

Arrays and Loops

Task 1 – Dynamic Scoreboard

Requirement:

1. Create an array called scores to store player scores (numbers).
Example Array : scores = [10, 15, 8]
2. Display all scores on the page using the DOM.
3. While displaying, assign **player labels dynamically** based on the index of the array
(e.g., index 0 → Player 1, index 1 → Player 2).
4. For each player, create two buttons: "Increase" and "Decrease".
5. Clicking "Increase" should add 1 to the player's score and update the array.
6. Clicking "Decrease" should subtract 1 from the player's score and update the array.

Example : UI Shows

Player 1: 10 [+][-]

Player 2: 15 [+][-]

Player 3: 8 [+][-]

Highest Score: 15

Lowest Score: 8

7. After each change, re-render the scores, so the UI always reflects the array.
8. Display the highest and lowest score below the scoreboard

Task 2 – Multi-Filter Product List

Requirements:

1. Create an array called products where each product is an object with properties:

- a. name (string)
- b. price (number)
- c. category (string)

Eg : let products = [

```
{ name: "Laptop", price: 50000, category: "Electronics" },
{ name: "Shirt", price: 1200, category: "Clothing" },
]
```

2. Display all products in a table or cards on the page using the DOM.
3. Create a dropdown filter for category and two input fields for minimum and maximum price.
4. Use loops to filter products based on the selected category and price range.

(Eg : **Filter selected:**

- Category: Electronics
- Min price: 10000
- Max price: 30000

Filtered Products : Phone – 20000)

5. Display only the filtered products dynamically in the UI.
6. Show a message "No products found" if the filter returns an empty array

Task 3 – Editable Table

Requirements:

1. Create an array called users where each element is an object with:
 - a. name (string)
 - b. age (number)
 - c. email (string)

(eg: users = [

```
{ name: "Ali", age: 22, email: "ali@mail.com" },
{ name: "Sara", age: 25, email: "sara@mail.com" }
])
```

2. Render the user data as a table on the page using the DOM.

3. Each row should have an "Edit" button.

4. The table should show like this:

Ali | 22 | ali@mail.com | Edit

Sara | 25 | sara@mail.com | Edit

5. Clicking "Edit" should allow the user to modify name or age inline.

6. After editing, update both the array and the table UI dynamically.

7. Add the "Save" button to finalize changes for each row.

Task 4 – Duplicate Highlighter

Requirements:

1. Display the Array Items

- a. Use DOM manipulation to display **all items** from the items array on the web page.
- b. Each item should be shown as a **list element** (for example, inside
 and - tags).

2. Find Duplicate Values

- a. Use **loops** to compare the items in the array.
- b. Identify which values appear **more than once** in the array (duplicate values).

3. Highlight Duplicate Items

- a. When displaying the list:
 - i. Items that appear **more than once** should be visually highlighted.
 - ii. For example, change their **background color**, **text color**, or add a **border**.

4. Items that appear **only once** should remain normal.

5. Expected Behavior

- a. "apple" appears twice → both "apple" list items should be highlighted.

Task 5 – Manual Pagination

Requirement:

1. Create an array called allItems with at least 20 string or number items.
(`allItems = [1,2,3,...,20]`)
2. Display only 5 items at a time on the page using the DOM.
3. Add "Next" and "Previous" buttons for navigation.
4. Clicking "Next" should show the next 5 items, clicking "Previous" should show the previous 5 items.
 - Page 1 shows : 1 2 3 4 5
 - Click Next : => Page 2
 - Page 2 shows : 6 7 8 9 10
5. Update the DOM each time the page changes using loops.
6. **Buttons disabled:**
 - Previous disabled on page 1
 - Next disabled on last page

Task 6 – Search + Highlight

Requirements:

1. Create an array called texts with multiple string values.
`texts = ["JavaScript", "Java", "Python", "React"]`
2. Display all text values in a list on the page using the DOM.
3. Add an input field for the user to type a search term.
4. Use a loop to check which text values contain the search term.
5. Display only the matching results dynamically.
 - **Search input:** ja
 - **UI result:**
 - JavaScript
 - Java
6. Update the display as the user types (live search).

7. If no match is found, show "No results found."

Task 7 – Dynamic Form Builder

Requirement:

1. Create an array describing form fields:

a) Input type

b) Label text

Ex: fields = [

{ type: "text", label: "Name" },

{ type: "email", label: "Email" }

]

a) Use a **loop** to go through the fields array.

b) For each object:

a. Create a **label**

b. Create an **input field** based on the type

c) Display the generated form on the page using the **DOM**.

2. On Form Submission

d) Read all the values entered by the user.

e) Store those values in a **new array** (for example: formData).

f) Show or log the stored data

Task 8 – Attendance Tracker

Requirement:

1. Create an array of students:

a) Name

b) Attendance status (true/false)

students = [

{ name: "Ali", present: true },

{ name: "Sara", present: false }]

2. Display each student with their status.

- Ali - Present
- Sara - Absent

Present: 1 Absent: 1

3. Toggle Attendance on Click

- Add Toggle Button for Each Students .When you click the button:
 - i. Change their present value from true to false or from false to true.
 - ii. Update:
 - Student status
 - Total present count
 - Total absent count

Task 9 – Inventory Manager

Requirement:

1. Create an array called inventory with objects containing:

- a. product (string)
- b. quantity (number)

