Async JavaScript — Lesson Notes

The Problem

JavaScript is single-threaded. Long operations (API calls, file reads) would block the UI if done synchronously. Async patterns solve this.

Callbacks

```
function fetchData(callback) {
  setTimeout(() => {
    callback({ name: "Alice" });
  }, 1000);
}
fetchData((data) => console.log(data));
```

Problem: Callback hell — deeply nested callbacks become unreadable.

Promises

```
const promise = new Promise((resolve, reject) => {
  setTimeout(() => resolve("Done!"), 1000);
});

promise
  .then(result => console.log(result))
  .catch(error => console.error(error))
  .finally(() => console.log("Finished"));
```

Async / Await

```
async function loadUser() {
  try {
    const response = await fetch("/api/user");
    const data = await response.json();
    return data;
  } catch (error) {
    console.error("Failed:", error);
  }
}
```

Promise.all — Parallel Requests

```
const [users, posts] = await Promise.all([
  fetch("/api/users").then(r => r.json()),
  fetch("/api/posts").then(r => r.json()),
]);
```

Key Takeaways

1. Prefer `async/await` over `.then()` chains for readability.
2. Always wrap `await` calls in `try/catch` for error handling.
3. Use `Promise.all()` when requests are independent (runs in parallel).
4. Never forget to handle rejected promises — unhandled rejections crash Node.
