Task-1

Aim: custom Module

Create a Node.js module for basic arithmetic operations and demonstrate its utilization in a main script.

Description:

a custom module is a separate JavaScript file that encapsulates a set of related functionalities. The goal is to organize and modularize your code, making it easier to manage, maintain, and reuse.

Here are the key components and concepts related to custom modules in Node.js:

1) Module Definition 2)Module Export 3)Module import 4)Path to Modules 5)Encapsulation 6) Reusability

Source Code:

```
const mathOperations = require('./mathOperations.js');

console.log(`Addition : ${mathOperations.add(5, 3)}`);
console.log(`Subtraction: ${mathOperations.subtract(8, 2)}`);
console.log(`Multiplication : ${mathOperations.multiply(4, 6)}`);
console.log(`Divison: ${mathOperations.divide(9, 3)}`);
```

Output:

```
PS C:\SEM-4\FSWD\WEEK-3\task-1> node task-1
Addition : 8
Subtraction: 6
Multiplication : 24
Divison: 3
```

Task-2

Aim: fs Module (File System)

Create a program that reads a directory and lists all files within it, displaying their names, sizes, and file types.

Implement functionalities to copy a file from one location to another, rename a file, and delete a file.

Provide an option to create a new directory and move specific files into it.

Description:

The fs module is not a standard module in Node.js, but it's commonly used as an abbreviation for the "file system" module, which is an integral part of Node.js core modules. The official name of the module is fs, which stands for "file system." It provides an API for interacting with the file system in a Node.js environment, allowing you to perform various file operations such as reading, writing, deleting, and manipulating files and directories.

Source Code:

```
const fs = require('fs');
const path = require('path');

// Reading a directory
const directoryPath = './Example';
fs.readdir(directoryPath, (err, files) => {
  if (err) {
    console.error('Error reading directory:', err);
    return;
  }

  // Displaying file details
  files.forEach((file) => {
    const filePath = path.join(directoryPath, file);
    const stats = fs.statSync(filePath);
    console.log(`${file} - Size: ${stats.size} bytes, Type: ${stats.isFile() ?
'File' : 'Directory'}`);
  });

  // Copying a file
  const sourceFile = path.join(directoryPath, 'example.txt');
  const destinationFile = path.join(directoryPath, 'example-copy.txt');
  fs.copyFileSync(sourceFile, destinationFile);
  console.log('File copied successfully.');

  // Renaming a file
  const oldFileName = path.join(directoryPath, 'example-copy.txt');
  const newFileName = path.join(directoryPath, 'renamed-file.txt');
  fs.renameSync(oldFileName, newFileName);
  console.log('File renamed successfully.');

  // Deleting a file
  const fileToDelete = path.join(directoryPath, 'renamed-file.txt');
```

```
  fs.unlinkSync(fileToDelete);
  console.log('File deleted successfully.');

  // Creating a new directory and moving files into it
  const newDirectoryName = path.join(directoryPath, 'new-directory');
  fs.mkdirSync(newDirectoryName);
   const filesToMove = ['index.html', 'style.css'];
   filesToMove.forEach((file) => {
     const sourcePath = path.join(directoryPath, file);
     const destinationPath = path.join(newDirectoryName, file);
     fs.renameSync(sourcePath, destinationPath);
   });
   console.log('Files moved to the new directory.');
});
```

Output:

```
PS C:\SEM-4\FSWD\WEEK-3\task-2> node task-2
example.txt - Size: 26 bytes, Type: File
index.html - Size: 0 bytes, Type: File
style.css - Size: 0 bytes, Type: File
File copied successfully.
File renamed successfully.
File deleted successfully.
Files moved to the new directory.
```

Task-3

Aim: path Module (Path Handling)

Design a tool that accepts a file path as input and extracts information like the file extension, filename, and directory path.

Normalize a given path and resolve a path by joining different segments.

Validate whether a path exists and whether it points to a file or a directory.

Description:

The path module is a core module in Node.js that provides utilities for working with file and directory paths. It helps in constructing, normalizing, and resolving file and directory paths in a platform-independent manner. This module is particularly useful for dealing with file paths in a

way that is compatible across different operating systems, as the path formats can vary (e.g., backslashes in Windows vs. forward slashes in Unix-like systems).

