```
 1  /**
 2   * Practice: some common programming idioms based
 3   * upon manipulating arrays ...
 4   */
 5
 6  var newArray; // global variable that is shared.
 7  var tempArray1, temparray2; // arrays that hold "intermediate" values --
     for testing swap, etc.
 8  /* A note on "global" variables.
 9   * Considered a bad practice in most modern programming texts,
10   * global variables expose implementation details to anyone
11   * who has access to the code. This means that any function
12   * can change the value of the variable from any place within
13   * the code, which results in difficult to understand/maintain
14   * software.
15   *
16   * Because the DOM/JavaScript interaction model is as it is,
17   * however, you will see frequent use of such constructions
18   * because ... well, it makes interprocess/interfunction
19   * communication easier.
20   */
21
22  //__________Methods__________
23
24  /*
25   * Note: each method is paired with its "event handler,"
26   * which is the method that is called from the Document,
27   * i.e., the HTML.
28   */
29  // ___private, helper methods, for input/output.
30  function formatArray( anyArray ) {
31     var description="[ ";
32     for( var index=0; index < anyArray.length - 1; index++ ) {
33        description += anyArray[ index ] + ", ";
34     }
35     return description + anyArray[ anyArray.length -1 ] + " ]";
36  }
37
38  function formatArrayUpToIndex( anArray, toIndex ) {
39     var start=0;
40     if( toIndex < 0 ) {
41        return "Target was not found in " + formatArray( anArray );
42     }
43     var returnString="[";
44     for( ; start < toIndex; start++ ) {
45        returnString += anArray[ start ] +", ";
46     }
47     if( start < anArray.length ) {
48        return returnString + anArray[ start ] + " ... ]"
49     } else {
50        return returnString + " ... ]";
51     }
```

```
 52    }
 53    //___private, helper methods ... end here.
 54
 55    /** ==> User provided methods paired with event-handers <=== **/
 56    /**
 57     * preconditions: n is a non-negative integer.
 58     * postconditions: a new array of size n is created and
 59     * populated with randomly generated integers, starting from 0
 60     * through n-1.
 61     *
 62     * A special note on using the Math.random() function:
 63     *    Math.random() returns a double (decimal) number between 0.0 and 1.
       0.
 64     *    Typically, we wish to generate random integers in a particular
 65     *    range: such as 0 through N. One way to do this is to
 66     *    multiply the Math.random() * N, which gives us another decimal
 67     *    number.
 68     */
 69    function genArray( n ) {
 70       var newArray = Array( n );
 71       for( var index=0; index < newArray.length; index++ ) {
 72          newArray[ index ] = Math.floor( ( Math.random() * n ) );
 73       }
 74       return newArray;
 75    }
 76    /*
 77     * Event Handler for genArray function:
 78     * Called from the document; feeds the genArray function
 79     * preconditions: input is non-negative integer, perhaps
 80     * no larger than 100 for sanity's sake ...
 81     */
 82    function testGenArray() {
 83       var size = parseInt( document.getElementById("input").value );
 84       var generatedArray = genArray( size );
 85       newArray = generatedArray; // remember this array for remainder of th
       e exercises.
 86       document.getElementById("exercise-
    1").innerHTML=formatArray( generate
       dArray );
 87    }
 88    /**
 89     * preconditions: the top-level newArray variable has been set to
 90     * a valid array as a result of completing the previous
 91     * exercise.
 92     * postconditions: the "index" of the "target" is returned; if the
 93     * target is not found in the newArray, then -1 is returned.
 94     */
 95    function indexOf( target ) {
 96       for( var index=0; index < newArray.length; index++ ) {
 97          if( newArray[ index ] === target ) {
 98             return index;
 99          }
100       }
101       return -1;
```

```
102    }
103    /**
104     * Event Handler for indexOf function.
105     * Retrieves input, tests your indexOf function, and
106     * pretty prints the results into the appropriate field.
107     */
108    function testIndexOf() {
109       var searchForTarget = parseInt( document.getElementById("target").val
       ue);
110       var locationInArray = indexOf( searchForTarget );
111       document.getElementById("exercise-
   2").innerHTML = formatArrayUpToInde
       x( newArray, locationInArray );
112    }
113
114    /**
115     * preconditions: the top-level newArray must have been created and must

116     * contain integers.
117     * postconditions: a new array (possibly empty) that contains only the e
       ven integers
118     * that appeared in the top-level newArray is returned. Note: this funct
       ion does
119     * not modify the top-level newArray!!
120     */
121    function filterEvens() {
122       var evensArray = Array(); //[];
123       for( var index = 0; index < newArray.length; index++ ) {
124          if( newArray[ index ] % 2 === 0 ) {
125             evensArray.push( newArray[ index ] );
126          }
127       }
128       return evensArray;
129    }
130
131    /**
132     * Event Handler: onlick.
133     * Tests the filterEvens() function.
134     */
135    function testFilterEvens() {
136       var onlyEvens = filterEvens();
137       document.getElementById("exercise-
   3").innerHTML = formatArray( onlyEv
       ens );
138    }
139    /**
140     * preconditions: anArray should be non-empty, and the parameters to and
        from must
141     * be valid indexes for anArray (less than anArray.length).
142     */
143    function swap( to, from, anArray ) {
144       var temp = anArray[ to ];
145       anArray[ to ] = anArray[ from ];
146       anArray[ from ] = temp;
```

```
147    }
148    // misc. function used to ensure separation
149    // of arrays (non-interference between arrays
150    // for swap testing.)
151    function copyArray( theArray ) {
152       var copy = Array( theArray.length );
153       for( var index = 0; index < theArray.length; index++ ) {
154          copy[ index ] = theArray[ index ];
155       }
156       return copy;
157    }
158    /**
159     * Event Handler: onlick.
160     * Tests the swap() procedure.
161     */
162    function testSwap() {
163       tempArray1 = genArray( 4 );
164       tempArray2 = copyArray( tempArray1 );
165       swap( 1, 3, tempArray1 );
166       document.getElementById("exercise-
4").innerHTML="Original array: " +
       formatArray( tempArray2 ) + " swapped elements at index 1 and index 3 =>
        " + formatArray( tempArray1 );
167
168    }
```