Lesson 3: Data Storage & State Management

Lesson Overview

Duration: ~2 hours

Goal: Teach students how to store and manage program state using JSON and local storage in JavaScript. Introduce how data persists across sessions and how to structure stored data effectively.

Format: Explanation + Interactive Coding Exercises + Mini Project

---

1. Understanding State and Variables (20 min)

What is State?

- **State** refers to the data that an application maintains while running. This data can change based on user actions or program execution.
- **Examples of State:**
    - Current score in a game
    - User preferences (theme, language settings)
    - Shopping cart contents

Using Variables for State Management

- **Declaring and updating state variables:**

```
let score = 0;
console.log("Initial score:", score);

// Updating the state
score += 10;
console.log("Updated score:", score);
```

- **State Management Best Practices:**
  - Always initialize state variables.
  - Use meaningful variable names.
  - Keep track of state changes systematically.

---

2. Introduction to JSON (30 min)

What is JSON?

- **JSON (JavaScript Object Notation)** is a lightweight format for storing and transporting data.
- It is commonly used for saving structured data and communicating with web APIs.

Converting Data to JSON Format

```
const user = {
   name: "Alice",
   age: 25,
   score: 90
};

const jsonString = JSON.stringify(user); // Convert object to JSON string
console.log(jsonString);
```

Parsing JSON Back into an Object

```
const parsedUser = JSON.parse(jsonString); // Convert JSON string back to o
bject
console.log(parsedUser.name); // Alice
```

What JSON looks like?

```
{
  "name": "Alice",
  "age": 25,
  "score": 90,
  "address": {
    "city": "Toronto",
    "country": "Canada"
  },
  "hobbies": ["reading", "coding", "gaming"]
}
```

Why Use JSON?

- Can be easily saved in local storage.
- Used in many web applications and APIs.
- Supports complex data structures (arrays, nested objects).

---

3. Local Storage for Data Persistence (30 min)

What is Local Storage?

- **Local Storage** allows data to be stored **permanently** in the browser.
- It remains **even after the page is reloaded or the browser is closed**.

Saving Data in Local Storage

```
localStorage.setItem("playerName", "Alice");
localStorage.setItem("highScore", "150");
```

Retrieving Data from Local Storage

```
const playerName = localStorage.getItem("playerName");
const highScore = localStorage.getItem("highScore");
```

```
console.log("Player:", playerName, "High Score:", highScore);
```

Storing and Retrieving JSON Data in Local Storage

```
const playerData = { name: "Alice", score: 150 };
localStorage.setItem("playerData", JSON.stringify(playerData));

const retrievedData = JSON.parse(localStorage.getItem("playerData"));
console.log("Player:", retrievedData.name, "Score:", retrievedData.score);
```

Clearing Data from Local Storage

```
localStorage.removeItem("playerData"); // Remove a specific item
localStorage.clear(); // Clear all stored data
```

---

4. Mini Project: Mood Tracker

Estimated Time: 40-60 minutes

Skills Practiced: JavaScript, JSON, Local Storage, DOM Manipulation

Project Overview

The **Mood Tracker** allows users to select a date from a **calendar**, record their **mood** and **notes**, and store this data using **local storage**. Users can also **export** and **import** their mood data as a JSON file.

Learning Objectives

By the end of this project, students will:

- Understand **DOM manipulation** to create and interact with elements dynamically.
- Work with **event listeners** to handle user input.
- Store and retrieve data using **local storage**.

- Format and process **JSON data** for saving and loading information.
- Implement **functions** for **saving, retrieving, and exporting data**.

Starter Code (Students Complete the Missing Parts)

Part 1: Setting Up the Calendar

The calendar dynamically generates months and days based on the current date. Each day is clickable, allowing users to select it and input their mood.

📌 Task 1: Understa the `selectDate` Function

When a user clicks on a date, the function should:

1. Highlight the selected date.
2. Update `selectedDateStr` to store the clicked date.
3. Retrieve mood data from `moodData` if available and display it in the input fields.

