Week-3 - Node.js

Task-1

Aim: OS Information:
Write a program that uses the os module to display the current user's username, home directory, and operating system platform.
Create a function that utilizes the os module to display the total system memory, free memory, and the percentage of free memory available.

Description:
The Node.js OS module provides a set of methods for interacting with the operating system. It offers various functions to retrieve information about the computer, network interfaces, and operating system. Here are some commonly used methods from the `os` module:

Source Code:

```
const os = require('os');

// Display current user's information
const userInfo = os.userInfo();
console.log('Current User Information:');
console.log(`Username: ${userInfo.username}`);
console.log(`Home Directory: ${userInfo.homedir}`);
console.log(`Platform: ${os.platform()}`);
console.log();

// Function to display system memory information
function displayMemoryInfo() {
  const totalMemory = os.totalmem();
  const freeMemory = os.freemem();
  const percentageFree = ((freeMemory / totalMemory) * 100).toFixed(2);

  console.log('System Memory Information:');
  console.log(`Total Memory: ${(totalMemory / 1024 / 1024).toFixed(2)} MB`);
  console.log(`Free Memory: ${(freeMemory / 1024 / 1024).toFixed(2)} MB`);
  console.log(`Percentage Free: ${percentageFree}%`);
```

```
}
displayMemoryInfo();
```

Output:

```
PS C:\Users\qaz12\Desktop\FSWD\week-3\task-2> node C:\Users\qaz12\Des
Current User Information:
Username: qaz12
Home Directory: C:\Users\qaz12
Platform: win32

System Memory Information:
Total Memory: 8089.29 MB
Free Memory: 989.24 MB
Percentage Free: 12.23%
PS C:\Users\qaz12\Desktop\FSWD\week-3\task-2>
```

Theoretical Background:

1. os.arch(): Returns the CPU architecture of the operating system (e.g., "x64", "arm", "ia32").
2. os.cpus(): Returns an array of objects containing information about each CPU/core, including model, speed, and times (user, nice, sys, idle, irq).
3. os.totalmem(): Returns the total system memory in bytes.
4. os.freemem(): Returns the amount of free system memory in bytes.
5. os.hostname(): Returns the hostname of the operating system.
6. os.networkInterfaces(): Returns an object containing information about the network interfaces on the system, including IP addresses, MAC addresses, and family (IPv4 or IPv6).
7. os.platform(): Returns the operating system platform (e.g., "win32", "linux", "darwin").
8. os.release(): Returns the operating system release.
9. os.tmpdir(): Returns the operating system's default directory for temporary files.
10. os.type(): Returns the operating system name (e.g., "Windows_NT", "Linux", "Darwin").
11. os.uptime(): Returns the system uptime in seconds.
12. These are just a few examples of the available methods in the os module. You can refer to the official Node.js documentation for a complete list of methods and their descriptions.

Task-2

Aim : Experiment with chalk,upper-case any other External Modules

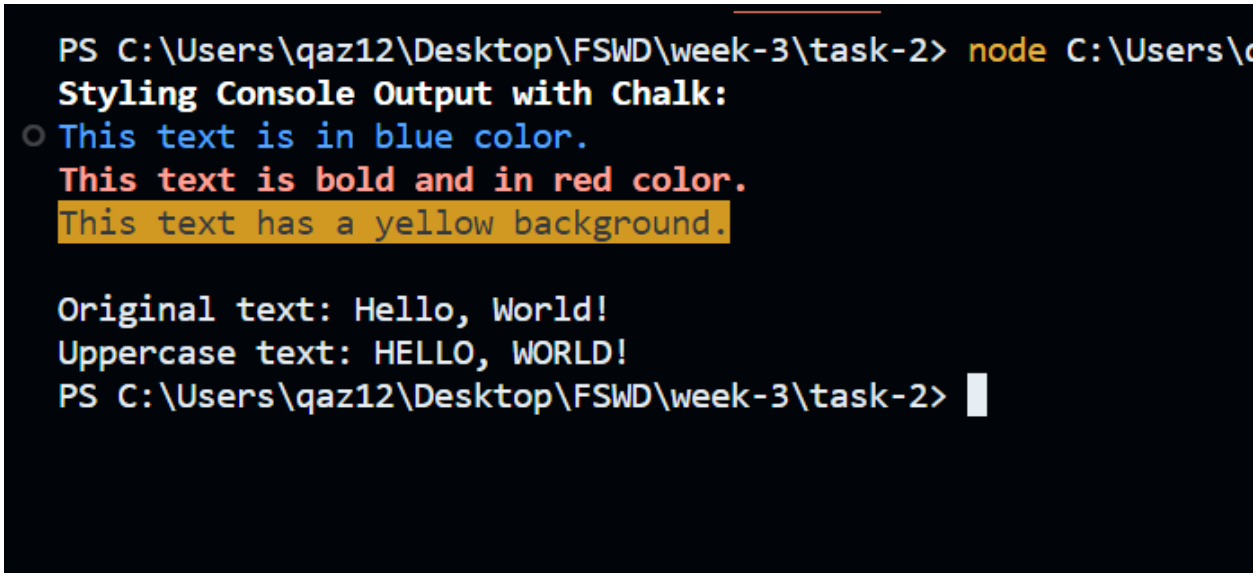
Theory:-

These modules, chalk and upper-case, provide convenient functionality for enhancing the appearance of console output and transforming text to uppercase, respectively. By utilizing these modules, developers can achieve better visual presentation and improved string handling in their JavaScript projects.

