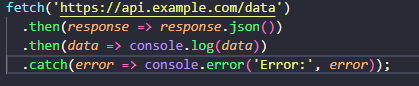
**Task 13 – Date 02/03/2025**

**Fetch API & Async/Await**

1. **Fetch API -** Making API calls using fetch()
   * The **Fetch API** in JavaScript is a modern, built-in interface for making HTTP requests like GET, POST, PUT, DELETE to servers and handling responses easily.
   * It is promise-based, making it simpler to work with asynchronous requests compared to the older XMLHttpRequest method.
   * It is promise-based, meaning it returns a promise that resolves to a response object, making it easy to handle asynchronous requests using then and catch.
   * Example:

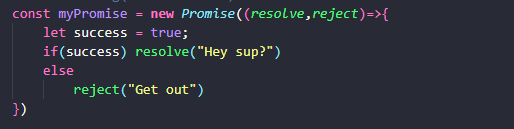
****

* **fetch() makes a GET request to the URL.**
* **response.json() converts the response to JSON format.**
* **then handles the data if the request is successful.**
* **catch handles any errors if the request fails.**

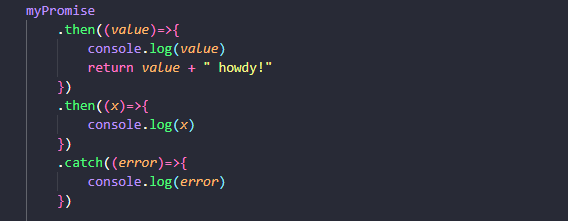
1. **Async/Await**

* Async and await are keywords in JavaScript that make working with promises easier and code more readable by allowing asynchronous code to be written in a synchronous-looking way.
* They are used with functions that return promises, simplifying the process of handling asynchronous operations like fetching data from an API or reading files.
* **How async and await Work**
  + **async:**
    - Used to declare a function as asynchronous.
    - An async function always returns a promise.
    - If the function returns a value, the promise is resolved with that value.
    - If the function throws an error, the promise is rejected.
  + **await:**
    - Can only be used inside an async function.
    - Pauses the execution of the async function until the promise is resolved or rejected.
    - Makes the code wait for the promise to settle before moving to the next line.

**Handling promises**

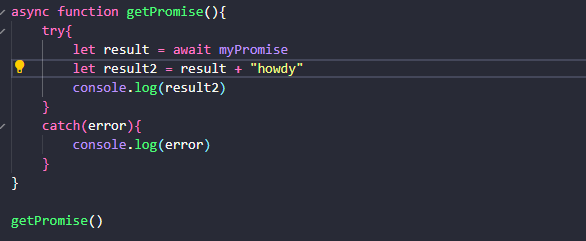
* A promise in JavaScript is an object that represents the eventual completion or failure of an asynchronous operation.
* It acts as a placeholder for a value that will be available in the future.
* Promises help manage asynchronous code (like API calls or file reading) without causing callback hell, making the code easier to read and maintain.
* **A promise can be in one of these three states:**
  1. **Pending**: The initial state, neither fulfilled nor rejected.
  2. **Fulfilled**: The operation completed successfully, and the promise has a value.
  3. **Rejected:** The operation failed, and the promise has a reason for failure (usually an error).
* A promise can be created in following way using Promise constructor
  + 
* resolve(value): Marks the promise as fulfilled and passes value to .then().
* reject(error): Marks the promise as rejected and passes error to .catch().
* There are two main ways to handle a promise
  1. **Using then() and catch()**
  2. **Using async and await**

**Using then() and catch()**

* Using .then() and .catch() involves callback functions to handle the resolved or rejected states of a promise.
* **.then():** Runs if the promise is fulfilled. Receives the value passed to resolve().
* .**catch():** Runs if the promise is rejected. Receives the error passed to reject().
* There can be multiple **then()** and for all of the then()s there will be only one **catch(),** if any error is thrown at any place the catch block will be able to catch the error thrown.
* Example continuing with the code of above:
  + 
  + 

**Using async and await**

* The async and await approach makes asynchronous code look synchronous and is easier to read.
* Example:



* + **async function**: Allows using await inside it.
  + **await keyword:** Pauses the function execution until the promise is settled.
  + **try and catch:** Used for error handling.

**Practical Exercise**  
  
**Objective: Fetch and display a random joke from an API (**[**https://official-joke-api.appspot.com/random\_joke**](https://official-joke-api.appspot.com/random_joke)**).**

**Source code:**

**HTML:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<h1>

Press the button below for a Dad Joke

</h1>

<p></p>

<button>Crack me up</button>

<script src="index.js"></script>

</body>

</html>

**JS:**

async function fetchAJoke(){

try{

result = await fetch("https://official-joke-api.appspot.com/random\_joke")

if(!result.ok){

throw new Error("Failed to fetch")

}

const joke= await result.json()

document.querySelector("p").innerHTML = `<h3>${joke.setup}</h3><h3>${joke.punchline}</h3>`

console.log(joke)

}

catch(error){

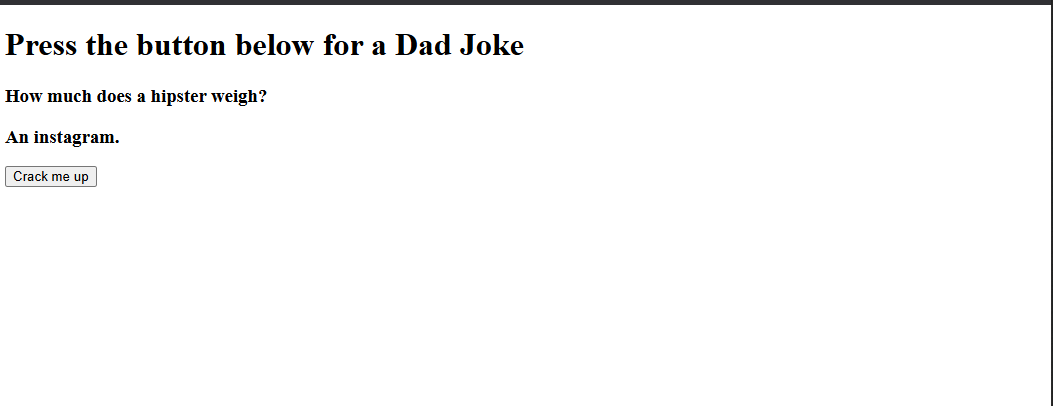
console.log(error)

}

}

document.querySelector("button").addEventListener("click",fetchAJoke)

**Output:**

****