Asynchronous JavaScript Assignment

1. **Basic `setTimeout`:**

   - Write a function called `delayedMessage` that takes a message and a delay (in milliseconds) as arguments. Use `setTimeout` to log the message to the console after the specified delay.

2. **Basic `setInterval`:**

   - Create a function called `countdown` that takes a number as an argument. Use `setInterval` to log the countdown from that number to zero, decrementing by 1 every second. Stop the interval when it reaches zero.

3. **Clearing Intervals:**

   - Extend the `countdown` function from the previous question to also accept a callback function that gets executed when the countdown reaches zero. Ensure the interval is cleared after the countdown completes.

4. **Using Promises with `setTimeout`:**

   - Write a function called `wait` that returns a Promise. The Promise should resolve after a delay (in milliseconds) that is passed as an argument to the function. Test this function by logging a message to the console once the promise resolves.

5. **Promise-based Countdown:**

   - Refactor the `countdown` function to return a Promise that resolves when the countdown completes. Use the `wait` function from the previous question to handle the delay between countdown steps.

6. **Chaining Promises:**

   - Create a function `delayedLogSequence` that takes an array of messages and delays (in milliseconds). The function should use Promises to log each message to the console in sequence, waiting for the specified delay between each log.

7. **Using `async/await` with Promises:**

   - Rewrite the `delayedLogSequence` function using `async/await` instead of chaining `.then()` methods. Ensure the function logs each message with the specified delay.

8. **Combining `setTimeout` and Promises:**

   - Create a function called `randomDelayMessage` that logs a message to the console after a random delay between 1 and 5 seconds. Use `setTimeout` and a Promise to implement this.

9. **Using `setInterval` with Promises:**

   - Write a function called `repeatAction` that takes an action (a function), an interval (in milliseconds), and a duration (in milliseconds). Use `setInterval` to repeatedly execute the action at the specified interval, but stop after the specified duration. Return a Promise that resolves when the repeating action stops.