Ex :

```
inventory = [  
  { product: "Pen", quantity: 10 },  
  { product: "Notebook", quantity: 5 },  
  { product: "Eraser", quantity: 0 }  
];
```

2. Display each product and quantity in a table or list.

3. Add "Increase" and "Decrease" buttons for each product.

4. Clicking buttons should update the array and the UI dynamically.

5. Prevent quantity from going below zero.

Task 10 – Live Character Counter

Requirement:

1. Create an empty array called inputs to store text input values.
(`inputs = []`).
2. Add a textarea input on the page Using DOM.
 - **User types:** "Hello"
 - Display the live character count as the user types.
 - Characters: 5
 - Longest Input: Hello
3. Store each submitted input in the inputs array.
On Submit : `inputs = ["Hello"]`

Task 11 – Sorting Visualizer

Requirement:

1. Create an array called numbers with random numbers.
2. Display the numbers visually (e.g., as bars or a list).
3. Implement a sorting algorithm (e.g., bubble sort) using loops.
4. Show each step of the sorting visually in the DOM.

For Example : `numbers = [5, 2, 4]`

- Sorting steps shown visually:

5 2 4

2 5 4

2 4 5

5. Update the display after every swap in the sorting process.

Task 12 – Quiz Engine

Requirement:

1. Create an array called quiz where each object has:
 - a. question (string)

- b. options (array of strings)
- c. answer (string)

```
Ex: quiz = [
{
    question: "What is JavaScript?",
    options: ["Language", "Framework", "Database"],
    answer: "Language"
},
{
    question: "Which keyword declares a variable?",
    options: ["var", "loop", "print"],
    answer: "var"
};
];
```

2. Display one question at a time with clickable options.
3. Check if the selected option is correct.
4. Track the user's score in a variable.
5. Display the next question after answering.
6. Show the final score at the end of the quiz.

Task 13 – Comment System

Requirement:

1. Create an empty array called comments.
2. Add an input field and "Submit" button for users to enter comments On the page using the DOM.
3. If the Comments Already exists Display the latest 5 comments on the page (newest first) Otherwise Display “No Comments In the Array”.
4. Store new comments in the array.
5. Update the displayed comments dynamically after submission.

Task 14 – Price Calculator

Requirement:

1. Create an array called prices with numeric values.
Ex: `prices = [100, 200, 300];`
2. Calculate the total price using a loop.
3. Apply tax and discount rules (e.g., 10% tax, 5% discount) using calculations.
4. Calculation :
 - Total = 600
 - Tax (10%) = 60 => 600 + 60 = 660
 - Discount (5%) = 30 => 660 - 30 = 630
 - Final = 630
5. Display each individual price and the total price in the DOM.

Task 15 – Password Rule Checker

Requirement:

1. Add an input field for password entry.
2. Create an array called rules with objects:
 - a. rule (description string)
 - b. isPassed (boolean)

Ex:

```
rules = [  
  { rule: "Minimum 8 characters", isPassed: false },  
  { rule: "At least one uppercase letter", isPassed: false },  
  { rule: "At least one number", isPassed: false },  
  { rule: "At least one special character", isPassed: false }  
];
```

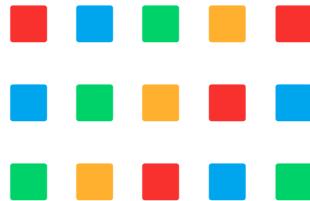
3. Check password against multiple rules (e.g., minimum length, uppercase, number, special character).
4. Store failed rules in an array.
5. Display which rules passed or failed dynamically as the user types.

Task 16 – Color Pattern Generator

Requirement:

1. Create an array called colors with multiple color values.
Ex: colors = ["red", "blue", "green", "yellow"];
2. Create a grid layout (e.g., 3x3) using the DOM.
3. Fill the grid using a repeating pattern of colors from the array.
4. Use nested loops to generate rows and columns.
5. Display the colored grid dynamically on the Page Using DOM.

Ex:



Task 17 – Expense Tracker

Requirement:

1. Create an array called expenses with objects containing:
 - a. category (string)
 - b. amount (number)

Ex:

```
expenses = [  
  { category: "Food", amount: 200 },  
  { category: "Travel", amount: 500 },
```

```
{ category: "Food", amount: 300 }]
```

2. Display all expenses in a table or list Using DOM.
3. Use loops to calculate the total expense per category.
4. Display the total for each category in the DOM like this :
 - Food: ₹500
 - Travel: ₹500
 - Total: ₹1000
5. Add a grand total of all expenses.

Task 18 – Drag Order Simulator

Requirements:

1. Create an array called items with string values representing order.
Ex: items = ["Item A", "Item B", "Item C", "Item D"];
2. Display the items on the page using the DOM as a list.
3. Add "Move Up" and "Move Down" buttons for each item.
4. Clicking buttons should update the array and reorder items dynamically.
5. Prevent moving the first item up or last item down.

Task 19 – Dashboard Summary

Requirements:

1. Create arrays:
 - a. users (number of users or objects)
 - b. orders (array of order amounts)
 - c. revenue (array of revenue numbers)

Ex :

```
users = [1,2,3,4]
orders = [500, 1200, 800]
revenue = [5000, 3000]
```

2. Calculate totals users,orders,revenue using loops
3. Display the totals in separate summary cards on the page.
 - a. Total Users: 4
 - b. Total Orders: 3
 - c. Total Revenue: ₹8000
4. Update summary dynamically if any array changes.