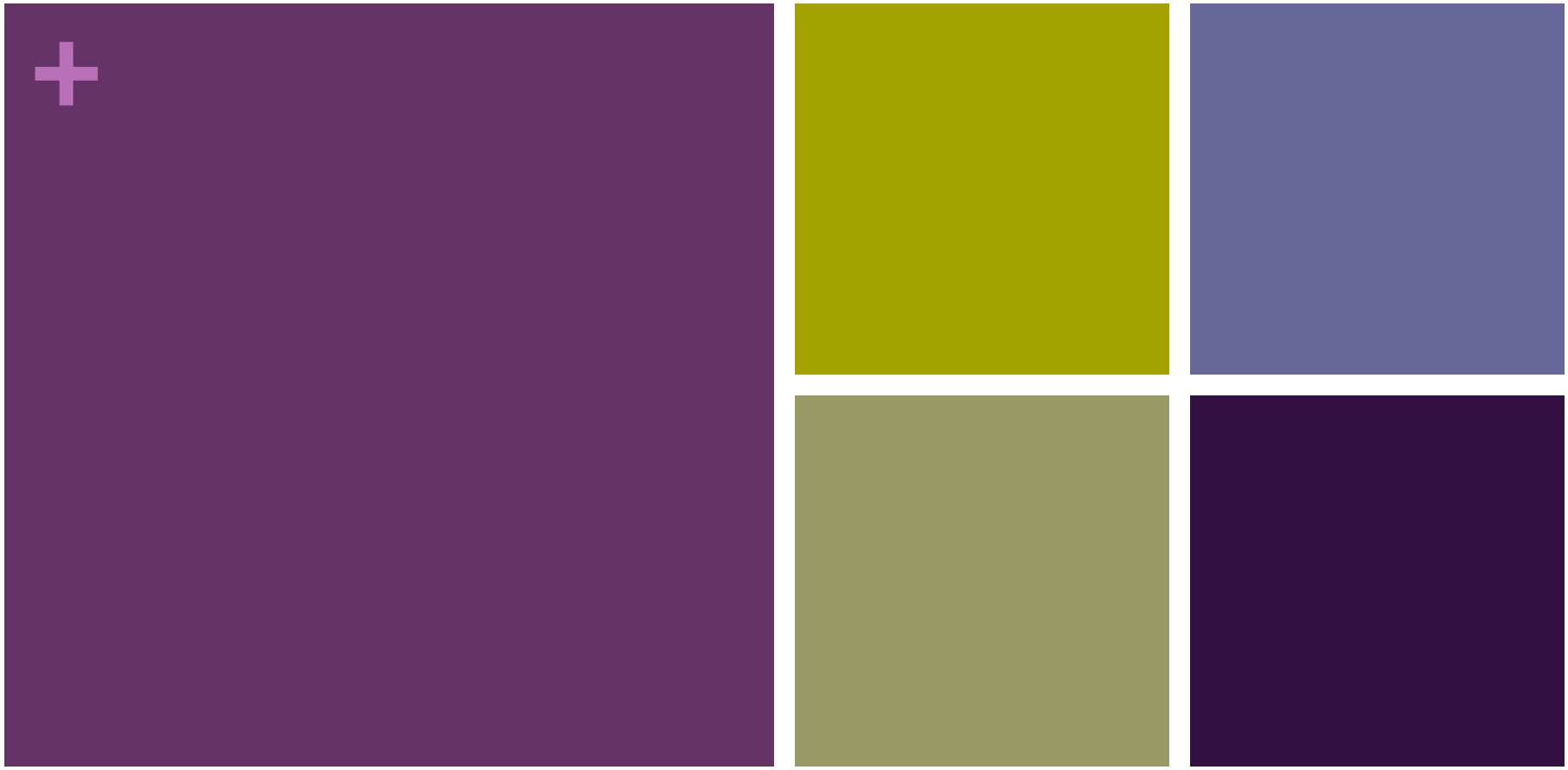


JavaScript: Arrays

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.

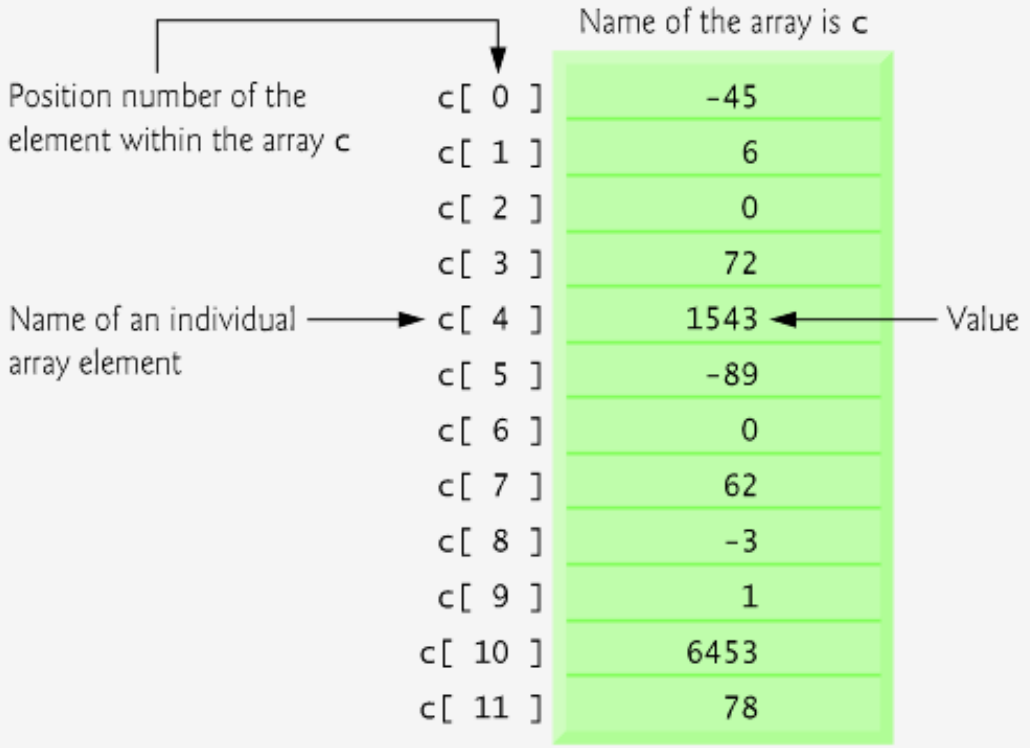
Introduction

- Arrays
  - Data structures consisting of related data items
  - Sometimes called collections of data items
- 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
  - An element may be referred to by giving the name of the array followed by index of the element in square brackets ([])
- JavaScript arrays
  - "dynamic" entities that can change size after they are created

Introduction (…)

- 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

Example: Array with 12 elements.



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.

Four ways to create an array

- You can use an array literal:
  let colors = ["red", "green", "blue"];
- You can use new Array() to create an empty array:
  - let colors = new Array();
  - You can add elements to the array later:
    colors[0] = "red"; colors[2] = "blue"; colors[1]="green";
- You can use new Array(***n***) with a single numeric argument to create an array of that size
  - let colors = new Array(3);
- You can use new Array(…) with *two or more* arguments to create an array containing those values:
  - let colors = new Array("red","green", "blue");

Examples

- An array literal can appear anywhere an expression can appear.

```
let empty = [];
let numbers = [ 'zero', 'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine' ];
        empty[1]                // undefined
        numbers[1]              // 'one'
        empty.length            // 0
        numbers.length          // 10
```

- If you put two commas in a row, the array has an "empty" element in that location
  - Example: color = ["red", , , "green", "blue"];
    - color has 5 elements
  - However, a single comma at the end is ignored
    - Example: color = ["red", , , "green", "blue",]; still has 5 elements

Another properties of arrays

- In most languages, the elements of an array are all required to be of the same type.
- JavaScript allows an array to contain any mixture of values:

```
let misc = [ 'string', 98.6, true, false, null, undefined, ['nested', 'array'], {object: true}, NaN, Infinity ];

misc.length                    // 10
```

The length of an array

- JavaScript's array length is not an upper bound.
  - If you store an element with a subscript that is greater than or equal to the current length, the length will increase to contain the new element.

