WEB TECHNOLOGIES

Async / Await

Prof. Pavan A C

Department of Computer Science and Engineering

Acknowledgement:
Teaching Assistants(Harini B and Chandana MS)

Async / Await

Introduction

- **Async and Await in JavaScript** are used to simplify handling asynchronous operations using promises. By enabling asynchronous code to appear synchronous, they enhance code readability and make it easier to manage complex asynchronous flows.

- The **async** function allows us to write promise-based code as if it were synchronous. This ensures that the execution thread is not blocked.
- The keyword async before a function makes the function return a promise:

```
async function Function1() {
  return "Returns a promise";
}
```

Async / Await

Introduction

```
const getData = async () => {
    let data = "Hello World";
    return data;
}

getData().then(data => console.log(data));
```

- If a function returns a value, the Promise is resolved with that value.
- If an exception is thrown, the Promise is rejected.

- The async function getData returns "Hello World" wrapped in a Promise.
- .then() is used to get the resolved result and print it.
- Output: Hello World

Async / Await

Await

The await keyword is used to wait for a promise to resolve. It can only be used within an async block. Await makes the code wait until the promise returns a result

- It pauses the async function until the Promise resolves/rejects.
- Returns the resolved value or throws an error if the Promise is rejected.

```
async function getData() {
  let response = await fetch("https://api.example.com/data");
  let data = await response.json();
  return data;
}
```

Async / Await

Await

```
// Practical example
async function fetchUser(userId) {
  try {
    const response = await fetch(`https://jsonplaceholder.typicode.com/users/${userId}`);
    const user = await response.json();
    console.log(user);
  } catch (error) {
    console.error("Failed to fetch user:", error);
  }
}
fetchUser(1);
```

Async / Await

Await

```
async function getUserAndPosts(userId) {
  try {
    const userResponse = await fetch(`https://jsonplaceholder.typicode.com/users/${userId}`);
    const user = await userResponse.json();
    const postsResponse = await fetch(`https://jsonplaceholder.typicode.com/posts?userId=${userId}`);
    const posts = await postsResponse.json();
    console.log(user);
    console.log(posts);
  } catch (error) {
    console.error('Error:', error);
  }
}
getUserAndPosts(1);
```

Sequential API calls

Async / Await

Await

```
async function fetchMultiple() {
  try {
    const [user, post] = await Promise.all([
      fetch('https://jsonplaceholder.typicode.com/users/1').then(res => res.json()),
      fetch('https://jsonplaceholder.typicode.com/posts/1').then(res => res.json())
    ]);
    console.log('User:', user);
    console.log('Post:', post);
  } catch (err) {
    console.error('Failed to fetch:', err);
  }
}
fetchMultiple();
```

Promise.all for parallel requests

Async / Await

Await

```
function wait(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}

async function demoDelay() {
  console.log('Waiting...');
  await wait(2000);
  console.log('2 seconds later!');
}
demoDelay();
```

- `demoDelay` is an async function.
- It first logs `"Waiting..."` immediately.
- Then it `await`s the `wait(2000)` call, pausing execution inside the function until the Promise resolves after 2000 milliseconds (2 seconds).
- After waiting 2 seconds, it proceeds to log `"2 seconds later!"`.

O/P
Waiting...
(2 second pause)
2 seconds later!

Error Handling with try / catch in Async Functions

- When using async / await, errors from awaited Promises can be caught just like synchronous code errors.
- Wrapping your awaited code inside a try block lets you handle errors gracefully in the accompanying catch block.
- This approach avoids scattered .catch() calls and makes your async error handling clearer and easier to manage.
- If any awaited Promise rejects, control jumps to catch, enabling centralized error processing.

Similar to the above example

Async / Await

Multiple Await

Multiple Await Statements

```
async function process() {
  let a = await doA();
  let b = await doB(a);
  let c = await doC(b);
  return c;
}
```

Sequential

```
async function getBoth() {
  let [user, posts] = await Promise.all([
    fetchUser(),
    fetchPosts()
  ]);
  // both fetched in parallel
}
```

Parallel

Best Practices

- Always use try/catch for error handling.
- Avoid blocking the main thread—await is for I/O network, not CPU-heavy tasks.
- Use async/await for serial operations, Promise.all for parallel.
- Every async function returns a Promise.

Real-World Use Cases

- Fetching API data in web apps
- Database queries in Node.js servers
- Reading files asynchronously
- Chained and dependent HTTP requests

Async / Await

MCQ

1. **What will "async function example() { return 5; }" return when called?**
   A) 5
   B) A Promise that resolves with 5
   C) null
   D) undefined

1. **What happens if a Promise awaited with await rejects, and the code is NOT inside a try/catch?**

   A) The function stops without error

   B) JavaScript ignores the error

   C) The async function returns a rejected Promise with that error

   D) The function returns undefined

Async / Await

MCQ

1. **Which of the following is a correct usage of `await`?**
   A) At the top level in any JavaScript file
   B) Only inside an `async` function
   C) Inside any normal function
   D) Inside setTimeout

1. **4. To execute three API calls in parallel and wait for all to finish, which is the best approach?**
   A) Placing three `await` statements in sequence
   B) Using Promise.all([...]) around the three Promises with await
   C) Running three asynchronous functions in three scripts
   D) Using only callbacks

Async / Await

MCQ Answers

1. **Answer 1:** B) A Promise that resolves with 5
2. **Answer 2:** C) The async function returns a rejected Promise with that error
3. **Answer 3:** B) Only inside an async function
4. **Answer 4:** B) Using Promise.all([...]) around the three Promises with await

THANK YOU

Prof. Pavan A C

Department of Computer Science and Engineering