⏱ JavaScript Async Programming

Practice Problems - setTimeout, Callbacks & Asynchronous Operations

Total Problems: 15 | Difficulty: Beginner to Intermediate

🚀 Section 1: Basic Level

Difficulty: ⭐ Beginner | Focus: setTimeout fundamentals + basic conditions

1 Delayed Greeting

Create a function `greetUser(name)` that implements delayed greeting with validation.

Tasks:

- Wait 2 seconds before displaying the greeting
- Then print: `Hello <name>`
- If name is empty → print `Invalid Name` immediately (without timeout)

Concepts: Conditional logic + setTimeout + string interpolation

2 Countdown Timer

Write a program that prints a countdown sequence.

Output Required:

- 5 (after 1 second)
- 4 (after 2 seconds)
- 3 (after 3 seconds)
- 2 (after 4 seconds)
- 1 (after 5 seconds)
- Go! (after 6 seconds)

Rules:

- Each number should appear after 1 second interval
- Use loop + setTimeout
- Do NOT use setInterval

Concepts: Loop with setTimeout + closure + delay calculation

3 Delayed Array Processing

Process an array after a delay and calculate the sum.

```
const numbers = [2, 4, 6, 8];
```

Tasks:

- After 3 seconds, print the sum of array elements
- If array is empty → print `"No Data"`
- Use reduce method for calculation

Concepts: Array methods (reduce) + setTimeout + validation

🚀 Section 2: Intermediate Level

Difficulty: ⭐⭐ Intermediate | Focus: Complex conditions + async logic

4 Fake Login System

Create a function `login(username, password)` that simulates an authentication system.

Behavior:

- Immediately print `"Checking credentials..."`
- After 2 seconds check credentials:
  - If username = "admin" AND password = "1234" → `"Login Success"`
  - Else → `"Login Failed"`

Concepts: Conditional logic + setTimeout + logical operators (&&)

5 Filter Array with Async

```
const numbers = [10, 25, 30, 45, 60];
```

Tasks:

- Create a function that filters numbers greater than 30
- Use setTimeout to return result after 2 seconds
- Print `"No numbers found"` if result array is empty

💡 Expected output: [45, 60] (after 2 seconds)

Concepts: Array filter + setTimeout + callback/promise pattern

6 Student Pass/Fail Checker

```
const marks = [40, 55, 70, 20];
```

Tasks:

- Write a function that checks each mark
- If mark ≥ 35 → print `"Pass"`
- Else → print `"Fail"`
- Display result one by one every second using setTimeout

💡 Output format: "Student 1: Pass", "Student 2: Pass", "Student 3: Pass", "Student 4: Fail"
(Each appearing 1 second apart)

Concepts: Loop + setTimeout + conditional + delay multiplication

7 Async Fetch Simulation

Create a fake async function `getUsers()` that simulates fetching user data.

```
const users = [ {name: "Alice", age: 25}, {name: "Bob", age: 17}, {name: "Charlie", age: 30}, {name: "David", age: 16} ];
```

Tasks:

- Return array of users after 2 seconds
- Loop through users
- If age > 18 → print `"[Name] is Adult"`
- Else → print `"[Name] is Minor"`

Concepts: Async simulation + array iteration + object access + conditions

8 Shopping Cart Logic

```
const prices = [100, 250, 75, 300];
```

Tasks:

- Create async function `calculateTotal()`
- Use setTimeout to simulate processing delay (2 seconds)
- Calculate total price
- If total > 500 → print `"Discount Applied - Total: [amount]"`
- Else → print `"No Discount - Total: [amount]"`

Concepts: Array reduce + setTimeout + conditional output + calculation

9 Sequential Delays

```
const letters = ["A", "B", "C", "D"];
```

Tasks:

- Print each letter with 1-second gap
- Use function + setTimeout
- If letter is "C" → print `"C - Special Letter"`
- Otherwise → print just the letter

Concepts: Sequential setTimeout + array iteration + special case handling

🔥 Section 3: Advanced Challenges

Difficulty: ★★★ Advanced | Focus: Complex async patterns + multiple operations

10 Multi-Step Order Processing

Simulate a multi-step order processing system with different delays.

