**Asynchronous Programming in Java Script**

**Description**

Asynchronous programming is often confusing when first learning JavaScript. In this SuperSKill, Asynchronous Programming in JavaScript,, you’ll gain the ability to handle async programming in JavaScript.  
First, you’ll explore consuming promises.  
Next, you’ll discover creating your own promises.  
Finally, you’ll learn how to use async/await to handle asynchronous programming.  
When you’re finished with this SuperSKill, you’ll have the skills and knowledge of JavaScript promises needed to handle asynchronous code.

# Understanding Promises

# Solving the Race Condition

# Problems with Callback Pyramid

# Solving the Callback Pyramid

# Consuming Promises

# Handling Errors with Promises

# Chaining Promises

# Catching Errors in a Chain

# Performing One Last Operation

https://www.youtube.com/watch?v=9HrhgW9tJe0

# Queuing Promises

# Settling All Promises

# Waiting for a Single Response

# Waiting for the First Response

https://youtu.be/T\_TdLSOec4w

# Creating Promises

# Understanding Promise States

# Rejecting a Promise

https://youtu.be/Mkfnto9Ab5o

# Iterating with Async / Await

# Awaiting a Call

# Handling Errors with Async / Await

# Chaining Async / Await

# Awaiting Concurrent Requests

# Awaiting Parallel Calls

# Review

# https://youtu.be/1m3H3rKfXGs

# An Overview of Error Handling in JavaScript

# Demo: Different Types of Errors

# Demo: A Closer Look at the Error Object

# Demo: Handling Errors with a Try, Catch, Finally Block

# Demo: Asynchronous Error Handling

# <https://youtu.be/Dus1OKNw41o>

# Conclusion

1. **Event Loop Mechanism:** Asynchronous programming in JavaScript relies on the event loop, a mechanism that allows non-blocking execution of code. The event loop continuously checks the message queue for events or tasks and executes them in a sequential manner.
2. **Callbacks:** Callback functions are a fundamental concept in asynchronous JavaScript. They are functions passed as arguments to other functions, to be executed later when a specific event occurs. Callbacks enable the execution of code after asynchronous operations, such as file I/O or network requests, are completed.
3. **Promises:** Promises are a more structured way to handle asynchronous operations compared to callbacks. They represent a value that might be available now, or in the future, or never. Promises have states (pending, fulfilled, or rejected) and provide methods like **.then()** and **.catch()** for handling the result or error of an asynchronous operation.
4. **Async/Await:** Introduced in ECMAScript 2017, the **async** and **await** keywords simplify asynchronous code and make it look more synchronous. An **async** function returns a promise, and within it, the **await** keyword is used to pause execution until a promise is resolved or rejected. This syntax enhances code readability and maintainability.
5. **Error Handling:** Asynchronous programming in JavaScript requires careful error handling due to the nature of non-blocking operations. Callbacks often involve error-first patterns (callback takes an error as its first parameter), while Promises use **.catch()** for error handling. Async/Await simplifies error handling by allowing the use of try/catch blocks within asynchronous code.