SHREE H.N.SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University and G.T.U)

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Lt. Shree Chimanbhai Shukla

MScIT SEM-3 :: CS-14: Node.js

Shree H.N.Shukla College
Street No. 2, Vaishali Nagar,
Nr. Amrapali Under Bridge,
Raiya Road, Rajkot.
Ph. (0281)2440478, 2472590

Shree H.N.Shukla College
Street No. 3, Vaishali Nagar,
Nr. Amrapali Under Bridge,
Raiya Road, Rajkot.
Ph. (0281)2471645

Shree H.N.Shukla College of I.T & Management "Sky is the Limit"

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

CS – 14: NODE JS

MSc(IT) SEMESTER : 03 :: CS-14: Node.js		
No	**Topics**	**Details**
2.	**Node.js Basic, Node.js Modules, Node Package Manager (NPM)**	• Primitive Types • Object Literal, Functions, Buffer, Access Global Scope • Module, Module Types: Core Modules, Local Modules, Third Party Modules, Module Exports. • Using Modules in a Node.js File • Using the Built in HTTP, URL, Query String Module • Creating a Custom Module • NPM, Installing Packages Locally • Adding dependency in package.json • Installing packages globally Updating package

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Primitive Types

What are the primitive types in JavaScript, and how are they used in Node.js?

Primitive types in JavaScript include:

1. **String**
2. **Number**
3. **Boolean**
4. **Undefined**
5. **Null**
6. **Symbol** (introduced in ES6)
7. **BigInt** (introduced in ES2020)

Examples:

```
// String
let name = "John Doe";

// Number
let age = 30;

// Boolean
let isStudent = false;
```

```
// Undefined
let car;

// Null
let job = null;

// Symbol
let sym = Symbol('unique');
```

```
// BigInt let bigNumber = BigInt(9007199254740991);


// Usage in Node.js
console.log(`Name: ${name}, Age: ${age}, Is Student: ${isStudent}, Car: ${car}, Job: ${job}, Symbol: ${sym.toString()}, Big Number: ${bigNumber}`);
```

OUTPUT:

Name: John Doe, Age: 30, Is Student: false, Car: undefined, Job: null, Symbol: Symbol(unique), Big Number: 9007199254740991n

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

OBJECT LITERAL

What is an object literal in JavaScript, and how is it used in Node.js?

An object literal is a comma-separated list of name-value pairs wrapped in curly braces.

```
let person = {
        firstName: "John",
        lastName: "Doe",
        age: 25,
        getFullName: function() {
                return this.firstName + " " + this.lastName;
        }
};

// Usage in Node.js
console.log(person.getFullName()); // Output: John Doe
```

Functions

How are functions defined and used in Node.js? Provide examples.

Functions can be defined in several ways in JavaScript, including function declarations, function expressions, and arrow functions.

```
// Function Declaration
function greet(name) {
  return `Hello, ${name}`;
}

// Function Expression
const greetExp = function(name) {
  return `Hello, ${name}`;
};
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

```
// Arrow Function
const greetArrow = (name) => `Hello, ${name}`;

// Usage in Node.js
console.log(greet("John")); // Output: Hello, John
console.log(greetExp("Jane")); // Output: Hello, Jane
console.log(greetArrow("Doe")); // Output: Hello, Doe
```

Buffer

What is a Buffer in Node.js, and how is it used?

A Buffer is a class in Node.js to handle binary data. Buffers are used to read or manipulate binary data streams.

Example:

```
// Creating a Buffer from a string
const buf = Buffer.from('Hello World');

// Logging buffer content
console.log(buf); // Output: <Buffer 48 65 6c 6c 6f 20 57 6f 72 6c 64>

// Converting buffer back to string
console.log(buf.toString()); // Output: Hello World
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

ACCESS GLOBAL SCOPE

How do you access the global scope in Node.js?

