Async and Await in JavaScript

What is `async` ?

- `async` is a keyword written before a function.
- An `async` function always returns a **promise**.
- If an `async` function returns a string or any other value, it is automatically wrapped in a **promise** before being returned.

Example:

```
async function getData() {
    return "Hello";
}
const dataPromise = getData();

console.log(dataPromise);
// Output:
// Promise {<fulfilled>: 'Hello'}
// [[Prototype]]: Promise
// [[PromiseState]]: "fulfilled"
// [[PromiseResult]]: "Hello"

dataPromise.then(res => {
    console.log(res); // Hello
});
```

What is `await` ?

- `await` can only be used inside an `async` function.
- It pauses the execution of an `async` function until the promise is resolved.

Example:

```
const p1 = new Promise((resolve) => {
    setTimeout(() => { resolve("p1 resolved"); }, 2000);
});

const p2 = new Promise((resolve) => {
    setTimeout(() => { resolve("p2 resolved"); }, 1000);
});

async function handlePromise() {
    const p1Data = await p1;
    console.log("First promise");
    console.log(p1Data);

    const p2Data = await p2;
    console.log("Second promise");
    console.log(p2Data);
}

handlePromise();
```

Execution Flow of `async` and `await`

1. `handlePromise()` is added to the **call stack**.
2. Execution reaches `await p1;`, suspending `handlePromise()` and removing it from the call stack.
3. Once `p1` is resolved, `handlePromise()` is moved back to the call stack and resumes execution at `await p1`.
4. Execution reaches `await p2;`, suspending `handlePromise()` again.
5. Once `p2` is resolved, `handlePromise()` resumes at `await p2` and continues execution.
6. If `p1` takes more time than `p2`, the function **does not move to `p2` immediately** since execution is paused at `await p1`.
7. If `p2` resolves before `p1`, it still waits for `p1` to complete before moving forward.

Error Handling in `async` / `await`

Example 1: Using `try...catch`

```
async function process() {
    try {
        const user = await fetchUser(url);
    } catch (err) {
        console.error('Error:', err);
    }
}
process();
```

Example 2: Using `.catch()`

```
async function process() {
    const user = await fetchUser(url);
}

process().catch(err => console.log(err));
```

Additional Resources

Understanding Async/Await