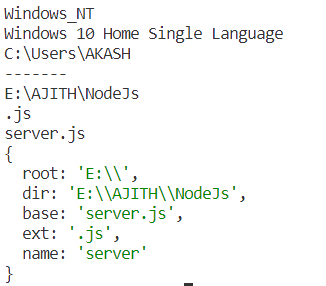
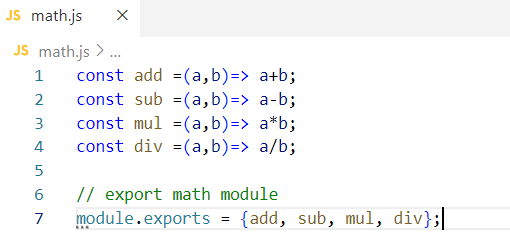
**Path Module**

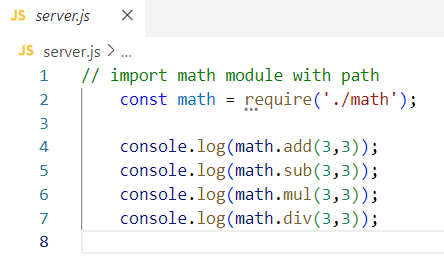
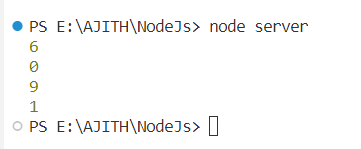


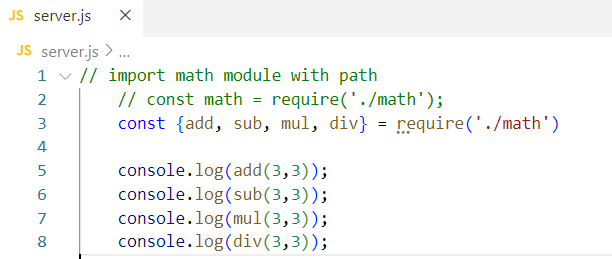
**User defined Math module**

1. Create math.js file
2. Create all maths operation function.
3. In Node.js, the **module.exports** object is used to define what a module exports and makes available for other modules to require and use.



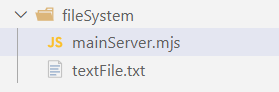
1. In server.js file
2. Import with **path**



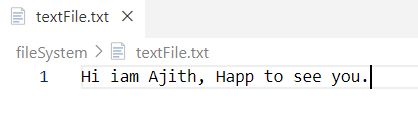
**Another way to access**

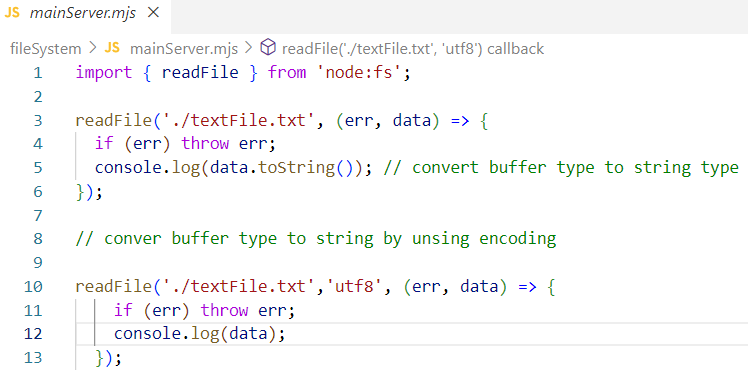
**2. File System**

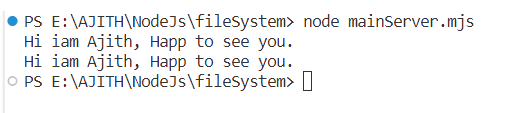
In Node.js, the **File System (fs)** module provides functionality for interacting with the file system. This module allows you to perform various operations on files and directories, such as **reading**, **writing**, **updating**, and **deleting**.