- there is no such thing as an array out-of-bounds error
  - get an element out of bounds → undefined
  - set an element out of bounds → length increases to fit
    - any elements in between old/new lengths are undefined

```
let a = [42, 10];
a[10] = 5;
a     // 42,10,,,,,,,,,5

typeof(a[6])    // Undefined
```

The length of an array

- If myArray is an array, its length is given by myArray.length
- Array length can be changed by assignment beyond the current length
  - Example: let myArray = new Array(5); myArray[10] = 3;
- Arrays are sparse, that is, space is only allocated for elements that have been assigned a value
  - Example: myArray[50000] = 3; is perfectly OK
  - But indices must be between 0 and 2^{32}-1
- you can set `length`;
  - if smaller, truncates the array to the new smaller size
  - if larger, all new elements will be `undefined`

```
a.length = 2;
```

```
1  <?xml version = "1.0" encoding = "utf-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 10.3: InitArray.html -->
6  <!-- Initializing the elements of an array. -->
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8     <head>
9        <title>Initializing an Array</title>
10       <style type = "text/css">
11          table { width: 10em }
12          th    { text-align: left }
13       </style>
14       <script type = "text/javascript">
15          <!--
16          // create (declare) two new arrays
17          var n1 = new Array( 5 ); // allocate five-element Array
18          var n2 = new Array(); // allocate empty Array
19
20          // assign values to each element of Array n1
21          for ( var i = 0; i < n1.length; ++i )
22             n1[ i ] = i;
23
24          // create and initialize five elements in Array n2
25          for ( i = 0; i < 5; ++i )
26             n2[ i ] = i;
27
28          outputArray( "Array n1:", n1 );
29          outputArray( "Array n2:", n2 );
30
```

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

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

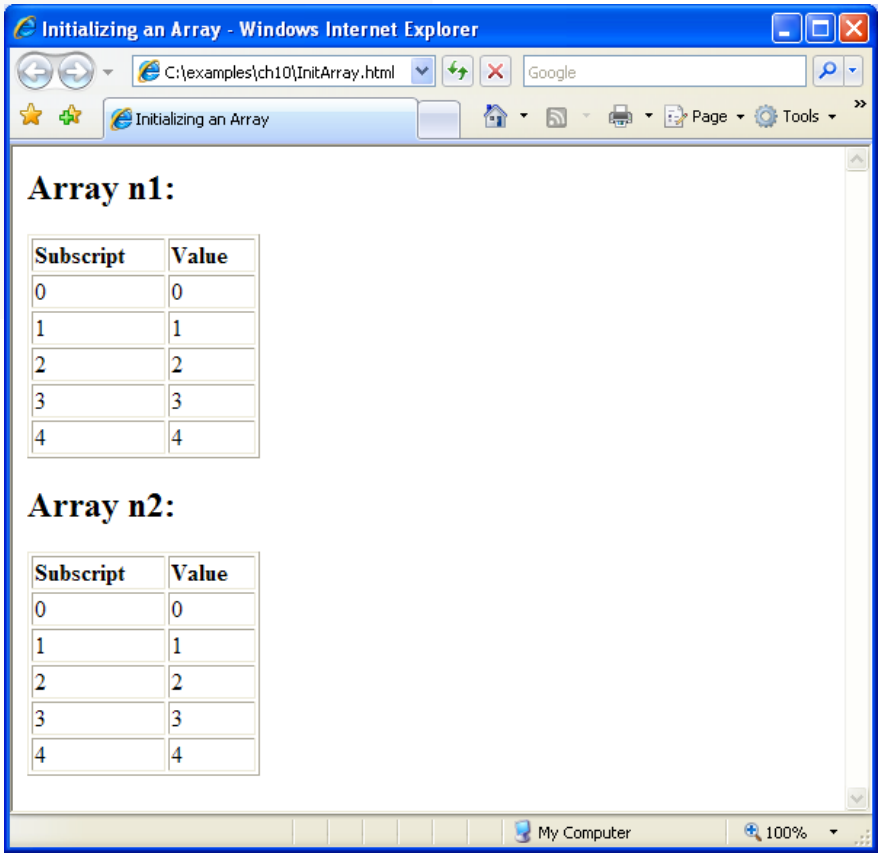
Zero-based counting used in `for` loop to set each element's value equal to its subscript

Five elements added and initialized in `n2`, which dynamically expands

Example.
Initialization

```
31          // output the heading followed by a two-column table
32          // containing subscripts and elements of "theArray"
33          function outputArray( heading, theArray )
34          {
35             document.writeln( "<h2>" + heading + "</h2>" );
36             document.writeln( "<table border = \"1\"" );
37             document.writeln( "<thead><th>Subscript</th>" +
38                "<th>Value</th></thead><tbody>" );
39
40             // output the subscript and value of each array element
41             for ( var i = 0; i < theArray.length; i++ )
42                document.writeln( "<tr><td>" + i + "</td><td>" +
43                   theArray[ i ] + "</td></tr>" );
44
45             document.writeln( "</tbody></table>" );
46          } // end function outputArray
47          // -->
48       </script>
49    </head><body></body>
50 </html>
```

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



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.

Example 2. Initialization

```
1  <?xml version = "1.0" encoding = "utf-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 10.4: InitArray2.html -->
6  <!-- Declaring and initializing arrays. -->
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8     <head>
9        <title>Initializing an Array with a Declaration</title>
10       <style type = "text/css">
11          table { width: 15em }
12          th    { text-align: left }
13       </style>
14       <script type = "text/javascript">
15          <!--
16          // Initializer list specifies the number of elements and
17          // a value for each element.
18          var colors = new Array( "cyan", "magenta","yellow", "black" );
19          var integers1 = [ 2, 4, 6, 8 ];
20          var integers2 = [ 2, , , 8 ];
21
22          outputArray( "Array colors contains", colors );
23          outputArray( "Array integers1 contains", integers1 );
24          outputArray( "Array integers2 contains", integers2 );
25
26          // output the heading followed by a two-column table
27          // containing the subscripts and elements of theArray
28          function outputArray( heading, theArray )
29          {
```

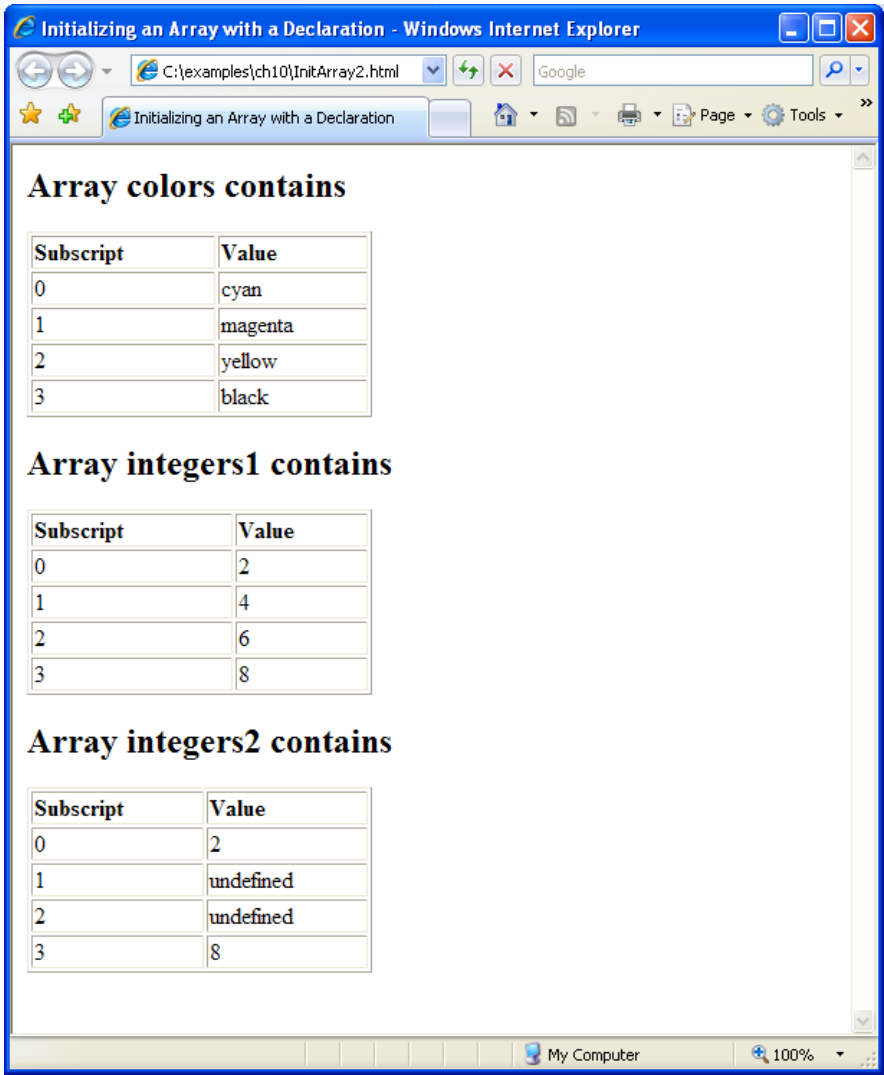
Creates an array with four elements, all of which are defined

