

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>Array Example 4</title>
6
7 <!--
8   File:  \\cssd.Local\Student Resources\Website Development I\CodeSamples\ArrayExample_4.html
9   Copyright (c) 2022 by Jesse Heines. All rights reserved. May be freely
10  copied or excerpted for educational purposes with credit to the author.
11  updated by JMH on June 6, 2022 at 10:27 AM
12 -->
13
14 <script>
15   //// CODE TO INITIALIZE THE ARRAY OF DATE WE WANT TO WORK WITH
16
17   /* This statement declares variable arrCSSD to be an array. It will actually be a
18   two-dimensional (2D) array by the time we finish populating it with data. A 2D
19   array is equivalent to a table with rows and column. Each top level element of
20   the array represents a row, and the data in the secondary array represents the
21   columns. */
22   var arrCSSD = new Array() ;
23   /* The statements that follow "push" onto the array, which means they add data at the
24   end. Each statement pushes an entire subarray onto the top level array, resulting
25   in a data structure with the 4 "columns". These columns, in order, are:
26       | Room # | Instructor Name | Course Name | Time Slot |
27   where "Day Period" is "A" for 7:10-8:30, "B" for 8:40=10:00, "C" for 10:10-11:30,
28   "D" for 11:40-1:00, and "E" for 1:10-2:30. */
29   arrCSSD.push( [ 1, "Lees", "Economics", "A" ] ) ;
30   arrCSSD.push( [ 1, "Lees", "Geography", "B" ] ) ;
31   arrCSSD.push( [ 2, "Burzycki", "Biology IA", "D" ] ) ;
32   arrCSSD.push( [ 2, "Burzycki", "Biology IB", "E" ] ) ;
33   arrCSSD.push( [ 3, "Latorella", "World Literature IA", "A" ] ) ;
34   arrCSSD.push( [ 3, "Latorella", "Special Units", "B" ] ) ;
35   arrCSSD.push( [ 3, "Latorella", "American Literature IA", "D" ] ) ;
36   arrCSSD.push( [ 3, "Latorella", "Tolkien", "E" ] ) ;
37   arrCSSD.push( [ 4, "Jones", "CDL Prep.", "B" ] ) ;
38   arrCSSD.push( [ 4, "Jones", "CDL Prep.", "D" ] ) ;
39   arrCSSD.push( [ 6, "Mosher", "Algebra IA", "A" ] ) ;
40   arrCSSD.push( [ 6, "Mosher", "Computation for Business", "B" ] ) ;
41   arrCSSD.push( [ 7, "Piper", "Short Story", "D" ] ) ;
42   arrCSSD.push( [ 7, "Piper", "Mythology", "E" ] ) ;
43   arrCSSD.push( [ 8, "Kroll", "Purchasing", "D" ] ) ;
44   arrCSSD.push( [ 8, "Kroll", "Nutrition for Culinary", "E" ] ) ;
45   arrCSSD.push( [ 10, "Spires", "Small Business Management", "A" ] ) ;
46   arrCSSD.push( [ 10, "Spires", "Advanced Microsoft Applications", "D" ] ) ;
47   arrCSSD.push( [ 10, "Spires", "Career Awareness", "E" ] ) ;
48
49
50   //// FUNCTIONS TO DISPLAY THE DATA TABLE
51
52   // This is a "helper" function that returns the time slot given its letter identifier.
53   // Note that the string returned is actually a table itself that goes inside the last
54   // column of the master table.
55   var TimeSlot = function( strLetterID ) {
56     var strReturn = "<table id='timeslot' cellpadding='0' cellspacing='0'><tr><td>" ;
57     switch ( strLetterID ) {
58       case "A" : strReturn += "7:10 AM</td><td>&nbsp;&ndash;&nbsp;  </td><td>8:30 AM" ; break ;
59       case "B" : strReturn += "8:40 AM</td><td>&nbsp;&ndash;&nbsp;  </td><td> 10:00 AM" ; break ;
60       case "C" : strReturn += "10:10 AM</td><td>&nbsp;&ndash;&nbsp;  </td><td> 11:30 AM" ; break ;
61       case "D" : strReturn += "11:40 AM</td><td>&nbsp;&ndash;&nbsp;  </td><td> 1:00 PM" ; break ;
62       case "E" : strReturn += "1:10 PM</td><td>&nbsp;&ndash;&nbsp;  </td><td> 2:30 PM" ; break ;
63     }
64     strReturn += "</td></tr></table>" ;

```

