



Welcome to this session: Task Walkthrough - Promises, Async and Await

The session will start shortly...

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



Safeguarding & Welfare

We are committed to all our students and staff feeling safe and happy; we want to make sure there is always someone you can turn to if you are worried about anything.

If you are feeling upset or unsafe, are worried about a friend, student or family member, or you feel like something isn't right, speak to our safeguarding team:



Ian Wyles
Designated Safeguarding
Lead



Simone Botes



Nurhaan Snyman



Rafiq Manan



Ronald Munodawafa



Tevin Pitts

Scan to report a
safeguarding concern



or email the Designated
Safeguarding Lead:
Ian Wyles

safeguarding@hyperiondev.com

Skills Bootcamp Cloud Web Development

- 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**

Skills Bootcamp Cloud Web Development

- 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](#)**
- If you are hearing impaired, please kindly use your computer's function through Google chrome to enable captions.

Learning Outcomes

- ❖ **Use promises to handle asynchronous data requests** and understand how to work with API responses.
- ❖ **Implement `async/await` syntax** to simplify asynchronous operations and improve code readability.
- ❖ **Extract and present data from API responses** in a user-friendly format.
- ❖ **Incorporate error handling in API calls** to manage unexpected issues in data retrieval.

Lecture Overview

- Presentation of the Task
- Promises
- Async
- Await
- Task Walkthrough




Promises Task

Ever wondered what it takes to become a weather forecaster? 🌤️ In this task, you'll create a Weather Update Fetcher using JavaScript promises. Choose any city you like and fetch the latest weather information from the Weather API!

You'll be able to pull up details like temperature, weather conditions, and humidity levels—just like a real meteorologist.

Whether it's sunny skies in London or rain showers in New York, you'll get the full forecast with a few lines of code. This is your chance to bring real-time weather data into your app, impressing your friends and honing your data-fetching skills!

Async/Await Task


Get ready to add a bit of daily inspiration to your console!  In this task, you'll build a Random Quotes Fetcher using `async/await`. Each time you run the function, you'll get a brand-new, uplifting quote straight from a Random Quotes API. It's like having your own digital pocketbook of wisdom, ready whenever you need a pick-me-up.

Perfect for keeping the creativity flowing, this task shows you how to work with `async/await` for fetching data from APIs—and who doesn't love an inspiring quote to brighten their day?



What is the primary purpose of an API in web development?

- A. To style the webpage.
- B. To fetch or send data to an external source.
- C. To create forms.
- D. To add animations.



What does a JavaScript promise help with?

- A. Handling asynchronous operations like data fetching.
- B. Stopping the code.
- C. Making code synchronous.
- D. Adding elements to HTML.



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.

Asynchronous Tasks

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



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.

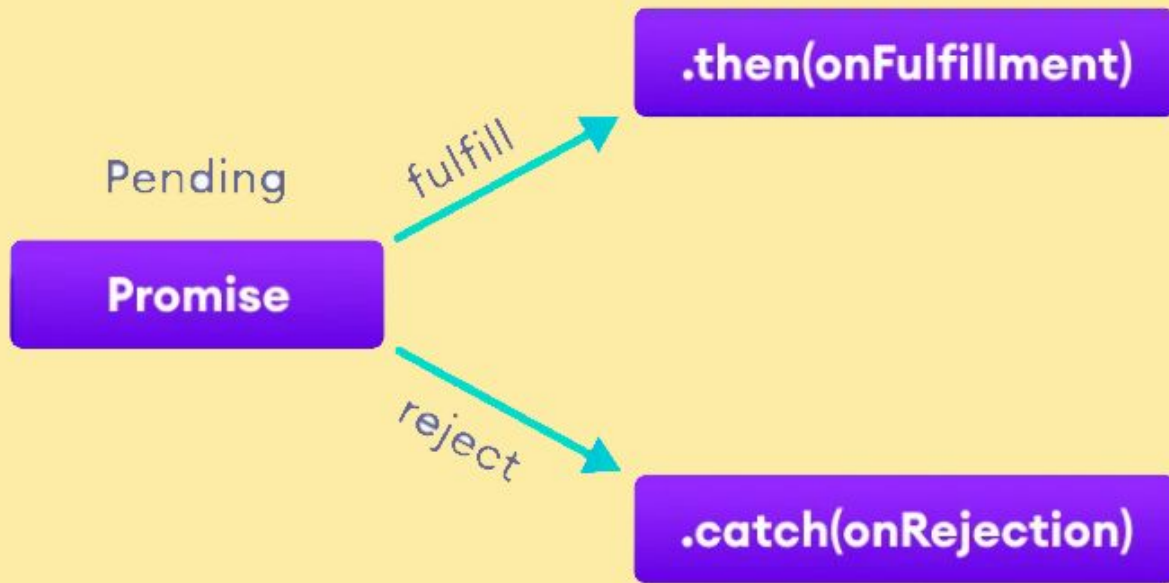


Creating a Promise

- ❖ 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  
});
```

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()**.

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.");
  });
```

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);  
    }  
);
```

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.');
```

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  
}
```

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).

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();
```


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.



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);  
}
```



Promises Task

Ever wondered what it takes to become a weather forecaster? 🌤️ In this task, you'll create a Weather Update Fetcher using JavaScript promises. Choose any city you like and fetch the latest weather information from the Weather API!

You'll be able to pull up details like temperature, weather conditions, and humidity levels—just like a real meteorologist.

Whether it's sunny skies in London or rain showers in New York, you'll get the full forecast with a few lines of code. This is your chance to bring real-time weather data into your app, impressing your friends and honing your data-fetching skills!

Async/Await Task

Get ready to add a bit of daily inspiration to your console!  In this task, you'll build a Random Quotes Fetcher using `async/await`. Each time you run the function, you'll get a brand-new, uplifting quote straight from a Random Quotes API. It's like having your own digital pocketbook of wisdom, ready whenever you need a pick-me-up.

Perfect for keeping the creativity flowing, this task shows you how to work with `async/await` for fetching data from APIs—and who doesn't love an inspiring quote to brighten their day?

What does `async/await` improve in JavaScript code?

- A. Synchronous operations.
- B. Code readability and handling of asynchronous tasks.
- C. CSS styling.
- D. HTML structuring.



What does `.catch()` do in a promise chain?

- A. Adds elements to an array.
- B. Logs errors if the promise fails.
- C. Pauses code execution.
- D. Removes data from the DOM.

CoGrammar

Q & A SECTION

**Please use this time to ask
any questions relating to the
topic, should you have any.**

Thank you for attending



CoGrammar



Department
for Education