Source Code:

```
const path = require('path')
const fs = require('fs')
const filepath = 'C:/SEM-4/FSWD/WEEK-3/TASK-2/Example/example.txt';
console.log('File Extension:', path.extname(filepath));
console.log('File Name:', path.basename(filepath));
console.log('Directory Path:', path.dirname(filepath));

const normalizedPath = path.normalize(filepath);
console.log('Normalized Path:', normalizedPath);
const resolvedPath = path.resolve(__dirname, 'files', 'example.txt');
console.log('Resolved Path:', resolvedPath);

console.log('Path Exists:', fs.existsSync(filepath));
 console.log('Is a Directory:', fs.statSync(filepath).isDirectory());
 console.log('Is a File:', fs.statSync(filepath).isFile());
```

Output:

```
PS C:\SEM-4\FSWD\WEEK-3\task-3> node task-3
File Extension: .txt
File Name: example.txt
Directory Path: C:/SEM-4/FSWD/WEEK-3/TASK-2/Example
Normalized Path: C:\SEM-4\FSWD\WEEK-3\TASK-2\Example\example.txt
Resolved Path: C:\SEM-4\FSWD\WEEK-3\task-3\files\example.txt
Path Exists: true
Is a Directory: false
Is a File: true
```

Task-4

Aim: url Module (URL Handling)

Create a program that parses a given URL into its components such as protocol, host, pathname, and query parameters.

Build a URL by combining its components (protocol, hostname, pathname, query) into a valid URL string.

Validate URLs and check if they are well-formed.

Description:

The url module in Node.js provides utilities for URL parsing and formatting. It is used to work with URLs (Uniform Resource Locators), allowing you to parse URL strings, construct URLs from components, and manipulate URL components.

Source Code:

```
const url = require('url')
// Parsing a URL
const urlstring = "https://mail.google.com/mail/u/0/?tab=rm&ogbl#inbox";
const parsedUrl = url.parse(urlstring, true);
console.log('Protocol:', parsedUrl.protocol);
console.log('Host:', parsedUrl.host);
console.log('Pathname:', parsedUrl.pathname);
console.log('Query Parameters:', parsedUrl.query);

// Building a URL
const builtUrl = url.format({
    protocol: 'https',
    host: 'www.example.com',
    pathname: '/newpath',
    query: { param1: 'value1', param2: 'value2' },
  });
  console.log('Built URL:', builtUrl);

  // Validating a URL
const isValidUrl = url.parse(urlstring).protocol !== null;
console.log('Is Valid URL:', isValidUrl);
```

Output:

```
PS C:\SEM-4\FSWD\WEEK-3\task-4> node task-4
Protocol: https:
Host: mail.google.com
Pathname: /mail/u/0/
Query Parameters: [Object: null prototype] { tab: 'rm', ogbl: '' }
Built URL: https://www.example.com/newpath?param1=value1&param2=value2
Is Valid URL: true
```

Task-5

Aim: util Module (Utilities)

Develop a script that uses the util module to create custom error objects with defined error codes and messages.
Utilize the util.inspect() function to log objects and nested structures for debugging purposes.
Demonstrate error handling techniques using the util.promisify() method for callback-based functions.

Description:

The util module in Node.js provides a set of utility functions that are commonly used in various applications. It includes functions that simplify common programming tasks, such as formatting and inspecting objects, working with asynchronous code, and more.

Source Code:

```
const util = require('util');
const fs = require('fs');

// Custom error objects
class CustomError extends Error {
  constructor(code, message) {
    super(message);
    this.code = code;
  }
}
  const myError = new CustomError(404,"Page Not Found");
  console.log('Custom Error:', myError);
// Logging objects with util.inspect()
const objToInspect = { name: 'John', age: 30, address: { city: 'Example City',
zip: '12345' } };
console.log('Inspected Object:', util.inspect(objToInspect, { depth: null }));

// Error handling with util.promisify()
const readFileAsync = util.promisify(fs.readFile);

async function readAndHandleFile(filePath) {
  try {
    const content = await readFileAsync(filePath, 'utf8');
    console.log('File Content:', content);
  } catch (error) {
    if (error.code === 'ENOENT') {
```

```
      console.error('File not found:', error.message);
    } else {
      console.error('Error reading file:', error);
    }
  }
}
// Example usage
readAndHandleFile('./example.txt');
```

Output:

```
PS C:\SEM-4\FSWD\WEEK-3\task-5> node task-5
Custom Error: CustomError: Page Not Found
    at Object.<anonymous> (C:\SEM-4\FSWD\WEEK-3\task-5\task-5.js:11:19)
    at Module._compile (node:internal/modules/cjs/loader:1376:14)
    at Module._extensions..js (node:internal/modules/cjs/loader:1435:10)
    at Module.load (node:internal/modules/cjs/loader:1207:32)
    at Module._load (node:internal/modules/cjs/loader:1023:12)
    at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:135:12)
    at node:internal/main/run_main_module:28:49 {
  code: 404
}
Inspected Object: {
  name: 'John',
  age: 30,
  address: { city: 'Example City', zip: '12345' }
}
File Content: Hello my name is Trushang
```

Task-6

Aim: buffer Module (Binary Data)

Build a program that reads a text file and converts its content into a buffer.
Manipulate the buffer (e.g., split, slice, concatenate) and convert it back to text.
Demonstrate encoding and decoding functionalities (e.g., UTF-8, Base64) with buffers.

