**Callback hell code:-**

*```javascript*

*// Function simulating an asynchronous task*

*function asynchronousTask(callback) {*

*setTimeout(function() {*

*console.log("Asynchronous task completed");*

*callback();*

*}, 1000);*

*}*

*// Nested callbacks*

*asynchronousTask(function() {*

*console.log("First task completed");*

*asynchronousTask(function() {*

*console.log("Second task completed");*

*asynchronousTask(function() {*

*console.log("All tasks completed");*

*});*

*});*

*});*

*```*

In this code:

- `asynchronousTask` simulates an asynchronous operation using `setTimeout`.

- We have three instances of `asynchronousTask` called in sequence, each nested within the callback of the previous one.

- As each task completes, it triggers the next one in the sequence.

- This nesting structure creates callback hell, making the code harder to read and maintain, especially as more tasks are added.

**here's the same code using promises:**

*```javascript*

*// Function simulating an asynchronous task*

*function asynchronousTask() {*

*return new Promise(function(resolve) {*

*setTimeout(function() {*

*console.log("Asynchronous task completed");*

*resolve();*

*}, 1000);*

*});*

*}*

*// Using promises to handle asynchronous tasks sequentially*

*asynchronousTask()*

*.then(function() {*

*console.log("First task completed");*

*return asynchronousTask();*

*})*

*.then(function() {*

*console.log("Second task completed");*

*return asynchronousTask();*

*})*

*.then(function() {*

*console.log("All tasks completed");*

*})*

*.catch(function(error) {*

*console.error("An error occurred:", error);*

*});*

*```*

In this code:

- `asynchronousTask` now returns a Promise.

- The `then` method is used to chain promises, ensuring that each task is executed sequentially.

- If any promise in the chain rejects (i.e., an error occurs), it will jump to the `catch` block for error handling.

- Promises provide a cleaner and more readable way to handle asynchronous code compared to nested callbacks.

**Here's the code using `async` and `await`:**

*```javascript*

*// Function simulating an asynchronous task*

*function asynchronousTask() {*

*return new Promise(function(resolve) {*

*setTimeout(function() {*

*console.log("Asynchronous task completed");*

*resolve();*

*}, 1000);*

*});*

*}*

*// Using async/await to handle asynchronous tasks sequentially*

*async function executeTasks() {*

*try {*

*await asynchronousTask();*

*console.log("First task completed");*

*await asynchronousTask();*

*console.log("Second task completed");*

*await asynchronousTask();*

*console.log("All tasks completed");*

*} catch (error) {*

*console.error("An error occurred:", error);*

*}*

*}*

*executeTasks();*

*```*

In this code:

- The `asynchronousTask` function remains the same, returning a Promise.

- The `executeTasks` function is defined as `async`, allowing the use of `await` inside it.

- Inside `executeTasks`, `await` is used to wait for each asynchronous task to complete sequentially.

- If any error occurs in any of the awaited promises, it's caught in the `try...catch` block for error handling.