1. Introduction to JSON

What is JSON?

- JSON (**JavaScript Object Notation**) is a lightweight data-interchange format.
- It is **easy to read** and **write**.
- Used for data storage and communication between a client and a server.

Why JSON?

- Simple and lightweight.
- Human-readable format.
- Used by most programming languages.
- Common in APIs (like REST APIs).

Example JSON Data

```
{
  "name": "Ved",
  "age": 30,
  "city": "Mumbai"
}
```

Teaching JSON (Without Express.js)

This revised approach will cover **JSON concepts** without using a backend framework.

1. Introduction to JSON

- JSON (**JavaScript Object Notation**) is used to store and exchange data.
- It is text-based and easy for both humans and machines to read.
- Used commonly in **web development**, **config files**, and **APIs**.

Example JSON Data

```
{
  "name": "Ved",
  "age": 30,
  "city": "Mumbai"
}
```

2. JSON Syntax

Basic Rules

- **Key-Value Pairs**: { "key": "value" }
- **Keys must be in double quotes** "key"
- **Values** can be:
    - Strings "text"
    - Numbers 123
    - Boolean true/false
    - Arrays [ ]
    - Objects { }
    - null

Example

```
{
  "student": {
    "name": "Rahul",
    "age": 21,
    "subjects": ["Math", "Science", "English"],
    "marks": { "Math": 90, "Science": 85 },
    "graduated": false
  }
}
```

3. JSON vs XML

Feature	JSON	XML
Format	Key-Value	Tag-Based
Readability	Easy	Complex
Data Size	Smaller	Larger
Parsing	Fast	Slow
Use in Web	APIs, Config Files	Used in some old APIs

4. JSON Data Types

```
{
  "name": "John",
  "age": 25,
  "isStudent": false,
  "grades": [85, 90, 95],
  "address": { "city": "Mumbai", "pincode": 400001 },
```

```
  "phone": null
}
```

5. JSON Parsing & Stringify (JavaScript Only)

Convert JSON to JavaScript Object

```
let jsonData = '{"name": "Ved", "age": 30}';
let obj = JSON.parse(jsonData);
console.log(obj.name); // Ved
```

Convert JavaScript Object to JSON

```
let student = { name: "Rahul", age: 21 };
let jsonString = JSON.stringify(student);
console.log(jsonString); // {"name":"Rahul","age":21}
```

6. JSON Arrays

```
{
  "students": [
    { "name": "Rahul", "age": 21 },
    { "name": "Anjali", "age": 22 }
  ]
}
```

Accessing JSON Array in JavaScript

```
let data = `{
  "students": [
    { "name": "Rahul", "age": 21 },
    { "name": "Anjali", "age": 22 }
  ]
}`;

let obj = JSON.parse(data);
console.log(obj.students[0].name); // Rahul
```

7. JSON in HTML (Without Backend)

You can use **JavaScript Fetch API** to load JSON into a webpage.

Example: Fetch API with JSON

```
<!DOCTYPE html>
<html>
<head>
  <title>JSON Example</title>
</head>
<body>
  <h2>Student Details</h2>
  <pre id="output"></pre>

  <script>
    let jsonData = `{
      "students": [
        { "name": "Rahul", "age": 21 },
        { "name": "Anjali", "age": 22 }
      ]
    }`;

    let students = JSON.parse(jsonData);
    document.getElementById("output").textContent =
JSON.stringify(students, null, 2);
  </script>
</body>
</html>
```

8. Using JSON in Postman (Without Backend)

Even without Express.js, students can still **send JSON requests in Postman** using mock APIs.

Step 1: Use a Free Online Mock API

- Visit https://mockapi.io/
- Create a free mock API.
- Use the given API URL in Postman.

Step 2: Sending a GET Request

- Open **Postman**.
- Set method to **GET**.
- Enter **Mock API URL** (e.g., https://example.mockapi.io/students).
- Click **Send**.
- You'll receive a **JSON response**.

Step 3: Sending a POST Request

- Set method to **POST**.
- Go to **Body > raw** > select **JSON**.

Enter:

```
{
  "name": "Amit",
  "age": 23
}
```

- Click **Send**.

9. Assignment for Students

Task 1: JSON Parsing & Stringify

- Write a **JavaScript program** that:
  - Takes a JSON string.
  - Converts it to an object.
  - Modifies the object.
  - Converts it back to JSON.

Task 2: JSON Arrays in HTML

- Load JSON data into an HTML page and display it dynamically.