Description:

The buffer module in Node.js provides a way to work with binary data directly, without having to first convert it to a string. It is a part of the core modules in Node.js and is commonly used in scenarios where handling binary data, such as reading from or writing to files, dealing with network protocols, or manipulating raw binary data, is necessary.

Source Code:

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

// Reading a text file and converting its content to a buffer
const textContent = 'Hello, this is a text file.';
```

```
const filePath = './textFile.txt';
fs.writeFileSync(filePath, textContent);
const buffer = fs.readFileSync(filePath);
console.log("Buffer: ",buffer)
console.log('Buffer in utf8:', buffer.toString('utf8'));

// Manipulating the buffer
const slicedBuffer = buffer.slice(0, 5);
console.log('Sliced Buffer:', slicedBuffer.toString());

const concatenatedBuffer = Buffer.concat([buffer, Buffer.from(' Appended
text.')]);
console.log('Concatenated Buffer:', concatenatedBuffer.toString());

// Encoding and Decoding with buffers
const utf8Encoded = buffer.toString('utf8');
console.log('UTF-8 Encoded:', utf8Encoded);

const base64Encoded = buffer.toString('base64');
console.log('Base64 Encoded:', base64Encoded);

const base64Decoded = Buffer.from(base64Encoded, 'base64');
console.log('Base64 Decoded:', base64Decoded.toString('utf8'));
```

Output:

```
PS C:\SEM-4\FSWD\WEEK-3\task-6> node task-6
Buffer:  <Buffer 48 65 6c 6c 6f 2c 20 74 68 69 73 20 69 73 20 61 20 74 65 78 74 20 66 69 6c 65 2e>
Buffer in utf8: Hello, this is a text file.
Sliced Buffer: Hello
Concatenated Buffer: Hello, this is a text file. Appended text.
UTF-8 Encoded: Hello, this is a text file.
Base64 Encoded: SGVsbG8sIHRoaXMgaXMgYSB0ZXh0IGZpbGUu
Base64 Decoded: Hello, this is a text file.
```

Task-7

Aim: os Module (Operating System)

Create a Node.js script that fetches and displays system information such as CPU architecture, total memory, operating system platform, and uptime.

Implement a function that monitors system resources (CPU, memory) and logs their usage periodically during script execution.

Description:

The os module in Node.js provides a set of utility functions to interact with the operating system. It allows you to access information about the operating system, such as platform, architecture, and network interfaces. The module is part of the core Node.js modules and is available for use without the need for additional installations.

Source Code:

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

console.log(`Hostname : ${os.hostname}`)
console.log(`CPU Architecture : ${os.arch()}`);
console.log(`Total Memory(GB) : ${os.totalmem/(1024*1024*1024)}`)
console.log(`Operating System Platform: ${os.platform}`)
console.log(`Uptime(second): ${os.uptime}`)
console.log(`Operating System type:${os.type}`)
//console.log(`CPU : ${os.cpus}`)

setInterval(() => {
    console.log('CPU Usage:', os.cpus());
    console.log('Free Memory (bytes):', os.freemem());
  }, 5000);
```

