Line wrap ☐

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <title>Array Example 4b</title>

<!--
  http://cssdweb.edu/StudentResources/CodeSamples/ArrayExample_4b.html
  Copyright (c) 2022-2023 by Jesse Heines.  All rights reserved.  May be freely
    copied or excerpted for educational purposes with credit to the author.
  updated by JMH on June 12, 2022 at 8:06 PM
  updated by JMH on May 29, 2023 at 8:34 AM
-->

  <!-- read the stylesheet for this application -->
  <link rel="stylesheet" href="ArrayExample_4b.css">

  <script>
    //// INITIALIZE THE ARRAY WITH THE DATA WE WANT TO WORK WITH

    /* This statement declares variable arrCSSD to be an array.  It will actually be a
     * two-dimensional (2D) array by the time we finish populating it with data.  A 2D
     * array is equivalent to a table with rows and column.  Each top level element of
     * the array represents a row, and the data in the secondary array represents the
     * columns.
     */
    var arrCSSD = new Array() ;

    /* The array data is read from an auxiliary JavaScript file so that it can be easily
     * changed from one semester to another.  Putting the data in an auxiliary file
     * separates the program LOGIC from the DATA that the program works on.  This is
     * analogous to specifying all program layout via CSS to separate program CONTENT from
     * RENDITION.
     *
     * The statements in the auxiliary JavaScript file "push" onto the array, which means
     * they add data at the end of the array.  Each statement pushes an entire subarray
     * onto the top level array, resulting in a data structure with the 4 "columns".  These
     * columns are, in order:
     *      | Room # | Instructor Name | Course Name | Time Slot |
     * where "Day Period" is:
     *      "A" for  7:10- 8:30
     *      "B" for  8:40-10:00
     *      "C" for 10:10-11:30
     *      "D" for 11:40- 1:00
     *      "E" for  1:10- 2:30
     */
  </script>

  <!-- load the data for this application -->
  <script src="ArrayExample_4b_2023-04.js"></script>

  <script>
    //// SET THE DESIRED SORT COLUMN FROM THE URL SEARCH STRING

    // nSortColumn is the number of the column by which the data will be sorted.  The
    // default is 1, but this can be changed by clicking a different column header cell.
    var nSortColumn = 1 ;

    // The following code determines whether a query (search) string has been supplied
    // and, if so, whether that query string contains a SortColumn parameter and a legal
    // value (1-4). If it does, the value of the SortColumn parameter replaces the
    // nSortColumn value so that the table data is sorted as the user desires.
    //    Note that for simplicity, the following code does not contain a lot of error
    // checking.  In a real application you have to have a lot of error checking whenever
    // you get input from the user to ensure that the data he or she enters is valid.  In
    // many of the production quality programs that I have written personally, the amount
    // of error-checking code EXCEEDS the amount of processing code!
    //    The following code could also be made more efficient, but I have written it our
    // step-by-step for clarity.

    // (1) get the search (query) string, which is the part of the URL that begins with a
    //     question mark (?)
    var strSearchString = window.location.search ;

    // (2) if the search string exists, process it, otherwise the following code is skipped
    //     and the default sort column defined above is used
    if ( strSearchString != "" ) {

      // (3) call the built-in URLSearchParams function to parse the query string into an
      //     data structure of key/value pairs, where the key is the name of the parameter
      //     and the value is the string assigned to that parameter
      var urlParams = new URLSearchParams( strSearchString ) ;

      // (4) retrieve the value we want: the one associated with key "SortColumn"
      var strSortColumn = urlParams.get( "SortColumn" ) ;

      // (5) CRITICALLY IMPORTANT!  Values extracted from query strings are ALWAYS STRINGS.
      //     For use in our program, we must therefore convert string variable strSortColumn
      //     to a number (variable nSortColumn) using the built-in eval function.
      nSortColumn = eval( strSortColumn ) ;
    }


    //// COMPARISON FUNCTIONS TO SORT THE TABLE DATA ON THE DESIRED COLUMN

    // Note:  The following 4 functions could be combined into a single function with some
    // additional programming, but I have left them as separate functions for clarity.  We
    // could also write additional functions that sort the specified columns in descending
    // order.

