Welcome to the CoGrammar Lecture: Promises

The session will start shortly...

Questions? Drop them in the chat. We'll have dedicated moderators answering questions.

CoGrammar

Full Stack Web Development Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

Full Stack Web Development Session Housekeeping cont.

- For all **non-academic questions**, please submit a query: www.hyperiondev.com/support

- Report a **safeguarding** incident: www.hyperiondev.com/safeguardreporting

- We would love your **feedback** on lectures: [Feedback on Lectures]()

Skills Bootcamp 8-Week Progression Overview

Fulfil 4 Criteria to Graduation

✅ **Criterion 1: Initial Requirements**

Timeframe: First 2 Weeks
Guided Learning Hours (GLH): Minimum of 15 hours
Task Completion: First four tasks

Due Date: 24 March 2024

✅ **Criterion 2: Mid-Course Progress**

60 Guided Learning Hours

Data Science - **13 tasks**
Software Engineering - **13 tasks**
Web Development - **13 tasks**

Due Date: 28 April 2024

CoGrammar

Skills Bootcamp Progression Overview

✅ Criterion 3: Course Progress

Completion: All mandatory tasks, including Build Your Brand and resubmissions by study period end
Interview Invitation: Within 4 weeks post-course
Guided Learning Hours: Minimum of 112 hours by support end date (10.5 hours average, each week)

✅ Criterion 4: Demonstrating Employability

Final Job or Apprenticeship Outcome: Document within 12 weeks post-graduation
Relevance: Progression to employment or related opportunity

CoGrammar

Agenda

- ❖ Recognize the purpose and structure of promises in JavaScript.
- ❖ Implement basic asynchronous operations using promises.
- ❖ Convert existing promise-based code to utilize the async/await syntax for improved readability.
- ❖ Utilize error handling techniques within promise chains and async/await functions.

CoGrammar

Sync-Async-Sync

❖ **Synchronous programming** executes code line by line in sequential order.

❖ **Asynchronous programming** allows tasks to be executed concurrently, without waiting for each other to complete.

CoGrammar

Asynchronous Tasks in Web Development

- Fetching data from external APIs without halting other operations.
- Processing user input while simultaneously performing computations in the background.

CoGrammar

JavaScript Promise

- ❖ A Promise is an object representing the eventual completion or failure of an asynchronous operation.
- ❖ A promise is a returned object to which you attach callbacks, instead of passing callbacks into a function.
- ❖ In JavaScript, a promise is a good way to handle asynchronous operations.

```
fetch(url, requestOptions) Promise<Response>
  .then((response) => {
    if (!response.ok) {
      alert('You were logged out due to ina
      throw Error(response.statusText);
    }
    return response.json();
  }) Promise<any>
  .then(data => this.setState( state: {
    data,
    loading: false,
  })) Promise<void>
  .catch(() => this.setState( state: { loadin
```

CoGrammar

JavaScript Promise

- It is used to find out if the asynchronous operation is successfully completed or not.
- A promise may have one of three states: Pending, Fulfilled or Rejected.
- A promise starts in a pending state. That means the process is not complete.