Creates an array with four elements, all of which are defined in an initializer list

Creates an array with four elements, two of which reserve space for values to be specified later

Example 2. Initialization

```
30            document.writeln( "<h2>" + heading + "</h2>" );
31            document.writeln( "<table border = \"1\"" );
32            document.writeln( "<thead><th>Subscript</th>" +
33               "<th>Value</th></thead><tbody>" );
34
35            // output the subscript and value of each array element
36            for ( var i = 0; i < theArray.length; i++ )
37               document.writeln( "<tr><td>" + i + "</td><td>" +
38                  theArray[ i ] + "</td></tr>" );
39
40            document.writeln( "</tbody></table>" );
41         } // end function outputArray
42         // -->
43      </script>
44   </head><body></body>
45 </html>
```



Array methods

- If myArray is an array,
  - myArray.sort() sorts the array alphabetically
  - myArray.sort(function(a, b) { return a - b; }) sorts numerically
  - myArray.reverse() reverses the array elements
  - myArray.push(…) adds any number of new elements to the end of the array, and increases the array's length
  - myArray.pop() removes and returns the last element of the array, and decrements the array's length
  - myArray.toString() returns a string containing the values of the array elements, separated by commas

Array methods example

```
let a = ["Stef", "Jay"];      // Stef, Jay
a.push("Bob");                // Stef, Jay, Bob
a.unshift("Kelly");           // Kelly, Stef, Jay, Bob
a.pop();                      // Kelly, Stef, Jay
a.shift();                    // Stef, Jay
a.sort();                     // Jay, Stef
```

Split and join example

```
var s = "quick brown fox";

var a = s.split(" ");        // ["quick", "brown", "fox"]

a.reverse();                 // ["fox", "brown", "quick"]

s = a.join("!");             // "fox!brown!quick"
```

- split breaks a string into an array using a delimiter
  - can also be used with regular expressions (seen later)
- join merges an array into a single string, placing a delimiter between them

Array Methods: sort( )

Sorts the elements in alphabetical order

```
x = new Array ( 4 ) ;

x[ 0 ] = "Waseem" ;

x[ 1 ] = "Waqar" ;

x[ 2 ] = "Saqlain" ;

x[ 3 ] = "Shoaib" ;

x.sort( ) ;

for ( k = 0 ; k < x.length; k = k + 1 ) {document.write( x[ k ] + "<BR>" ) ;}
```

Saqlain

Shoaib

Waqar

Waseem

What if you wanted to arrange them in the reverse order?

Array Methods: reverse( )

Reverses the order of the elements

```
x = new Array ( 4 ) ;
x[ 0 ] = "Waseem" ;
x[ 1 ] = "Waqar" ;
x[ 2 ] = "Saqlain" ;
x[ 3 ] = "Shoaib" ;
x.reverse( ) ;
x.sort( ) ;
for ( k = 0 ; k < x.length; k = k + 1 ) {document.write( x[ k ] + "<BR>") ;}
```

Saqlain
Shoaib
Waqar
Waseem

Is this the required result?

Array Methods: reverse( )
Reverses the order of the elements

```
x = new Array ( 4 ) ;
x[ 0 ] = "Waseem" ;
x[ 1 ] = "Waqar" ;
x[ 2 ] = "Saqlain" ;
x[ 3 ] = "Shoaib" ;
x.sort( ) ;
x.reverse( ) ;
for ( k = 0 ; k < x.length; k = k + 1 ) { document.write( x[ k ] + "<BR>") ; }
```

Waseem

Waqar

Shoaib

Saqlain

for-of loop (like for-each)

- You can loop through all *elements* of an array with for

for (***name*** of ***expr***) { ***statements***; }

- E.g.

```
let ducks = ["Huey", "Dewey", "Louie"];
for (x of a) { console.log(x); }
```

"Huey"
"Dewey"
"Louie"

for-in loop (not recommended)

- You can loop through all indexes of an array with for

for (***name*** in ***expr***) { ***statements***; }

- JavaScript has a "for-each" loop, but it loops over each *index*, not each value, in the array.

```
let ducks = ["Huey", "Dewey", "Louie"];
for (x in a) { print(x); }
```

0
1
2

Example. For-each loop

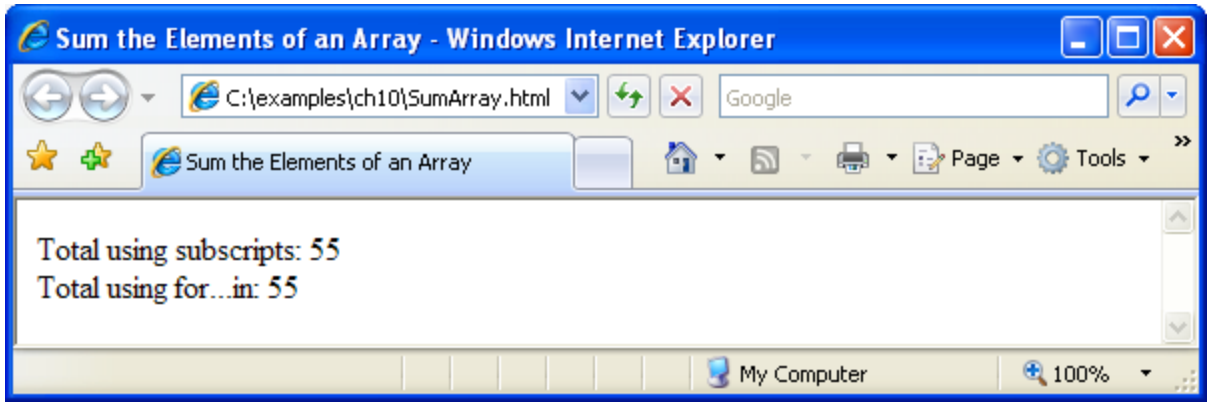
```
1  <?xml version = "1.0" encoding = "utf-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 10.5: SumArray.html -->
6  <!-- Summing elements of an array. -->
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8     <head>
9        <title>Sum the Elements of an Array</title>
10
11       <script type = "text/javascript">
12          <!--
13          var theArray = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ];
14          var total1 = 0, total2 = 0;
15
16          // iterates through the elements of the array in order and adds
17          // each element's value to total1
18          for ( var i = 0; i < theArray.length; i++ )
19             total1 += theArray[ i ];
20
21          document.writeln( "Total using subscripts: " + total1 );
22
23          // iterates through the elements of the array using a for... in
24          // statement to add each element's value to total2
25          for ( var element in theArray )
26             total2 += theArray[ element ];
```

Sums the values of all the elements in `theArray` by iterating through the elements in order

Sums the values of all the elements in `theArray` by having JavaScript automatically iterate over its elements

Example. For-each loop

```
27
28          document.writeln( "<br />Total using for...in: " + total2 );
29          // -->
30       </script>
31    </head><body></body>
32 </html>
```



When iterating over all 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.

