Gmail Bikash Adhikari <virtual.bikash@gmail.com>

Task 1.3: Async Task with Callback Function Create...

1 message

Bikash <bikash.caim@gmail.com> Sat, Mar 8, 2025 at 6:29 PM
To: virtual.bikash@gmail.com

JavaScript

Task 3: Async Task with Callback Function

Create a function `fetchDataWithCallback(callback)` that simulates fetching data asynchronously using setTimeout (after 2 seconds).

Once the data is “fetched”, it should invoke the provided callback function with "Fetched data" as an argument.

```
function fetchDataWithCallback(callback) {
  setTimeout(() => {
    const data = "Fetched data";
    callback(data);
  }, 2000); // Simulate 2 seconds delay
}

// Example usage:
function processData(data) {
  console.log("Data received:", data);
}

console.log("Fetching data...");
fetchDataWithCallback(processData);
console.log("Fetch initiated");
```

Explanation:

1. `fetchDataWithCallback(callback)` **Function:**
   - This function takes a `callback` function as an argument.
   - It uses `setTimeout` to simulate an asynchronous data fetching operation that takes 2 seconds.
   - Inside the `setTimeout` callback:
     - It creates a `data` variable with the value "Fetched data".
     - It calls the provided `callback` function, passing the `data` as an argument.
2. `processData(data)` **Callback Function:**
   - This is an example callback function that will be executed when the data is "fetched".
   - It takes the `data` as an argument and logs it to the console.
3. **Example Usage:**

- `console.log("Fetching data...");` is called, indicating the start of the data fetching process.
- `fetchDataWithCallback(processData);` calls the `fetchDataWithCallback` function, passing the `processData` function as the callback.
- `console.log("Fetch initiated");` is run. This demonstrates that the code does not wait for the timeout to finish, but continues execution.
- After 2 seconds, the `setTimeout` callback executes, and the `processData` function is called, logging the "Fetched data".