JavaScript ES6 Practice Projects

Welcome to your comprehensive JavaScript practice guide! This document contains 6 progressive projects designed to help you master ES6 features and modern JavaScript development patterns.

What You'll Learn

- **Array Methods**: `filter()`, `map()`, `reduce()`
- **Modern Syntax**: Rest/Spread operators, destructuring, template literals
- **Async Programming**: `async/await`, fetch API, error handling
- **Browser Storage**: localStorage, sessionStorage, cookies
- **Immutability**: Working without mutating original data
- **Real-world Applications**: Task management, shopping carts, weather apps

---

Utility Functions (Use These Across Projects)

Before starting the projects, familiarize yourself with these helper functions:

```
// ID generator (uses crypto.randomUUID if available)
const genId = () => (typeof crypto !== 'undefined' && crypto.randomUUID)
  ? crypto.randomUUID()
  : `id_${Date.now()}_${Math.floor(Math.random()*1e6)}`;

// Date utilities
const nowISO = () => new Date().toISOString();
const formatDateReadable = (iso = nowISO()) =>
  new Date(iso).toLocaleString();

// Deep clone utility
const deepClone = obj => JSON.parse(JSON.stringify(obj));

// localStorage wrapper
const storage = {
  get(key) {
    try { return JSON.parse(localStorage.getItem(key) || 'null'); }
    catch(e) { return null; }
  },
```

```
  set(key, value) {
    localStorage.setItem(key, JSON.stringify(value));
  },
  remove(key) { localStorage.removeItem(key); }
};

// sessionStorage wrapper
const session = {
  get(key) {
    try { return JSON.parse(sessionStorage.getItem(key) || 'null'); }
    catch(e) { return null; }
  },
  set(key, value) { sessionStorage.setItem(key, JSON.stringify(value)); }
  remove(key) { sessionStorage.removeItem(key); }
};

// Cookie utilities
const setCookie = (name, value, days=7) => {
  const expires = new Date(Date.now() + days*864e5).toUTCString();
  document.cookie = `${name}=${encodeURIComponent(value)}; expires=${expi
};

const getCookie = name => document.cookie.split('; ').reduce((r, v) => {
  const parts = v.split('=');
  return parts[0] === name ? decodeURIComponent(parts[1]) : r;
}, '');

const deleteCookie = name => setCookie(name, '', -1);

// Fetch with timeout
const fetchWithTimeout = async (url, options = {}, ms = 5000) => {
  const controller = new AbortController();
  const id = setTimeout(() => controller.abort(), ms);
  try {
    const res = await fetch(url, { ...options, signal: controller.signal
    clearTimeout(id);
    if (!res.ok) throw new Error(`HTTP ${res.status}`);
    return res.json();
  } catch (err) {
    clearTimeout(id);
    throw err;
  }
};
```

Project 1: Task Manager App

Difficulty: Moderate | **Focus**: Array methods, rest/spread operators, reduce()

Project Overview

Create a simple task manager to organize personal and work tasks. This project will help you practice filtering, mapping, and data aggregation.

Learning Objectives

- Use `filter()` and `map()` for data transformation
- Apply rest parameters and spread operator
- Implement `reduce()` for data aggregation

Data Structure

```
// Task Schema
{
  id: "string",             // genId()
  title: "string",
  status: "Pending"|"Completed"|"InProgress",
  tags: ["string", ...],
  createdAt: "ISO timestamp"
}
```

Seed Data

```
const TASKS_SEED = [
  { id: "t1", title: "Learn ES6", status: "Pending", tags: ["JS","Study"]
  { id: "t2", title: "Build Portfolio", status: "Completed", tags: ["Proj
  { id: "t3", title: "Buy Groceries", status: "Pending", tags: ["Personal
];
```

Requirements

1. Filter Pending Tasks

```
const getPendingTasks = tasks => {
  // TODO: Filter pending tasks and return only their titles
  // Expected output: ["Learn ES6", "Buy Groceries"]
};
```

2. Add Multiple Tasks

```
const addTask = (tasks, ...newTasks) => {
  // TODO: Add multiple tasks without mutating original array
  // Validate: title exists, set id with genId(), set createdAt if missir
  // Return: new array with all tasks
};
```

3. Generate Tag Summary

```
const tagSummary = tasks => {
  // TODO: Count occurrences of each tag across all tasks
  // Expected output: { JS:1, Study:1, Project:1, Web:1, Personal:1 }
};
```