Example. For loop

```
1  <?xml version = "1.0" encoding = "utf-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 10.6: RollDie.html -->
6  <!-- Dice-rolling program using an array instead of a switch. -->
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8     <head>
9        <title>Roll a Six-Sided Die 6000 Times</title>
10       <style type = "text/css">
11          table { width: 15em }
12          th    { text-align: left }
13       </style>
14       <script type = "text/javascript">
15          <!--
16          var face;
17          var frequency = [ , 0, 0, 0, 0, 0, 0 ]; // leave frequency[0]
18                                                  // uninitialized
19
20          // summarize results
21          for ( var roll = 1; roll <= 6000; ++roll )
22          {
23             face = Math.floor( 1 + Math.random() * 6 );
24             ++frequency[ face ];
25          } // end for
26
27          document.writeln( "<table border = \"1\"><thead>" );
28          document.writeln( "<th>Face</th>" +
29             "<th>Frequency</th></thead><tbody>" );
30
```

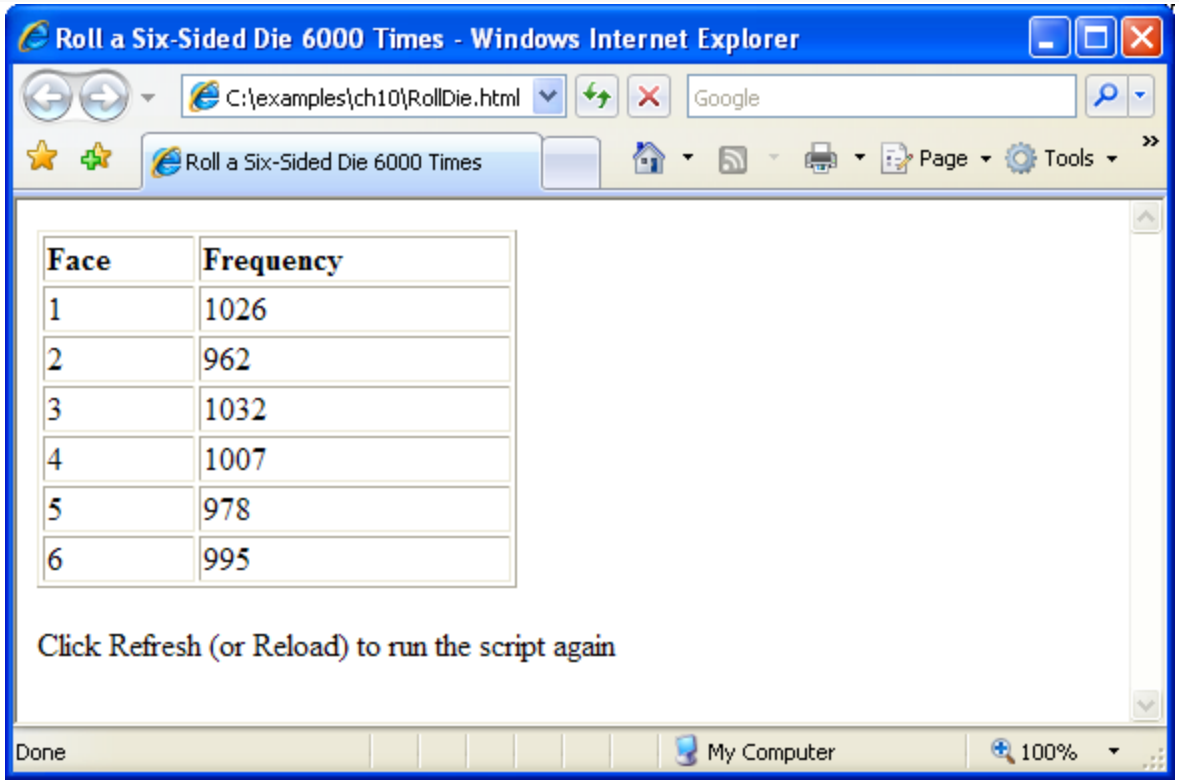
Creates a `frequency` array with each element's index corresponding to a face value (we leave index 0 uninitialized because the lowest face value is 1)

Randomly picks a face of the die and increments the value of the element with the corresponding index in the `frequency` array

Example. For loop

```
31          // generate entire table of frequencies for each face
32          for ( face = 1; face < frequency.length; ++face )
33             document.writeln( "<tr><td>" + face + "</td><td>" +
34                frequency[ face ] + "</td></tr>" );
35
36          document.writeln( "</tbody></table>" );
37          // -->
38       </script>
39    </head>
40    <body>
41       <p>Click Refresh (or Reload) to run the script again</p>
42    </body>
43 </html>
```

Outputs results in a table



Example.

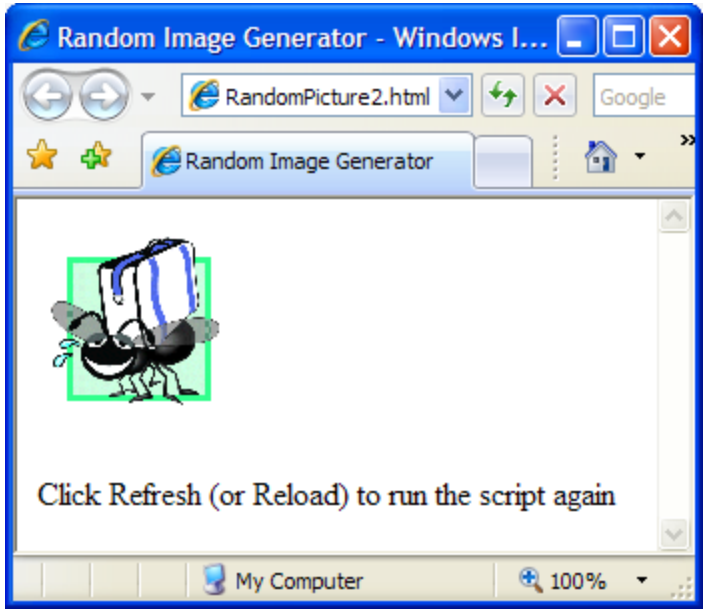
```
1  <?xml version = "1.0" encoding = "utf-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 10.7: RandomPicture2.html -->
6  <!-- Random image generation using arrays. -->
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8     <head>
9        <title>Random Image Generator</title>
10       <style type = "text/css">
11          table { width: 15em }
12          th    { text-align: left }
13       </style>
14       <script type = "text/javascript">
15          <!--
16          var pictures =
17             [ "CPE", "EPT", "GPP", "GUI", "PERF", "PORT", "SEO" ];
18
19          // pick a random image from the pictures array and displays by
20          // creating an img tag and appending the src attribute to the
21          // filename
22          document.write ( "<img src = \"" +
23             pictures[ Math.floor( Math.random() * 7 ) ] + ".gif\" />" );
24          // -->
25       </script>
26    </head>
27    <body>
28       <p>Click Refresh (or Reload) to run the script again</p>
29    </body>
30 </html>
```

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

- Random image generator
  - Uses a pictures array to store the names of the image files as strings
  - Accesses the array using a randomized index

Example. For loop



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.
- 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
  - 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
- Arrays are passed to a function by reference
  - a called function can access the elements of the caller's original Arrays.