CoGrammar

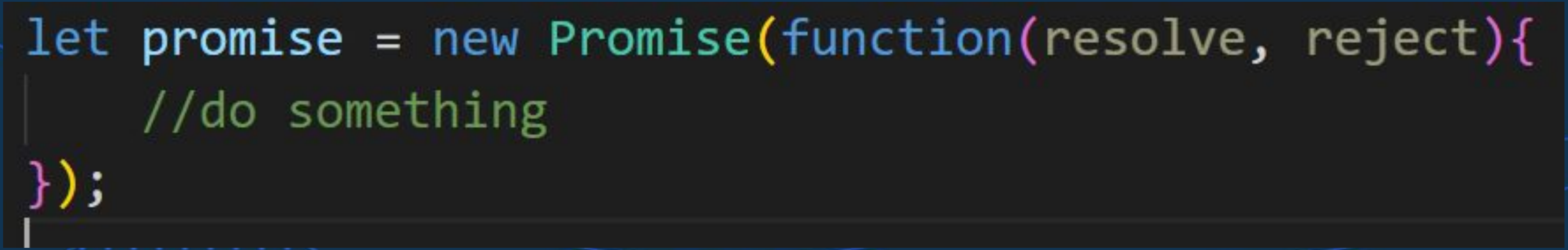
JavaScript Promise

- If the operation is successful, the process ends in a fulfilled state.
- If an error occurs, the process ends in a rejected state.
- For example, when you request data from the server by using a promise, it will be in a pending state.
- When the data arrives successfully, it will be in a fulfilled state.
- If an error occurs, then it will be in a rejected state.

CoGrammar

Creating a Promise

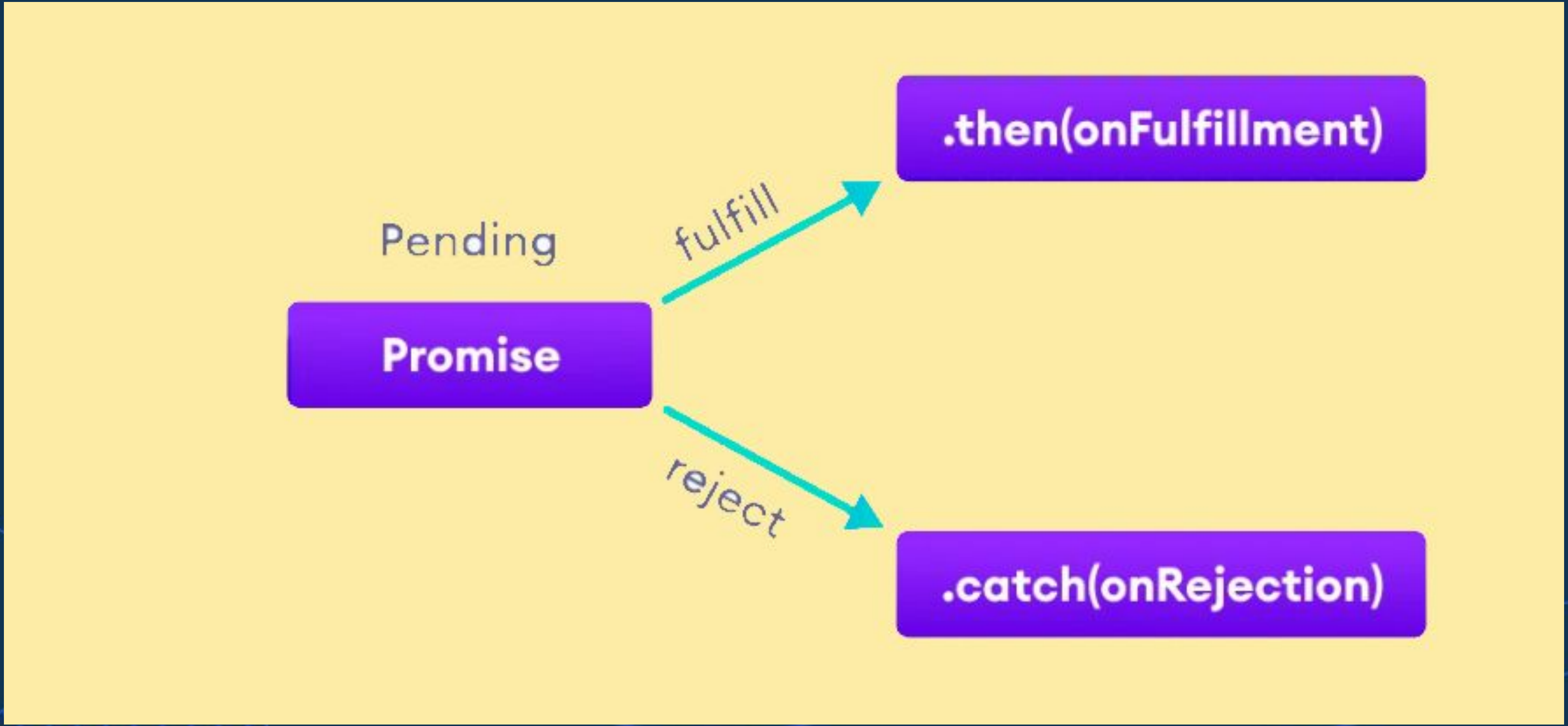
- To create a promise object, we use the **Promise()** constructor.
- The **Promise()** constructor takes a function as an argument.
- The function also accepts two functions **resolve()** and **reject()**.
- If the promise returns successfully, the **resolve()** function is called.
- If an error occurs, the **reject()** function is called.



```
let promise = new Promise(function(resolve, reject){
    //do something
});
```