Source Code:

```
const chalk = require('chalk');
const upperCase = require('upper-case');

// Styling console output using chalk
console.log(chalk.bold('Styling Console Output with Chalk:'));
console.log(chalk.blue('This text is in blue color.'));
console.log(chalk.red.bold('This text is bold and in red color.'));
console.log(chalk.yellow.inverse('This text has a yellow background.'));

console.log(); // Empty line for separation

// Using upper-case module to convert text to uppercase
const text = 'Hello, World!';
console.log(`Original text: ${text}`);
console.log(`Uppercase text: ${upperCase.upperCase(text)}`);
```

Output:



Theoretical Background:

Chalk is a popular JavaScript library used for styling command-line output with colors and formatting in Node.js applications.
It provides an easy and convenient way to add colors, styles (bold, italic, underline), and backgrounds to the text displayed in the console.
Chalk works by chaining its methods to the desired text, allowing you to apply different styles in a fluent manner.
The upper-case module is a simple JavaScript library that provides a function to convert text to uppercase.
It is particularly useful when you need to transform strings to uppercase in a convenient and reliable way.
The upper-case module handles various language-specific uppercase conversions correctly, taking into account locale-specific rules.
It supports Unicode characters and can handle strings containing multi-byte characters correctly.

Task-3

Aim: Create your own custom module and import/export it to the main module.

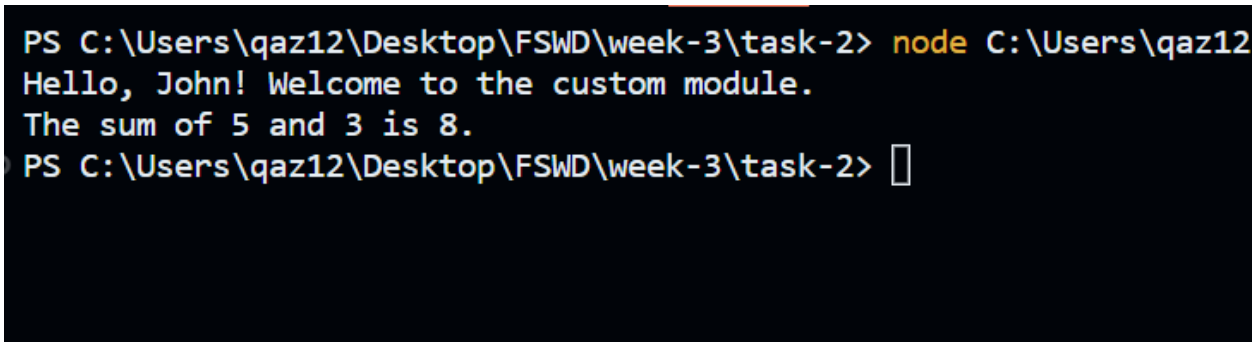
Source Code:

```
// customModule.js

// Function to greet a person
function greetPerson(name) {
    console.log(`Hello, ${name}! Welcome to the custom module.`);
  }

  // Function to add two numbers
  function addNumbers(a, b) {
    return a + b;
  }

  // Export the functions to make them accessible from other modules
  module.exports = {
    greetPerson,
    addNumbers,
  };
```

```
// main.js

// Import the custom module
const customModule = require('./customModule');

// Call the greetPerson function from the custom module
customModule.greetPerson('John');

// Call the addNumbers function from the custom module
const result = customModule.addNumbers(5, 3);
console.log(`The sum of 5 and 3 is ${result}.`);
```

Output:-

=> Using Browser



Theoretical Background:

In JavaScript, a module is a self-contained unit of code that encapsulates related functionality. It allows you to organize and structure your code into reusable and maintainable components.
A custom module is a module that you create to encapsulate specific functionality or a set of related functions.

Benefits of using custom modules:

Modularity: Custom modules promote modularity by breaking down your code into smaller, manageable units. This makes it easier to understand, maintain, and reuse code across different parts of your application.
Encapsulation: Custom modules encapsulate functionality, keeping the implementation details hidden and providing a clean interface to interact with the module.
Reusability: By creating custom modules, you can write code once and reuse it in multiple parts of your application or even in different projects.

Learning Outcome:-

CO1 : Understand various technologies and trends impacting single page web applications.
CO4 : Demonstrate the use of JavaScript to fulfill the essentials of front-end development To back-end development