1. **Readfile**

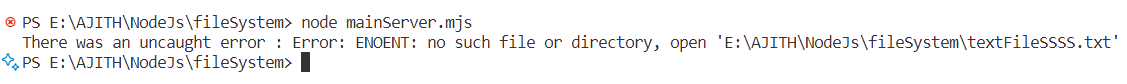




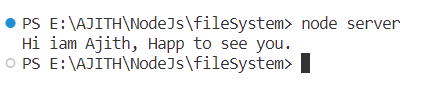


**To handle uncaught error by using inbuilt process function**



**Use Path module to provide path to readFile and server.js file**



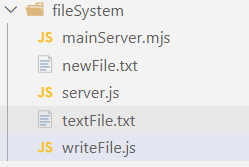


1. **WriteFile**

The **fs.writeFile()** function in Node.js is used to asynchronously write data to a file.

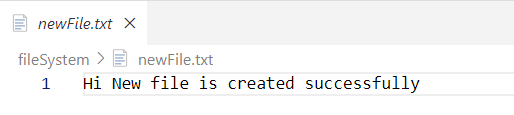
**fs.writeFile(file, data, [options], callback)**

* **file**: A string representing the file name or file descriptor where the data will be written.
* **data**: The data that will be written to the file. It can be a string or a buffer.
* **options** (optional): An object that can contain encoding information (e.g., 'utf8') and a flag for specifying file system behavior (e.g., 'w' for writing).
* **callback**: A callback function that will be called once the operation is complete. It takes one parameter, which is an error object. If there is no error, the file has been successfully written.









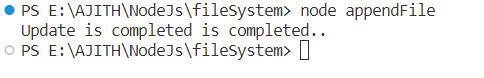
1. **appendFile**

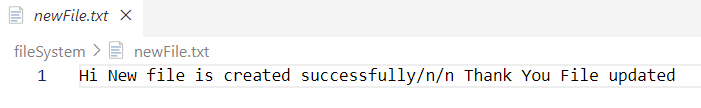
In Node.js, the **fs.appendFile()** function is used to asynchronously append data to a file.

fs.appendFile(file, data, [options], callback)

* **file**: A string representing the file name or file descriptor where the data will be appended.
* **data**: The data that will be appended to the file. It can be a string or a buffer.
* **options** (optional): An object that can contain encoding information (e.g., 'utf8').
* **callback**: A callback function that will be called once the operation is complete. It takes one parameter, which is an error object. If there is no error, the data has been successfully appended.

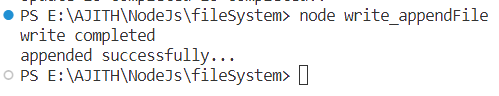


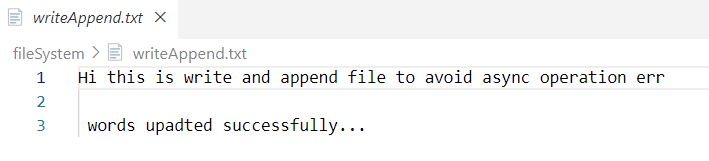




**Write and append in single file avoid async error**







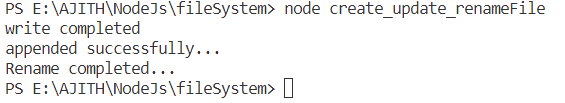
1. **Rename**

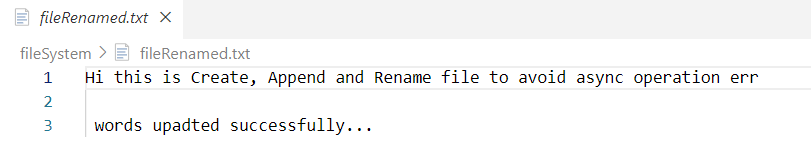
In Node.js, the **fs.rename()** function is used to asynchronously rename or move a file or directory.

