

Session 9

Web page that accepts a matrix and computes its transpose

This session produces one HTML file with inline CSS and JavaScript, no libraries. The page asks for the number of rows and columns, builds that many text boxes when the user clicks Input Elements, and shows the transpose when the user clicks Compute Transpose. It is the first web page in the course and the page you test in Session 10. The point is not the transpose, which is one loop; the point is creating form controls at run time, wiring events to them, and reading a grid of inputs back into an array.

Objectives

✖ Do not copy. Read for understanding and the viva

- Build a page whose second stage of inputs does not exist until the first stage is submitted
- Create elements with `document.createElement` and attach them with `appendChild`
- Attach click handlers with `addEventListener` and pass the dimensions into the handler
- Read a grid of text boxes into a two-dimensional array by id
- Validate at both stages and show the error in the page, not in an alert

Problem Statement

■ Write in lab record

Design a web page that accepts a matrix as input and computes its transpose. The web page should have two text boxes and a submit button labelled as Input Elements. After entering the number of rows of the input matrix in the first text box and number of columns of the input matrix in the second text box of the web page, SUBMIT button should be clicked. Once clicked, a number of text boxes which are equivalent to the number of elements in the matrix will appear along with a submit button at the bottom labelled as Compute Transpose. When the Compute Transpose button is clicked, the transpose of the input matrix has to be displayed.

Concept

✗ Do not copy. Read for understanding and the viva

Two stages, one page

The page has a fixed part (rows, columns, Input Elements) written in HTML, and a generated part (the grid, Compute Transpose, the result) that JavaScript creates after the first click. Nothing is submitted to a server; there is no `form` tag and no page reload. Every click is handled in the browser.

Creating elements at run time

`document.createElement("input")` makes a text box that is not yet on the page. Setting its properties (`type` , `id`) and calling `parent.appendChild(box)` puts it in the document. The grid is a `table` built row by row: for each row a `tr` , for each column a `td` containing one `input` . Giving every box an id of the form `a<row>_<col>` (for example `a1_2` for row 1, column 2, counting from zero) means the second handler can find each box with `getElementById` without keeping any array of references.

Passing the dimensions to the second handler

The Compute Transpose button is created inside the first click handler, where `r` and `c` are known. Its click handler is a small anonymous function that calls `computeTranspose(r, c)` . The anonymous function remembers `r` and `c` because JavaScript functions keep the variables of the scope they were created in. No global state is needed.

Reading the grid and transposing

`computeTranspose` reads every box into `a[i][j]` , then builds `t` with `t[j][i] = a[i][j]` . If `a` is $r \times c$ then `t` is $c \times r$. It renders `t` as a plain table with borders so the result is visibly a matrix and not a line of numbers.

Reading the grid back in order

The boxes are visited row by row in `computeTranspose` using the same `i` and `j` loops that created them, so the array `a` has the same shape as the grid on screen. Trace for the 2×3

example: `a0_0` to `a0_2` fill `a[0]` , `a1_0` to `a1_2` fill `a[1]` . Then `t[0] = [a[0][0], a[1][0]] = [1, 4]` , `t[1] = [2, 5]` , `t[2] = [3, 6]` . Three rows of two, which is what the result table shows.

Validation at both stages

Stage one accepts only whole numbers from 1 to 10. The upper bound is a design choice: 100 boxes still fit a screen, 10,000 do not. A regular expression `^\d+$` rejects blanks, decimals, signs, and letters in one test. Stage two rejects an empty box or a non-numeric value and names the exact cell in the error message so the user can fix it. Errors are written into a `p` element in the page, which is easier to test than an alert.