    // This function compares the first (0th) items in the subarrays, which are the
    // ROOM NUMBERS.  It returns -1 if the 1st item should come before the 2nd. +1 if the
    // 1st item should come after the 2nd, 0 if their current order is OK.
    var fnRoomAscending = function( a, b ) {
      if ( a[0] < b[0] ) {
        return -1 ;
      } else if ( a[0] > b[0] ) {
        return +1 ;
      } else {
        return 0 ;
      }
    }

    // This function compares the second (index 1) items in the subarrays, which are the
    // INSTRUCTOR NAMES.  It returns -1 if the 1st item should come before the 2nd. +1 if
    // the 1st item should come after the 2nd, 0 if their current order is OK.
    var fnNameAscending = function( a, b ) {
      if ( a[1] < b[1] ) {
        return -1 ;
      } else if ( a[1] > b[1] ) {
        return +1 ;
      } else {
        return 0 ;
      }
    }

    // This function compares the third (index 2) items in the subarrays, which are the
    // COURSE NAMES.  It returns -1 if the 1st item should come before the 2nd. +1 if the
    // 1st item should come after the 2nd, 0 if their current order is OK.
    var fnCourseAscending = function( a, b ) {
      if ( a[2] < b[2] ) {
        return -1 ;
      } else if ( a[2] > b[2] ) {
        return +1 ;
      } else {
        return 0 ;
      }
    }

    // This function compares the fourth (index 3) items in the subarrays, which are the
    // TIME SLOTS.  Remember that the time slots are encoded as "A", "B", "C", "D", or "E".
    // Just like the other functions, this one also returns -1 if the 1st item should come
    // before the 2nd. +1 if the 1st item should come after the 2nd, 0 if their current
    // order is OK.
    var fnTimeAscending = function( a, b ) {
      if ( a[3] < b[3] ) {
        return -1 ;
      } else if ( a[3] > b[3] ) {
        return +1 ;
      } else {
        return 0 ;
      }
    }


    //// SORT THE ARRAY ON THE DESIRED COLUMN

    // note the need to use the appropriate comparison function
    switch ( nSortColumn ) {
      case 1 :  // sort by Room #
        arrCSSD.sort( fnRoomAscending ) ;
        break ;
      case 2 :  // sort by Instructor Name
        arrCSSD.sort( fnNameAscending ) ;
        break ;
      case 3 :  // sort by Course Name
        arrCSSD.sort( fnCourseAscending ) ;
        break ;
      case 4 :  // sort by Time Slot
        arrCSSD.sort( fnTimeAscending ) ;
        break ;
    }


    //// FUNCTIONS TO DISPLAY THE DATA TABLE

    // This is a "helper" function that returns the time slot given its letter identifier.
    // Note that the string returned is actually a table itself that goes inside the last
    // column of the master table.
    var TimeSlot = function( strLetterID ) {
      var strReturn = "<table id='timeslot' cellpadding='0' cellspacing='0'><tr><td>" ;
      switch ( strLetterID ) {
        case "A" : strReturn +=  "7:10 AM</td><td>&nbsp;&ndash;&nbsp;</td><td> 8:30 AM" ; break ;
        case "B" : strReturn +=  "8:40 AM</td><td>&nbsp;&ndash;&nbsp;</td><td>10:00 AM" ; break ;
        case "C" : strReturn += "10:10 AM</td><td>&nbsp;&ndash;&nbsp;</td><td>11:30 AM" ; break ;
        case "D" : strReturn += "11:40 AM</td><td>&nbsp;&ndash;&nbsp;</td><td> 1:00 PM" ; break ;
        case "E" : strReturn +=  "1:10 PM</td><td>&nbsp;&ndash;&nbsp;</td><td> 2:30 PM" ; break ;
      }
      strReturn += "</td></tr></table>" ;
      // the following statement was used for debugging
      // console.log( strReturn ) ;
      return strReturn ;
    }

    // This function displays the table in whatever order it is currently sorted.
    // https://www.tutorialspoint.com/How-to-add-rows-to-a-table-using-JavaScript-DOM
    var DisplayTable = function( nSortColumn ) {
      // Loop through the entire array, displaying the subarray in each top-level element
      // in a separate row.
      for ( var k = 0 ; k < arrCSSD.length ; k++ ) {
        let table = document.getElementById( "master") ;
        // insertRow parameter -1 causes the new row is to be added at the end of the table
        let row = table.insertRow( -1 ) ;
        row.insertCell().innerHTML = arrCSSD[k][0] ;
        row.insertCell().innerHTML = arrCSSD[k][1] ;
        // if a room is closed, display that message in italics
        if ( arrCSSD[k][2] == "Room Closed" ) {
          row.insertCell().innerHTML = "<em>" +arrCSSD[k][2] + "</em>" ;
        } else {
          row.insertCell().innerHTML = arrCSSD[k][2] ;
        }
        row.insertCell().innerHTML = TimeSlot( arrCSSD[k][3] ) ;
      }