Steps:

- Step 1: Print `"Order Received"` (immediately)
- Step 2: Print `"Processing Payment..."` (after 1 second)
- Step 3: Print `"Payment Confirmed"` (after 3 seconds total)
- Step 4: Print `"Order Shipped"` (after 5 seconds total)
- Step 5: Print `"Order Delivered"` (after 7 seconds total)

Concepts: Nested setTimeout + sequential async operations + accumulating delays

11 Traffic Light Simulator

Create a traffic light sequence that runs once.

Sequence:

- `"🔴 RED"` (0 seconds - start immediately)
- `"🟡 YELLOW"` (after 3 seconds)
- `"🟢 GREEN"` (after 5 seconds total)
- `"🟡 YELLOW"` (after 10 seconds total)
- `"🔴 RED - Cycle Complete"` (after 12 seconds total)

Concepts: Sequential timing + state management + visual feedback

12 Async Data Validator

```
const formData = { email: "user@example.com", password: "pass123", age: 25 };
```

Tasks:

- Validate email (must contain @) - takes 1 second
- Validate password (must be at least 6 characters) - takes 1 second
- Validate age (must be >= 18) - takes 1 second
- Each validation should run sequentially
- If all pass → print `"Validation Success"`
- If any fails → print `"Validation Failed: [reason]"` and stop

> **Concepts:** Sequential validation + early termination + string/number checks

13 Race Condition Simulator

Simulate three async tasks with different completion times.

```
const tasks = [ {name: "Task A", duration: 3000}, {name: "Task B", duration: 1000}, {name: "Task C", duration: 2000} ];
```

Tasks:

- Start all tasks simultaneously
- Print `"[Task Name] completed"` when each finishes
- Track and print which task finishes first
- When all complete, print `"All tasks completed"`

> **Concepts:** Parallel setTimeout + completion tracking + race conditions

14 Retry Mechanism

Create a function that simulates API calls with retry logic.

Requirements:

- Create `fetchData()` that randomly succeeds or fails
- If it fails, retry up to 3 times
- Wait 2 seconds between retries
- Print attempt number each time
- If all retries fail → print `"Failed after 3 attempts"`
- If successful → print `"Success on attempt [number]"`

💡 Use Math.random() > 0.5 to simulate success/failure

Concepts: Retry logic + recursive setTimeout + random simulation + counter

15 Bank Transaction Queue

```
const transactions = [ {type: "deposit", amount: 1000}, {type: "withdraw", amount: 500},
{type: "deposit", amount: 200}, {type: "withdraw", amount: 1500} ];
```

Tasks:

- Starting balance: 1000
- Process each transaction after 1 second interval
- For deposit: add amount and print new balance
- For withdraw: check if sufficient balance
  - If yes: deduct and print new balance
  - If no: print `"Insufficient funds"` and skip
- Print final balance when all transactions complete

Concepts: State management + sequential processing + balance validation + accumulator

💎 Bonus Section: Real-World Scenarios

Difficulty: ★★★ Advanced | Focus: Practical applications

B1 Typing Animation Effect

Create a typing effect that displays text character by character.

```
const message = "Welcome to JavaScript Async Programming!";
```

Tasks:

- Display each character after 100ms delay
- When complete, print `"✓ Message displayed"`

B2 Progress Bar Simulation

Simulate a download progress bar from 0% to 100%.

Tasks:

- Start at 0%, increment by 10% every 500ms
- Print progress: `"Downloading... 10%"`, `"Downloading... 20%"`, etc.
- When reaching 100%, print `"Download Complete!"`

B3 Auto-Save System

Simulate an auto-save feature that saves data every 3 seconds, 5 times total.

Tasks:

- Print `"Auto-saving... [timestamp]"` every 3 seconds
- Save 5 times total
- After 5 saves, print `"Auto-save disabled"`

Total Practice Problems: 18

Basic (3) • Intermediate (6) • Advanced (6) • Bonus (3)

🎯 Focus Areas: setTimeout • Callbacks • Async Logic • Conditional Statements • Array Methods • State Management

Created for JavaScript Async Programming Practice | Good luck! 🚀