```

65     // the following statement was used for debugging
66     // console.log( strReturn );
67     return strReturn ;
68 }
69
70 // This function displays the table in whatever order it is currently sorted.
71 var DisplayTable = function( nSortColumn ) {
72     // Loop through the entire array, displaying the subarray in each top-level element
73     // in a separate row.
74     for ( var k = 0 ; k < arrCSSD.length ; k++ ) {
75         document.writeln( "<tr>" ) ;
76         document.writeln( " <td>" + arrCSSD[k][0] + "</td>" ) ;
77         document.writeln( " <td>" + arrCSSD[k][1] + "</td>" ) ;
78         document.writeln( " <td>" + arrCSSD[k][2] + "</td>" ) ;
79         document.writeln( " <td>" + TimeSlot( arrCSSD[k][3] ) + "</td>" ) ;
80         document.writeln( "</tr>" ) ;
81     }
82
83     // Highlight the header cell of the column on which the data is sorted.
84     // Be careful! The getElementsByTagName function returns a *collection*, which is
85     // similar to, but not quite the same as an *array*!
86     var colHeaderCells = document.getElementsByTagName("th") ;
87     colHeaderCells[nSortColumn-1].setAttribute(
88         "style", "background-color: lightgreen ; color: black" ) ;
89 }
90
91
92 //// FUNCTIONS TO SORT THE TABLE DATA ON THE DESIRED COLUMN
93
94 // Note: The following 4 functions could be combined into a single function with some
95 // additional programming, but I have left them as separate functions for clarity. We
96 // could also write additional functions which sort the specified columns in descending
97 // order.
98
99 // This function compares the first (0th) items in the subarrays, which are the
100 // ROOM NUMBERS. It returns 1 if the 1st item should come after the 2nd, 0 if their
101 // current order is OK, and -1 if the 1st item should come after the 2nd.
102 var fnRoomAscending = function( a, b ) {
103     if ( a[0] < b[0] ) {
104         return -1 ;
105     } else if ( a[0] > b[0] ) {
106         return +1 ;
107     } else {
108         return 0 ;
109     }
110 }
111
112 // This function compares the second (index 1) items in the subarrays, which are the
113 // INSTRUCTOR NAMES. It returns 1 if the 1st item should come after the 2nd, 0 if
114 // their current order is OK, and -1 if the 1st item should come after the 2nd.
115 var fnNameAscending = function( a, b ) {
116     if ( a[1] < b[1] ) {
117         return -1 ;
118     } else if ( a[1] > b[1] ) {
119         return +1 ;
120     } else {
121         return 0 ;
122     }
123 }
124

```

```
125 // This function compares the third (index 2) items in the subarrays, which are the
126 // COURSE NAMES. It returns 1 if the 1st item should come after the 2nd, 0 if their
127 // current order is OK, and -1 if the 1st item should come after the 2nd.
128 var fnCourseAscending = function( a, b ) {
129     if ( a[2] < b[2] ) {
130         return -1 ;
131     } else if ( a[2] > b[2] ) {
132         return +1 ;
133     } else {
134         return 0 ;
135     }
136 }
137
138 // This function compares the fourth (index 3) items in the subarrays, which are the
139 // TIME SLOTS. Remember that the time slots are encoded as "A", "B", "C", "D", or "E".
140 // Just like the other functions, this one also returns 1 if the 1st item should come
141 // after the 2nd, 0 if their current order is OK, and -1 if the 1st item should come
142 // after the 2nd.
143 var fnTimeAscending = function( a, b ) {
144     if ( a[3] < b[3] ) {
145         return -1 ;
146     } else if ( a[3] > b[3] ) {
147         return +1 ;
148     } else {
149         return 0 ;
150     }
151 }
152
153
154 //// FUNCTION TO RELOAD THE PAGE WITH A SPECIFIED SORT COLUMN
155
156 // This function is executed when the user clicks on a column header cell. The
157 // parameter passed to this function is the number of the column on which we wish to
158 // sort the table.
159 var ReloadPage = function( nSortColumn ) {
160     // The following statement returns:
161     //     "/StudentResources/CodeSamples/ArrayExample_4.html"
162     var strThisPagePath = window.location.pathname ;
163
164     // To this we want to add a query parameter, which is part of a query string.
165     // The question mark begins the query string part of a URL.
166     // Each parameter is then written in name=value format.
167     // Thus, what we want is the page path returned by the above statement, then a
168     // question mark, then the query parameter name ("SortColumn") followed by an
169     // equals sign (=) and finally followed by the parameter value, which in our
170     // current case is the number of the desired sort column passed to this function.
171     var strFullPagePath = strThisPagePath + "?SortColumn=" + nSortColumn ;
172
173     // The final step is to replace the current page with the full path that we just
174     // constructed. In our case, this is the same page that we are currently viewing,
175     // but with a parameter specifying the desired sort column.
176     window.location.replace( strFullPagePath ) ;
177 }
178
```