CoGrammar

Creating a Promise



Promise Chaining

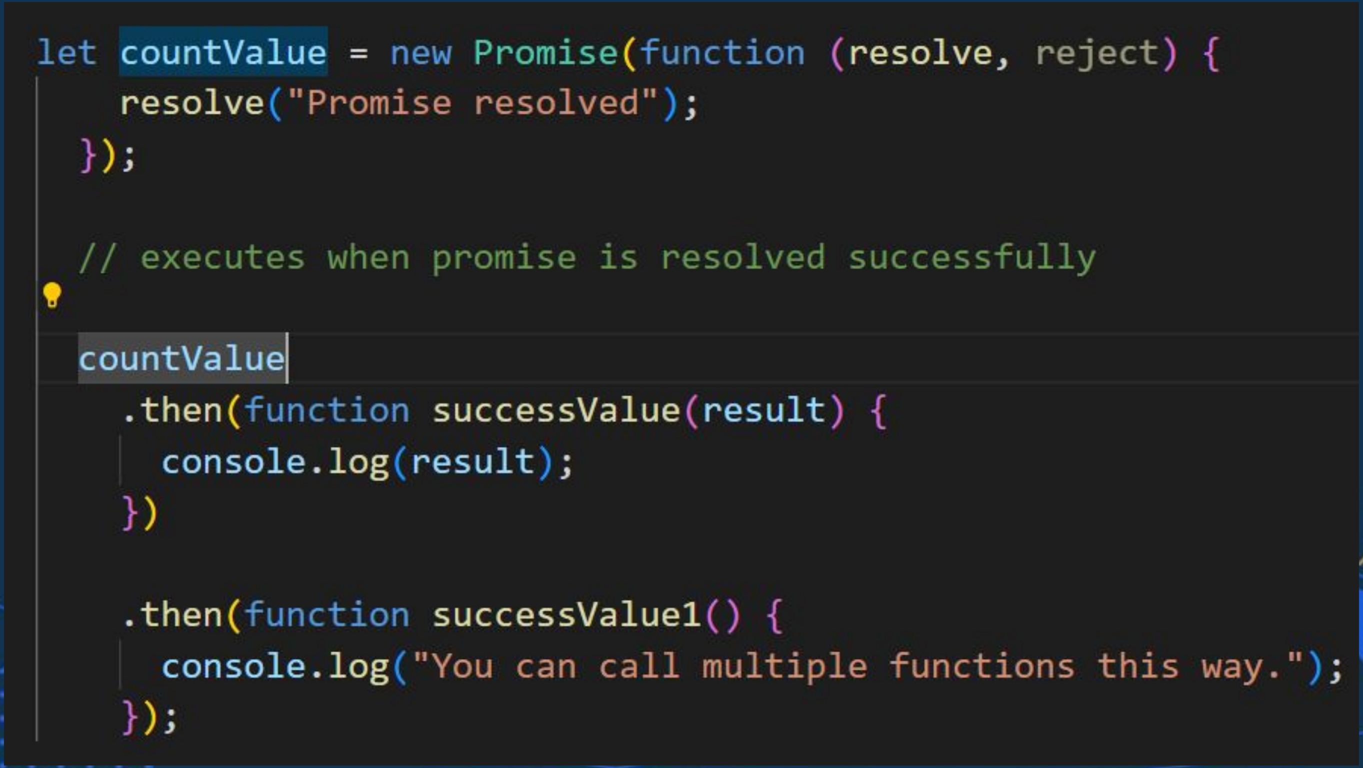
- Promises are useful when you have to handle more than one asynchronous task, one after another. For that, we use promise chaining.
- You can perform an operation after a promise is resolved using methods **then()**, **catch()** and **finally()**.