Web Page

■ Write in lab record

transpose.html

```
1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <title>Matrix Transpose</title>
6     <style>
7       body { font-family: sans-serif; padding: 16px; max-width: 640px; }
8       label { display: inline-block; width: 90px; }
9       input[type="number"] { width: 70px; padding: 4px; margin: 4px 0; }
10      button { padding: 6px 14px; margin-top: 8px; }
11      .grid { border-collapse: collapse; margin-top: 8px; }
12      .grid td { padding: 2px; }
13      .grid input { width: 50px; }
14      .error { color: #b00020; margin-top: 8px; }
15      #result td { border: 1px solid #333; padding: 4px 10px; text-align: ri
16      h3 { margin: 12px 0 4px; }
17    </style>
18  </head>
19  <body>
20    <h2>Matrix Transpose</h2>
21
22    <div>
23      <label for="rows">Rows:</label>
24      <input id="rows" type="number" min="1" max="10" />
25    </div>
26    <div>
27      <label for="cols">Columns:</label>
28      <input id="cols" type="number" min="1" max="10" />
29    </div>
30    <button id="inputBtn">Input Elements</button>
31    <p id="dimError" class="error"></p>
32
33    <div id="elements"></div>
34    <p id="elemError" class="error"></p>
35    <div id="output"></div>
36
37    <script>
38      var rowsBox = document.getElementById("rows");
39      var colsBox = document.getElementById("cols");
40      var elements = document.getElementById("elements");
41      var output = document.getElementById("output");
42      var dimError = document.getElementById("dimError");
43      var elemError = document.getElementById("elemError");
```

```
44
45 // Returns the integer value of a text box, or null if it is not an
46 // integer between 1 and 10 (the limit keeps the grid readable).
47 function readDim(box) {
48     var s = box.value.trim();
49     if (!/^\d+$/.test(s)) return null;
50     var n = parseInt(s, 10);
51     return n ≥ 1 && n ≤ 10 ? n : null;
52 }
53
54 // Step 1: build an r x c grid of text boxes and the second button.
55 document.getElementById("inputBtn").addEventListener("click", function
56     dimError.textContent = "";
57     elemError.textContent = "";
58     elements.innerHTML = "";
59     output.innerHTML = "";
60
61     var r = readDim(rowsBox);
62     var c = readDim(colsBox);
63     if (r === null || c === null) {
64         dimError.textContent = "Rows and columns must be whole numbers fro
65         return;
66     }
67
68     var table = document.createElement("table");
69     table.className = "grid";
70     for (var i = 0; i < r; i++) {
71         var tr = document.createElement("tr");
72         for (var j = 0; j < c; j++) {
73             var td = document.createElement("td");
74             var box = document.createElement("input");
75             box.type = "number";
76             box.id = "a" + i + "_" + j;
77             box.title = "Element (" + (i + 1) + ", " + (j + 1) + ")";
78             td.appendChild(box);
79             tr.appendChild(td);
80         }
81         table.appendChild(tr);
82     }
83     var title = document.createElement("h3");
84     title.textContent = "Enter " + r + " x " + c + " elements";
85     elements.appendChild(title);
86     elements.appendChild(table);
87
```

```
88     var btn = document.createElement("button");
89     btn.id = "transposeBtn";
90     btn.textContent = "Compute Transpose";
91     btn.addEventListener("click", function () { computeTranspose(r, c);
92     elements.appendChild(btn);
93 });
94
95 // Step 2: read the grid, swap rows and columns, display the result.
96 function computeTranspose(r, c) {
97     elemError.textContent = "";
98     output.innerHTML = "";
99
100     var a = [];
101
102     for (var i = 0; i < r; i++) {
103
104         a.push([]);
105
106         for (var j = 0; j < c; j++) {
107
108             var v = document.getElementById("a" + i + "_" + j).value.trim();
109
110             if (v === "" || isNaN(Number(v))) {
111
112                 elemError.textContent = "Element (" + (i + 1) + ", " + (j + 1)
113
114                 return;
115
116             }
117
118             a[i].push(Number(v));
119
120         }
121     }
122
123     // t[j][i] = a[i][j]; the transpose is c rows by r columns.
124
125     var t = [];
126
127     for (var j2 = 0; j2 < c; j2++) {
```

```
11
6      t.push([]);
11
7      for (var i2 = 0; i2 < r; i2++) t[j2].push(a[i2][j2]);
11
8  }
11
9
12
0      var title = document.createElement("h3");
12
1      title.textContent = "Transpose (" + c + " x " + r + ")";
12
2      output.appendChild(title);
12
3      var table = document.createElement("table");
12
4      table.id = "result";
12
5      for (var p = 0; p < c; p++) {
12
6          var tr = document.createElement("tr");
12
7          for (var q = 0; q < r; q++) {
12
8              var td = document.createElement("td");
12
9              td.textContent = t[p][q];
13
0              tr.appendChild(td);
13
1          }
13
2          table.appendChild(tr);
13
3      }
13
4      output.appendChild(table);
13
5  }
13
6  </script>
```

```
13  
7   </body>  
13  
8   </html>
```

Run

Save the file as `transpose.html` and open it in any browser with File then Open, or double-click it. No server is needed. To see the script errors while testing, open the browser console with F12.