```

179      //// CODE TO SET THE DESIRED SORT COLUMN
180
181      // nSortColumn is the number of the column by which the data will be sorted. The
182      // default is 1, but this can be changed by clicking on a different column header cell.
183      var nSortColumn = 1 ;
184
185      // The following code determines whether a query (search) string has been supplied
186      // and, if so, whether that query string contains a SortColumn parameter and a legal
187      // value (1-4). If it does, the value of the SortColumn parameter replaces the
188      // nSortColumn value so that the table data is sorted as the user desires.
189      // Note that for simplicity, the following code does not contain a lot of error
190      // checking. In a real application you have to have a lot of error checking whenever
191      // you get input from the user to ensure that the data he or she enters is valid. In
192      // many of the production quality programs that I have written personally, the amount
193      // of error-checking code EXCEEDS the amount of processing code!
194      // The following code could also be made more efficient, but I have written it our
195      // step-by-step for clarity.
196
197      // (1) get the query (search) string, which starts with a question mark (?)
198      var strSearchString = window.location.search ;
199      if ( strSearchString != "" && strSearchString.startsWith( "?SortColumn" ) ) {
200          // (2) split the search string on the equals sign (=), which returns an array
201          arrSearchString = strSearchString.split( "=" ) ;
202          // (3) the value we want is the 2nd (index 1) element of the array
203          strSortColumn = arrSearchString[1] ;
204          // (4) CRITICAL! Note that values extracted from query strings are ALWAYS STRINGS,
205          // not numbers. For use in our program we must therefore convert string variable
206          // strSortColumn to a number (variable nSortColumn) using the built-in eval function.
207          nSortColumn = eval( strSortColumn ) ;
208      }
209 </script>
210
211 <style>
212     h2#title {                               /* main page title */
213         color: blue ;
214         margin-bottom: 0.2em ;
215     }
216     h3#semester {                             /* page subtitle (semester dates) */
217         color: blue ;
218         margin-top: 0 ;
219     }
220     table#master tr th {                       /* formatting for the table header row */
221         background-color: black ;
222         color: white ;
223     }
224     table#master tr th:hover {                 /* formatting change when a header cell is moused over */
225         background-color: yellow ;
226         color: black ;
227     }
228     /* Note that the nth-child pseudo-selector's argument begins at 1, not 0. (Sigh.) */
229     table#master tr td:nth-child(1) {         /* formatting for column 1 */
230         text-align: center ;
231     }
232     table#master tr td:nth-child(4) {         /* formatting for column 4 */
233         text-align: right ;
234     }
235     table#timeslot tr td:nth-child(1),        /* formatting for the Time Slot subtable */
236     table#timeslot tr td:nth-child(3) {      /* that is the data in column 4 */
237         width: 4.09em ;
238         text-align: right ;
239     }
240 </style>
241
242 </head>
243

```

```

244 <body>
245 <!-- display the page title and semester dates -->
246 <h2 id="title">Corrections Special School District</h2>
247 <h3 id="semester">Semester:&nbsp; April 4 to June 17, 2022</h3>
248
249 <script>
250 // The following code displays a message stating how the data is sorted.
251 // Note the use of document.write instead of document.writeln to prevent extra spaces
252 // from being added where not desired, such as before the period.
253 // -- document.writeln outputs text followed by a carriage return, which the browser
254 // displays as a space.
255 // -- document.write outputs text without a carriage return.
256 document.write( "<p>This is option " + nSortColumn + ", in which the data is " +
257 "ordered by:&nbsp; <strong style='background-color: lightgreen'>&nbsp;&nbsp;&nbsp;</strong>" );
258 switch ( nSortColumn ) {
259 case 1 : // sorted on the Room Number
260 document.write( "Room Number" ); break ;
261 case 2 : // sorted on the Instructor Name
262 document.write( "Instructor Name" ); break ;
263 case 3 : // sorted on the Course name
264 document.write( "Course Name" ); break ;
265 case 4 : // sorted on the Time Slot
266 document.write( "Time Slot" ); break ;
267 }
268 document.writeln( "&nbsp;</strong></p>" );
269 </script>
270
271 <!-- This is the table of room numbers, instructor names, course names, and time slots. -->
272 <table id="master" cellpadding="5" cellspacing="0" border="1">
273 <tr>
274 <!--
275 Note that each table header cell has an onclick parameter. The value of this
276 parameter is the name of a function that will be called when that table cell is
277 clicked. To that function we pass the number of the column by which we want the
278 table data to be sorted.
279 -->
280 <th onclick="ReloadPage( 1 )">Room #</th>
281 <th onclick="ReloadPage( 2 )">Instructor</th>
282 <th onclick="ReloadPage( 3 )">Course</th>
283 <th onclick="ReloadPage( 4 )">Time Slot</th>
284 </tr>
285
286 <script>
287 // Before displaying the table, we sort the array on the desired column using the
288 // appropriate comparison function.
289 switch ( nSortColumn ) {
290 case 1 : // sort by Room #
291 arrCSSD.sort( fnRoomAscending ); break ;
292 case 2 : // sort by Instructor Name
293 arrCSSD.sort( fnNameAscending ); break ;
294 case 3 : // sort by Course Name
295 arrCSSD.sort( fnCourseAscending ); break ;
296 case 4 : // sort by Time Slot
297 arrCSSD.sort( fnTimeAscending ); break ;
298 }
299
300 // We are now finally ready to display the table! :)
301 DisplayTable( nSortColumn );
302 </script>
303 </table>
304 </body>
305
306 </html>

```