And then() I said to myself...

- The **then()** method is used with the callback when the promise is successfully fulfilled or resolved.
- The **then()** method is called when the promise is resolved successfully.
- You can chain multiple then() methods with the promise.

CoGrammar

then() method

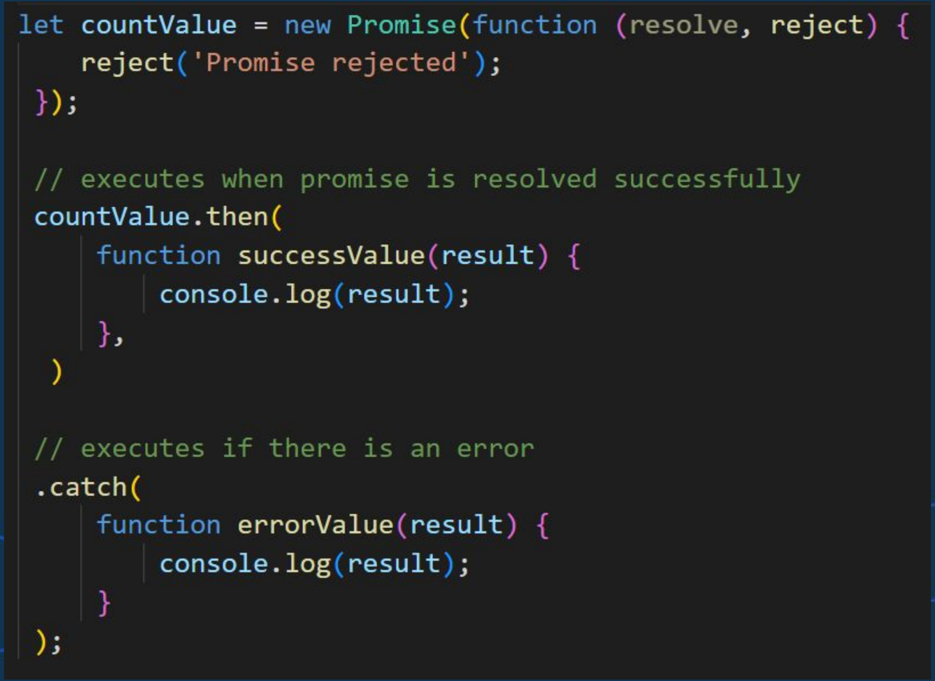


```
let countValue = new Promise(function (resolve, reject) {
    resolve("Promise resolved");
  });

  // executes when promise is resolved successfully

  countValue
    .then(function successValue(result) {
      console.log(result);
    })

    .then(function successValue1() {
      console.log("You can call multiple functions this way.");
    });
```

CoGrammar

catch() me if you can...

❖ The **catch()** method is used with the callback when the promise is **rejected** or if an **error** occurs.

