Unit 1: Introduction to Node.js, Modules and Events

1. What is Node.js

Node.js is an open-source, server-side runtime environment that allows JavaScript to run outside the browser. It is mainly used to build fast and scalable web applications.

---

2. Advantages of Node.js

- Uses JavaScript for both client and server
- Asynchronous and non-blocking I/O
- High performance and scalability
- Suitable for real-time applications
- Large ecosystem of libraries (NPM)

---

3. Node.js Process Model

Node.js follows a **single-threaded, event-driven** process model. It uses an event loop to handle multiple client requests efficiently without creating multiple threads.

---

4. Traditional Web Server Model

In the traditional model, each client request creates a new thread. This approach consumes more memory and becomes slower when handling many users simultaneously.

---

5. Setup Development Environment

Setting up Node.js includes installing Node.js, configuring environment variables, and using tools like text editors or IDEs to write and run Node.js programs.

---

6. Installation of Node.js on Windows

Node.js is installed on Windows by downloading the installer from the official website and following the setup steps. It includes Node.js and NPM.

---

7. Working in REPL

REPL stands for **Read–Eval–Print–Loop**. It allows users to execute JavaScript code line by line directly in the Node.js command prompt.

---

8. Node.js Console

The console is used to display output or debug information using commands like console.log(), console.error(), etc.

---

9. Standard Callback Pattern

Callbacks are functions passed as arguments to other functions. They are executed after a task is completed, mainly used for asynchronous operations.

---

10. Event Emitter Pattern

The Event Emitter pattern allows objects to emit named events and other objects to listen and respond to those events.

---

11. Event Types

Events represent actions like data reception, completion of tasks, or errors. Common events include data, end, and error.

---

12. Event Emitter API

The Event Emitter API provides methods like:

- **on()** – listen to events
- **emit()** – trigger events
- **once()** – listen only once

---

13. Creating an Event Emitter

An event emitter is created by importing the events module and creating an object of the EventEmitter class.

---

14. Defer Execution of a Function

Deferring execution means delaying function execution using methods like setTimeout() or setImmediate().

---

15. Cancel Execution of a Function

Execution can be cancelled using functions like clearTimeout() or clearInterval() to stop scheduled tasks.

---

16. Self-Learning Topics: Additional Events

Additional events include custom events created by developers to handle specific application logic.

Unit 2: File Handling & HTTP Web Server (with code)

---

1. File Paths

File paths specify the location of a file or directory in the system. Node.js provides path handling to access files correctly across different operating systems.

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

console.log(path.join(__dirname, 'files', 'data.txt'));
```

---

2. fs Module

The fs (File System) module is used to work with files in Node.js. It allows creating, reading, writing, updating, and deleting files.

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

---

3. Opening a File

Opening a file means making it available for reading or writing. Node.js provides methods to open files in different modes.

```
fs.open('data.txt', 'r', (err, fd) => {
  if (!err) console.log("File opened");
});
```

---

4. Reading from a File

Reading a file involves retrieving its contents. Node.js can read files synchronously or asynchronously.

```
fs.readFile('data.txt', 'utf8', (err, data) => {
  console.log(data);
});
```

---

5. Writing to a File

Writing to a file means storing data into it or overwriting existing content.

```
fs.writeFile('data.txt', 'Hello Node.js', () => {
  console.log("File written");
});
```

---

6. Closing a File

Closing a file releases system resources after file operations are completed.

```
fs.close(fd, () => {
  console.log("File closed");
});
```

---

7. HTTP Request Object

The HTTP request object contains information sent by the client such as URL, headers, and method.

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

http.createServer((req, res) => {
  console.log(req.method);
  console.log(req.url);
}).listen(3000);
```

---

8. HTTP Response Object

The HTTP response object is used by the server to send data back to the client.

```
res.write("Hello Client");
res.end();
```

---

9. HTTP Headers

Headers are key-value pairs that provide metadata about the request or response.

res.writeHead(200, { 'Content-Type': 'text/plain' });

---

10. Piping

Piping is used to transfer data from one stream to another.

fs.createReadStream('data.txt').pipe(res);

---

11. Shutting Down the Server

Shutting down the server stops accepting new requests and closes connections safely.

```
server.close(() => {
  console.log("Server shut down");
});
```

---

12. Self-Learning Topics: TCP Server

A TCP server allows direct communication using the TCP protocol.