**fs.rename(oldPath, newPath, callback)**

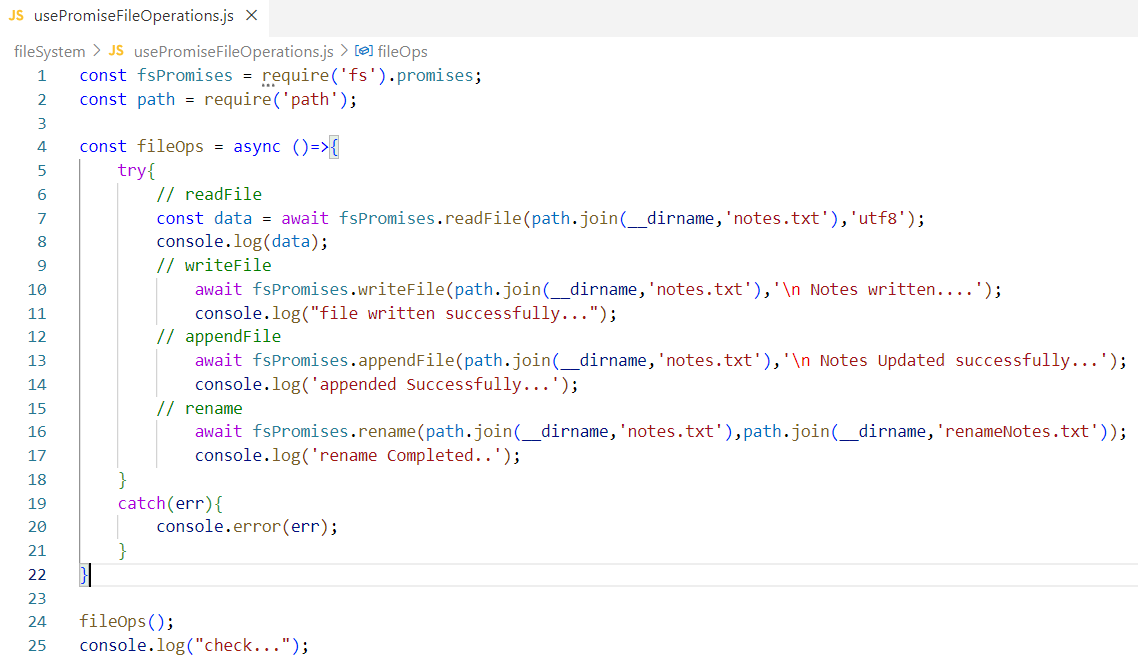
* **oldPath**: A string representing the current path or file name.
* **newPath**: A string representing the new path or file name.
* **callback**: A callback function that will be called once the operation is complete. It takes one parameter, which is an error object. If there is no error, the renaming or moving operation has been successfully completed.

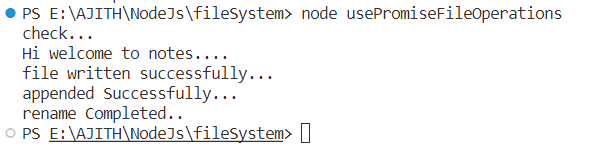


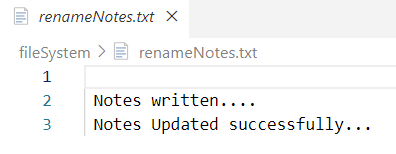




**Use promises in FileSystem**







1. **unlink(delete)**

In Node.js, the **fs.unlink()** function is used to asynchronously delete a file. It has the following signature:

**fs.unlink(path, callback);**

* **path**: A string representing the path to the file you want to delete.
* **callback**: A callback function that will be called once the deletion operation is complete. It takes one parameter, which is an error object. If there is no error, the file has been successfully deleted.



**3. Node Package Manager**

Website : <https://www.npmjs.com/>

1. **Nodemon**

**nodemon** is a tool for Node.js that helps in development by automatically restarting the Node application when file changes in the directory are detected. It is particularly useful during the development phase because it eliminates the need to manually stop and restart the Node server every time you make changes to your code.

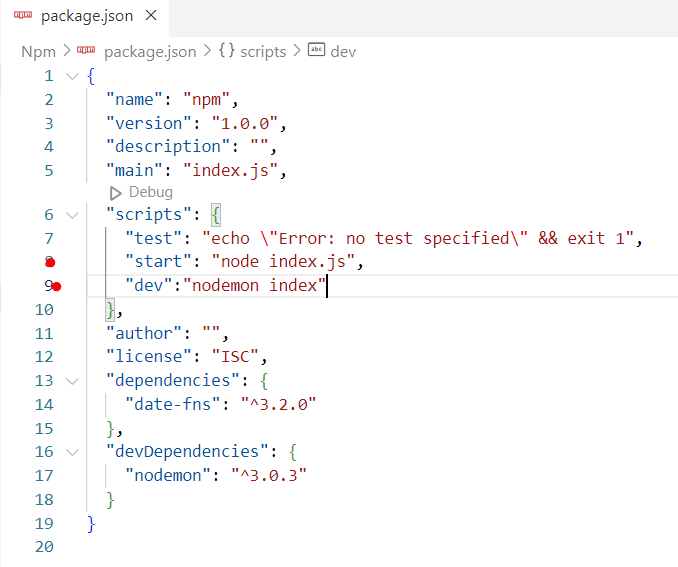
**Install comment : npm I nodemon -g**

Here are some key features and use cases for **nodemon**:

1. **Automatic Restart:**
   * The primary purpose of **nodemon** is to monitor files in your Node.js application. When it detects changes, such as modifications to your JavaScript files, it automatically restarts the Node.js application.
2. **Development Workflow:**
   * During development, developers frequently make changes to their code. Manually stopping and restarting the Node server every time you make a change can be time-consuming and disrupt the development flow. **nodemon** helps streamline this process.
3. **Configuration:**
   * **nodemon** can be configured through a **nodemon.json** file or by using command-line arguments. This allows you to customize its behavior based on your project's specific requirements.
4. **Support for Various File Types:**
   * **nodemon** can monitor various file types, not just JavaScript files. This includes configuration files, templates, and other assets that might be part of your project.
5. **Command-Line Interface (CLI):**
   * **nodemon** is run from the command line, and you can use it by replacing the **node** command with **nodemon**. For example, instead of running **node app.js**, you can use **nodemon app.js**. This makes it easy to integrate into existing projects.
6. **Integration with Build Tools:**
   * **nodemon** can be integrated into build scripts and development workflows. It is often used in conjunction with task runners like npm scripts or tools like Gulp and Grunt.

**After install of npm init-y**

The **npm i nodemon -D** command is a shorthand way of installing the **nodemon** package as a **development dependency** in a Node.js project. Let's break down the components of this command:



Now Run comment is **npm run dev**

1. **Npm init -y**

**npm init** is a command-line tool provided by npm (Node Package Manager) that allows you to initialize a new Node.js project by creating a **package.json** file. The **package.json** file is a metadata file that describes your project, its dependencies, and various configuration settings.

When you run **npm init** in your project directory, npm will interactively prompt you for information about your project and use your responses to generate a **package.json** file. This file contains essential information such as the **project's name, version, entry point (main script), test command, repository, keywords, author, and dependencies.**

Install comment : **npm init -y** y - means for all yes

1. **Date-fns npm**

**date-fns** is a JavaScript library for working with dates. It provides a set of utility functions to manipulate and format dates in a simpler and more developer-friendly way compared to the native JavaScript Date object.

**date-fns** is typically used for tasks such as:

1. **Date Formatting:**
   * **date-fns** provides functions to format dates into strings according to various formats. For example, you can format a date as "YYYY-MM-DD" or "DD/MM/YYYY".
2. **Date Manipulation:**
   * You can perform various operations on dates, such as adding or subtracting days, months, or years. This is useful for tasks like calculating future or past dates.
3. **Locale-Specific Formatting:**
   * **date-fns** supports localization, allowing you to format dates according to different locales and languages.
4. **Difference between Dates:**
   * You can calculate the difference between two dates, such as the number of days, months, or years between them.
5. **Parsing Dates:**
   * **date-fns** can parse date strings into JavaScript Date objects, making it easier to work with user input or data from external sources.

Install : **npm i date-fns**

1. **uuid npm**

Running **npm i uuid** installs the **uuid** package as a dependency in your Node.js project. UUID stands for "Universally Unique Identifier," and the **uuid** package is used for generating unique identifiers conforming to different UUID specifications.

UUIDs are often used in software development for various purposes, including:

1. **Database Records:**
   * UUIDs are commonly used as primary keys for database records, ensuring a unique identifier for each record across distributed systems.
2. **Session Management:**
   * In web applications, UUIDs can be used to generate unique session IDs for user sessions, enhancing security.
3. **Message Queues:**
   * When working with message queues or distributed systems, UUIDs help uniquely identify messages and transactions.
4. **File Naming:**
   * UUIDs can be used to generate unique file names to avoid naming conflicts.
5. **Random Token Generation:**
   * UUIDs are suitable for generating random tokens for authentication or authorization purposes.
6. **Temporary Unique Identifiers:**
   * When you need to generate temporary, unique identifiers for a specific period, UUIDs can be useful.