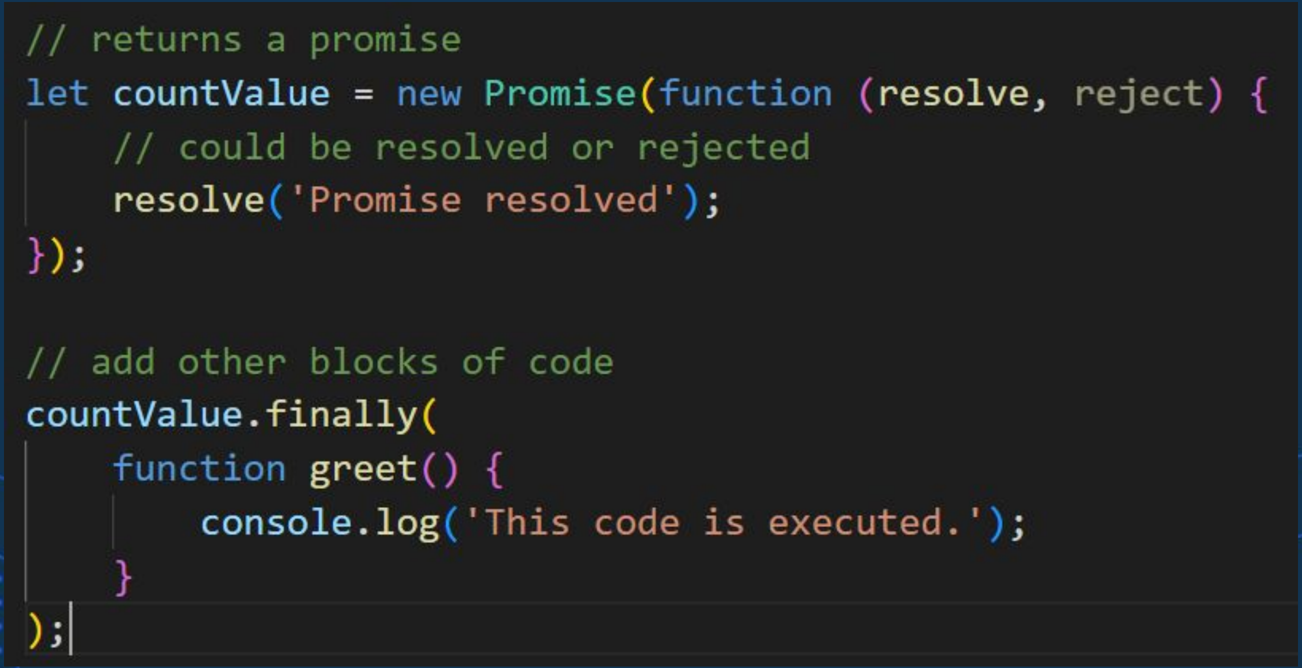


```
let countValue = new Promise(function (resolve, reject) {
    reject('Promise rejected');
 });

 // executes when promise is resolved successfully
 countValue.then(
     function successValue(result) {
         console.log(result);
     },
  )

 // executes if there is an error
 .catch(
     function errorValue(result) {
         console.log(result);
     }
 );
```

CoGrammar

OMG, finally()!

❖ The **finally()** method gets executed when the promise is either resolved successfully or rejected.



```
// returns a promise
let countValue = new Promise(function (resolve, reject) {
    // could be resolved or rejected
    resolve('Promise resolved');
});

// add other blocks of code
countValue.finally(
    function greet() {
        console.log('This code is executed.');
    }
);
```

CoGrammar

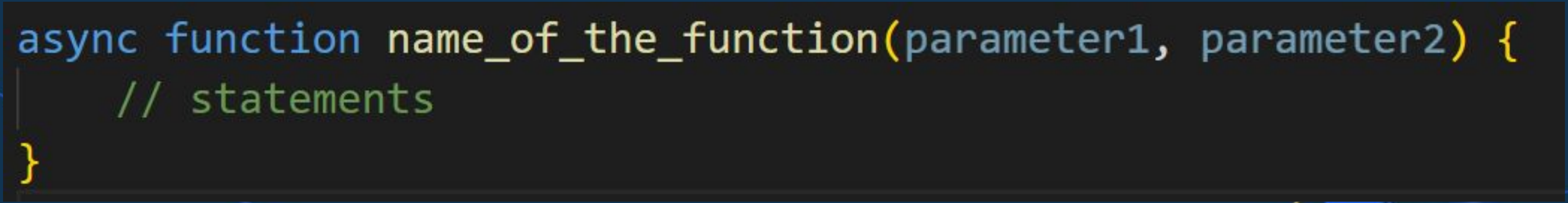
Let’s Breathe!