Output:

```
PS C:\SEM-4\FSWD\WEEK-3\task-7> node task-7
Hostname : LAPTOP-M24R3J6J
CPU Architecture : ia32
Total Memory(GB) : 7.6790618896484375
Operating System Platform: win32
Uptime(second): 245129.109
Operating System type:Windows_NT
CPU Usage: [
  {
    model: '12th Gen Intel(R) Core(TM) i5-1235U',
    speed: 2496,
    times: {
      user: 1979750,
      nice: 0,
      sys: 4222062,
      idle: 46621640,
      irq: 300281
    }
  },
  {
    model: '12th Gen Intel(R) Core(TM) i5-1235U',
    speed: 2496,
    times: {
      user: 1086000,
      nice: 0,
      sys: 1317453,
      idle: 50419375,
      irq: 53437
    }
  },
  {
```

```
  {
    model: '12th Gen Intel(R) Core(TM) i5-1235U',
    speed: 2496,
    times: {
      user: 2267437,
      nice: 0,
      sys: 3772125,
      idle: 46783312,
      irq: 182265
    }
  },
  {
    model: '12th Gen Intel(R) Core(TM) i5-1235U',
    speed: 2496,
    times: {
      user: 1155437,
      nice: 0,
      sys: 1378125,
      idle: 50289312,
      irq: 35203
    }
  },
  {
    model: '12th Gen Intel(R) Core(TM) i5-1235U',
    speed: 2496,
    times: { user: 204328, nice: 0, sys: 226640, idle: 52391921, irq: 7437 }
  },
  {
    model: '12th Gen Intel(R) Core(TM) i5-1235U',
    speed: 2496,
    times: { user: 200906, nice: 0, sys: 199218, idle: 52422703, irq: 5562 }
  },
```

```
  {
    model: '12th Gen Intel(R) Core(TM) i5-1235U',
    speed: 2496,
    times: { user: 208421, nice: 0, sys: 219562, idle: 52394812, irq: 8343 }
  },
  {
    model: '12th Gen Intel(R) Core(TM) i5-1235U',
    speed: 2496,
    times: { user: 199421, nice: 0, sys: 237109, idle: 52386312, irq: 6593 }
  },
  {
    model: '12th Gen Intel(R) Core(TM) i5-1235U',
    speed: 2496,
    times: { user: 202093, nice: 0, sys: 244062, idle: 52376703, irq: 7531 }
  },
  {
    model: '12th Gen Intel(R) Core(TM) i5-1235U',
    speed: 2496,
    times: { user: 200500, nice: 0, sys: 231703, idle: 52390625, irq: 6234 }
  },
  {
    model: '12th Gen Intel(R) Core(TM) i5-1235U',
    speed: 2496,
    times: { user: 194125, nice: 0, sys: 231359, idle: 52397328, irq: 5671 }
  },
  {
    model: '12th Gen Intel(R) Core(TM) i5-1235U',
    speed: 2496,
    times: { user: 200406, nice: 0, sys: 246890, idle: 52375562, irq: 5296 }
  }
]
Free Memory (bytes): 1655717888
```

Task-8

Aim: HTTP Module (HTTP Server)

Set up an HTTP server that listens on a specified port 8080 and handles GET and POST requests. Create endpoints for handling GET requests to retrieve data and POST requests to update data. Implement error handling for different status codes and invalid routes.

Description:

The http module in Node.js is a core module that allows you to create and run HTTP servers, handle HTTP requests, and send HTTP responses. It provides the fundamental building blocks for creating web applications and handling network communication over the HTTP protocol.

Source Code:

```
const http = require('http');
const url = require('url');
const querystring = require('querystring');

// Sample data
let data = {
  name: 'John Doe',
  age: 25,
};

// Creating an HTTP server
const server = http.createServer((req, res) => {
  const parsedUrl = url.parse(req.url, true);
  const pathname = parsedUrl.pathname;

  if (req.method === 'GET') {
    if (pathname === '/getData') {
      res.writeHead(200, { 'Content-Type': 'application/json' });
      res.end(JSON.stringify(data));
    } else {
      res.writeHead(404, { 'Content-Type': 'text/plain' });
      res.end('Not Found');
    }
  } else if (req.method === 'POST') {
    if (pathname === '/updateData') {
      let body = '';

      req.on('data', (chunk) => {
```

```
        body += chunk.toString();
      });
      req.on('end', () => {
        const postData = querystring.parse(body);
        data = { ...data, ...postData };
        res.writeHead(200, { 'Content-Type': 'application/json' });
        res.end(JSON.stringify(data));
      });
    } else {
      res.writeHead(404, { 'Content-Type': 'text/plain' });
      res.end('Not Found');
    }
  } else {
    res.writeHead(405, { 'Content-Type': 'text/plain' });
    res.end('Method Not Allowed');
  }
});
// Listening on port 8080
const PORT = 8080;
server.listen(PORT, () => {
  console.log(`Server listening on port ${PORT}`);
});
```

Output:

```
PS C:\SEM-4\FSWD\WEEK-3\task-8> node task-8
Server listening on port 8080
```

Learning Outcome:

Overall, the provided code examples cover various important aspects of web development, including custom Module, fs Module, path Module (Path Handling), url Module (URL Handling), util Module (Utilities), buffer Module (Binary Data), os Module (Operating System) and HTTP Module (HTTP Server)