📊 Promise vs Async/Await in JavaScript

🔢	🧠 Feature	🤝 Promise	🕰️ Async/Await
1️⃣	🧾 **Definition**	Represents an **intermediate state** of an async operation	A **syntactic sugar** over Promises to write cleaner, synchronous-looking code
2️⃣	🔁 **States**	Has **3 states**: `pending`, `fulfilled`, `rejected`	Doesn't have its own states (it wraps a Promise internally)
3️⃣	⛓️ **Execution Flow**	`then()` continues execution after previous task is done	`await` **pauses** execution until the Promise resolves
4️⃣	🛠️ **Chaining**	Uses `.then()`, `.catch()` for chaining and error handling	Uses `try...catch` blocks for better error handling
5️⃣	😖 **Complexity**	Can lead to **callback/promise hell** with deeply nested `.then()` chains	Easier to **read and maintain** with sequential flow
6️⃣	🧑‍💻 **Example**	`fetchData().then(res => doStuff(res)).catch(err => handleErr(err))`	`const res = await fetchData(); doStuff(res)`
7️⃣	🐞 **Error Handling**	`.catch()` to handle errors	`try...catch` block catches errors like `try { await fetch() } catch {}`
8️⃣	🧪 **Debugging**	Harder to trace through `.then()` chains	Easier to step through line by line using debugger
9️⃣	⛳ **Best Use Case**	When you need **parallel async calls** using `Promise.all()`	When you want **sequential** async operations (one after another)
🔟	🌈 **Code Style Preference**	Better for **one-liners or simple chains**	Better for **complex logic and multiple awaits**

✅ Summary

- Use **Promise** when:
  - You want **parallel operations**.
  - The chain is simple.
  - You're using `Promise.all()` or `Promise.race()`.
- Use **Async/Await** when:
  - You want **cleaner**, **linear flow**.
  - You have **multiple sequential async calls**.
  - You want better **error handling and debugging**.