Test Cases

```
// Test your functions with these:
console.log(getPendingTasks(TASKS_SEED)); // ["Learn ES6", "Buy Groceries
console.log(tagSummary(TASKS_SEED)); // { JS:1, Study:1, Project:1, Web:1

const newTasks = [
  { title: "Call Mom", tags: ["Personal"] },
  { title: "Debug App", tags: ["JS", "Project"] }
];
console.log(addTask(TASKS_SEED, ...newTasks)); // Should return array wit
```

Edge Cases to Handle

- Duplicate tags across tasks (count cumulatively)
- Invalid status values
- Missing title in new tasks

Project 2: Shopping Cart System

Difficulty: Moderate | **Focus**: Immutability, map(), template literals

Project Overview

Build a shopping cart system for an e-commerce site. Focus on immutable operations and formatted output.

Learning Objectives

- Practice immutability with spread operator
- Transform data using `map()`
- Create formatted output with template literals

Data Structure

```
// Cart Item Schema
{
  id: "string",
  product: "string",
  price: number,
  qty: number
}
```

Seed Data

```
const CART_SEED = [
  { id: "c1", product: "Book", price: 200, qty: 2 },
  { id: "c2", product: "Pen", price: 10, qty: 5 }
];
```

Requirements

1. Add Item to Cart

```
const addItem = (cart, newItem) => {
  // TODO: Add item without mutating original cart
  // Generate ID if not provided
  // If product already exists, consider incrementing quantity
  // Return: new cart array
};
```

2. Update Item Quantity

```
const updateQty = (cart, id, qty) => {
  // TODO: Update quantity of specific item
  // If qty is 0 or negative, remove item
  // Return: new cart array
};
```

3. Cart Summary

```
const cartSummary = cart => {
  // TODO: Generate formatted summary using template literals
  // Format: "Product (xQty) → Total"
  // Include grand total at bottom
};
```

Expected Output

```
Cart Summary:
Book (x2) → 400
```

```
Pen (x5) → 50
Total = 450
```

Test Cases

```
// Test your functions:
const newCart = addItem(CART_SEED, { product: "Notebook", price: 50, qty:
console.log(newCart.length); // Should be 3

const updatedCart = updateQty(CART_SEED, "c1", 3);
console.log(updatedCart.find(item => item.id === "c1").qty); // Should be

console.log(cartSummary(CART_SEED));
```

Edge Cases to Handle

- Negative quantity (remove item)
- Non-numeric price (validation)
- Duplicate products (decide behavior)

Project 3: Notes Application with localStorage

Difficulty: Moderate | **Focus**: localStorage, JSON handling, CRUD operations

Project Overview

Create a notes application where users can write, save, and manage their notes with persistent storage.

Learning Objectives

- Work with localStorage for data persistence
- Handle JSON parsing/stringifying safely
- Implement CRUD operations (Create, Read, Delete)

Data Structure

```
// Note Schema
{
  id: "string",
  title: "string",
  content: "string",
  createdAt: "ISO timestamp"
}
```

Seed Data

```
const NOTES_SEED = [
  {
    id: "n1",
    title: "Shopping list",
    content: "Milk, Eggs, Bread",
    createdAt: "2025-09-05T09:00:00Z"
  }
];

// Storage key
const NOTES_KEY = 'app_notes_v1';
```

Requirements

1. Add Note

```
const addNote = (title, content) => {
  // TODO: Create new note with unique ID and timestamp
  // Merge with existing notes in localStorage (don't overwrite)
  // Validate: title must not be empty
};
```

2. Get All Notes

```
const getNotes = () => {
  // TODO: Retrieve all notes from localStorage
  // Return empty array if no notes exist
  // Handle JSON parsing errors gracefully
};
```

3. Delete Note

```
const deleteNote = (id) => {
  // TODO: Remove note by ID from localStorage
  // Update the stored array without the deleted note
};
```

4. Initialize Notes (Optional)

```
const initializeNotes = () => {
  // TODO: Load seed data if no notes exist in localStorage
};
```

Test Cases

```
// Test your functions:
addNote("Meeting Notes", "Discuss project timeline and deliverables");
console.log(getNotes().length); // Should increase by 1

const notes = getNotes();
const firstNoteId = notes[0].id;
deleteNote(firstNoteId);
console.log(getNotes().length); // Should decrease by 1
```

Edge Cases to Handle

- Empty title (validation error)
- localStorage quota exceeded
- Invalid JSON in localStorage
- Large content (consider length limits)

Project 4: Weather Dashboard

Difficulty: Hard | **Focus**: Async/await, API handling, sessionStorage caching

Project Overview

Build a weather dashboard that fetches live weather data with caching to reduce API calls.