Let’s take a small break before moving on to the next topic.

CoGrammar

Async

- We use the async keyword with a function to represent that the function is an asynchronous function.
- The async function returns a promise.



```
async function name_of_the_function(parameter1, parameter2) {
    // statements
}
```

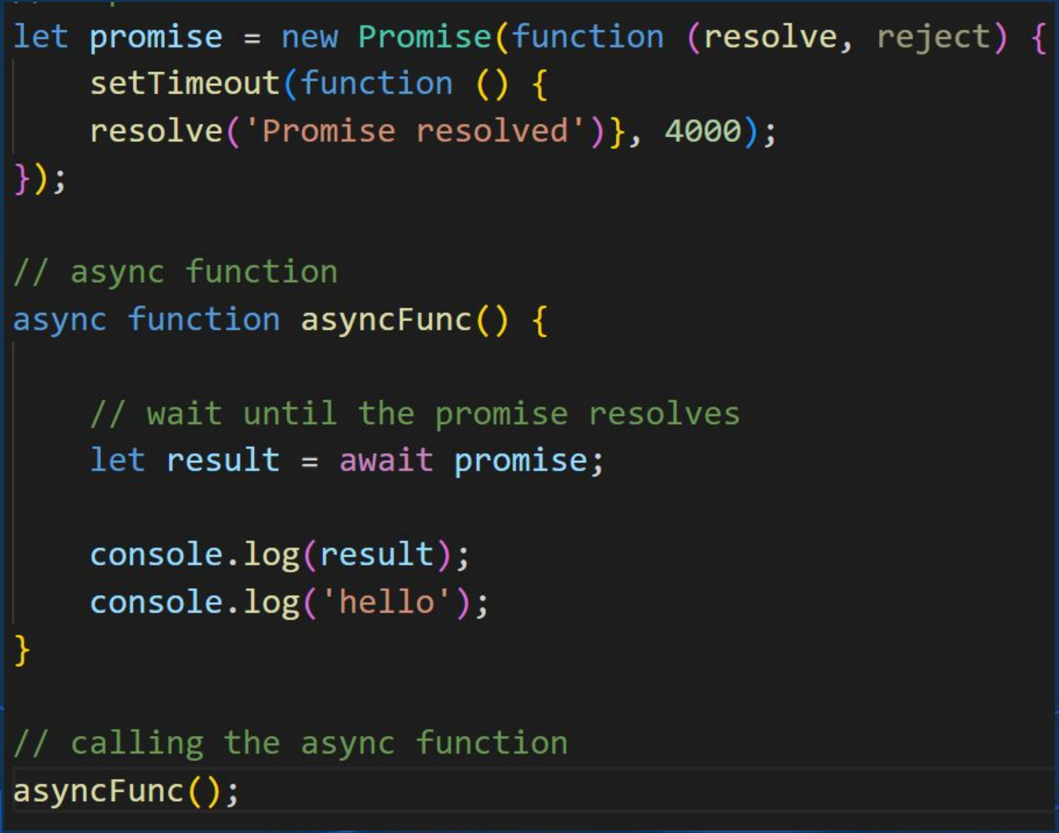
CoGrammar

Await

- The await keyword is used inside the async function to wait for the asynchronous operation.
- The use of await pauses the async function until the promise returns a result (resolve or reject) value.
- You can use await only inside of async functions.
- The await keyword waits for the promise to be complete (resolve or reject).

