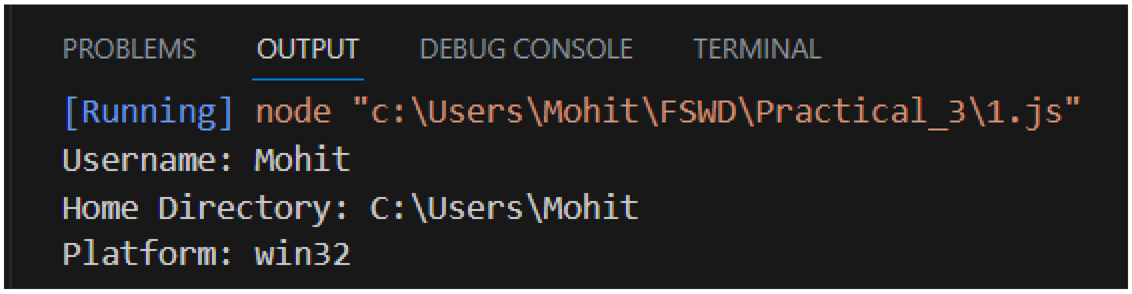
Task-1

Aim:
Write a program that uses the os modules to display the current user's username, home directory, and operating system platform.

Source Code:

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

// Get the current user's username
const username = os.userInfo().username;

// Get the home directory of the current user
const homeDirectory = os.homedir();

// Get the operating system platform
const platform = os.platform();

// Display the information
console.log('Username:', username);
console.log('Home Directory:', homeDirectory);
console.log('Platform:', platform);
```

Output:



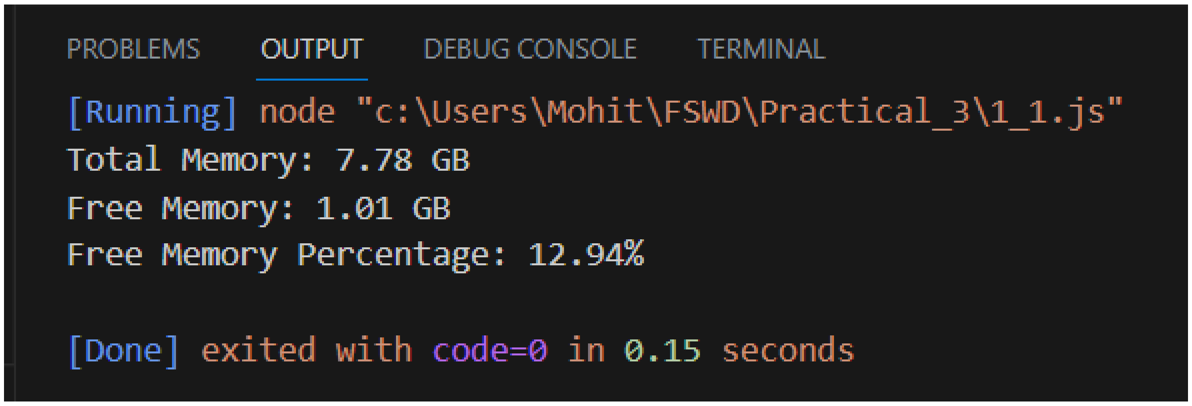
```
[Running] node "c:\Users\Mohit\FSWD\Practical_3\1.js"
Username: Mohit
Home Directory: C:\Users\Mohit
Platform: win32
```

Aim:
Create a function that utilizes the os module to display the total system memory, free memory, and the percentage of free memory available

Source Code:

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

function displayMemoryInfo() {
  // Get the total system memory
  const totalMemory = os.totalmem();

  // Get the free memory
  const freeMemory = os.freemem();

  // Calculate the percentage of free memory
  const freeMemoryPercentage = (freeMemory / totalMemory) * 100;

  // Display the information
  console.log('Total Memory:', formatBytes(totalMemory));
  console.log('Free Memory:', formatBytes(freeMemory));
  console.log('Free Memory Percentage:', freeMemoryPercentage.toFixed(2) + '%');
}

// Helper function to format bytes to human-readable format
function formatBytes(bytes) {
  const units = ['B', 'KB', 'MB', 'GB', 'TB'];
  let i = 0;
  while (bytes >= 1024 && i < units.length - 1) {
    bytes /= 1024;
    i++;
  }
  return bytes.toFixed(2) + ' ' + units[i];
}

// Call the function to display memory information
displayMemoryInfo();
```

Output:



```
[Running] node "c:\Users\Mohit\FSWD\Practical_3\1_1.js"
Total Memory: 7.78 GB
Free Memory: 1.01 GB
Free Memory Percentage: 12.94%

[Done] exited with code=0 in 0.15 seconds
```

Theoretical Background:

We use OS modules in Node.js as it provides functions and properties to access information about the operating system on which your Node.js application is running. It allows you to gather system information, such as memory, CPU, and network details. It also helps in managing resources, optimizing performance, and making decisions based on available system resources. The os module abstracts away operating system differences, making your code work consistently across platforms. Additionally, it provides user-specific information and functions for interacting with the operating system. Overall, the os module is useful for system-level programming, cross-platform development, and accessing operating system-related information in your Node.js application.