      // Highlight the header cell of the column on which the data is sorted.
      // Be careful!  The getElementsByTagName function returns a *collection*, which is
      // similar to, but not quite the same as an *array*!
      var colHeaderCells = document.getElementsByTagName("th") ;
      colHeaderCells[nSortColumn-1].setAttribute(
        "style", "background-color: lightgreen ; color: black" ) ;
    }


    //// FUNCTION TO RELOAD THE PAGE WITH A SPECIFIED SORT COLUMN

    // This function is executed when the user clicks a column header cell.  The parameter
    // passed to this function is the number of the column on which to sort the table.
    var ReloadPage = function( nSortColumn ) {
      // The following statement returns:
      //    "/StudentResources/CodeSamples/ArrayExample_4b.html"
      var strThisPagePath = window.location.pathname ;

      // To this we want to add a parameter, which is part of the search (or query) string.
      // A question mark begins the search string part of a URL.
      // Each parameter is then written in name=value format (without quotes).
      // Thus, what we want is the page path returned by the above statement, then a
      //   question mark, then the search parameter name ("SortColumn") followed by an
      //   equals sign (=) and finally followed by the parameter value, which in our
      //   program is the number of the desired sort column passed to this function.
      var strFullPagePath = strThisPagePath + "?SortColumn=" + nSortColumn ;

      // The final step is to replace the current page with the full path that we just
      // constructed.  In our program, this is the same page that we are currently viewing,
      // but with a parameter specifying the desired sort column.
      window.location.replace( strFullPagePath ) ;
    }
  </script>

</head>

<body>

  <!-- display the page title and semester dates -->
  <h2 id="title">Corrections Special School District<br>
    Career &amp; Technical Education Center</h2>

  <!-- display the name of this HTML file and its search string -->
  <p><em>This is file:&nbsp;</em><strong> <span id="filename"></span></strong><br>
    <em>The search string is:&nbsp;</em><strong> <span id="searchstring"></span></strong></p>

  <!-- display the term for which data has been loaded -->
  <h3 id="semester"><em>Term:</em>&nbsp; <span id="term"></span></h3>

  <!-- display the skeleton of the message stating how the data is sorted -->
  <p>This is option <span id="OptionNumber"></span>, in which the data is ordered by
    <span id="OptionName"></span>.</p>

  <script>
    // the following code completes the above message stating how the data is sorted
    document.getElementById( "OptionNumber" ).innerHTML = nSortColumn ;
    switch ( nSortColumn ) {
      case 1 :  // sorted on the Room Number
        document.getElementById( "OptionName" ).innerHTML = "Room Number" ;
        break ;
      case 2 :  // sorted on the Instructor Name
        document.getElementById( "OptionName" ).innerHTML = "Instructor Name" ;
        break ;
      case 3 :  // sorted on the Course name
        document.getElementById( "OptionName" ).innerHTML = "Course Name" ;
        break ;
      case 4 :  // sorted on the Time Slot
        document.getElementById( "OptionName" ).innerHTML = "Time Slot" ;
        break ;
    }
  </script>

  <!-- This is the table of room numbers, instructor names, course names, and time slots. -->
  <table id="master" cellpadding="5" cellspacing="0" border="1">
    <tr>
      <!--
        Note that each table header cell has an onclick parameter.  The value of this
        parameter is the name of a function that will be called when that table cell is
        clicked.  To that function we pass the number of the column by which we want the
        table data to be sorted.
      -->
      <th onclick="ReloadPage( 1 )">Room #</th>
      <th onclick="ReloadPage( 2 )">Instructor</th>
      <th onclick="ReloadPage( 3 )">Course</th>
      <th onclick="ReloadPage( 4 )">Time Slot</th>
    </tr>

  </table>

  <script>
    // insert the string showing the name of the current file
    document.getElementById( "filename" ).innerHTML = location.pathname ;

    // insert the string showing the search string
    if ( location.search == "" ) {
      document.getElementById( "searchstring" ).innerHTML = "{there is no search string}" ;
    } else {
      document.getElementById( "searchstring" ).innerHTML = location.search ;
    }

    // insert the string showing the term for this data
    document.getElementById( "term" ).innerHTML = strTerm ;

    // we are now finally ready to display the table! :)
    DisplayTable( nSortColumn ) ;
  </script>

</body>

</html>
```