Passing Arrays to Functions

- Pass an array as an 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

Example. Pass by...

```
1  <?xml version = "1.0" encoding = "utf-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 10.8: PassArray.html -->
6  <!-- Passing arrays and individual array elements to functions. -->
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8     <head>
9        <title>Passing arrays and individual array
10          elements to functions</title>
11       <script type = "text/javascript">
12          <!--
13          var a = [ 1, 2, 3, 4, 5 ];
14
15          document.writeln( "<h2>Effects of passing entire " +
16             "array by reference</h2>" );
17          outputArray( "Original array: ", a );
18
19          modifyArray( a );  // array a passed by reference
20
21          outputArray( "Modified array: ", a );
22
23          document.writeln( "<h2>Effects of passing array " +
24             "element by value</h2>" +
25             "a[3] before modifyElement: " + a[ 3 ] );
26
27          modifyElement( a[ 3 ] ); // array element a[3] passed by value
28
29          document.writeln( "<br />a[3] after modifyElement: " + a[ 3 ] );
30
```

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

Passes array element `a[3]` to function `modifyElement` by value

Example. Pass by…

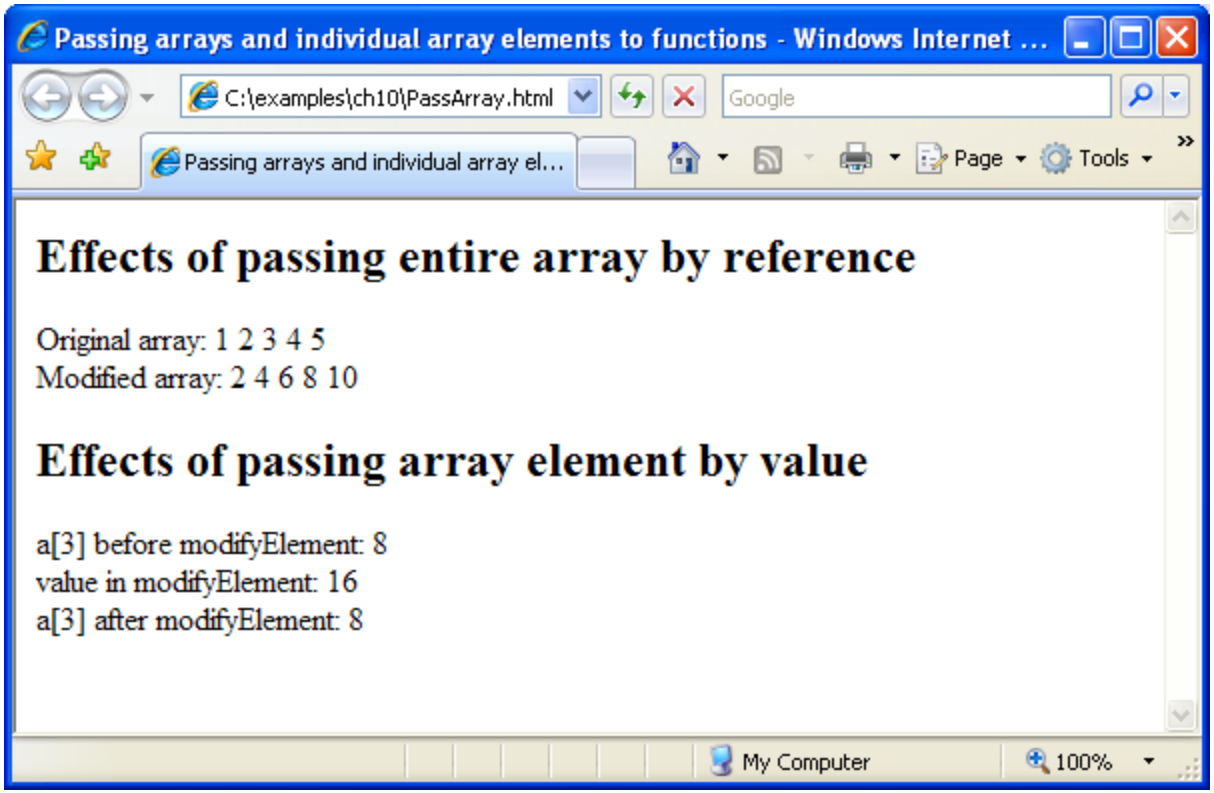
```
31          // outputs heading followed by the contents of "theArray"
32          function outputArray( heading, theArray )
33          {
34             document.writeln(
35                heading + theArray.join( " " ) + "<br />" );
36          } // end function outputArray
37
38          // function that modifies the elements of an array
39          function modifyArray( theArray )
40          {
41             for ( var j in theArray )
42                theArray[ j ] *= 2;
43          } // end function modifyArray
44
45          // function that modifies the value passed
46          function modifyElement( e )
47          {
48             e *= 2; // scales element e only for the duration of the
49                     // function
50             document.writeln( "<br />value in modifyElement: " + e );
51          } // end function modifyElement
52          // -->
53       </script>
54    </head><body></body>
55 </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

Passing arrays and individual array elements to functions



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.

Sorting Arrays

- 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 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
  - They can be assigned to variables, stored in Arrays and passed to functions just like other data

Example. Sort

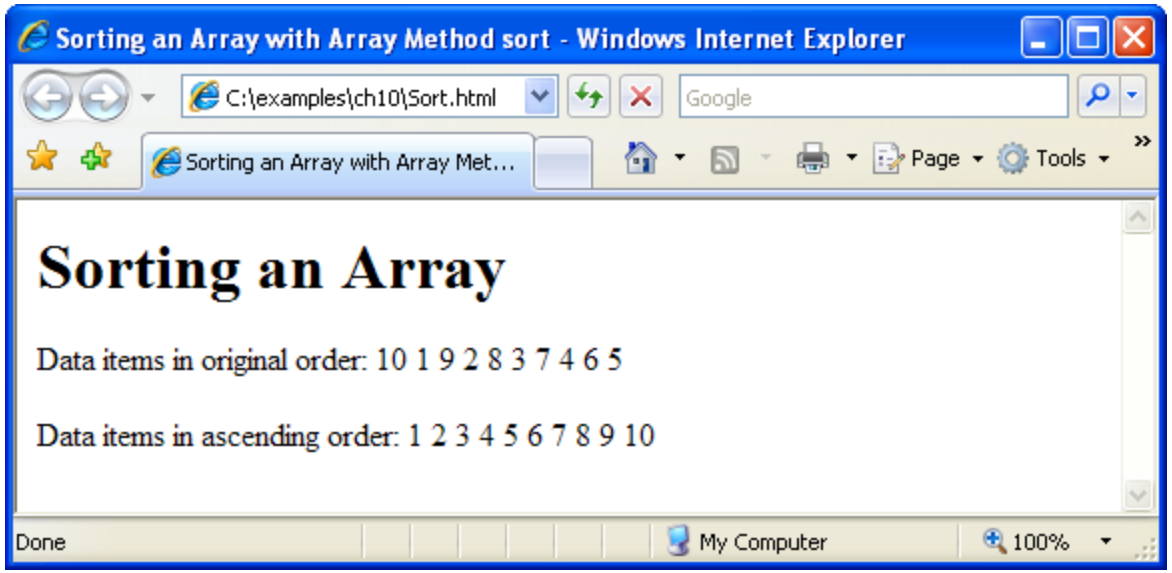
```
1  <?xml version = "1.0" encoding = "utf-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 10.9: Sort.html -->
6  <!-- Sorting an array with sort. -->
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8     <head>
9        <title>Sorting an Array with Array Method sort</title>
10       <script type = "text/javascript">
11          <!--
12          var a = [ 10, 1, 9, 2, 8, 3, 7, 4, 6, 5 ];
13
14          document.writeln( "<h1>Sorting an Array</h1>" );
15          outputArray( "Data items in original order: ", a );
16          a.sort( compareIntegers );  // sort the array
17          outputArray( "Data items in ascending order: ", a );
18
19          // output the heading followed by the contents of theArray
20          function outputArray( heading, theArray )
21          {
22             document.writeln( "<p>" + heading +
23                theArray.join( " " ) + "</p>" );
24          } // end function outputArray
25
```

Passes function `compareIntegers` to method `a.sort` to arrange the elements of `a` in ascending numerical order

Example. Sort

```
26          // comparison function for use with sort
27          function compareIntegers( value1, value2 )
28          {
29             return parseInt( value1 ) - parseInt( value2 );
30          } // end function compareIntegers
31          // -->
32       </script>
33    </head><body></body>
34 </html>
```

Defines a function comparing integers to be passed to method `sort` (to replace the default string comparison function)



Searching Arrays: Linear Search and Binary Search

- 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, the program will have to compare the search key with half the elements of the array

Example. Search

