**Assignment 1**

**Global object # 1 Filesystem.**

The **fs (filesystem)** module in Node.js is a core module that provides an interface for working with the file system. It allows you to perform various operations on files and directories, such as reading, writing, updating, and deleting. The module includes both synchronous and asynchronous methods, giving developers flexibility in handling file-related tasks.

This module can be used:

**Reading a File.** The fs.readFile method is commonly used to read the contents of a file asynchronously. It takes the file path and an optional encoding parameter. In my assignment I demonstrate this method (See fs.js file).

**Writing to a File.** The fs.writeFile method allows to write data to a file. It takes the file path, data to be written, and an optional encoding parameter. If the file already exists, it will be overwritten. In my assignment I demonstrate this method (See writeFile.js file).

**Checking if a File or Directory Exists.** The fs.existsSync method can be used to check if a file or directory exists synchronously.

I demonstrate this method (See exist.js file).

**Global object # 2 Operating system.**

The **os (operating system)** module in Node.js provides a set of utility methods for interacting with the operating system. It allows developers to access information about the underlying operating system, such as system architecture, platform, memory usage, and various system-related parameters. This module simplifies tasks related to system information retrieval and provides a platform-independent way of obtaining OS-specific details.

One key use of the os module is to gather information about the system on which a Node.js application is running. For example, you can retrieve the current operating system platform, version, and architecture. Additionally, the os module allows you to obtain details about the system's memory usage, including total and free memory.

I demonstrate this method (See os.js file).

The OS module can provide other useful data, including:

**os.cpus().** Returns information about the system's central processing units (CPUs), including their count, model, speed, and other characteristics.

**os.networkInterfaces**(). Returns information about the system's network interfaces, including IP addresses, subnet masks, and other parameters.

**os.hostname().** Returns the hostname of the system.

**os.loadavg().** Returns an array with three numeric values representing the system's average load over the last 1, 5, and 15 minutes.

I demonstrate all this methods (See os.js file).

**Global object # 3 Console.**

The **console** global object in Node.js is a fundamental tool for developers, providing a set of methods for interacting with the standard output and error streams. It serves as a versatile mechanism for logging information, debugging, and monitoring the execution of a Node.js application. The primary methods of the console object include log(), error(), warn(), and info(), each designed for specific types of output.

The console.log() method is commonly used for general logging purposes. Developers use it to output informative messages to the console during the execution of their code.

**Console.error()** is specifically designed for displaying error messages, aiding in the identification and resolution of issues within the application.

**Console.warn()** method is utilized for issuing warnings, alerting developers to potential problems or best practice violations.

**Console.info()** method is employed for conveying general information about the application's state or operation.

**Console.dir()** method in Node.js is used to display an interactive listing of the properties of JavaScript objects. It allows you to inspect the structure of an object in a more detailed and interactive way compared to console.log().

**Console.clear()** method in Node.js is used to clear the console, removing all previously logged messages and resetting the display. When this method is called, it results in a clean console window, making it useful for improving the readability of the console output, especially in environments where logs can accumulate.

**Console.time()** method in Node.js is used to mark the beginning of a timer. It allows developers to measure the duration of a specific operation or block of code. This method takes a label as its argument, which serves as a unique identifier for the timer.

**Console.timeEnd()** method s used to stop a timer that was previously started with the console.time() method. It calculates and logs the elapsed time since the corresponding console.time() call with the specified label.

**Console.count()** method in Node.js is used to log the number of times that console.count() has been called with a specific label. It's a simple way to track and count occurrences of a particular event or section of code.

Altogether, the console global object is an indispensable ally for Node.js developers, contributing significantly to the development and debugging workflow.

I demonstrate all this methods (See console.js file).

**Assignment 2**

According to this assignment I choose axios module.

Axios is a popular JavaScript library that simplifies the process of making HTTP requests in both web browsers and Node.js environments. It is promise-based and offers a clean and consistent API for handling asynchronous operations. Axios supports various HTTP methods, including GET, POST, PUT, DELETE, and more, making it versatile for interacting with RESTful APIs and other web services.

Axios leverages the Promise API, allowing developers to handle asynchronous tasks in a more readable and maintainable way. By returning promises, Axios enables the use of .then() and .catch() to manage successful responses and error handling, providing a straightforward and intuitive approach to working with asynchronous code.

Axios provides features such as interceptors, request and response transformation, automatic JSON data parsing, and the ability to cancel requests using CancelTokens. Interceptors allow developers to execute custom logic before a request is sent or after a response is received, facilitating tasks like adding headers or logging. The library's flexibility makes it suitable for a wide range of use cases, from simple GET requests to more complex interactions with APIs.

In my example, **(index.js file)** I used to make an HTTP GET request to a public API endpoint, and the response data is logged to the console.