Title: Promises Deep Dive

1. Why Promises
   JavaScript often needs to wait for things — like fetching data from an API. Without Promises, this can get messy and hard to manage. Promises give us a clean way to handle asynchronous code.

2. Creating a Promise
   A Promise represents a value that will be available now, later, or never.
   Example: const promise = new Promise((resolve, reject) => { let success = true; if (success) resolve("Data loaded"); else reject("Error loading data"); });

3. Using .then() and .catch()
   You can handle the result of a Promise like this:
   promise.then(result => console.log(result)).catch(error => console.error(error))

4. Chaining Promises
   You can run multiple asynchronous actions one after another.
   Example: fetch("data.json").then(res => res.json()).then(data => console.log(data)).catch(err => console.log("Error:", err))

5. Async / Await
   Async and await make Promises easier to read.
   Example:
   async function getData() {
   try {
   const res = await fetch("data.json");
   const data = await res.json();
   console.log(data);
   } catch (error) {
   console.log("Error:", error);
   }
   }

6. Promise States
   A Promise can be:

- Pending still working

- Fulfilled finished successfully
- Rejected finished with an error