**Practical-11**

**Aim:-**

Write a program that demonstrates asynchronous behavior using a callback   
function. For example, create a function that simulates fetching data from an   
API and invokes a callback with the fetched data.

**Code:-**

function fetchDataFromAPI(callback) {

  setTimeout(() => {

    const data = { message: "Data fetched from API is as bellow",data:{Name:"Vasu Bhut",Id:"21CE014"} };

    callback(null, data);

  }, 2000);

}

function handleFetchedData(error, data) {

  if (error) {

    console.error("Error fetching data:", error);

  } else {

      console.log("Fetched data:", data);

      console.log("My name is:",data.data.Name);

      console.log("My Id No is:",data.data.Id);

  }

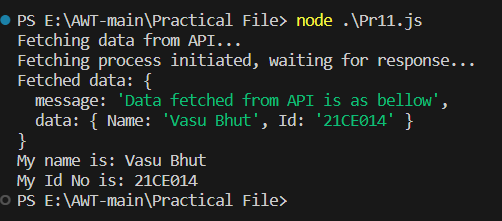
}

console.log("Fetching data from API...");

fetchDataFromAPI(handleFetchedData);

console.log("Fetching process initiated, waiting for response...");

**Output:-**



**Practical-12**

**Aim:-**

Create a program that reads a file asynchronously using callbacks and displays   
its contents.

**Code:-**

const fs = require("fs");

function readFileAsync(filename, callback) {

  fs.readFile(filename, "utf8", (err, data) => {

    if (err) {

      callback(err, null);

      return;

    }

    callback(null, data);

  });

}

function displayFileContents(err, data) {

  if (err) {

    console.error("Error reading the file:", err.message);

  } else {

    console.log("File contents:");

    console.log(data);

  }

}

const filename = "example.txt";

console.log(`Reading file: ${filename}`);

readFileAsync(filename, displayFileContents);

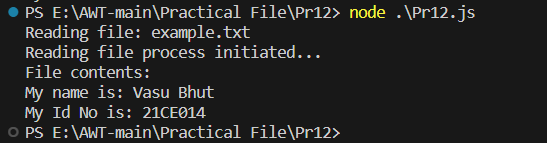
console.log("Reading file process initiated...");

**example.txt**

My name is: Vasu Bhut

My Id No is: 21CE014

**Output:-**



**Practical-13**

**Aim:-**

Write a program that uses Promises to handle asynchronous operations. For   
example, create a function that returns a Promise to fetch data from an API and   
resolve it with the fetched data.   
   
Implement error handling using Promises by rejecting a Promise with an error   
message in case of failure.  
**Code:-**

const fetchFromAPI = () => {

  return new Promise((resolve, reject) => {

    const success = true;

    setTimeout(() => {

      if (success) {

        const data = { message: "Data fetched from the API" };

        resolve(data);

      } else {

        reject(new Error("Failed to fetch data from the API"));

      }

    }, 2000);

  });

};

fetchFromAPI()

  .then((data) => {

    console.log("API call successful:", data);

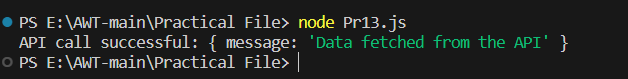
  })

  .catch((error) => {

    console.error("API call failed:", error.message);

  });

**Output:-**



**Practical-14**

**Aim:-**

Convert a Promise-based asynchronous function into an async/await style   
function. For example, rewrite a function that fetches data from an API using   
async/await.   
   
Write a program that utilizes multiple async/await functions to fetch data from   
different APIs sequentially and display the combined results.

**Code:-**

const fetchFromAPI = () => {

  return new Promise((resolve, reject) => {

    const success = true;

    setTimeout(() => {

      if (success) {

        const data = { message: "Data fetched from the API" };

        resolve(data);

      } else {

        reject(new Error("Failed to fetch data from the API"));

      }

    }, 2000);

  });

};

const fetchFromAnotherAPI = () => {

  return new Promise((resolve, reject) => {

    const success = true;

    setTimeout(() => {

      if (success) {

        const data = { message: "Data fetched from another API" };

        resolve(data);

      } else {

        reject(new Error("Failed to fetch data from another API"));

      }

    }, 1500);

  });

};

const fetchDataSequentially = async () => {

  try {

    const apiData = await fetchFromAPI();

    console.log("API 1 data:", apiData);

    const anotherApiData = await fetchFromAnotherAPI();

    console.log("API 2 data:", anotherApiData);

  } catch (error) {

    console.error("Error:", error.message);

  }

};

fetchDataSequentially();

**Output:-**