Expected Output

Test input: a 2×3 matrix with elements 1 2 3 in the first row and 4 5 6 in the second. The three screen states are:

```

1 State 1: page loaded
2 +-----+
3 | Matrix Transpose |
4 | Rows:    [    ] |
5 | Columns: [    ] |
6 | [Input Elements] |
7 +-----+
8
9 State 2: after typing 2 and 3 and clicking Input Elements
10 +-----+
11 | Matrix Transpose |
12 | Rows:    [ 2  ] |
13 | Columns: [ 3  ] |
14 | [Input Elements] |
15 | Enter 2 x 3 elements |
16 |   [ 1 ] [ 2 ] [ 3 ] |
17 |   [ 4 ] [ 5 ] [ 6 ] |
18 | [Compute Transpose] |
19 +-----+
20
21 State 3: after clicking Compute Transpose
22 +-----+
23 | ... (everything from state 2) ... |
24 | Transpose (3 x 2) |
25 |   +---+---+ |
26 |   | 1 | 4 | |
27 |   | 2 | 5 | |
28 |   | 3 | 6 | |
29 |   +---+---+ |
30 +-----+

```

Error states:

```

1 Rows = 0, click Input Elements:
2   "Rows and columns must be whole numbers from 1 to 10." (no grid appears)
3
4 Rows = 2, Columns = 3, box (1,2) left blank, click Compute Transpose:
5   "Element (1,2) must be a number." (no result tabl
6
7 Rows = 11, click Input Elements:
8   "Rows and columns must be whole numbers from 1 to 10." (no grid appears)

```

Clicking Input Elements again with new dimensions clears the old grid, the old result, and both error messages before building the new grid.

Viva Questions

✗ Do not copy. Read for understanding and the viva

- **Q:** Why is there no `form` element? **A:** Nothing is sent to a server. A form would reload the page on submit and lose the generated boxes.
- **Q:** How does the Compute Transpose handler know the dimensions? **A:** It is created inside the Input Elements handler and closes over `r` and `c`.
- **Q:** Why give each box an id like `a1_2`? **A:** So `computeTranspose` can locate any cell with `getElementById` using the row and column numbers.
- **Q:** What are the dimensions of the transpose of an $r \times c$ matrix? **A:** $c \times r$.
- **Q:** Why limit rows and columns to 10? **A:** Larger grids do not fit on screen; the limit is a usability decision and is stated in the error message.
- **Q:** Why does `readDim` use a regular expression instead of `parseInt` alone? **A:** `parseInt("2.5")` returns 2 silently; the regular expression rejects anything that is not a plain whole number.
- **Q:** Why clear `elements.innerHTML` before building a new grid? **A:** Otherwise a second click appends a second grid below the first.
- **Q:** Where does the error text appear and why not `alert`? **A:** In a `p` element with class `error`; page text can be checked by a test and does not block the browser.

Common Mistakes

✗ Do not copy. Read for understanding and the viva

- Using a `form` with a submit button, so the page reloads and the generated grid vanishes.
- Building the grid as one long row of boxes; the user cannot tell where a row ends. Use a table.
- Reading dimensions with `parseInt` alone, which accepts `2.5` and `3abc`.
- Keeping `r` and `c` in globals set by the first handler; if the user changes the dimension boxes without clicking Input Elements again, the second handler reads the wrong grid.
- Displaying the transpose with `alert` or `console.log` instead of in the page.

- Labelling the buttons differently from the manual. The examiner looks for Input Elements and Compute Transpose.

Session Summary

■ Write in lab record

- The complete `transpose.html` listing
- A drawing of the three screen states for a 2×3 example, with the transpose result
- The error messages and when each one appears
- A note of the browser used to run the page

[✎ Edit this page](#)

< Previous
Session 8

Next >
Session 10