In Node.js, the global scope refers to the top-level scope that is accessible from anywhere within your Node.js application. It is the scope outside of any function or module.
For Example:

```
// app.js

// Declare a global variable
global.counter = 0;

// Import the counter module
const counterModule = require('./counter');

// Increment the counter 5 times
for (let i = 0; i < 5; i++) {
  counterModule.incrementCounter();
}

// Log the final value of the counter

console.log(`Final counter value: ${global.counter}`);
```

```
// counter.js Module

// Function to increment the global counter
function incrementCounter() {
  global.counter++;
  console.log(`Counter incremented. Current value: ${global.counter}`);
}

// Export the incrementCounter function
module.exports = {   incrementCounter };
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Module

What are modules in Node.js, and what are the different types of modules?

Modules in Node.js are individual JavaScript files that contain code. They can be loaded using the `require()` function.

Module Types:

1. **Core Modules**: Built-in modules provided by Node.js.
2. **Local Modules**: Custom modules created locally in your application.
3. **Third Party Modules**: Modules installed via npm.

Module Exports

How do you export and import modules in Node.js?

Modules can be exported using `module.exports` and imported using `require()`.

Example:

```
// math.js (Local Module)
module.exports.add = (a, b) => a + b;

// app.js
const math = require('./math'); // importing local module

console.log(math.add(5, 3)); // Output: 8
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Setup a new Project

- mkdir student-greeting-app
- cd student-greeting-app
- npm init –y

```
// student.js  (Local Module in the app folder student-greeting-app)
const students = [];

function addStudent(name) {
   students.push(name);
}

function greetStudents() {
   return students.map(student => `Hello, ${student}! Welcome to the Node.js
application.`).join('\n');
}

module.exports = {    addStudent,    greetStudents  };

============ Main Application ==============
// app.js
const studentModule = require('./student');  // the file student.js – dot js is not necessary

// Adding students
studentModule.addStudent('Alice');
studentModule.addStudent('Bob');
studentModule.addStudent('Charlie');

// Greeting students
const greetings = studentModule.greetStudents();
console.log(greetings);
```

Output:

```
Hello, Alice! Welcome to the Node.js application.
Hello, Bob! Welcome to the Node.js application.
Hello, Charlie! Welcome to the Node.js application.
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Using Modules in a Node.js File

How do you use modules in a Node.js file?

Modules are used in a Node.js file by requiring them at the beginning of the file and using their exported functionality.

Example:

```
// file: server.js

const http = require('http');

const server = http.createServer((req, res) => {

   res.statusCode = 200;

   res.setHeader('Content-Type', 'text/plain');

   res.end('Hello World\n');

});

server.listen(3000, '127.0.0.1', () => {

   console.log('Server running at http://127.0.0.1:3000/');

});
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Using the Built-in HTTP, URL, Query String Modules

How do you use the built-in HTTP, URL, and Query String modules in Node.js?

- Core modules in Node.js are built-in modules that provide essential functionality for building applications.
- They are an integral part of the Node.js platform and are available out-of-the-box, without the need to install any external packages

Core modules are useful because they:

- **Provide essential functionality**: They offer a wide range of capabilities out-of-the-box, allowing developers to focus on building their applications rather than reinventing the wheel.
- **Are optimized for performance:** Core modules are compiled into Node.js's binary distribution and are designed to be efficient and fast.
- **Promote code reuse:** By providing common functionality, core modules encourage code reuse and consistency across Node.js applications.
- **Simplify development:** By abstracting away low-level details and providing a high-level API, core modules make it easier to build complex applications

Some of the most commonly used core modules in Node.js include:

1. **fs (File System):** Provides functionality for interacting with the file system, allowing you to create, read, update, and delete files and directories.
2. **http:** Enables you to create servers and clients that communicate using the HTTP protocol. It is crucial for building web servers and RESTful APIs.
3. **url:** Allows you to parse, construct, and manipulate URLs, making it easier to work with web addresses and extract or modify various URL components.
4. **os (Operating System):** Provides information about the operating system, such as CPU architecture, memory, and network interfaces.
5. **path:** Provides utilities for working with file paths and directories.
6. **process:** Provides information about the current Node.js process and allows you to control it.
7. **buffer:** Provides functionality for handling binary data.
8. **stream:** Enables you to work with streaming data.

