JSNAD-Practice-Assignments

Chapters

Event Emitters 2

Promises 3

Buffers 4

Streams and Piping 5

fs (File System) 6

Process and OS 7

package.json Dependencies and Semver 8

Child Processes 9

Exploring Promisify 10

Jest and Assertions 11

Event Emitters

This chapter aims to teach the basics and advanced concepts of using EventEmitter in Node.js.

1 - Basic EventEmitter

To understand how to create an EventEmitter instance, register a listener, and emit an event.

Create a simple EventEmitter instance and register a listener for a custom event.

2 - Removing Event Listeners

To practice adding and removing listeners and observe how only the remaining listeners are called when the event is emitted.

Learn to remove specific event listeners.

3 - One-time Event Listeners

To ensure the one-time listener is only called once.

Use the once method to register a one-time listener.

4 - Passing Arguments to Listeners

To understand how to pass arguments to listeners.

Pass arguments to the event listeners when emitting an event.

5 - Handling Errors

To learn how to handle errors in EventEmitter and ensure the application doesn't crash.

Handle errors using the 'error' event.

6 - Wildcard Event Listener (Advanced)

To understand how to extend EventEmitter functionality and handle multiple events with a single listener.

Implement a wildcard listener that listens to all events.

7 - Event Listener with Priority (Advanced)

To learn how to manage and execute listeners based on their priority.

Implement a priority system for event listeners.

8 - Event Emitter with Asynchronous Listeners

To understand how to manage asynchronous operations within event listeners.

Handle asynchronous event listeners.

9 - Custom Event Emitter Class

To learn how to extend EventEmitter and add custom methods.

Extend the EventEmitter class to add custom functionality.

10 - Real-World Use Case - File Processing

To apply your knowledge of EventEmitter to a practical use case and see how it can be used to manage complex workflows.

Use EventEmitter for a real-world scenario, such as processing a file.

Promises

This chapter aims to provide a comprehensive understanding of promises and asynchronous programming in JavaScript.

1 - Basic Promise Creation

To understand the basic creation and resolution of promises.

Create a basic promise that resolves after 2 seconds with a message 'Hello, World!'.

2 - Handling Promise Rejection

To learn how to handle promise rejections.

Create a promise that rejects with an error message 'Something went wrong!' and handle the rejection.

3 - Chaining Promises

To practice chaining promises for sequential operations.

Chain two promises, where the first one resolves with a number and the second one adds 10 to that number.

4 - Using Promise.all

To understand how to use Promise.all to handle multiple promises.

Create three promises that resolve with different values after different intervals. Use Promise.all to wait for all of them to complete.

5 - Using Promise.race

To learn how to use Promise.race to handle the fastest promise.

Create three promises that resolve after different intervals. Use Promise.race to get the result of the first promise that resolves.

6 - Promise in a Function

To practice returning promises from functions.

Create a function that returns a promise. The promise should resolve with the square of a number after 2 seconds.

7 - Error Handling in Promises

To understand how to handle errors in promise-based operations.

Create a promise that may either resolve or reject based on a condition. Handle both the success and error cases.

8 - Nested Promises

To learn how to handle nested promises.

Create a promise that returns another promise. The inner promise should resolve with a value after 1 second, and the outer promise should resolve after the inner promise resolves.

9 - Async/Await Syntax

To understand how to use async/await for handling asynchronous operations.

Rewrite the previous assignment using async/await syntax.

10 - Sequential Promise Execution

To learn how to execute promises sequentially using reduce.

Create an array of numbers. Write a function that returns a promise to multiply each number by 2 after 1 second. Use reduce to execute these promises sequentially.

Buffers

This chapter aims to teach the fundamentals of working with buffers in Node.js.

1 - Create a Buffer

To understand how to create and initialize buffers.

Create a buffer of length 10 and fill it with zeros. Log the buffer to the console.

2 - Write to a Buffer

To learn how to write data to buffers.

Create a buffer of length 15. Write the string 'Hello' to the buffer starting at position 0. Log the buffer to the console.

3 - Read from a Buffer

To practice reading data from buffers.

Create a buffer containing the string 'Node.js'. Read and log the first 4 bytes as a string.

4 - Copy Buffers

To understand how to copy data between buffers.

Create two buffers of length 10 each. Fill the first buffer with the string 'BufferOne'. Copy the contents of the first buffer into the second buffer. Log both buffers to the console.

5 - Concatenate Buffers

To learn how to concatenate multiple buffers.

Create three buffers with the strings 'Node', '.', and 'js'. Concatenate these buffers into a single buffer. Log the resulting buffer as a string.