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

net.createServer(socket => {
  socket.write("Hello TCP Server");
}).listen(4000);
```

Unit 3: Databases (MySQL – Connect, Communicate & CRUD Operations)

---

1. Connect and Communicate with MySQL Database

Node.js connects to a MySQL database using a MySQL driver. This allows applications to send queries and receive results.

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


const con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "",
  database: "testdb"
});


con.connect(err => {
  if (!err) console.log("Connected to MySQL");
});
```

---

2. Adding Data to the Database

Data can be inserted into a database table using SQL INSERT queries from Node.js.

```
const sql = "INSERT INTO student (id, name) VALUES (1, 'Sujit')";


con.query(sql, (err, result) => {
  if (!err) console.log("Record inserted");
});
```

---

3. CRUD Operations

CRUD stands for **Create, Read, Update, Delete**, which are basic database operations.

---

a) Create (Insert Data)

Adds new records to the table.

```
con.query(
  "INSERT INTO student (id, name) VALUES (2, 'Amit')"
);
```

---

b) Read (Retrieve Data)

Fetches data from the database.

```
con.query("SELECT * FROM student", (err, result) => {
  console.log(result);
});
```

---

c) Update (Modify Data)

Updates existing records.

```
con.query(
  "UPDATE student SET name='Rahul' WHERE id=2"
);
```

---

d) Delete (Remove Data)

Deletes records from the table.

```
con.query(
  "DELETE FROM student WHERE id=2"
);
```

---

4. Self-Learning Topics: Working with Any Other Database

Apart from MySQL, Node.js can work with other databases like MongoDB, PostgreSQL, and SQLite using their respective drivers.

```
// Example (MongoDB connection idea)
const { MongoClient } = require('mongodb');
```

Unit 4: Introduction to ReactJS

---

1. Introduction to ReactJS

ReactJS is a JavaScript library used to build **user interfaces**, especially **single-page applications**. It is component-based and developed by Facebook.

React makes UI fast, reusable, and easy to manage.

---

2. Features of ReactJS

- Component-based architecture
- Virtual DOM for better performance
- One-way data binding
- Reusable UI components

React focuses only on the view layer.

---

3. Setting Up React Environment

React environment is set up using **Node.js** and **npm**. The recommended way is using **Create React App**.

npm install -g create-react-app

---

4. Create React App

Create React App is a tool that sets up a React project with all required configurations.

npx create-react-app myapp
cd myapp
npm start

---

5. Folder Structure of React App

- src → contains application code
- public → static files
- package.json → project dependencies

src/App.js is the main component file.

---

6. Hello World Program in React

A simple React program that displays text on the browser.

function App(){ return <h1>Hello World</h1>; }

---

7. Understanding JSX

JSX stands for **JavaScript XML**. It allows writing HTML inside JavaScript.

const element = <h1>Welcome</h1>;

---

8. Rendering Elements

Rendering displays React elements on the web page using ReactDOM.

ReactDOM.render(<App />, document.getElementById('root'));

---

9. Components in React

Components are reusable pieces of UI. They can be **functional** or **class** components.

function MyComponent(){ return Component ; }

---

10. Self-Learning Topic: XML

XML is a markup language used to store and transport data in a structured format.