```
1  <?xml version = "1.0" encoding = "utf-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 10.10: LinearSearch.html -->
6  <!-- Linear search of an array. -->
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8     <head>
9        <title>Linear Search of an Array</title>
10       <script type = "text/javascript">
11          <!--
12          var a = new Array( 100 );  // create an Array
13
14          // fill Array with even integer values from 0 to 198
15          for ( var i = 0; i < a.length; ++i )
16             a[ i ] = 2 * i;
17
18          // function called when "Search" button is pressed
19          function buttonPressed()
20          {
21             // get the input text field
22             var inputVal = document.getElementById( "inputVal" );
23
24             // get the result text field
25             var result = document.getElementById( "result" );
26
27             // get the search key from the input text field
28             var searchKey = inputVal.value;
29
```

Creates a new array to search

Initializes each array element with a value double its index

Example. Search

```
30             // Array a is passed to linearSearch even though it
31             // is a global variable. Normally an array will
32             // be passed to a method for searching.
33             var element = linearSearch( a, parseInt( searchKey ) );
34
35             if ( element != -1 )
36                result.value = "Found value in element " + element;
37             else
38                result.value = "Value not found";
39          } // end function buttonPressed
40
41          // Search "theArray" for the specified "key" value
42          function linearSearch( theArray, key )
43          {
44             // iterates through each element of the array in order
45             for ( var n = 0; n < theArray.length; ++n )
46                if ( theArray[ n ] == key )
47                   return n;
48
49             return -1;
50          } // end function linearSearch
51          // -->
52       </script>
53    </head>
54
```

Calls function `linearSearch` on array `a` with the value input by the user

Iterates through every element of the array until the key is found

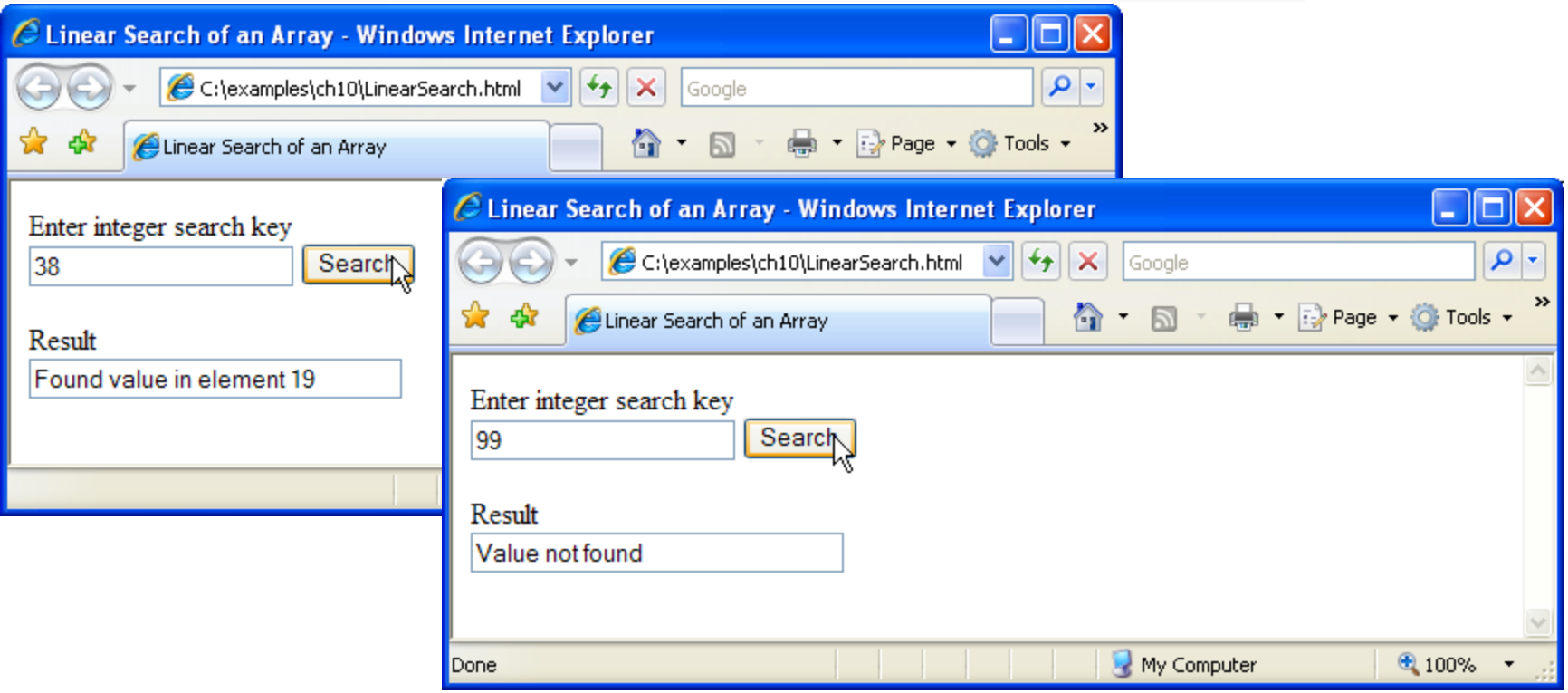
If the key is encountered, the index of the element with the key as its value is returned

If the key is not found, `-1` is returned

Example. Search

```
55    <body>
56       <form action  = "">
57          <p>Enter integer search key<br />
58          <input id = "inputVal" type = "text" />
59          <input type = "button" value = "Search"
60                 onclick = "buttonPressed()" /><br /></p>
61          <p>Result<br />
62          <input id = "result" type = "text" size = "30" /></p>
63       </form>
64    </body>
65 </html>
```

When the `Search` button is pressed, calls function `buttonPressed`



Searching Arrays: Linear Search and Binary Search (…)

- Binary search algorithm
  - More efficient than the linear search algorithm
  - Requires that the array be sorted
  - 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
- 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

Example. Search

```
1  <?xml version = "1.0" encoding = "utf-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 10.11: BinarySearch.html -->
6  <!-- Binary search of an array. -->
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8     <head>
9        <title>Binary Search</title>
10       <script type = "text/javascript">
11          <!--
12          var a = new Array( 15 );
13
14          for ( var i = 0; i < a.length; ++i )
15             a[ i ] = 2 * i;
16
17          // function called when "Search" button is pressed
18          function buttonPressed()
19          {
20             var inputVal = document.getElementById( "inputVal" );
21             var result = document.getElementById( "result" );
22             var searchKey = inputVal.value;
23
24             result.value = "Portions of array searched\n";
25
```

Example. Search

```
26            // Array a is passed to binarySearch even though it
27            // is a global variable. This is done because
28            // normally an array is passed to a method
29            // for searching.
30            var element =
31               binarySearch( a, parseInt( searchKey ) );
32
33            if ( element != -1 )
34               result.value += "\nFound value in element " + element;
35            else
36               result.value += "\nValue not found";
37         } // end function buttonPressed
38
39      // binary search function
40      function binarySearch( theArray, key )
41         {
42            var low = 0; // low subscript
43            var high = theArray.length - 1; // high subscript
44            var middle; // middle subscript
45
46            while ( low <= high ) {
47               middle = ( low + high ) / 2;
48
49               // The following line is used to display the
50               // part of theArray currently being manipulated
51               // during each iteration of the binary
52               // search loop.
53               buildOutput( theArray, low, middle, high );
54
```

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

Example. Search

