Step 1: Define the Purpose of the Web App

Let's create a **Task Manager** web app. This app will allow users to:

- Add tasks.
- Mark tasks as completed.
- Delete tasks.
- Filter tasks (e.g., show only completed or pending tasks).
- Save tasks to local storage for persistence.

This app will cover key JavaScript functionalities like DOM manipulation, event listeners, arrays, objects, and local storage.

Step 2: Set Up the Project Structure

Create the following files in your project folder:

1. `index.html` - The HTML structure.
2. `style.css` - Styling for the app.
3. `script.js` - JavaScript logic.

Your folder structure should look like this:

```
/task-manager
  ├── index.html
  ├── style.css
  └── script.js
```

Step 3: Write the HTML

In `index.html`, create the basic structure of the app:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Task Manager</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
```

```
  <div class="container">
        <h1>Task Manager</h1>
        <div class="input-section">
        <input type="text" id="taskInput" placeholder="Enter a task">
        <button id="addTaskBtn">Add Task</button>
        </div>
        <div class="filters">
        <button id="allBtn">All</button>
        <button id="pendingBtn">Pending</button>
        <button id="completedBtn">Completed</button>
        </div>
        <ul id="taskList"></ul>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

Step 4: Style the App

In `style.css`, add some basic styling:

```
body {
  font-family: Arial, sans-serif;
  background-color: #f4f4f9;
  margin: 0;
  padding: 0;
}

.container {
  max-width: 600px;
  margin: 50px auto;
  padding: 20px;
  background: #fff;
  border-radius: 8px;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
}

h1 {
  text-align: center;
  color: #333;
}

.input-section {
```

```
  display: flex;
  justify-content: space-between;
  margin-bottom: 20px;
}

#taskInput {
  width: 70%;
  padding: 10px;
  border: 1px solid #ccc;
  border-radius: 4px;
}

button {
  padding: 10px 15px;
  border: none;
  border-radius: 4px;
  background-color: #007bff;
  color: white;
  cursor: pointer;
}

button:hover {
  background-color: #0056b3;
}

.filters {
  display: flex;
  justify-content: space-around;
  margin-bottom: 20px;
}

ul {
  list-style-type: none;
  padding: 0;
}

li {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 10px;
  margin-bottom: 10px;
  background: #f9f9f9;
  border-radius: 4px;
```

```
}

.completed {
  text-decoration: line-through;
  color: #888;
}
```

Step 5: Add JavaScript Functionality

In `script.js`, implement the logic for the app:

1. Initialize Variables

```
const taskInput = document.getElementById('taskInput');
const addTaskBtn = document.getElementById('addTaskBtn');
const taskList = document.getElementById('taskList');
const allBtn = document.getElementById('allBtn');
const pendingBtn = document.getElementById('pendingBtn');
const completedBtn = document.getElementById('completedBtn');

let tasks = JSON.parse(localStorage.getItem('tasks')) || [];
```

2. Render Tasks

```
function renderTasks(filter = 'all') {
  taskList.innerHTML = '';
  const filteredTasks = tasks.filter(task => {
        if (filter === 'all') return true;
        if (filter === 'pending') return !task.completed;
        if (filter === 'completed') return task.completed;
  });

  filteredTasks.forEach((task, index) => {
        const li = document.createElement('li');
        li.textContent = task.text;
        if (task.completed) li.classList.add('completed');

        const deleteBtn = document.createElement('button');
        deleteBtn.textContent = 'Delete';
        deleteBtn.onclick = () => deleteTask(index);

        const completeBtn = document.createElement('button');
```

```
        completeBtn.textContent = task.completed ? 'Undo' : 'Complete';
        completeBtn.onclick = () => toggleComplete(index);

        li.appendChild(deleteBtn);
        li.appendChild(completeBtn);
        taskList.appendChild(li);
  });
}
```

3. Add a Task

```
addTaskBtn.addEventListener('click', () => {
  const taskText = taskInput.value.trim();
  if (taskText) {
        tasks.push({ text: taskText, completed: false });
        taskInput.value = '';
        saveTasks();
        renderTasks();
  }
});
```

4. Toggle Task Completion

```
function toggleComplete(index) {
  tasks[index].completed = !tasks[index].completed;
  saveTasks();
  renderTasks();
}
```

5. Delete a Task

```
function deleteTask(index) {
  tasks.splice(index, 1);
  saveTasks();
  renderTasks();
}
```

6. Save Tasks to Local Storage

```
function saveTasks() {
  localStorage.setItem('tasks', JSON.stringify(tasks));
```

```
}
```

7. Filter Tasks

```
allBtn.addEventListener('click', () => renderTasks('all'));
pendingBtn.addEventListener('click', () => renderTasks('pending'));
completedBtn.addEventListener('click', () => renderTasks('completed'));
```

8. Initial Render

```
renderTasks();
```

Step 6: Test the App

1. Open `index.html` in your browser.
2. Add tasks using the input field and "Add Task" button.
3. Mark tasks as completed or undo completion.
4. Delete tasks.
5. Use the filter buttons to view all, pending, or completed tasks.
6. Refresh the page to ensure tasks persist using local storage.

Key JavaScript Functionalities Demonstrated

1. **DOM Manipulation** : Adding, removing, and updating elements dynamically.
2. **Event Handling** : Responding to button clicks and input changes.
3. **Arrays and Objects** : Managing tasks as an array of objects.
4. **Local Storage** : Persisting data across page reloads.
5. **Conditional Rendering** : Filtering tasks based on user selection.