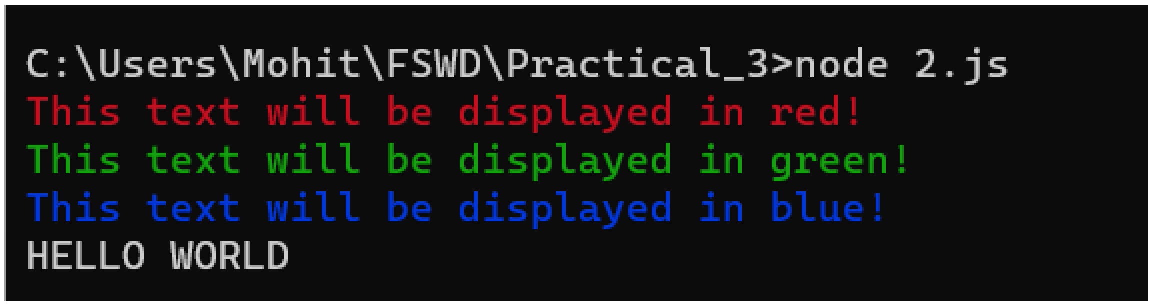
Task-2

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

Source Code:

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

console.log(chalk.red('This text will be displayed in red!'));
console.log(chalk.green('This text will be displayed in green!'));
console.log(chalk.blue('This text will be displayed in blue!'));

const text = 'hello, world!';
const uppercaseText = upperCase(text);

console.log(uppercaseText);
```

Output:



```
C:\Users\Mohit\FSWD\Practical_3>node 2.js
This text will be displayed in red!
This text will be displayed in green!
This text will be displayed in blue!
HELLO WORLD
```

Theoretical Background:

uppercase module: The uppercase module is used to convert text to uppercase. It provides the upperCase() function, which takes a string as input and returns the uppercase version of the string. This module can be helpful when you need to convert text to uppercase for various purposes, such as formatting, comparison, or display.

chalk module: The chalk module is used to add color and styling to console output. It provides various functions that allow you to apply different colors, background colors, and text formatting options to your text. This module can be useful when you want to enhance the visual appearance of your console logs or messages.

Task-3

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

Source Code:

tutorial.js

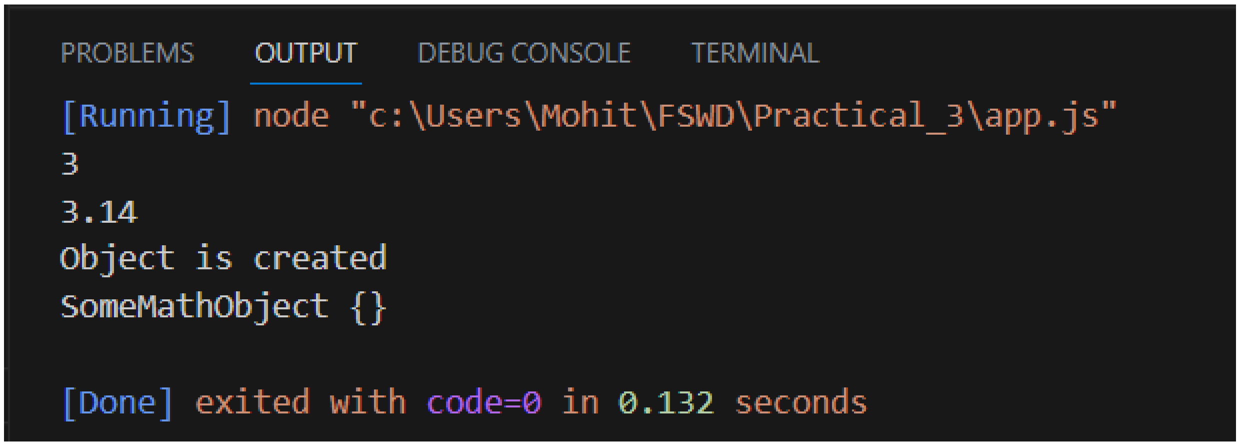
```
const sum = (num1,num2) => num1+num2;
const PI = 3.14;
class SomeMathObject{
    constructor(){
        console.log('Object is created')
    }
}

module.exports = {sum : sum, PI : PI, SomeMathObject: SomeMathObject}
```

Now we will import tutorial.js in app.js

app.js

```
const tutorial = require('./tutorial')
console.log(tutorial.sum(1,2))
console.log(tutorial.PI)
console.log(new tutorial.SomeMathObject())
```

Output :



```
[Running] node "c:\Users\Mohit\FSWD\Practical_3\app.js"
3
3.14
Object is created
SomeMathObject {}

[Done] exited with code=0 in 0.132 seconds
```

Theoretical Background :

Custom modules in Node.js are used to organize and encapsulate related functionality into reusable units of code. They allow you to create your own modules that can be imported and used in other parts of your application. By using custom modules, you can break down your code into smaller, manageable pieces and promote code reusability.
The module.exports statement is used to define the public interface of a custom module. It specifies which functions, objects, or variables should be accessible to other modules when they import the custom module. This way, you can control what parts of your module are exposed and what remains private.

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