6 - Convert Buffer to JSON

To understand how to convert buffers to JSON.

Create a buffer with the string 'Buffer to JSON'. Convert this buffer into a JSON object. Log the

JSON object.

7 - Allocate Unsafe Buffer

To learn how to allocate and use unsafe buffers.

Allocate an unsafe buffer of length 5. Fill it with the values [1, 2, 3, 4, 5]. Log the buffer to the console.

8 - Compare Buffers

To practice comparing buffers.

Create two buffers with the strings 'BufferA' and 'BufferB'. Compare the buffers using Buffer.compare(). Log the result of the comparison.

9 - Slice a Buffer

To understand how to slice buffers.

Create a buffer with the string 'Node.js Buffers'. Slice the buffer to get the word 'Buffers'. Log the sliced buffer as a string.

10 - Encode and Decode Buffers

To learn how to encode and decode buffers.

Create a buffer from a UTF-8 encoded string 'Encoding Buffers'. Convert this buffer to a base64 encoding. Decode the base64 encoded buffer back to a string. Log the final string.

Streams and Piping

This chapter aims to teach the basics and advanced concepts of streams and piping in Node.js.

1 - Create a Readable Stream

To understand how to create and use readable streams.

Create a readable stream that emits numbers from 1 to 5. Log the data emitted by the stream.

2 - Create a Writable Stream

To learn how to create and use writable streams.

Create a writable stream that logs data written to it. Write the string 'Hello Stream' to the writable stream.

3 - Pipe Streams

To practice piping data from a readable stream to a writable stream.

Create a readable stream that emits 'Hello, World!'. Create a writable stream that logs data. Pipe the readable stream to the writable stream.

4 - Transform Stream

To understand how to create and use transform streams.

Create a transform stream that converts input text to uppercase. Pipe the string 'node.js streams' through this transform stream. Log the transformed output.

5 - Read from a File

To learn how to read data from a file using streams.

Create a readable stream from a file named input.txt. Log the data read from the file.

6 - Write to a File

To understand how to write data to a file using streams.

Create a writable stream to a file named output.txt. Write the string 'Writing to a file using streams' to this writable stream.

7 - Duplex Stream

To learn how to create and use duplex streams.

Create a duplex stream that acts as both readable and writable. For every chunk written to it, it should respond with 'Echo: ' + chunk. Test the duplex stream by writing and reading data.

8 - Pipe Multiple Streams

To practice piping data through multiple streams.

Create a readable stream from a file named input.txt. Create a transform stream that reverses the content. Create a writable stream to a file named output.txt. Pipe the readable stream through the transform stream and then to the writable stream.

9 - Handle Stream Errors

To understand how to handle errors in streams.

Create a readable stream from a non-existent file. Handle the error event to log an appropriate message.

10 - Stream Performance

To compare the performance of streams with traditional file reading methods.

Create a script to read a large file using a stream and measure the time taken. Compare the performance with reading the same file using fs.readFileSync.

fs (File System)

This chapter aims to familiarize users with basic file system operations using scripts.

1 - Reading a File

To learn how to read files and output their content to the console.

Create a script that reads the contents of a file named example.txt and logs it to the console.

2 - Writing to a File

To understand how to write data to a file.

Create a script that writes the string 'Hello, File System!' to a file named output.txt.

3 - Appending to a File

To practice appending data to an existing file.

Create a script that appends the string 'Appended content' to the output.txt file.

4 - Deleting a File

To learn how to check for file existence and delete files.

Create a script that deletes the output.txt file if it exists.

5 - Creating a Directory

To understand how to create directories programmatically.

Create a script that creates a directory named testDir.

6 - Listing Directory Contents

To explore how to list and log directory contents.

Create a script that lists the contents of the current directory and logs them to the console.

7 - Checking if a File Exists

To learn how to verify the existence of a file and handle it accordingly.

Create a script that checks if example.txt exists and logs an appropriate message.

8 - Copying a File

To practice copying files within the file system.

Create a script that copies example.txt to a new file named exampleCopy.txt.

9 - Watching a File for Changes

To understand how to monitor files for changes and handle those events.

Create a script that watches example.txt for any changes and logs a message when a change is detected.

10 - Reading and Writing JSON Files

To learn how to handle JSON files, including reading, parsing, modifying, and writing data.

Create a script that reads a JSON file named data.json, parses it, modifies a property, and writes it back to the file.

Process and OS

This chapter aims to teach the management and utilization of process and OS modules in Node.js projects.

1 - Read Environment Variables

To understand how to access environment variables in a Node.js application.