CoGrammar

Await



```
let promise = new Promise(function (resolve, reject) {
    setTimeout(function () {
    resolve('Promise resolved')}, 4000);
});

// async function
async function asyncFunc() {

    // wait until the promise resolves
    let result = await promise;

    console.log(result);
    console.log('hello');
}

// calling the async function
asyncFunc();
```

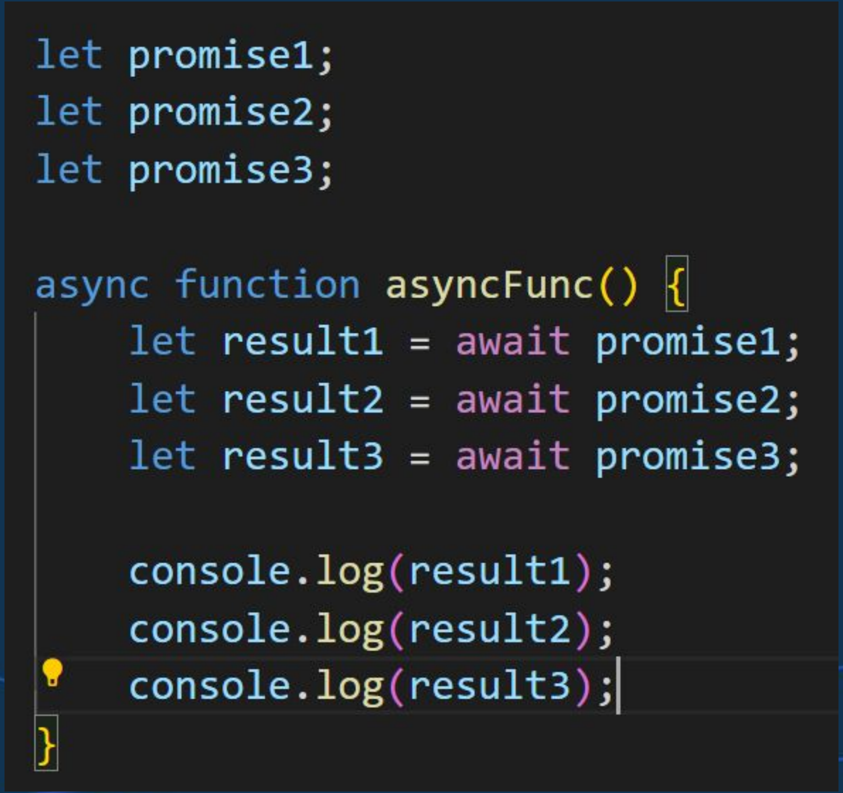
CoGrammar

Async/Await Syntax

- ❖ The async/await syntax allows the asynchronous method to be executed in a seemingly synchronous way.
- ❖ Though the operation is asynchronous, it seems that the operation is executed in synchronous manner.
- ❖ This can be useful if there are multiple promises in the program.

CoGrammar

Async/Await Syntax

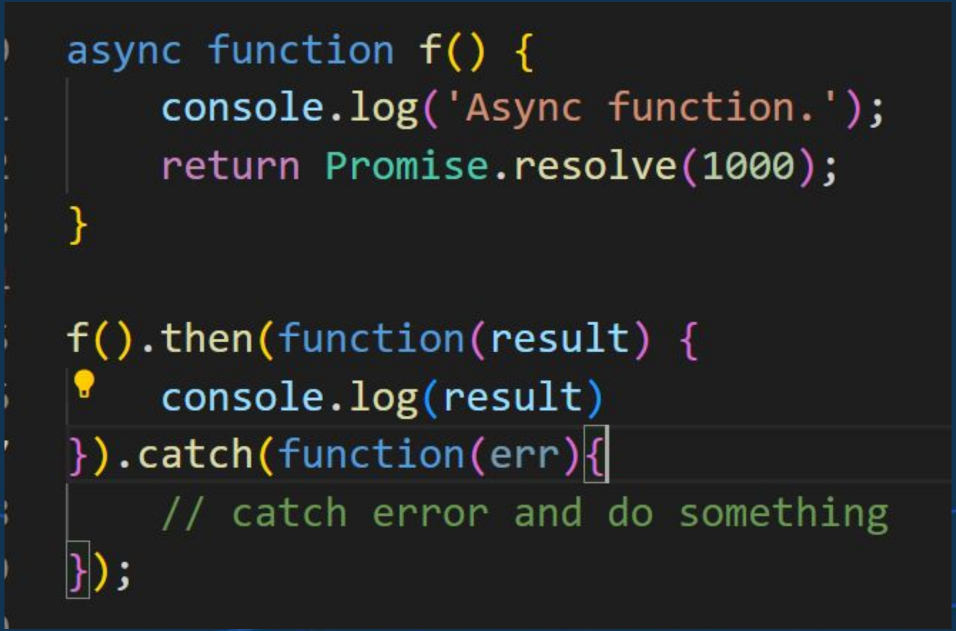


```
let promise1;
let promise2;
let promise3;

async function asyncFunc() {
    let result1 = await promise1;
    let result2 = await promise2;
    let result3 = await promise3;

    console.log(result1);
    console.log(result2);
    console.log(result3);
}
```

