Lecture 20: Async Await in Javascript

What is async/await?

async/await is a modern JavaScript feature (ES2017) that provides a cleaner way to work with promises. Instead of chaining `.then()` calls, you can write asynchronous code that reads like synchronous code.

The Basics

`async` **functions** always return a promise:

```
async function myFunction() { return "Hello"; } // Equivalent to: function myFunction() { return Promise.resolve("Hello"); }
```

`await` pauses execution until a promise resolves:

```
async function fetchData() { const response = await fetch('https://api.example.com/data'); const data = await response.json(); return data; }
```

Comparing Promises vs Async/Await

With Promises:

```
function getUser() { return fetch('/api/user') .then(response => response.json()) .then(user => { return fetch(`/api/posts/${user.id}`); }) .then(response => response.json()) .then(posts => { console.log(posts); return posts; }) .catch(error => console.error(error)); }
```

With Async/Await:

```
async function getUser() { try { const response = await fetch('/api/use
r'); const user = await response.json(); const postsResponse = await fetch
(`/api/posts/${user.id}`); const posts = await postsResponse.json(); conso
le.log(posts); return posts; } catch (error) { console.error(error); } }
```

Error Handling

Use `try/catch` blocks for error handling, which is more intuitive than `.catch()`:

```
async function riskyOperation() { try { const result = await mightFail();
return result; } catch (error) { // Handle specific errors if (error.code
=== 'NETWORK_ERROR') { console.error('Network failed'); } throw error; //
Re-throw if needed } }
```

Important Rules

1. `await` **only works inside** `async` **functions** (or top-level in modules)
2. `await` **pauses execution** of that function, but doesn't block the entire program
3. **You can only await promises** (or values, which get wrapped in resolved promises)

Parallel vs Sequential Execution

Sequential (slower):

```
const user = await fetchUser(); // Wait const posts = await fetchPosts();
// Then wait
```

Parallel (faster):

```
const [user, posts] = await Promise.all([ fetchUser(), fetchPosts() ]);
```

Common Patterns

Handling multiple operations:

```
async function loadDashboard() { try { const [user, notifications, stats] = await Promise.all([ getUser(), getNotifications(), getStats() ]); return { user, notifications, stats }; } catch (error) { console.error('Failed to load dashboard:', error); } }
```

Conditional awaits:

```
async function getData(useCache) { if (useCache) { return getCachedData(); } return await fetchFreshData(); }
```

The beauty of async/await is that it makes asynchronous code much easier to read and reason about, especially when you have complex logic with multiple asynchronous operations. Does this make sense? Want to dive deeper into any specific aspect?

What Gets Caught

When you use `try/catch` with async/await, the `catch` block handles:

1. **Rejected promises** - when a promise you `await` rejects
2. **Thrown errors** - any error thrown with `throw` inside the `try` block
3. **Synchronous errors** - runtime errors like `TypeError`, `ReferenceError`, etc.

Examples

Rejected Promise:

```
async function example1() { try { const result = await Promise.reject(new Error('Promise rejected!')); } catch (error) { console.log(error.message); // "Promise rejected!" } }
```

Thrown Error:

```
async function example2() { try { throw new Error('Something went wrong');
} catch (error) { console.log(error.message); // "Something went wrong" }
}
```

Runtime Error:

```
async function example3() { try { const obj = null; console.log(obj.proper
ty); // TypeError } catch (error) { console.log(error.message); // "Cannot
read property 'property' of null" } }
```

Real-world example with fetch:

```
async function fetchUser(id) { try { const response = await fetch(`/api/us
er/${id}`); // fetch doesn't reject on HTTP errors (404, 500, etc.) // So
you need to check manually: if (!response.ok) { throw new Error(`HTTP erro
r! status: ${response.status}`); } const user = await response.json(); ret
urn user; } catch (error) { // This catches: // - Network failures (promis
e rejection from fetch) // - The thrown error for bad HTTP status // - JSO
N parsing errors console.error('Failed to fetch user:', error); } }
```

Important Note about `fetch()`

A common gotcha: `fetch()` only rejects on **network failures**, not on HTTP error statuses like 404 or 500. You need to check `response.ok` manually and throw an error if needed.

So to answer your question directly: yes, rejected promises go into the `catch` block, along with any other errors thrown during execution of the `try` block!