Week 1 Coding Problems

1️⃣ Callbacks and Arrow Functions – Useful Task Manager

Scenario:
You are building a task manager for an app. When a new task is added, the system should not only update the task list but also provide a summary report. Write a function `addTask` that:

- Accepts an **array of task objects**, a **new task object**, and a **callback function**.
- Adds the new task to the list.
- **Computes a summary** that includes:
  - The **total number** of tasks.
  - The number of **completed** tasks.
  - The number of **pending** tasks.
- Calls the callback function, passing both the **updated tasks array** and the **summary object**.

Task Object Structure:

```
{ id: number, title: string, completed: boolean }
```

Starter Data:

```
let tasks = [
    { id: 1, title: 'Complete assignment', completed: false },
    { id: 2, title: 'Attend meeting', completed: true }
];
```

Example Usage:

```
addTask(tasks, { id: 3, title: 'Start project', completed: false }, (updatedTasks,
summary) => {
    console.log("Updated Task List:", updatedTasks);
    // Expected Summary: { total: 3, completed: 1, pending: 2 }
    console.log("Summary:", summary);
});
```

In this challenge, the callback function should receive both the updated list of tasks and a summary object that shows how many tasks are total, completed, and pending.

2️⃣ Arrays and Array Methods (Callbacks) – Temperature Conversion

Scenario:
You're developing a weather app that displays daily temperatures in Celsius. You are given an array of temperatures in Fahrenheit. Your task is to:

1. **Convert** the temperatures from Fahrenheit to Celsius using the `map()` method.
2. **Sort** the converted temperatures in ascending order with `sort()`.
3. **Extract** the three lowest (first three) temperatures using `slice()`.
4. **Display** the final array using `console.log()`.

Conversion Formula:
$\text{Celsius} = (\text{Fahrenheit} - 32) \times 95$

Starter Data:

```
let temperatures = [32, 45, 50, 84, 90, 78, 68];
```

Expected Outcome:
A new array containing the three lowest temperatures in Celsius (rounded as needed).

3️⃣ Strings and String Manipulation – Survey Code Generator

Scenario:
Create a unique survey code generator by combining a person's name and their date of birth. Write a function `createCode(name, dob)` that performs the following steps:

1. **Extract & Convert:** Take the first three letters of the name and convert them to uppercase.
2. **Extract:** From the date of birth (in `"YYYY-MM-DD"` format), extract the year and take its last two digits.
3. **Concatenate:** Combine these two parts to form a unique survey code.

Starter Data Example:

- **Name:** `"John"`
- **Date of Birth:** `"1995-06-15"`

Example Usage:

```
const code = createCode("John", "1995-06-15");
// Expected output: "JOH95"
```

4️⃣ Maps and Map Operations – Student Directory

Scenario:
You are creating a student directory app. Each student has a name and a grade. You must:

1. **Store** student data using a `Map` where the key is the student's name and the value is their grade.
2. **Check** if at least one student has an "A" grade.
3. **List** all the names of students who have a "B" grade.

Starter Data Generation:
For a more robust test, work with a dataset of 200 students. Use the following code snippet to generate the data:

```
// Generate a dataset of 200 students with random grades.
const grades = ['A', 'B', 'C', 'D', 'F'];
const studentData = new Map();
for (let i = 1; i <= 200; i++) {
    // Student names: "Student1", "Student2", ..., "Student200"
    const randomGrade = grades[Math.floor(Math.random() * grades.length)];
    studentData.set(`Student${i}`, randomGrade);
}
```

Expected Outcomes:

1. A boolean indicating whether at least one student received an "A".
2. An array containing the names of all students with a "B" grade.

5️⃣ JSON Handling – Earthquake Data Query

Scenario:
You are provided with earthquake data in JSON format. Your tasks are to write functions that:

1. **Identify** the country with the most earthquake records.
2. **Retrieve** the details (country, magnitude, date) of the earthquake with the highest magnitude.
3. **Calculate** the total number of earthquake records.

Data Generation:
`The Json File containing the earthquake Data has been attached along with this file.`

Expected Outcomes:

1. The **country** that appears most frequently in the dataset.
2. The details (country, magnitude, date) of the earthquake record with the **highest magnitude**.
3. The **total count** of earthquake records (which should be 200).

6️⃣ Async Programming – Simulated API Data Fetcher

Scenario:
Imagine you're developing an application that retrieves user data from an external source. Since there's no real API available, you'll simulate an asynchronous API call using a Promise with a `setTimeout`. Your tasks are as follows:

1. **Simulate Data Fetching:**
   Write an asynchronous function `fetchUserData` that returns a Promise which resolves with a predefined array of user objects after a **2-second delay**.
2. **Process the Data:**
   Write another asynchronous function `processUserData` that:
   - Awaits the data from `fetchUserData`.
   - Processes the data to compute:
     - The **total number** of users.
     - The **number of active** users (where `active` is `true`).
   - Logs a summary message to the console, e.g., `"Total users: 10, Active users: 6"`.

Tasks:

- Use `setTimeout` to simulate the delay inside `fetchUserData`.
- Implement `fetchUserData` using Promises.
- Use `async/await` syntax in `processUserData`.
- Process the data using appropriate array methods (e.g., `filter`).

Starter Data:
Simulate the following array of 10 user objects:

```
const userData = [
    { id: 1, name: "Alice", active: true },
    { id: 2, name: "Bob", active: false },
    { id: 3, name: "Charlie", active: true },
    { id: 4, name: "David", active: false },
    { id: 5, name: "Eva", active: true },
    { id: 6, name: "Frank", active: true },
    { id: 7, name: "Grace", active: false },
    { id: 8, name: "Hannah", active: true },
    { id: 9, name: "Ian", active: false },
    { id: 10, name: "Jane", active: true }
];
```

Expected Outcome:
After calling `processUserData`, the console should display a summary similar to:

```
Total users: 10, Active users: 6
```

Example Usage:

```
processUserData();
```