**Experiment Number : 2 Date: 20/01/25**

## **Develop Asynchronous Functions with Callbacks and Error Handling**

### **PRE LAB EXERCISE**

**QUESTIONS:**

1. **What is the difference between synchronous and asynchronous programming?**

**Synchronous programming** executes tasks sequentially, blocking further execution until the current task completes.

**Asynchronous programming** allows tasks to run concurrently, enabling non-blocking execution and improving efficiency, especially for I/O operations.

1. **What are callbacks in Node.js?**

Callbacks in Node.js are functions passed as arguments to other functions, executed after the completion of an asynchronous operation. They help handle non-blocking tasks like file I/O, database queries, or API calls.

**Example:**

const fs = require('fs');

fs.readFile('file.txt', 'utf8', (err, data) => {

if (err) throw err;

console.log(data); // Executes after file reading is complete

});

1. **How do you handle errors in asynchronous code?**

**Callbacks –** Pass an error as the first argument in a callback function.

**Promises (.catch)** – Use .catch() to handle errors in Promises.

**Async/Await with Try-Catch** – Wrap await calls inside try-catch blocks.

1. **What is the event loop in Node.js?**

The event loop in Node.js is a mechanism that handles asynchronous operations by continuously checking and executing tasks in different phases. It allows Node.js to be non-blocking and single-threaded, efficiently managing I/O operations.

**How it Works:**

* **Call Stack** – Executes synchronous code.
* **Task Queue** – Stores asynchronous callbacks.
* **Microtask Queue** – Handles Promises (.then(), .catch()).
* **Event Loop** – Moves tasks from the queue to the call stack when it's empty.

1. **Why is it important to use non-blocking code in Node.js?**

Using non-blocking code in Node.js is important because:

* **Efficient Resource Utilization** – Node.js runs on a single-threaded event loop, so non-blocking code prevents it from getting stuck on one task.
* **Better Scalability** – Handles multiple I/O operations (file system, database, API requests) concurrently, improving performance.
* **Faster Execution** – Non-blocking operations like fs.readFile(), fetch(), or database queries run asynchronously without waiting for the previous task to finish.
* **Improved User Experience** – Reduces delays and lag, making applications more responsive.

### **IN LAB EXERCISE**

#### **OBJECTIVE:**

* To implement asynchronous functions using callbacks.
* To handle errors in asynchronous functions.

#### **RESOURCES:**

* Node.js installed
* Any text editor

#### **PROGRAM LOGIC & IMPLEMENTATION:**

1. Create an asynchronous function to read a file.
2. Implement error handling using a callback.

### **PROGRAM: Asynchronous File Read with Callbacks**

const fs = require('fs');

function readFileAsync(fileName, callback) {

fs.readFile(fileName, 'utf8', (err, data) => {

if (err) {

callback(err, null);

} else {

callback(null, data);

}

});

}

// Using the function

readFileAsync('example.txt', (err, data) => {

if (err) {

console.log('Error reading file:', err.message);

} else {

console.log('File content:', data);

}

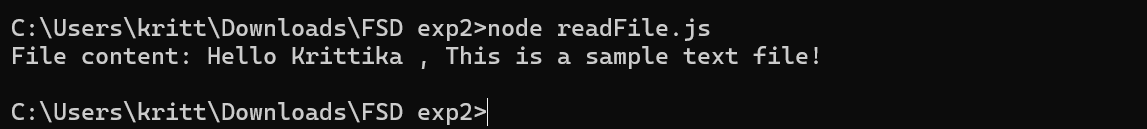
});

A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer screen

AI-generated content may be incorrect.



### **POST LAB EXERCISE**

1. **What are the advantages of asynchronous programming in Node.js?**

 Non-blocking I/O

 Improved Performance

 Better Scalability

 Faster Execution

 Efficient Resource Utilization

 Enhanced User Experience

1. **Rewrite the program using Promises instead of callbacks.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

1. **What happens if fs.readFile() is called on a file that does not exist?**

If fs.readFile() is called on a file that does not exist, an error will occur. Specifically, the error will be of type ENOENT, which stands for "Error NO ENTry" (i.e., no such file or directory).

**Behaviour:**

* **Callback-based approach:** The error is passed to the callback function as the first argument.
* **Promise-based approach:** The Promise is rejected with an ENOENT error.

1. **Explain the difference between callbacks and promises.**

**Syntax**:

**Callbacks**: A function passed as an argument and executed later.

fs.readFile('file.txt', (err, data) => { /\* handle result \*/ });

**Promises**: An object representing the result of an asynchronous operation, with .then() and .catch() for handling success and errors.

let promise = new Promise((resolve, reject) => { /\* operation \*/ });

promise.then(result => { /\* success \*/ }).catch(error => { /\* error \*/ });

**Error Handling**:

**Callbacks**: Errors are handled inside the callback.

**Promises**: Errors are handled with .catch().

**Chaining**:

**Callbacks**: Difficult to chain and prone to "callback hell."

**Promises**: Easily chainable for multiple operations.

**Multiple Operations**:

**Callbacks**: Harder to manage multiple async tasks.

**Promises**: Promise.all() simplifies handling multiple tasks.

**State**:

**Callbacks**: No built-in state management.

**Promises**: States: **pending**, **resolved**, **rejected**.

**ASSESSMENT PATTERN.**

|  |  |  |
| --- | --- | --- |
| **Description** | **Max Marks** | **Marks Awarded** |
| Pre Lab Exercise | **5** |  |
| In Lab Exercise | **10** |  |
| Post Lab Exercise | **5** |  |
| Viva | **10** |  |
| **Total** | **30** |  |
| **Faculty Signature** | |  |