🚀 Find this code in `mood_tracker.js`:

```
cell.addEventListener('click', function() {
    if (selectedCell) {
        selectedCell.classList.remove('selected');
    }
    selectedCell = cell;
    selectedCell.classList.add('selected');
    selectedDateStr = cell.dataset.date;
    document.getElementById('selectedDate').textContent = selectedDateStr;

     if (moodData[selectedDateStr]) {
     document.getElementById('mood').value = moodData[selectedDateStr].mood;
     document.getElementById('note').value = moodData[selectedDateStr].note;
    } else {
     document.getElementById('mood').value = "";
     document.getElementById('note').value = "";
    }
});
```

✅ **Student Task**: Understand this function to load saved mood and note data for the selected da

Part 2: Storing and Retrieving Mood Data

Users can select a date, pick a mood, and write a note. This data should be stored in an object (`moodData`) and saved in **local storage**.

📌 Task 2: Implement the `cacheSaveBtn` Function

When the **Save** button is clicked:

1. Ensure a date is selected.
2. Retrieve the mood and note values from the form.
3. Store the data in the `moodData` object using the date as a key.
4. Save `moodData` to **local storage**.

🚀 **Find this code in `mood_tracker.js`:**

```
document.getElementById('cacheSaveBtn').addEventListener('click', function
() {
    if (!selectedDateStr) {
        alert("Please select a date first!");
        return;
    }

    const mood = document.getElementById('mood').value;
    const note = document.getElementById('note').value;

    // TODO: Save the mood and note data to the moodData object and store in
local storage
});
```

✅ **Student Task**: Complete this function so that user input is stored and retrieved even after refreshing the page.

💡 **Hint**: Use `localStorage.setItem("moodData", JSON.stringify(moodData))` to store data persistently.

Part 3: Exporting Mood Data

Users should be able to download their mood records as a JSON file.

📌 Task 3: Implement the `exportJsonBtn` Function

When the **"Save to JSON"** button is clicked:

1. Convert `moodData` to a JSON string.
2. Create a downloadable JSON file.
3. Automatically trigger the file download.

🚀 **Find this code in `mood_tracker.js`:**

```
document.getElementById('exportJsonBtn').addEventListener('click', function
() {
  // TODO: Convert moodData to JSON and trigger a download
});
```

✅ **Student Task**: Complete this function to export the mood data.

💡 **Hint**: Use `encodeURIComponent(JSON.stringify(moodData, null, 4))` to create a JSON file.

Part 4: Importing Mood Data

Users should be able to upload a **previously saved JSON file** and load their mood records.

📌 Task 4: Implement the `importJsonBtn` and `fileInput` Functions

When the **"Load from JSON"** button is clicked:

1. Open a file selection dialog.
2. Read the selected JSON file.

3. Parse and store the data in `moodData`.
4. Update the calendar display.

🚀 **Find this code in `mood_tracker.js`:**

```
document.getElementById('importJsonBtn').addEventListener('click', function
() {
   // TODO: Open the file selection dialog
});

document.getElementById('fileInput').addEventListener('change', function(eve
nt) {
   // TODO: Read the JSON file and update moodData
});
```

✅ **Student Task:** Complete these functions to allow data import.

💡 **Hint:** Use `FileReader()` to read and parse JSON data.

Final Challenge: Display Mood on Calendar

Once a mood is saved, the corresponding date should display an emoji **(**😊**,** 😐**,** 😞**)** to reflect the mood.

📌 Task 5: Modify `generateCalendar` to Show Mood

1. If a date in `moodData` has a recorded mood, display it inside the calendar cell.
2. Update `generateCalendar()` to check for existing moods.

🚀 **Find this code in `mood_tracker.js`:**

```
cell.textContent = date;
// TODO: If mood data exists for this date, display the mood emoji inside the c
ell
```

✅ **Student Task:** Modify this code to show moods directly on the calendar.

💡 **Hint:** Use `cell.innerHTML = date + " " + moodData[dateStr].mood;` to show the mood icon.

Stretch Goals

If students finish early, they can try these extra challenges:

- **Add a Reset Button**: Allow users to clear all stored mood data.
- **Improve UI**: Style the calendar cells based on mood (e.g., different background colors for each mood).
- **Support Multiple Users**: Allow different users to save and retrieve their own mood records.

---

5. Recap & Q&A (10 min)

- Quick review of JSON, local storage, and persistent state.
- Discuss limitations of local storage and introduce **session storage vs. local storage**.
- Provide hints for the next lesson (working with databases and APIs).