Part 3 of 3).