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Example:

1. Create a simple HTTP server.
2. Serve student data in JSON format.
3. Use the URL module to parse query parameters.

Create a new Project folder

- **mkdir** student-data-server
- **cd** student-data-server
- **npm** init –y
- **create a file index.js**

Write the Server Code

```
// index.js
const http = require('http');
const url = require('url');

// Sample student data
const students = [
  { id: 1, name: 'Alice', age: 20, major: 'Computer Science' },
  { id: 2, name: 'Bob', age: 22, major: 'Mathematics' },
  { id: 3, name: 'Charlie', age: 21, major: 'Physics' },
];

// Create an HTTP server
const server = http.createServer((req, res) => {
// Parse the URL
  const parsedUrl = url.parse(req.url, true);

// Handle different routes
  if (parsedUrl.pathname === '/students') {
    // If a query parameter 'id' is provided, filter the student data
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

```
    const studentId = parsedUrl.query.id;
    if (studentId) {
      const student = students.find(s => s.id == studentId);
      if (student) {
        res.writeHead(200, { 'Content-Type': 'application/json' });
        res.end(JSON.stringify(student));
      } else {
        res.writeHead(404, { 'Content-Type': 'text/plain' });
        res.end('Student not found');
      }
    } else {
      // Return all students if no query parameter is provided
      res.writeHead(200, { 'Content-Type': 'application/json' });
      res.end(JSON.stringify(students));
    }
  } else {
    // Handle 404 for other routes
    res.writeHead(404, { 'Content-Type': 'text/plain' });
    res.end('Route not found');
  }
});

// Start the server
const PORT = 3000;
server.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

Start your server: ***node index.js***

Access the server:

1) To get all students:
   http://localhost:3000/students

2) To get a specific student by ID (e.g., ID 1):
http://localhost:3000/students?id=1

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

CUSTOM MODULES

Discuss usefulness of Custom module in Node.js app development and How do you create a custom module in Node.js?

- Custom modules in Node.js are used to organize code into manageable sections, improve reusability, and simplify the maintenance and scaling of applications by separating concerns.

- They allow developers to encapsulate specific functionality into a separate file that can be imported and used in other parts of the application

Some key uses of custom modules in Node.js include:

1) **Code Organization:**
2) **Reusability:**
3) **Encapsulation:**
4) **Dependency Management:**
5) **Testability:**
6) **Sharing Code:**

Example:

Create a node.js project structure

```
student-data/
├── index.js
├── modules/
│   ├── studentData.js
│   └── utils.js
└── data/
        └── students.json
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Students.json

```
[
 {
  "id": 1,
  "name": "John Doe",
  "age": 20,
  "grade": 85
 },
 {
  "id": 2,
  "name": "Jane Smith",
  "age": 19,
  "grade": 92
 },
 {
  "id": 3,
  "name": "Michael Johnson",
  "age": 21,
  "grade": 78
 },
 {
  "id": 4,
  "name": "Emily Davis",
  "age": 18,
  "grade": 90
 }

]
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Custom Modules

studentData.js

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

const getStudentData = () => {
  const filePath = path.join(__dirname, '../data/students.json');
  const data = fs.readFileSync(filePath, 'utf8');
  return JSON.parse(data);
};

const getStudentById = (id) => {
  const students = getStudentData();
  return students.find((student) => student.id === id);
};

module.exports = {   getStudentData,
                     getStudentById,   };
```

========== ========== ========== ==========

utils.js

```
const calculateAverage = (grades) => {
  const sum = grades.reduce((total, grade) => total + grade, 0);
  return sum / grades.length;
};

module.exports = {
  calculateAverage,
};
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Main Entry Point : the ***index.js*** file

```
const { getStudentData, getStudentById } = require('./modules/studentData');
const { calculateAverage } = require('./modules/utils');

// Get all student data
const students = getStudentData();
console.log('All Students:', students);

// Get a specific student by ID
const studentId = 2;
const student = getStudentById(studentId);
console.log(`Student ${studentId}: ${student.name}`);

// Calculate the average grade
const grades = students.map((student) => student.grade);
const averageGrade = calculateAverage(grades);
console.log('Average Grade:', averageGrade);
```

OUTPUT:

```
All Students: [
  { id: 1, name: 'John Doe', age: 20, grade: 85 },
  { id: 2, name: 'Jane Smith', age: 19, grade: 92 },
  { id: 3, name: 'Michael Johnson', age: 21, grade: 78 },
  { id: 4, name: 'Emily Davis', age: 18, grade: 90 }
]
Student 2: Jane Smith

Average Grade: 86.25
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

NPM, Installing Packages Locally

How do you install packages locally using npm?

Steps:

1. Initialize a new Node.js project:

```
npm init -y
```

2. Install a package locally:

```
npm install express
```

Example:

```
// file: app.js
const express = require('express');
const app = express();

app.get('/', (req, res) => {
    res.send('Hello World!');
});

app.listen(3000, () => {
    console.log('Server running on port 3000');
});
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Adding Dependency in package.json

How do you add a dependency in package.json?

