**1.What are the key differences between synchronous and asynchronous programming in JavaScript?**

**Synchronous Programming**:

* In synchronous code, tasks are performed one after the other, with each task blocking the execution of subsequent tasks until it completes.
* Example: If a function takes time to complete, it will block the program from executing the next line until it finishes.
* It is predictable, but long-running tasks can freeze the UI in web applications.

**Asynchronous Programming**:

* In asynchronous code, tasks can be initiated and then continue executing other tasks without waiting for the previous one to complete.
* Example: A long-running task, such as fetching data from a server, can be performed without stopping the execution of other tasks.
* It allows non-blocking execution, making it more efficient for operations like I/O or fetching data from APIs.

**2. explain the differences between callbacks, promises, and async/await in JavaScript.**

**Callbacks**:

* Functions passed as arguments to other functions, executed after an operation completes.
* Often leads to "callback hell" when multiple nested callbacks are needed.
* Example:

function fetchData(callback) {

setTimeout(() => {

callback('Data fetched');

}, 1000);

}

fetchData((data) => console.log(data));

**Promises**:

* Objects that represent the eventual completion or failure of an asynchronous operation.
* They can be in one of three states: pending, resolved (fulfilled), or rejected.
* Promises avoid callback hell by chaining .then() and .catch() methods.
* Example:

function fetchData() {

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

setTimeout(() => {

resolve('Data fetched');

}, 1000);

});

}

fetchData().then((data) => console.log(data)).catch((error) => console.error(error));

**Async/Await**:

* Syntactic sugar over promises, making asynchronous code look synchronous.
* Makes code easier to read and maintain, especially when chaining multiple asynchronous operations.

Example

async function fetchData() {

try {

const data = await new Promise((resolve) => {

setTimeout(() => resolve('Data fetched'), 1000);

});

console.log(data);

} catch (error) {

console.error(error);

}

}

fetchData();

**3. Describe how a callback function works in JavaScript, and give an example of a situation where callbacks might lead to "callback hell."**

* **Callback Function**:
  + A callback is a function passed as an argument to another function and is executed after the main function has completed its task.
  + Example:

function getUserData(callback) {

setTimeout(() => {

const user = { id: 1, name: 'John' };

callback(user);

}, 1000);

}

getUserData((user) => {

console.log(user); // This callback runs after getUserData finishes fetching the user data

});

* **Callback Hell**:
  + Occurs when multiple asynchronous operations are nested within one another, leading to deeply indented and hard-to-read code.
  + Example of callback hell:

getUserData((user) => {

getOrderDetails(user.id, (order) => {

getProductDetails(order.productId, (product) => {

processPayment(order, product, (confirmation) => {

console.log('Payment successful:', confirmation);

});

});

});

});

* + Callback hell makes it difficult to manage and read the code, and handling errors becomes messy.

**4. Convert the following callback-based function into one that returns a promise:**

**function fetchData(callback) {**

**setTimeout(() => {**

**const data = { id: 1, value: 'sample' };**

**callback(data);**

**}, 500);**

**}**

**Usage of the promise-based fetchData function:**

fetchData()

.then((data) => console.log(data))

.catch((error) => console.error(error));

**5. Using async/await syntax, write a function that fetches data from a simulated API. Include error handling to manage potential failures (e.g., network errors).**

async function fetchData() {

try {

// Simulating a delay with a promise

const data = await new Promise((resolve, reject) => {

setTimeout(() => {

const success = Math.random() > 0.2; // 80% chance of success

if (success) {

resolve({ id: 1, value: 'sample' });

} else {

reject('Failed to fetch data');

}

}, 1000);

});

console.log('Data:', data); // This will run if the promise is resolved

} catch (error) {

console.error('Error:', error); // This will run if the promise is rejected

}

}

fetchData();