# Async/Await in JavaScript

## Overview

Async/Await is a feature introduced in ES8 (ECMAScript 2017) that makes working with asynchronous code more readable   
and easier to understand. It allows you to write asynchronous code in a way that looks synchronous,   
which improves the clarity and maintainability of your code.

## Key Concepts

### 1. async Function

The `async` keyword is used to define a function that returns a promise. Any function marked with `async` will implicitly   
return a `Promise`, even if it doesn't have one.

### 2. await Expression

The `await` keyword is used inside an `async` function to pause the execution of the function until a Promise is resolved.   
It can only be used within an `async` function, and it waits for the promise to resolve before proceeding.

## Syntax

async function myAsyncFunction() {  
 // code here  
}  
  
let result = await someAsyncFunction();

## Basic Example with Async/Await

// Simulating an asynchronous task (e.g., network request)  
function fetchData() {  
 return new Promise((resolve, reject) => {  
 setTimeout(() => {  
 resolve("Data fetched!");  
 }, 2000);  
 });  
}  
  
// Async function to handle the asynchronous task  
async function getData() {  
 console.log("Fetching data...");  
 const data = await fetchData(); // Pauses here until the promise resolves  
 console.log(data); // Logs "Data fetched!" after 2 seconds  
}  
  
// Call the async function  
getData();

Expected Output:

Fetching data...  
(Data fetched! after 2 seconds)

## Handling Errors with Async/Await

async function getData() {  
 try {  
 console.log("Fetching data...");  
 const data = await fetchData();  
 console.log(data);  
 } catch (error) {  
 console.error("Error:", error); // If there's any error, it will be caught here  
 }  
}  
  
getData();

## Chaining Async Functions with Async/Await

function fetchUserData() {  
 return new Promise((resolve) => {  
 setTimeout(() => {  
 resolve("User data fetched!");  
 }, 1500);  
 });  
}  
  
function fetchOrders() {  
 return new Promise((resolve) => {  
 setTimeout(() => {  
 resolve("Orders fetched!");  
 }, 1000);  
 });  
}  
  
async function processData() {  
 const userData = await fetchUserData();  
 console.log(userData);  
   
 const orders = await fetchOrders();  
 console.log(orders);  
   
 console.log("All data processed successfully!");  
}  
  
processData();

## Case Study: User Registration Process

Imagine a user registration process where a user must:  
1. Register an account (an asynchronous task).  
2. Send a welcome email (another asynchronous task).  
3. Notify the admin (yet another asynchronous task).  
  
Each step must be completed successfully before moving to the next. If any step fails, the process should stop.

async function userRegistrationProcess(username) {  
 try {  
 await registerAccount(username); // Wait for account registration to finish  
 await sendWelcomeEmail(username); // Wait for the email to be sent  
 await notifyAdmin(username); // Wait for admin notification  
 console.log(`${username}'s registration process completed successfully!`);  
 } catch (error) {  
 console.log("Registration process failed:", error);  
 }  
}  
  
userRegistrationProcess("JohnDoe");