Learning Objectives

- Master async/await with proper error handling
- Practice destructuring complex objects
- Implement caching strategy with sessionStorage
- Create multiline formatted output

Expected API Response

```
// Mock API response structure
{
  "location": "Bengaluru",
  "data": {
    "temperature": 30,
    "humidity": 70,
    "description": "Cloudy"
  },
  "timestamp": "2025-09-06T10:00:00Z"
}
```

Storage Configuration

```
const WEATHER_CACHE = 'weather_bengaluru_v1';
const CACHE_DURATION = 10 * 60 * 1000; // 10 minutes in milliseconds
```

Requirements

1. Fetch Weather Data

```
const fetchWeather = async (city) => {
  try {
    // TODO: Use fetchWithTimeout to get weather data
    // Destructure the response to extract temperature, humidity, descrip
    // Handle network errors with try/catch
    // Return formatted weather object
  } catch (error) {
    // TODO: Return error object or throw with meaningful message
  }
};
```

2. Cache Management

```
const getCachedWeather = (city) => {
  // TODO: Check sessionStorage for cached weather data
  // Validate cache age (not older than CACHE_DURATION)
  // Return cached data if valid, null if stale or missing
};

const setCachedWeather = (city, weatherData) => {
  // TODO: Store weather data in sessionStorage with timestamp
};
```

3. Weather Display

```
const displayWeather = (weatherData) => {
  // TODO: Format weather data using template literals
  // Create multiline display with temperature, humidity, condition
};
```

4. Main Weather Function

```
const getWeather = async (city) => {
  // TODO: Check cache first, fetch if needed
  // Store fresh data in cache
```

```
    // Return formatted weather display
  };
```

Expected Output

```
Weather Report:
Temperature: 30°C
Humidity: 70%
Condition: Cloudy
Last updated: 10:30 AM
```

Test Cases

```
  // Test your functions:
  getWeather('Bengaluru').then(console.log);

  // Test caching
  setTimeout(() => {
    getWeather('Bengaluru').then(data => {
      console.log('From cache:', data);
    });
  }, 1000);
```

Edge Cases to Handle

- API server down (use cached data or show error message)
- Invalid JSON response
- Network timeout
- Invalid city name
- Cache corruption

Project 5: Expense Tracker

Difficulty: Moderate-Hard | **Focus**: Array methods combination, localStorage, data aggregation

Project Overview

Build an expense tracker to help users manage spending with categorization and reporting features.

Learning Objectives

- Combine multiple array methods (`filter` , `map` , `reduce`)
- Implement comprehensive localStorage persistence
- Create data import/export functionality
- Build reporting and aggregation features

Data Structure

```
// Expense Schema
{
  id: "string",
  category: "string",
  amount: number,
  date: "ISO timestamp",
  note?: "string" // optional
}
```

Seed Data

```
const EXPENSES_SEED = [
  { id: "e1", category: "Food", amount: 200, date: "2025-09-01T12:00:00Z"
  { id: "e2", category: "Travel", amount: 500, date: "2025-09-02T08:00:0(
  { id: "e3", category: "Food", amount: 300, date: "2025-09-03T19:00:00Z"
];

const EXPENSES_KEY = 'app_expenses_v1';
```

Requirements

1. Calculate Total Expenses

```
const totalExpenses = expenses => {
  // TODO: Use reduce() to calculate total amount
  // Handle empty array case
};
```

2. Filter by Category

```
const filterByCategory = (expenses, category) => {
  // TODO: Filter expenses by specific category
  // Case-insensitive comparison
};
```

3. Format Expense List

```
const mapToStrings = expenses => {
  // TODO: Map expenses to formatted strings "Category → amount"
  // Include date if needed
};
```

4. Storage Functions

```
const saveExpenses = expenses => {
  // TODO: Save expenses array to localStorage
};

const loadExpenses = () => {
  // TODO: Load expenses from localStorage
  // Return empty array if none exist
};

const addExpense = (category, amount, note = '') => {
  // TODO: Add new expense to existing data
  // Generate ID and timestamp
  // Save updated list to localStorage
};
```

5. Import/Export Functions

```
const exportExpenses = () => {
  // TODO: Export expenses as JSON string for download
};

const importExpenses = (jsonString) => {
  // TODO: Parse and validate JSON data
  // Merge with existing expenses (handle duplicates)
  // Save to localStorage
};
```