```
55                if ( key == theArray[ middle ] )  // match
56                   return middle;
57                else if ( key < theArray[ middle ] )
58                   high = middle - 1; // search low end of array
59                else
60                   low = middle + 1; // search high end of array
61             } // end while
62
63             return -1;   // searchKey not found
64          } // end function binarySearch
65
66          // Build one row of output showing the current
67          // part of the array being processed.
68          function buildOutput( theArray, low, mid, high )
69          {
70             var result = document.getElementById( "result" );
71
72             for ( var i = 0; i < theArray.length; i++ )
73             {
74                if ( i < low || i > high )
75                   result.value += "    ";
76                else if ( i == mid ) // mark middle element in output
77                   result.value += theArray[ i ] +
78                      ( theArray[ i ] < 10 ? "*  " : "* " );
79                else
80                   result.value += theArray[ i ] +
81                      ( theArray[ i ] < 10 ? "   " : "  " );
82             } // end for
83
```

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

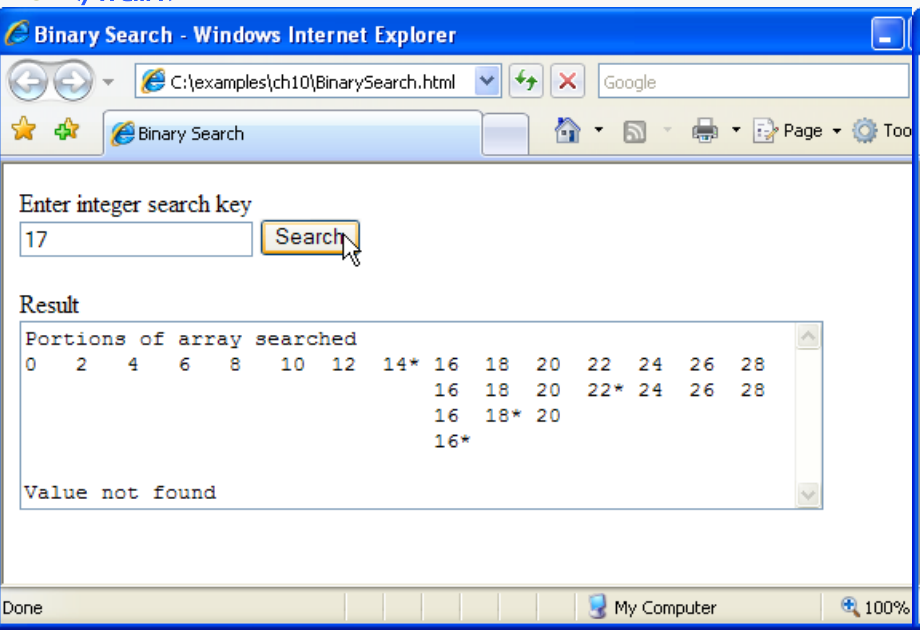
Otherwise, if the middle element's value is higher than the key, we only need to search the bottom half of the array

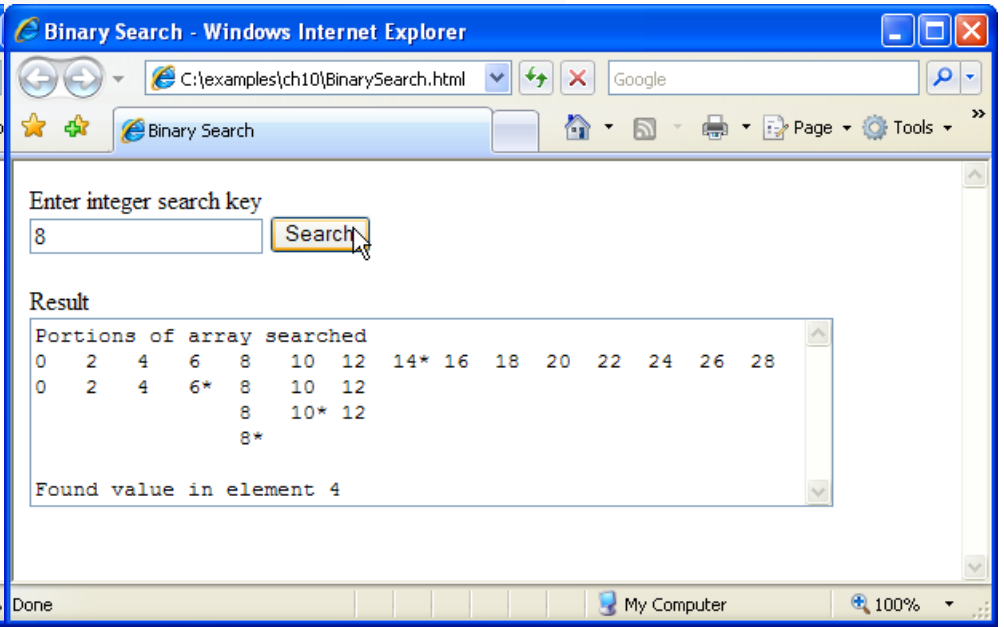
Otherwise, if the middle element's value is lower than the key, we only need to search the higher half of the array

If we've covered the whole array without encountering the key, return –1

Example. Search

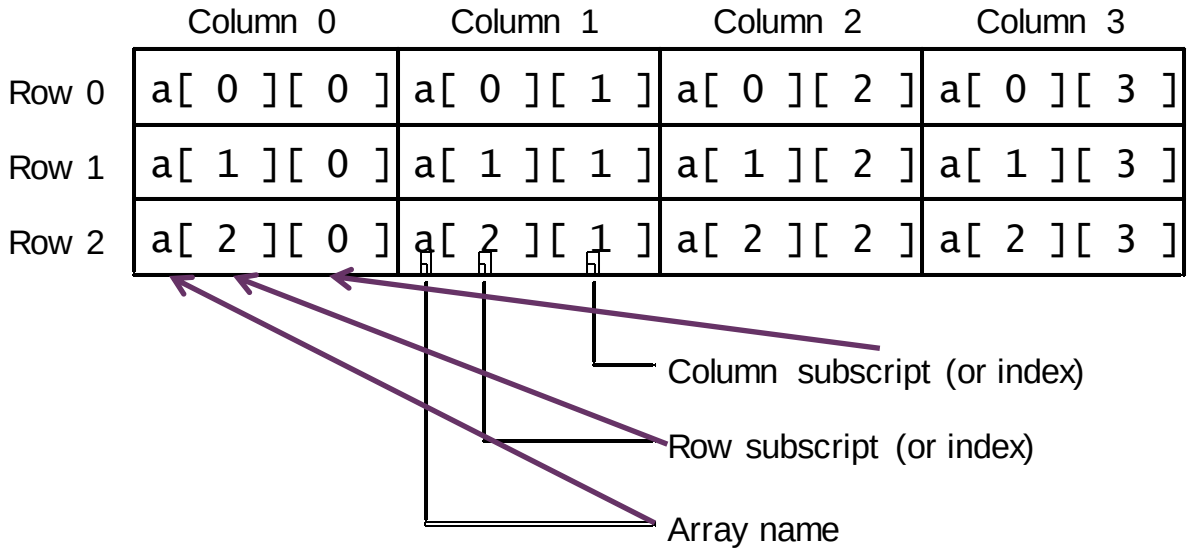
```
84             result.value += "\n";
85          } // end function buildOutput
86          // -->
87       </script>
88    </head>
89
90    <body>
91         <form action = "">
92          <p>Enter integer search key<br />
93          <input id = "inputVal" type = "text" />
94          <input type = "button" value = "Search"
95             onclick = "buttonPressed()" /><br /></p>
96          <p>Result<br />
97          <textarea id = "result" rows = "7" cols = "60">
98          </textarea></p>
99       </form>
100   </body>
101</html>
```





Multidimensional Arrays

- Two-dimensional arrays analogous to tables
  - Rows and columns
    - Specify row first, then column
  - Two subscripts



Multidimensional Arrays (…)

- 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

```
var matrix = [[10, 15, 20, 25],
              [30, 35, 40, 45],
              [50, 55, 60, 65]];

matrix[2][1]          // 55
matrix.length         // 3
matrix[1].length      // 4
```

- The rows of a two-dimensional array can vary in length

```
var b = [ [ 1, 2 ], [ 3, 4, 5 ] ];
```

Multidimensional Arrays

- A multidimensional array in which each row has a different number of columns can be allocated dynamically with operator new
  - Create array b with two rows, first with five columns and second with three:

```
var b;

b = new Array( 2 );
b[ 0 ] = new Array( 5 );
b[ 1 ] = new Array( 3 );
```

Multidimensional Arrays

- To identify a particular two-dimensional multidimensional array element
  - 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
- Two-dimensional array element accessed using an element name of the form a[i][j]
  - a is the name of the array
  - i and j are the subscripts that uniquely identify the row and column

Initializing multidimensional arrays