<name>Sujit</name>

Unit 5: Components and Events

---

1. Components (Theory)

Components are the basic building blocks of a React application. Each component represents a part of the user interface and can be reused multiple times. Components help in dividing the UI into independent and reusable pieces.

```
function Welcome() {
  return <h1>Welcome</h1>;
}
```

---

2. Functional Components (Theory)

Functional components are simple JavaScript functions that return JSX. They are easy to write, understand, and maintain.

```
function Hello() {
  return <p>Hello User</p>;
}
```

---

3. Class Components (Theory)

Class components are ES6 classes that extend React.Component. They can use lifecycle methods and maintain state.

```
class Message extends React.Component {
  render() {
    return <h2>Hello React</h2>;
  }
}
```

---

4. Rendering Components (Theory)

Rendering means displaying the component output on the browser. React uses ReactDOM.render() to render components inside the DOM.

```
ReactDOM.render(<Welcome />, document.getElementById('root'));
```

---

5. Components in Separate Files (Theory)

To improve code reusability and readability, components are created in separate files and imported when needed.

import Welcome from './Welcome';

---

6. Props (Theory)

Props (Properties) are used to pass data from a parent component to a child component. Props are read-only and cannot be modified inside the child component.

```
function Student(props) {
  return <h3>{props.name}</h3>;
}
```

---

7. Passing Props (Theory)

Props are passed to components as attributes.

```
<Student name="Sujit" />
```

---

8. Accessing Props (Theory)

Props are accessed using the props object inside the component.

```
props.name
```

---

9. DOM Events (Theory)

React supports handling DOM events similar to JavaScript. Event names are written in camelCase.

---

a) Click Event

```
<button onClick={clickMe}>Click</button>
```

---

b) Change Event

```
<input onChange={handleChange} />
```

---

c) Blur Event

```
<input onBlur={handleBlur} />
```

---

d) KeyUp Event

<input onKeyUp={handleKey} />

---

10. Component Life Cycle (Theory)

Lifecycle methods control different phases of a component such as creation, updating, and unmounting. These methods are mainly used in class components.

componentDidMount() {

console.log("Component Mounted");

}

---

11. Self-Learning Topics: CSS & SCSS (Theory)

CSS and SCSS are used to style React components. SCSS provides advanced features like variables and nesting.

Unit 6: Forms, Hooks and Routing

---

1. Forms (Theory)

Forms in React are used to collect user input such as text, numbers, and selections. React forms are usually **controlled components**, where form data is handled by React state.

```
function MyForm() {
  return (
    <form>
      <input type="text" />
    </form>
  );
}
```

---

2. Handling User Input with Forms (Theory)

User input is handled using state and event handlers. The value of input fields is controlled using useState.

```
import { useState } from 'react';


function FormExample() {
  const [name, setName] = useState("");


  return (
    <input
      type="text"
      value={name}
      onChange={(e) => setName(e.target.value)}
    />
  );
}
```

---

3. Form Validation Techniques (Theory)

Form validation ensures that the user enters correct and required data before submission. Validation can be done using conditions.

```
if (name === "") {
  alert("Name is required");
}
```

---

4. Hooks (Theory)

Hooks are special functions that allow functional components to use React features like state and lifecycle methods. Hooks simplify code and improve readability.

---

a) useState Hook (Theory)

useState is used to create and manage state in functional components.

```
const [count, setCount] = useState(0);
```

---

b) useEffect Hook (Theory)

useEffect is used to perform side effects such as data fetching or DOM updates.

```
useEffect(() => {
  console.log("Component Loaded");
}, []);
```

---

c) useContext Hook (Theory)

useContext is used to access data globally without passing props manually.

```
const value = useContext(MyContext);
```

---

5. React Routing (Theory)

Routing allows navigation between different pages without reloading the browser. React uses **React Router** for routing.

```
import { BrowserRouter, Route, Routes } from 'react-router-dom';


function App() {
  return (
    <BrowserRouter>
```

```
    <Routes>
      <Route path="/" element={<Home />} />
    </Routes>
  </BrowserRouter>
 );
}
```

---

6. Self-Learning Topic: Custom Hooks (Theory)

Custom Hooks are user-defined hooks that reuse common logic across components.

```
function useCounter() {
 const [count, setCount] = useState(0);
 return [count, setCount];
}
```