Easy: Basic Promise Usage

Objective: Understand how to create and use a Promise.

Task: Write a JavaScript function named `makeTea` that simulates the action of making tea. This function should return a Promise that:

1. Resolves after a delay of 2 seconds, simulating the time it takes to make tea.
2. Resolves with the string `"Tea is ready!"`.

Criteria for Success:

- Use the `new Promise` syntax.
- Use `setTimeout` to simulate the delay.
- Ensure the Promise resolves with the correct string.

Medium: Chaining Promises

Objective: Learn how to chain Promises using `.then()` and handle errors with `.catch()`.

Task: Write a JavaScript function named `prepareBreakfast` that simulates preparing a breakfast consisting of tea and toast. This function should:

1. Call the `makeTea` function from the Easy task and wait for it to complete.
2. Then, simulate making toast, which takes 3 seconds, and resolves with the string `"Toast is ready!"`.
3. Chain these actions so that they happen sequentially, not concurrently.
4. Catch any errors and log them to the console.

Criteria for Success:

- Use `.then()` for chaining.
- Use `setTimeout` to simulate the toast preparation.
- Ensure the proper sequence of breakfast preparation.
- Use `.catch()` to handle errors.

Hard: Using `Promise.all` and Async/Await

Objective: Understand how to handle multiple Promises concurrently and use async/await syntax for cleaner code.

Task: Write a JavaScript function named `prepareFullBreakfast` that simulates preparing a full breakfast of eggs, toast, and tea concurrently. This function should:

1. Have three separate functions that return Promises for making eggs (`makeEggs`), making toast (`makeToast`), and making tea (`makeTea`). Each function should resolve after a random delay between 1 to 5 seconds, simulating the unpredictable nature of cooking times.
2. Use `Promise.all` to wait for all three components of the breakfast to be ready.
3. After all components are ready, log `"Full breakfast is ready!"` to the console.
4. Use async/await syntax for cleaner code and error handling.

Criteria for Success:

- Properly implement `makeEggs` , `makeToast` , and reuse `makeTea` with random delays.
- Use `Promise.all` to handle concurrent Promise execution.
- Use async/await syntax for handling asynchronous code.
- Catch and log any errors.