```
1  <?xml version = "1.0" encoding = "utf-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 10.13: InitArray3.html -->
6  <!-- Initializing multidimensional arrays. -->
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8     <head>
9        <title>Initializing Multidimensional Arrays</title>
10       <script type = "text/javascript">
11          <!--
12          var array1 = [ [ 1, 2, 3 ], // first row
13                         [ 4, 5, 6 ] ]; // second row
14          var array2 = [ [ 1, 2 ], // first row
15                         [ 3 ], // second row
16                         [ 4, 5, 6 ] ]; // third row
17
18          outputArray( "Values in array1 by row", array1 );
19          outputArray( "Values in array2 by row", array2 );
20
21          function outputArray( heading, theArray )
22          {
23             document.writeln( "<h2>" + heading + "</h2><pre>" );
24
25             // iterates through the set of one-dimensional arrays
26             for ( var i in theArray )
27             {
28                // iterates through the elements of each one-dimensional
29                // array
30                for ( var j in theArray[ i ] )
31                   document.write( theArray[ i ][ j ] + " " );
```

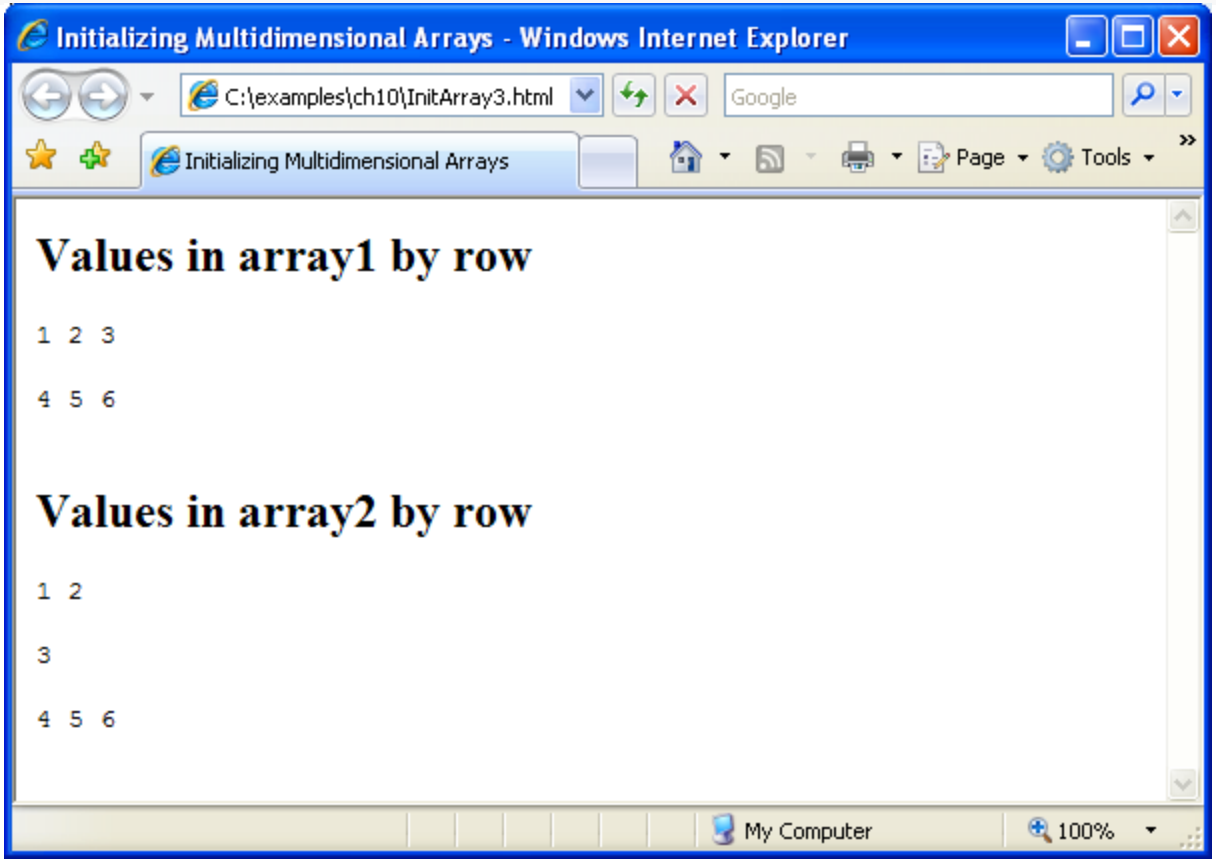
Initializes `array1` with an initializer list of sub initializer lists

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

Initializing multidimensional arrays

```
32
33                document.writeln( "<br />" );
34             } // end for
35
36             document.writeln( "</pre>" );
37          } // end function outputArray
38          // -->
39       </script>
40    </head><body></body>
41 </html>
```



More array methods…

- splice
  - arr.splice(start[, deleteCount, elem1, ..., elemN])
  - E.g.

```
let arr = ["I", "study", "JavaScript"];
arr.splice(1, 1); // from index 1 remove 1 element
alert( arr ); // ["I", "JavaScript"]
```

- slice
  - arr.slice([start], [end])
  - E.g.

```
let arr = ["t", "e", "s", "t"];
alert( arr.slice(1, 3) ); // e,s (copy from 1 to 3)
alert( arr.slice(-2) ); // s,t (copy from -2 till the end)
```

More array methods…

- concat
  - arr.concat(arg1, arg2...)
  - E.g.

```
let arr = [1, 2];
// create an array from: arr and [3,4]
alert( arr.concat([3, 4]) ); // 1,2,3,4
// create an array from: arr and [3,4] and [5,6]
alert( arr.concat([3, 4], [5, 6]) ); // 1,2,3,4,5,6
// create an array from: arr and [3,4], then add values 5 and 6
alert( arr.concat([3, 4], 5, 6) ); // 1,2,3,4,5,6
```

- forEach
  - arr.forEach(function(item, index, array) {// ... do something with item});
  - E.g.

```
["Bilbo", "Gandalf", "Nazgul"].forEach((item, index, array) => {
  alert(`${item} is at index ${index} in ${array}`);
});
```

Searching an array

- indexOf/lastIndexOf and includes
  - arr.indexOf(item, from) – looks for item starting from index from, and returns the index where it was found, otherwise -1
  - arr.includes(item, from) – looks for item starting from index from, returns true if found.
  - E.g.

```
let arr = [1, 0, false];


alert( arr.indexOf(0) ); // 1
alert( arr.indexOf(false) ); // 2
alert( arr.indexOf(null) ); // -1

alert( arr.includes(1) ); // true
```

Searching an array (2)

- find and findIndex/findLastIndex
  - let result = arr.find(function(item, index, array) {

    // if true is returned, item is returned and iteration is stopped

    // for falsy scenario returns undefined

    });
- The arr.findIndex method has the same syntax, but returns the index where the element was found instead of the element itself. The value of -1 is returned if nothing is found.
- The arr.findLastIndex method is like findIndex, but searches from right to left, similar to lastIndexOf.

Searching an array (3)

- filter
  - The find method looks for a single (first) element that makes the function return true.
  - If there may be many, we can use arr.filter(fn).
  - let results = arr.filter(function(item, index, array) {

    // if true item is pushed to results and the iteration continues

    // returns empty array if nothing found

    });
  - E.g.
    - let points = [12, 30, 70, 10, 90, 30];

    let passed = points.filter(item => item >= 50);

    console.log(passed);// [70, 90]

Transforming an array

- map
  - let result = arr.map(function(item, index, array) {
    // returns the new value instead of item
    });
  - E.g.
    let lengths = ["Bilbo", "Gandalf", "Nazgul"].map(item => item.length);
    alert(lengths); // 5,7,6
- reduce/reduceRight
  - These methods are used to calculate a single value based on the array.
  - let value = arr.reduce(function(accumulator, item, index, array) {
    // ...
    }, [initial])
  - E.g.
    let arr = [1, 2, 3, 4, 5];
    let result = arr.reduce((sum, current) => sum + current, 0);
    alert(result); // 15

Change an item

- with
  - let result = arr.with(index, replacingItem);
  - It returns a new array with the element at the given index replaced with the given value.
  - E.g.

```
let arr = ["Bilbo", "Gandalf", "Nazgul"];
arr.with(1, ‘Sauron’);

alert(arr); // 'Bilbo', 'Sauron', 'Nazgul'
```