Dependencies are added automatically when you install a package using npm. You can also manually add dependencies.

Example:

```
{
  "name": "example-project",
  "version": "1.0.0",
  "dependencies": {
    "express": "^4.17.1"
  }
}
```

Installing Packages Globally

How do you install packages globally using npm?

Command:

```
npm install -g nodemon
```

Updating Package

How do you update a package in Node.js?

Command:

```
npm update express
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Comparison Table for Module Types

Module Type	Description	Example
Core Modules	Built-in modules provided by Node.js	http, fs, path
Local Modules	Custom modules created locally in your application	require('./myModule')
Third Party Modules	Modules installed via npm	require('express')

Example Code for Using Core Modules

Using the HTTP Core Module:

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

const server = http.createServer((req, res) => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    res.end('Hello World\n');
});

server.listen(3000, '127.0.0.1', () => {
    console.log('Server running at http://127.0.0.1:3000/');
});
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Example Code for Using Local Modules

Creating and Using a Local Module:

```
// file: math.js
module.exports.add = (a, b) => a + b;

// file: app.js
const math = require('./math');
console.log(math.add(5, 3)); // Output: 8
```

Example Code for Using Third Party Modules

Using Express (Third Party Module):

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
    res.send('Hello World!');
});

app.listen(3000, () => {
    console.log('Server running on port 3000');
});
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Explain Export Module in Node.js:

Export Literals,
Export Objects,
Export Function,
Export Function as a Class

The ***module.exports*** is a special object which is included in every JavaScript file in the Node.js application by default.
The module is a variable that represents the current module, and exports is an object that will be exposed as a module.
So, whatever you assign to ***module.exports*** will be exposed as a module.

Export Literals

As mentioned above, exports is an object. So it exposes whatever you assigned to it as a module. For example, if you assign a string literal then it will expose that string literal as a module.

The following example exposes simple string message as a module in Message.js.

Message.js

```
module.exports = 'Hello world';
```

Now, import this message module and use it as shown here. the file **app.js**

```
var msg = require('./Message.js');

console.log(msg);
```

Run the above example and see the result, as shown here.

```
C:\> node app.js
Hello World
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

How to Export Object and Export Function?

1) Export Object

The exports is an object. So, you can attach properties or methods to it. The following example exposes an object with a string property in Message.js file.

```
exports.SimpleMessage = 'Hello world';

//or

module.exports.SimpleMessage = 'Hello world';
```

Import and use this module, as shown below in App.js

The require() function will return an object { SimpleMessage : 'Hello World'} and assign it to the msg variable.

So, now you can use msg.SimpleMessage.

```
var msg = require('./Messages.js');

console.log(msg.SimpleMessage);
```

Run the example ***node app.js*** in the command prompt.

```
C:\> node app.js
Hello World
```

We can also attach an object to **module.exports**

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Create a file ***data.js***

```
module.exports = {
    firstName: 'Tome',
    lastName: 'Carlos'
}
```

Create **app.js**

```
var person = require('./data.js');

console.log(person.firstName + ' ' + person.lastName);
```

Run app.js on command prompt

Output:

```
C:\> node app.js
Tome Carlos
```

2) Export Function

We can attach an anonymous function to exports object:

Create log.js file

```
module.exports = function (msg) {
    console.log(msg);
};
```

Create app.js

```
var msg = require('./Log.js');

msg('Hello World');
```

The **msg** variable becomes a function expression in the above example.
So, you can invoke the function using parenthesis ().
Run the above example and see the output as shown below.

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

```
C:\> node app.js
Hello World
```

3) Export Function as a Class

In JavaScript, a function can be treated like a class. The following example exposes a function that can be used like a class.

Create a file **Person.js**

```
module.exports = function (firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.fullName = function () {
        return this.firstName + ' ' + this.lastName;
    }
}
```

Create app.js

```
var person = require('./Person.js');

var person1 = new person('Abhishek', 'Sharma');

console.log(person1.fullName());
```

Output:

```
C:\> node app.js
Abhishek Sharma
```