Create a script that reads and logs environment variables from the process.env object.

2 - Process Arguments

To learn how to access and use command-line arguments in Node.js.

Write a script that logs the command-line arguments passed to it using process.argv.

3 - Exit a Process

To understand how to control process exit codes in Node.js.

Create a script that exits with code 0 if an environment variable 'EXIT_NOW' is set to 'true'. Log the exit code.

4 - Process Signals

To understand how to handle process signals in Node.js.

Write a script that listens for the SIGINT signal and logs a message before exiting.

5 - Log Process Information

To learn how to gather information about the current Node.js process.

Create a script that logs current process information, including process ID, execution path, and memory usage.

6 - Read OS Information

To learn how to retrieve basic information about the operating system using Node.js.

Use the os module to log the platform, CPU architecture, and total memory of the operating system.

7 - Monitor Process Memory Usage

To understand how to monitor memory usage in a Node.js application.

Write a script that logs the memory usage of the current Node.js process at regular intervals.

8 - Process Standard Input/Output

To learn how to handle standard input and output streams in Node.js.

Create a script that reads input from stdin, processes it, and writes the result to stdout.

9 - Handle Process Exit Events

To understand how to respond to process exit events in Node.js.

Write a script that listens for the 'exit' event on the process object and logs a message before the process exits.

10 - Create a Simple OS Utility

To apply knowledge of process and OS modules in building a simple Node.js utility.

Build a script that combines information from the process and os modules to log system uptime, current user information, and network interfaces.

package.json Dependencies and Semver

This chapter aims to teach the management of dependencies and semantic versioning in Node.js projects using package.json.

1 - Create a package.json

To understand how to create and initialize a package.json file.

Create a package.json file for a project with the name 'my-project' and version '1.0.0'.

2 - Add Dependencies

To learn how to add dependencies to a Node.js project.

Add a dependency to express version '^4.17.1' using CLI. Read the package.json file to verify the addition.

3 - Add Development Dependencies

To understand how to add development dependencies to a Node.js project.

Add a development dependency to mocha version '^8.3.2' using CLI. Log the package.json file to verify the addition.

4 - Use Semver Ranges

To learn the differences between various semantic versioning ranges.

Explain the difference between '^1.0.0', '~1.0.0', and '1.0.0' in package.json.

5 - Update a Dependency

To understand how to update dependencies in a Node.js project.

Update the express dependency to version '^4.18.0'. Log the package.json file to verify the update.

6 - Remove a Dependency

To learn how to remove dependencies from a Node.js project.

Remove the mocha development dependency. Log the package.json file to verify the removal.

7 - Add Scripts

To understand how to add scripts to a package.json file.

Add a script named 'start' that runs 'node index.js'. Log the package.json file to verify the addition.

8 - Use Peer Dependencies

To learn how to add peer dependencies to a Node.js project.

Add a peer dependency for react version '^17.0.1'. Log the package.json file to verify the addition.

9 - Add Optional Dependencies

To understand how to add optional dependencies to a Node.js project.

Add an optional dependency for fsevents version '^2.3.2'. Log the package.json file to verify the addition.

10 - Handle Conflicting Versions

To learn how to handle conflicting versions of dependencies in a Node.js project.

Explain how to handle conflicting versions of dependencies using resolutions in a package.json file.

Child Processes

This chapter aims to teach how to create, configure, handle input/output, and communicate with child processes in Node.js.

1 - Create a Child Process with exec

To understand how to create a child process using exec for executing shell commands.

Use the child_process module to execute a shell command using exec and log the output.

2 - Create a Child Process with spawn

To learn how to create a child process using spawn and handle its output.

Use spawn to start a new process that runs a command-line utility (e.g., ls or dir) and log the output.

3 - Create a Child Process with fork

To explore how to create a child process using fork for Node.js scripts and communicate with it.

Create a child process using fork to run a separate Node.js script and log the messages exchanged.

4 - Handle Child Process Output

To discover different ways to handle child process output and error streams.

Write a script that spawns a process and logs the standard output and standard error streams separately.

5 - Pass Arguments to Child Process

To understand how to pass and handle command-line arguments in child processes.

Create a child process using spawn and pass command-line arguments. Log the arguments received by the child process.

6 - Set Environment Variables for Child Process

To learn how to set and access environment variables in child processes.

Spawn a child process with specific environment variables and log them from within the child process.

7 - Communicate with Child Process

To understand how to establish communication between parent and child processes using fork.

Use fork to create a child process and set up a communication channel to send and receive messages.