CoGrammar

Error Handling

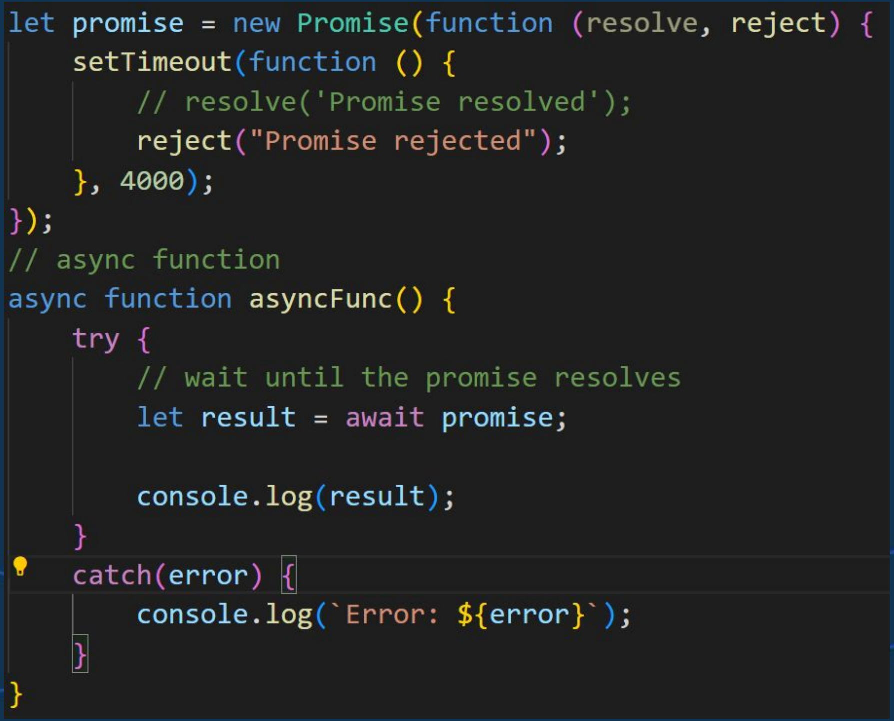
❖ You can also use the catch() method to catch the error.



```
async function f() {
    console.log('Async function.');
    return Promise.resolve(1000);
}

f().then(function(result) {
    console.log(result)
}).catch(function(err){
    // catch error and do something
});
```

Error Handling

❖ The other way you can handle an error is by using try/catch block.



```
let promise = new Promise(function (resolve, reject) {
    setTimeout(function () {
        // resolve('Promise resolved');
        reject("Promise rejected");
    }, 4000);
});
// async function
async function asyncFunc() {
    try {
        // wait until the promise resolves
        let result = await promise;

        console.log(result);
    }
    catch(error) {
        console.log(`Error: ${error}`);
    }
}
```

CoGrammar

Questions and Answers

CoGrammar

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar