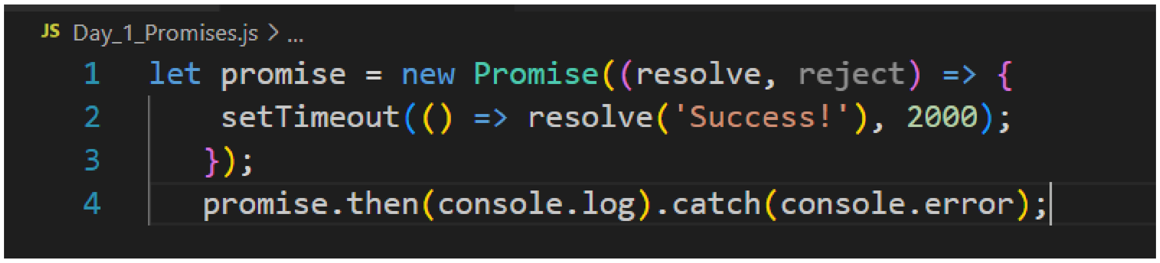
JavaScript Asynchronous Programming Assignment-01

Q1. What is a Promise in JavaScript? Describe its three states with examples.

A Promise in JavaScript is an object that handles asynchronous operations. It acts like a placeholder for a value that will be available in the future. It has three states:

1. **Pending** – Initial state, neither resolved nor rejected.
2. **Fulfilled (Resolved)** – The operation completed successfully.
3. **Rejected** – The operation failed.
   Example:



Day_1_Promises.js > ...

```javascript
let promise = new Promise((resolve, reject) => {
    setTimeout(() => resolve('Success!'), 2000);
  });
  promise.then(console.log).catch(console.error);
```

Q2. Difference between then().catch() and async/await with try/catch

a) .then().catch() vs async/await with try/catch

Feature	.then().catch()	async/await with try/catch
Syntax	Uses chaining with .then() and .catch()	Uses await inside async function
Readability	Harder to read with nested .then()	Easier to read, like synchronous code
Error Handling	Errors caught using .catch()	Errors caught using try/catch

Example using .then().catch():

```
fetchData()
 .then(console.log)
 .catch(console.error);
```

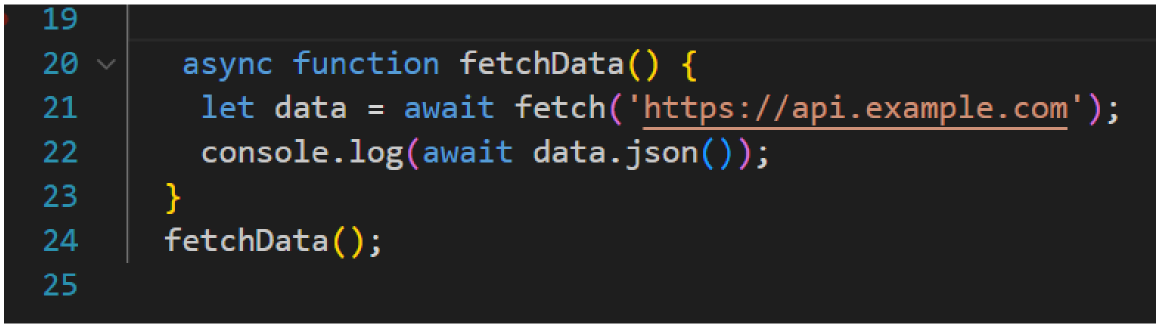
Example using async/await with try/catch:

```
async function getData() {
 try {
  let data = await fetchData();
  console.log(data);
 } catch (error) {
  console.error(error);
 }
}
```

Q3. What does the await keyword do inside an async function? Can it be used outside an async function?

The 'await' keyword pauses the execution of an async function until the Promise resolves. It cannot be used outside an async function, otherwise it will throw an error.

Example:

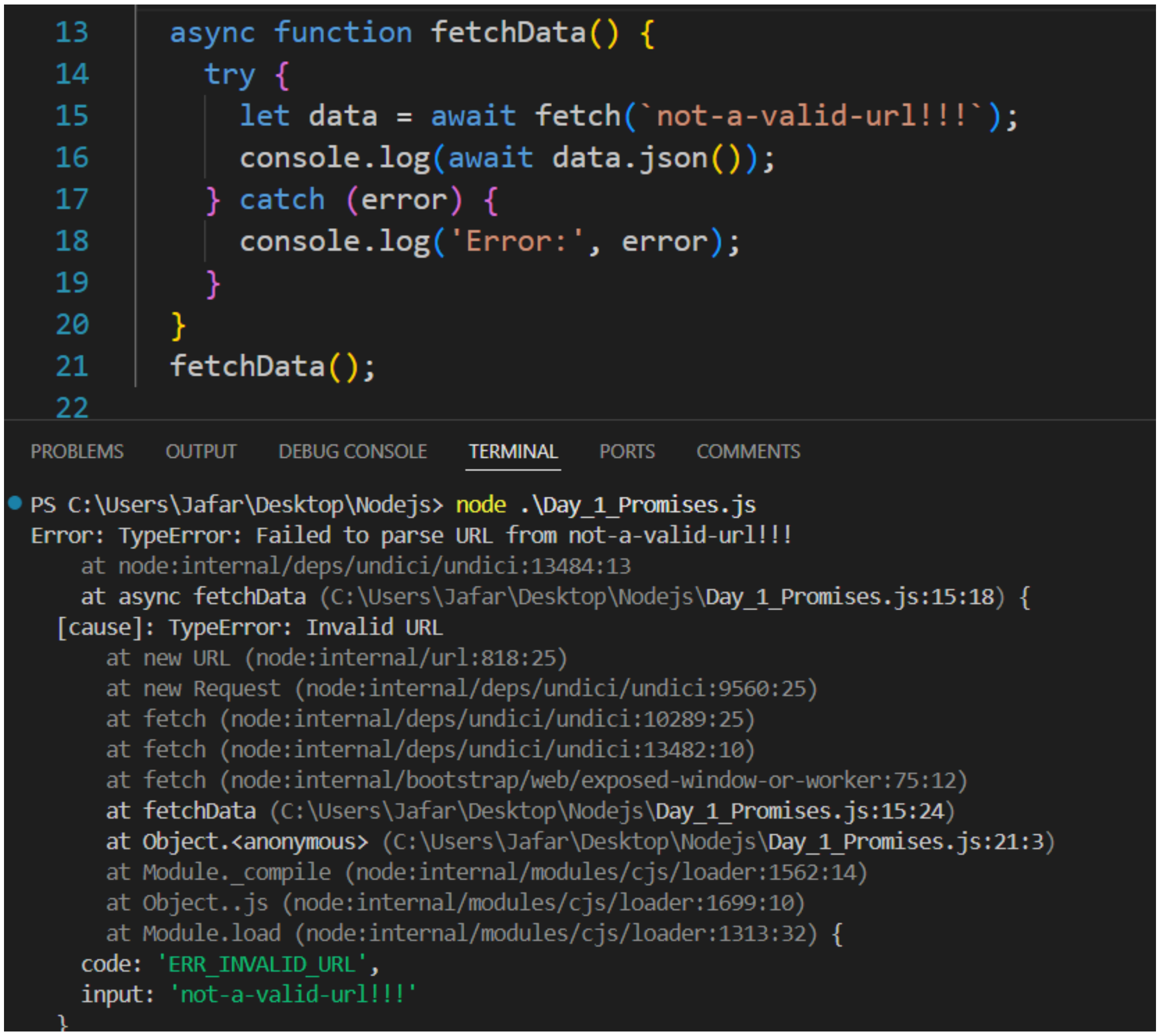


```

 async function fetchData() {
  let data = await fetch('https://api.example.com');
  console.log(await data.json());
}
fetchData();

```

Q4. Role of try...catch in asynchronous programming. Why is it better than just using .catch()?

try...catch handles errors in async/await code similarly to synchronous code, providing a clear and structured way to catch errors, often making error handling more intuitive than using .catch().

Example:



```
async function fetchData() {
  try {
    let data = await fetch(`not-a-valid-url!!!`);
    console.log(await data.json());
  } catch (error) {
    console.log('Error:', error);
  }
}
fetchData();
```

```
PS C:\Users\Jafar\Desktop\Nodejs> node .\Day_1_Promises.js
Error: TypeError: Failed to parse URL from not-a-valid-url!!!
    at node:internal/deps/undici/undici:13484:13
    at async fetchData (C:\Users\Jafar\Desktop\Nodejs\Day_1_Promises.js:15:18) {
  [cause]: TypeError: Invalid URL
      at new URL (node:internal/url:818:25)
      at new Request (node:internal/deps/undici/undici:9560:25)
      at fetch (node:internal/deps/undici/undici:10289:25)
      at fetch (node:internal/deps/undici/undici:13482:10)
      at fetch (node:internal/bootstrap/web/exposed-window-or-worker:75:12)
      at fetchData (C:\Users\Jafar\Desktop\Nodejs\Day_1_Promises.js:15:24)
      at Object.<anonymous> (C:\Users\Jafar\Desktop\Nodejs\Day_1_Promises.js:21:3)
      at Module._compile (node:internal/modules/cjs/loader:1562:14)
      at Object..js (node:internal/modules/cjs/loader:1699:10)
      at Module.load (node:internal/modules/cjs/loader:1313:32) {
    code: 'ERR_INVALID_URL',
    input: 'not-a-valid-url!!!'
  }
```

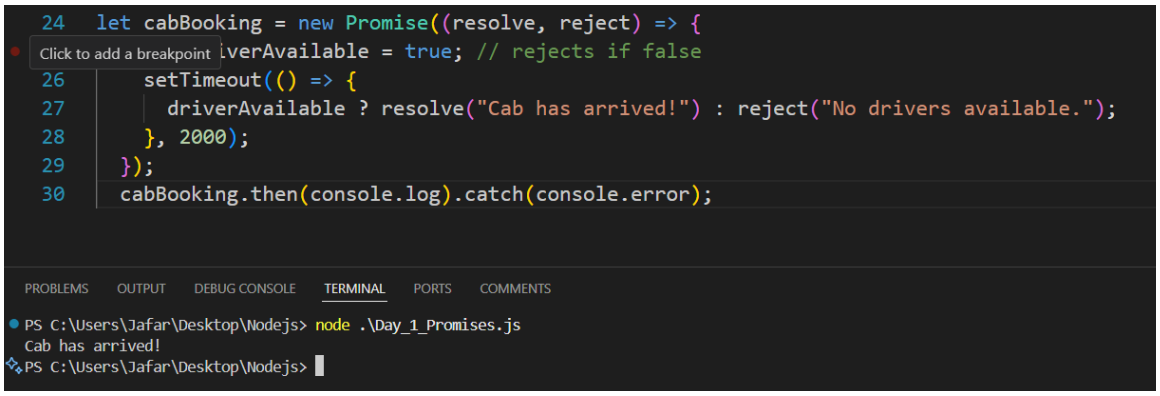
Q5. What is a callback function? Provide a real-world analogy and a coding scenario where callbacks are used.

A callback function is a function passed as an argument to another function and executed later.

Real-world analogy: A **Promise** is like booking a cab using a ride-hailing app.

- **Pending:** You book a cab and wait for confirmation.
- **Fulfilled:** The driver arrives, and you start your ride.
- **Rejected:** No drivers are available, and your booking is canceled.

Example:



```
let cabBooking = new Promise((resolve, reject) => {
    Click to add a breakpoint iverAvailable = true; // rejects if false
    setTimeout(() => {
      driverAvailable ? resolve("Cab has arrived!") : reject("No drivers available.");
    }, 2000);
  });
  cabBooking.then(console.log).catch(console.error);
```

```
PS C:\Users\Jafar\Desktop\Nodejs> node .\Day_1_Promises.js
Cab has arrived!
PS C:\Users\Jafar\Desktop\Nodejs> 
```

Q6. Difference between default export and named export

Default Export: Uses 'export default' and can be imported with any name. Named Export: Uses 'export' and must be imported with the exact exported name.
Example:

// Default export
export default function greet() { return 'Hello'; }

// Named export
export const PI = 3.14;

Q7. Identify the error in the code

const data = await getData();

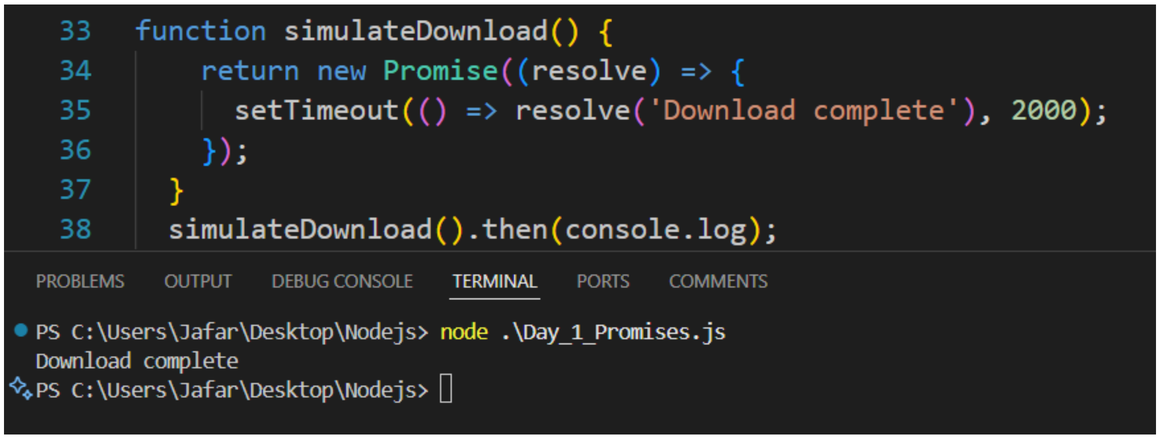
console.log(data);

→

Error: 'await' is used outside an async function.
Fix:

```
async function fetchData() {
  const data = await getData();
  console.log(data);
}
fetchData();
```

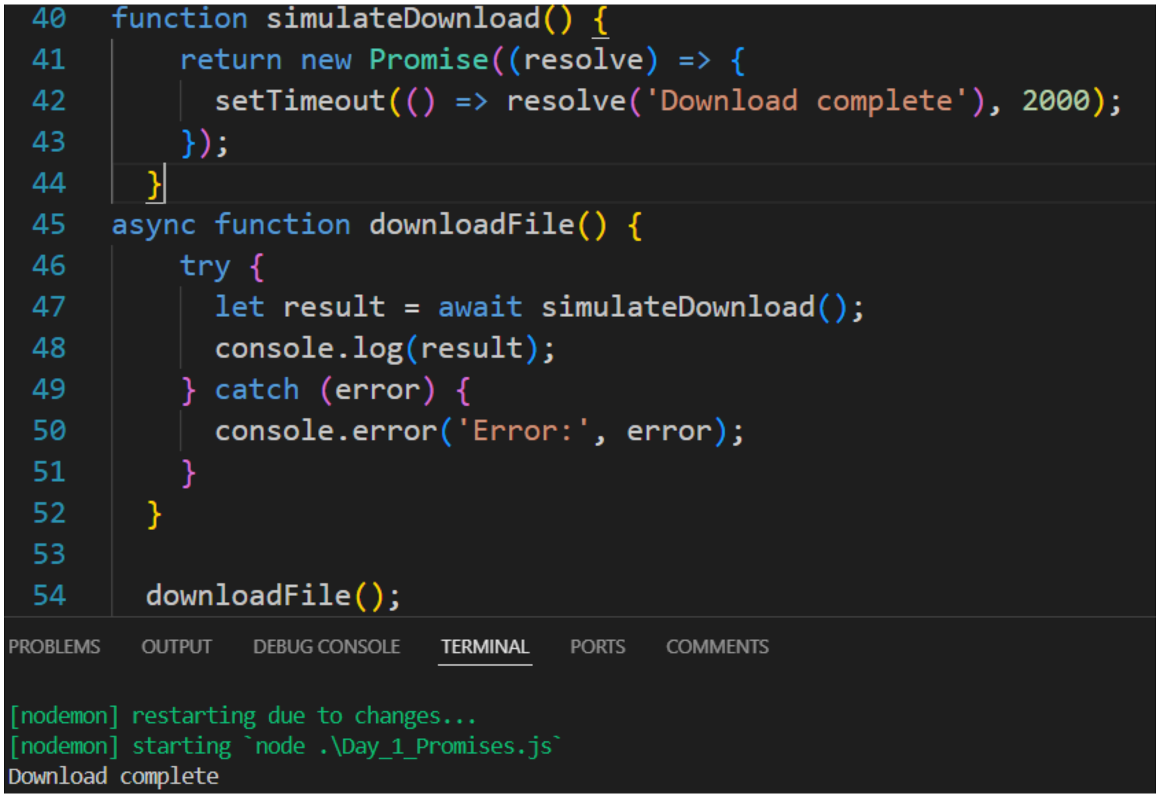
Q8. Create a Promise called simulateDownload() that resolves with "Download complete" after 2 seconds. Log the result using .then().



```javascript
function simulateDownload() {
    return new Promise((resolve) => {
      setTimeout(() => resolve('Download complete'), 2000);
    });
  }
  simulateDownload().then(console.log);
```

```
PS C:\Users\Jafar\Desktop\Nodejs> node .\Day_1_Promises.js
Download complete
PS C:\Users\Jafar\Desktop\Nodejs>
```

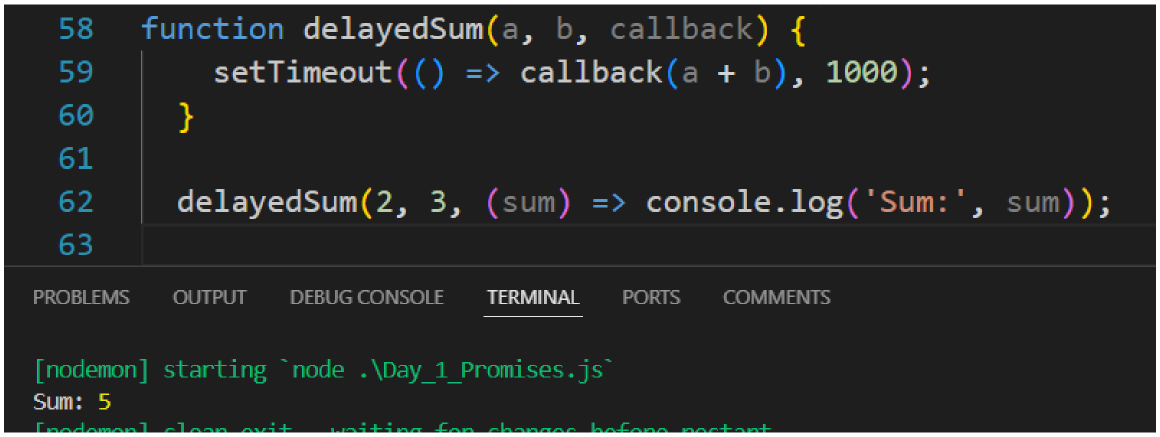
Q9. Re-write Q8 using async/await with try/catch



```javascript
function simulateDownload() {
    return new Promise((resolve) => {
      setTimeout(() => resolve('Download complete'), 2000);
    });
  }
async function downloadFile() {
    try {
      let result = await simulateDownload();
      console.log(result);
    } catch (error) {
      console.error('Error:', error);
    }
  }

  downloadFile();
```

```
[nodemon] restarting due to changes...
[nodemon] starting `node .\Day_1_Promises.js`
Download complete
```

Q10. Write a function delayedSum(a, b, callback) that adds two numbers after 1 second and then passes the result to the callback.



```javascript
function delayedSum(a, b, callback) {
    setTimeout(() => callback(a + b), 1000);
  }

  delayedSum(2, 3, (sum) => console.log('Sum:', sum));
```

```
[nodemon] starting `node .\Day_1_Promises.js`
Sum: 5
```

Q11. Export a constant PI = 3.14 and a function calculateArea(radius) from one file and import them into another file to calculate the area of a circle.

File: math.js

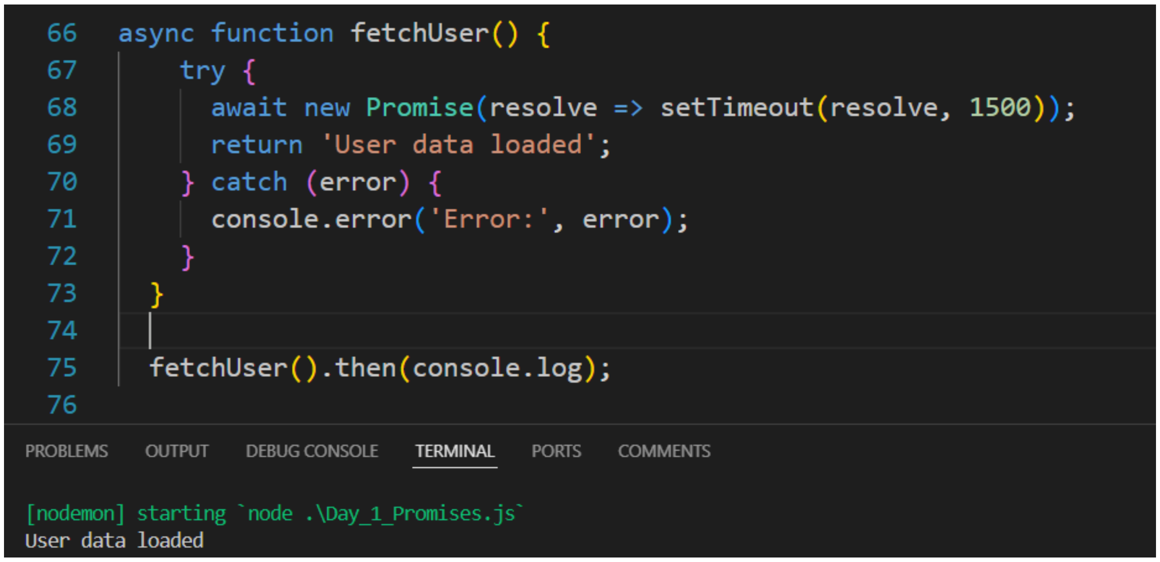
```javascript
export const PI = 3.14;
export function calculateArea(radius) {
  return PI * radius * radius;
}
```

File: app.js

```javascript
import { PI, calculateArea } from './math.js';
console.log('Area:', calculateArea(5));
```

Q12. Write an async function fetchUser() that:

- waits 1.5 seconds
- then returns "User data loaded"
- and handles error with try/catch



```
async function fetchUser() {
    try {
      await new Promise(resolve => setTimeout(resolve, 1500));
      return 'User data loaded';
    } catch (error) {
      console.error('Error:', error);
    }
  }
  
  fetchUser().then(console.log);
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   COMMENTS

```
[nodemon] starting `node .\Day_1_Promises.js`
User data loaded
```

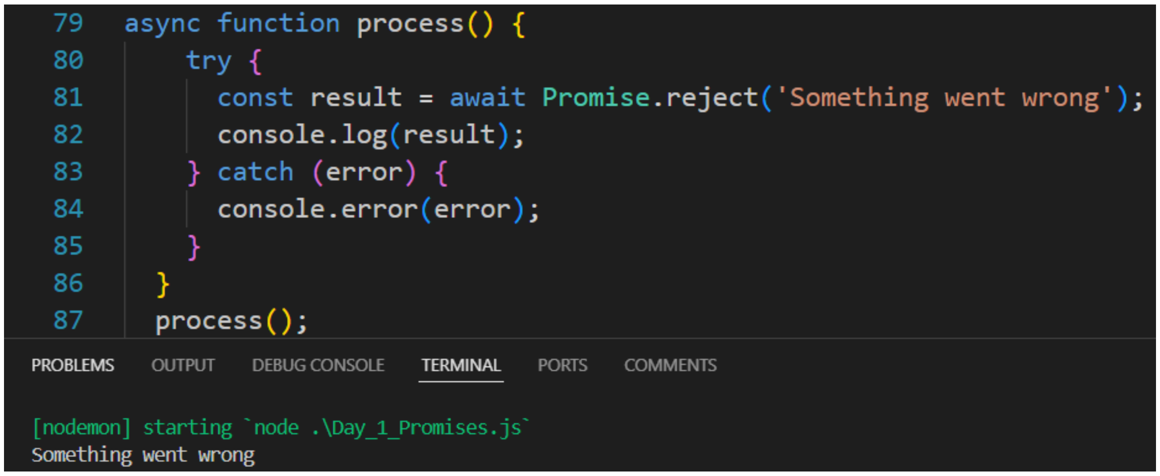
Q13. Given the following Promise function, write code that logs "Completed" if it resolves, and "Error occurred" if it rejects.

function loadData(success = true) { return new Promise((resolve, reject) => { setTimeout(() => { success ? resolve("Completed") : reject("Error occurred"); }, 1000); }); }

```
loadData(true)
  .then(console.log)
  .catch(console.error);
```

Q14. Modify the following code so that it catches and logs any error that might happen inside the async function:

async function process() { const result = await Promise.reject("Something went wrong"); console.log(result); }



```
async function process() {
    try {
      const result = await Promise.reject('Something went wrong');
      console.log(result);
    } catch (error) {
      console.error(error);
    }
  }
  process();
```

```
[nodemon] starting `node .\Day_1_Promises.js`
Something went wrong
```

Q15. Create a mini module

- File 1: Exports a function greet(name) that returns "Hello, ".
- File 2: Imports and uses the function to greet 2 different names.

File: greet.js

```
export function greet(name) {
  return `Hello, ${name}`;
}
```

File: app.js

```
import { greet } from './greet.js';
console.log(greet('Alice'));
console.log(greet('Bob'));
```