8 - Handle Child Process Exit

To learn how to handle child process termination and retrieve its exit code.

Spawn a child process and listen for its 'exit' event. Log the exit code and any errors.

9 - Pipe Streams Between Parent and Child Processes

To explore piping data between parent and child processes using streams.

Create a parent and child process where data is piped from the parent to the child and vice versa. Log the data at both ends.

10 - Implement a Worker Pool

To apply knowledge of child processes in building a system that handles tasks concurrently.

Create a worker pool using child processes to perform parallel tasks and aggregate the results.

Exploring Promisify

To learn how to convert callback-based functions to Promises using the promisify function in Node.js and apply them in various contexts.

1 - Introduction to Promisify

Understand how promisify works and why it's useful for modernizing asynchronous code.

Learn the basics of the promisify function by converting a simple callback-based function to a Promise-based one.

2 - Promisify a File System Operation

Apply promisify to real-world Node.js API functions that use callbacks, making them easier to work with using async/await.

Use promisify to convert the fs.readFile function to a Promise-based version and read a file asynchronously.

3 - Handling Errors with Promisify

Learn to handle errors effectively when using Promises, a crucial part of robust asynchronous code.

Promisify a callback function that might fail, and handle errors using async/await with try-catch blocks.

4 - Promisify Multiple Functions

Explore how promisify can be used to work with multiple asynchronous tasks simultaneously.

Convert multiple callback-based functions to Promises using promisify, and run them in parallel with Promise.all.

5 - Chaining Promisified Functions

Understand how to build complex asynchronous workflows by chaining promises.

Chain multiple promisified functions to perform a series of operations and return a final result.

6 - Integrating with Third-Party Libraries

Demonstrate the flexibility of promisify when working with various Node.js modules and libraries.

Use promisify to convert functions from a third-party library that relies on callbacks into Promise-based functions.

7 - Custom Promisify Function

Deepen understanding of how promisify works internally and the concept of converting callbacks to Promises.

Implement a custom version of the promisify function and use it to convert a callback function to a Promise.

8 - Promisify with Timeout

Explore advanced Promise patterns, such as handling timeouts, to enhance control over asynchronous operations.

Add a timeout to a promisified function to reject the promise if it takes too long to resolve.

9 - Combining Promisify with Streams

Demonstrate how promisify can be used to integrate with other Node.js features, like streams, for more complex scenarios.

Use promisify in conjunction with streams, such as reading a file stream asynchronously with Promise-based functions.

10 - Advanced Promisify Patterns

Gain a deeper understanding of how to use promisify in complex callback scenarios, expanding its utility in various contexts.

Explore advanced patterns such as promisifying functions with multiple callbacks, like those with error-first and data-second arguments.

Jest and Assertions

This chapter aims to teach the principles of assertions and how to use the Jest framework to write and run tests in Node.js projects.

1 - Introduction to Assertions

To understand the basic principles of assertions in Node.js.

Write a script using Node.js's built-in assert module to demonstrate basic assertions for a simple function.

2 - Run Tests with Node.js Assertions

To learn how to run tests using Node.js assertions without any additional frameworks.

Create a test script using the assert module to validate function outputs and run it directly using Node.js.

3 - Automate Test Execution with npm

To automate running tests using npm scripts and Node.js's built-in assert module.

Modify the package.json file to add a test script that uses Node.js to run the assertion-based tests.

4 - Advanced Assertions with Built-in Methods

To enhance understanding of assertions by testing more complex scenarios using built-in methods.

Explore more advanced assertions using Node.js's assert module to test complex objects and error handling.

5 - Configure npm Script for Advanced Assertions

To configure the npm package to run more complex tests written with the assert module.

Modify the package.json file to run the advanced assertion tests using an npm test script.

6 - Install Jest

To set up Jest as a test runner framework in a Node.js project.

Install the Jest test framework using npm and verify the installation by running a simple test.

7 - Write Basic Tests with Jest

To learn how to write and structure tests using the Jest framework.

Write a set of basic tests using Jest to validate the behavior of a simple function.

8 - Configure Jest in package.json

To configure Jest in a Node.js project for standardized test execution via npm.

Modify your package.json file to use Jest as the default test runner and run tests using npm scripts.

9 - Mock Functions with Jest

To understand how to use Jest's mocking capabilities for more effective unit testing.

Create mock functions using Jest to isolate and test specific parts of your code.

10 - Generate Test Coverage Reports with Jest

To learn how to leverage Jest's coverage tools to improve test completeness and quality.

Use Jest's built-in coverage feature to generate a report and identify untested parts of your code.