Test Cases

```
// Test your functions:
console.log(totalExpenses(EXPENSES_SEED)); // Should be 1000

console.log(filterByCategory(EXPENSES_SEED, "Food")); // Should return 2

console.log(mapToStrings(EXPENSES_SEED));
// Should return: ["Food → 200", "Travel → 500", "Food → 300"]

// Test persistence
saveExpenses(EXPENSES_SEED);
console.log(loadExpenses()); // Should return the saved expenses
```

Edge Cases to Handle

- Negative amounts (validation)
- Non-numeric amounts
- Invalid JSON during import
- localStorage quota exceeded
- Duplicate expense IDs during import

Project 6: Cookie Consent Banner

Difficulty: Easy-Moderate | **Focus**: Cookie management, DOM manipulation, ES6 utilities

Project Overview

Implement a cookie consent banner for a website with proper cookie management and user preferences.

Learning Objectives

- Work with browser cookies
- Practice DOM manipulation
- Implement conditional logic for user preferences
- Use arrow functions and template literals

Configuration

```
const CONSENT_COOKIE = 'site_consent';
const BANNER_ID = 'cookie-banner';
```

Requirements

1. Banner Display Logic

```
const showBanner = () => {
  // TODO: Create and display cookie banner
  // Message: "We use cookies to improve your experience. Accept?"
  // Include Accept and Decline buttons
};

const hideBanner = () => {
  // TODO: Hide/remove banner from page
};

const showBannerIfNoConsent = () => {
  // TODO: Show banner only if consent cookie doesn't exist
};
```

2. Consent Handling

```
const acceptConsent = () => {
  // TODO: Set consent cookie for 7 days
  // Hide banner
  // Optional: Enable tracking/analytics
};

const declineConsent = () => {
  // TODO: Set declined cookie for 1 day
  // Hide banner
  // Ensure no tracking cookies are set
};
```

3. Initialization

```
const initializeCookieBanner = () => {
  // TODO: Check consent status on page load
  // Show banner if needed
  // Set up event listeners
};
```

4. Utility Functions

```
const hasConsented = () => {
  // TODO: Check if user has accepted cookies
  // Return boolean
};

const resetConsent = () => {
  // TODO: Delete consent cookies (for testing)
};
```

HTML Structure (for testing)

```
<!DOCTYPE html>
<html>
<head>
    <title>Cookie Consent Test</title>
    <style>
```

```
        #cookie-banner {
            position: fixed;
            bottom: 0;
            left: 0;
            right: 0;
            background: #333;
            color: white;
            padding: 20px;
            text-align: center;
        }

        .banner-button {
            margin: 0 10px;
            padding: 10px 20px;
            border: none;
            cursor: pointer;
        }

        .accept { background: #4CAF50; color: white; }
        .decline { background: #f44336; color: white; }
    </style>
</head>
<body>
    <h1>Welcome to Our Website</h1>
    <p>This page demonstrates cookie consent functionality.</p>

    <button onclick="resetConsent()">Reset Consent (for testing)</button>

    <!-- Banner will be created dynamically -->

    <script>
        // Your cookie consent code here

        // Initialize on page load
        document.addEventListener('DOMContentLoaded', initializeCookieBar
    </script>
</body>
</html>
```

Test Cases

```
// Test your functions:
console.log(hasConsented()); // Should be false initially

// Simulate acceptance
acceptConsent();
console.log(hasConsented()); // Should be true

console.log(getCookie(CONSENT_COOKIE)); // Should return "true"

// Test banner display
resetConsent();
showBannerIfNoConsent(); // Should display banner
```

Edge Cases to Handle

- Cookies disabled in browser (fallback to sessionStorage)
- Multiple domains/subdomains (set proper path)
- Banner already exists (don't duplicate)
- Invalid cookie values

Completion Checklist

For each project, ensure you have:

- ☐ **Implemented all required functions**
- ☐ **Handled specified edge cases**
- ☐ **Tested with provided test cases**
- ☐ **Used appropriate ES6 features**
- ☐ **Followed proper error handling patterns**
- ☐ **Maintained code readability and comments**

Next Steps

After completing these projects, you'll have solid experience with:

- Modern JavaScript (ES6+) syntax and features
- Array manipulation and functional programming
- Browser storage APIs (localStorage, sessionStorage, cookies)
- Asynchronous programming with async/await

- Error handling and data validation
- Real-world application patterns

Good luck with your JavaScript journey!