```
/**
 * Scripts for managing the array-example-1 document. Note: this is pretty
 * rudimentary; the current logic does NOT attempt to reset and restart the
 * game through any user interaction with a control ... instead, the user
 * is asked to "reload" the page, which will clear and reset all of the
 * top-level variables and call the generateQuestion procedure.
 */

 /*
* Variable declarations
*/
var colorNames = [ "blue", "green", "red" ]; // array of color names (strings).
var currentIndex = 0; // keep track of currently selected color name.
var correctResponses = 0; // for keeping score (maybe later?)
//_______end variable declarations.

//_______methods (functions & procedures) definitions.
/**
* preconditions: assumes that the array and currentIndex are set
* by the caller BEFORE calling this procedure!
* postconditions: generates a question by choosing the current color
* name (by currentIndex) from the colorNames array.
*
* Note: called by both onload event and by the evalResponse() procedure.
*/
function generateQuestion() {
   updateScore();
   uncheckRadioButtons(); // clears any errant settings between questions
   /*
    * Because this could be called from evalResponse while at the
    * last element in the array (looking at the last color), we need
    * to check here.
    */
   if( currentIndex >= colorNames.length ) { // note the negation of "<"
      alert( "Out of colors. Reload this page to play again, or close the tab.");
      return;
   }
    document.getElementById("prompt").innerHTML="What is the complement for the color " + colorNames[ currentIndex ] + "? ";
}
/** preconditions: none.
*  postconditions: unchecks (clears) all radio buttons named "complement".
* Note: called only by the generateQuestion() procedure.
*/
function uncheckRadioButtons() {
   var radioButtons = document.getElementsByName("complement");
   /* an example of a "bounded iteration," because we
    * know the size (length) of the radioButtons array
    * at the time that the code below is executed.
    */
   for( var index=0; index < radioButtons.length; index++ ) {
       radioButtons[ index ].checked=false;
   }
}
/**
* preconditions: the colorNames array and the number of correctResponses have been
* defined and maintained.
* postconditions: updates information on the HTML element whose id="results" to
* show the number of correct/total number of responses.
*/
function updateScore() {
   document.getElementById("score").innerHTML="Current Score: " + correctResponses + " /" + colorNames.length;
}
/**
 * ---Main Entry Point: onclick event handler.
 *
 * preconditions: assumes that currentIndex is valid (within range).
 * postconditions: checks user's response by matching checked radio
 * button with chosen color.
 *
 * Presently increments the correctResponses variable ... but not reported
 * out.
 * Note: calls uncheckRadioButtons() and generateQuestion().
 */
function evalResponse() {
   if( currentIndex >= colorNames.length ) { // note the negation of "<"
      alert( "Out of colors. Reload this page to play again, or close the tab.");
      return;
   }
   /* retrieves the next color name from the
    * array, using the currentIndex. AFTER doing this, it
    * updates the currentIndex by 1.
    * This code can be written on two lines for clarity:
    * forColor = colorNames[ currentIndex ];
    * currentIndex = currentIndex + 1;
    */
   var forColor = colorNames[ currentIndex++ ];
   /*
    * This is the "pattern" for using radio buttons. You need to concern
    * yourself with the "details" at this time, unless you wish to
    * use radio buttons in your own documents.
    * Radio Buttons, by the way, are more commony used in "forms processing,"
    * which does not concern us because we are writing "client side" scripts.
    */
   if(( forColor == "red" && document.getElementById("cyan").checked === true ) ||
      ( forColor == "green" && document.getElementById("magenta").checked === true ) ||
      ( forColor == "blue" && document.getElementById("yellow").checked === true ) )
   {
       correctResponses++;
       /* we can either update the scores within this procedure,
        * or we can depend upon the common procedure "generateQuestion()"
        * to update the score before it generates the "next" question.
        * I leave this logic here for you to uncomment just to satisfy
        * your curiosity.
        */
       //updateScore();
   } else {
       correctResponses--;
       /* see the comment above. */
       //updateScore();
   }
   /*
    * Ask: why is it "safe" to call this here?
    * Hint: look at the first line of code in this procedure..
    */
   generateQuestion();
}
//_______end methods definitions.
```