/**
 * Practice: some common programming idioms based
 * upon manipulating arrays ... */

var newArray; // global variable that is shared.
var tempArray1, temparray2; // arrays that hold "intermediate" values --
// for testing swap, etc.
/* A note on "global" variables.
* Considered a bad practice in most modern programming texts,
* global variables expose implementation details to anyone
* who has access to the code. This means that any function
* can change the value of the variable from any place within
* the code, which results in difficult to understand/maintain
* software.
* Because the DOM/JavaScript interaction model is as it is,
* however, you will see frequent use of such constructions
* because ... well, it makes interprocess/interfunction
* communication easier. */

//__________Methods__________

/* Note: each method is paired with its "event handler,"
* which is the method that is called from the Document,
* i.e., the HTML. */
// ___private, helper methods, for input/output.
function formatArray( anyArray ) {
  var description="[ ";
  for( var index=0; index < anyArray.length - 1; index++ ) {
    description += anyArray[ index ] + ", ";
  }
  return description + anyArray[ anyArray.length -1 ] + "]";
}

function formatArrayUpToIndex( anArray, toIndex ) {
  var start=0;
  if( toIndex < 0 ) {
    return "Target was not found in " + formatArray( anArray );
  }
  var returnString="["
  for( ; start < toIndex; start++ ) {
    returnString += anArray[ start ] +" ", ";
  }
  if( start < anArray.length ) {
    return returnString + anArray[ start ] + " ... ]" 
  } else {
    return returnString + " ... ]";
}
```javascript
/** ==> User provided methods paired with event-handers <=== */
/**
* preconditions: n is a non-negative integer.
* postconditions: a new array of size n is created and
* populated with randomly generated integers, starting from 0
* through n-1.
* A special note on using the Math.random() function:
* Math.random() returns a double (decimal) number between 0.0 and 1.
* Typically, we wish to generate random integers in a particular
* range: such as 0 through N. One way to do this is to
* multiply the Math.random() * N, which gives us another decimal
* number.
*/
function genArray( n ) {
    var newArray = Array( n );
    for( var index=0; index < newArray.length; index++ ) {
        newArray[ index ] = Math.floor( ( Math.random() * n ) );
    }
    return newArray;
}

/*
* Event Handler for genArray function:
* Called from the document; feeds the genArray function
* preconditions: input is non-negative integer, perhaps
* no larger than 100 for sanity's sake ...
*/
function testGenArray() {
    var size = parseInt( document.getElementById("input").value );
    var generatedArray = genArray( size );
    newArray = generatedArray; // remember this array for remainder of th
    exercises.
    document.getElementById("exercise-
1").innerHTML=formatArray( generate
dArray );
}

/*
* preconditions: the top-level newArray variable has been set to
* a valid array as a result of completing the previous
* exercise.
* postconditions: the "index" of the "target" is returned; if the
* target is not found in the newArray, then -1 is returned.
*/
function indexOf( target ) {
    for( var index=0; index < newArray.length; index++ ) {
        if( newArray[ index ] === target ) {
            return index;
        }
    }
    return -1;
```
**Event Handler for indexOf function.**
* Retrieves input, tests your indexOf function, and
* pretty prints the results into the appropriate field.
*/

```javascript
function testIndexOf() {
    var searchForTarget = parseInt(document.getElementById("target").value);
    var locationInArray = indexOf(searchForTarget);
    document.getElementById("exercise-2").innerHTML = formatArrayUpToIndex
    x(newArray, locationInArray);
}
```

*/

**preconditions: the top-level newArray must have been created and must
contain integers.**
*postconditions: a new array (possibly empty) that contains only the even integers
* that appeared in the top-level newArray is returned. Note: this function does
* not modify the top-level newArray!!
*/

```javascript
function filterEvens() {
    var evensArray = Array(); //[];
    for( var index = 0; index < newArray.length; index++ ) {
        if( newArray[index] % 2 === 0 ) {
            evensArray.push( newArray[index] );
        }
    }
    return evensArray;
}
```

*/

**Event Handler: onlick.**
* Tests the filterEvens() function.
*/

```javascript
function testFilterEvens() {
    var onlyEvens = filterEvens();
    document.getElementById("exercise-3").innerHTML = formatArray(onlyEvens);
}
```

*/

**preconditions: anArray should be non-empty, and the parameters to and from must
be valid indexes for anArray (less than anArray.length).**
*/

```javascript
function swap( to, from, anArray ) {
    var temp = anArray[to];
    anArray[to] = anArray[from];
    anArray[from] = temp;
}
```
// misc. function used to ensure separation
// of arrays (non-interference between arrays
// for swap testing.)
function copyArray( theArray ) {
    var copy = Array( theArray.length );
    for( var index = 0; index < theArray.length; index++ ) {
        copy[ index ] = theArray[ index ];
    }
    return copy;
}

/**
 * Event Handler: onlick.
 * Tests the swap() procedure.
 */
function testSwap() {
    tempArray1 = genArray( 4 );
    tempArray2 = copyArray( tempArray1 );
    swap( 1, 3, tempArray1 );
    document.getElementById("exercise-4").innerHTML="Original array: " + 
        formatArray( tempArray2 ) + " swapped elements at index 1 and index 3 => " + formatArray( tempArray1 );
}