Task 3: Use Postman for API Testing

- Use **mock API** to test **GET & POST requests**.

Scenario-Based Questions for JSON

These questions will help students apply **JSON concepts in real-world scenarios**.

1. Data Storage Scenario

Scenario:
A school wants to store student information, including their name, roll number, subjects, and grades.

Task:

- Create a **JSON structure** to store student details.
- Write JavaScript code to **parse** the JSON data and access the student's **Math marks**.

```
{
  "students": [
    {
      "name": "Rahul",
      "rollNumber": 101,
      "subjects": ["Math", "Science", "English"],
      "grades": { "Math": 90, "Science": 85, "English": 88 }
    },
    {
      "name": "Anjali",
      "rollNumber": 102,
      "subjects": ["Math", "History", "English"],
      "grades": { "Math": 95, "History": 80, "English": 78 }
    }
  ]
}
```

JavaScript Code to Read Student Data

```
let jsonData = `{
  "students": [
    {
      "name": "Rahul",
      "rollNumber": 101,
      "subjects": ["Math", "Science", "English"],
      "grades": { "Math": 90, "Science": 85, "English": 88 }
    }
  ]
}`;

let students = JSON.parse(jsonData);
console.log("Rahul's Math Grade:",
students.students[0].grades.Math);
```

2. Product Catalog for E-commerce

Scenario:
An e-commerce website wants to store details of products in JSON format.

Task:

- Define a JSON structure for **three products** (name, price, stock, category).
- Write JavaScript to **convert** this data into a JSON string.

```
{
  "products": [
    { "id": 1, "name": "Laptop", "price": 50000, "stock": 10,
"category": "Electronics" },
    { "id": 2, "name": "Mobile", "price": 20000, "stock": 5,
"category": "Electronics" },
    { "id": 3, "name": "Book", "price": 500, "stock": 100,
"category": "Education" }
  ]
}
```

JavaScript Code to Convert to JSON

```
let products = [
  { id: 1, name: "Laptop", price: 50000, stock: 10, category:
"Electronics" },
  { id: 2, name: "Mobile", price: 20000, stock: 5, category:
"Electronics" },
  { id: 3, name: "Book", price: 500, stock: 100, category:
"Education" }
];

let jsonString = JSON.stringify(products, null, 2);
console.log(jsonString);
```

3. Weather API Response Parsing

Scenario:
A weather app fetches data from an API in JSON format.

Task:

- Given a JSON response, extract and display the **temperature and weather condition**.

```
{
  "city": "Mumbai",
  "temperature": 30,
  "condition": "Sunny",
  "humidity": 70
}

JavaScript Code to Extract Data
let weatherData = `{
  "city": "Mumbai",
  "temperature": 30,
  "condition": "Sunny",
  "humidity": 70
}`;

let weather = JSON.parse(weatherData);
console.log(`City: ${weather.city}`);
console.log(`Temperature: ${weather.temperature}°C`);
console.log(`Condition: ${weather.condition}`);
```

User Authentication Using JSON

Scenario:
A login system stores users in a JSON file.

Task:

- Given a JSON list of users, **check if login credentials are correct**.

```
{
  "users": [
    { "username": "admin", "password": "12345" },
    { "username": "guest", "password": "guest123" }
  ]
}

JavaScript Code for Login
let users = `{
  "users": [
    { "username": "admin", "password": "12345" },
    { "username": "guest", "password": "guest123" }
  ]
```

```
}`;

let userList = JSON.parse(users);
let enteredUsername = "admin";
let enteredPassword = "12345";

let isValidUser = userList.users.some(
  user => user.username === enteredUsername && user.password === 
enteredPassword
);

console.log(isValidUser ? "Login Successful!" : "Invalid 
Credentials");
```

Using JSON in React

Now, let's see how **React** handles JSON.

1. Display JSON Data in a React Component

React Code:

```
import React from "react";

const studentData = {
  name: "Rahul",
  age: 21,
  subjects: ["Math", "Science", "English"]
};

function StudentInfo() {
  return (
    <div>
      <h2>Student Information</h2>
      <p><strong>Name:</strong> {studentData.name}</p>
      <p><strong>Age:</strong> {studentData.age}</p>
      <p><strong>Subjects:</strong> {studentData.subjects.join(", 
")}</p>
    </div>
  );
}
```

```
export default StudentInfo;
```

2. Fetching JSON Data from an API

```
import React, { useEffect, useState } from "react";

function StudentList() {
  const [students, setStudents] = useState([]);

  useEffect(() => {
    fetch("https://67b43473392f4aa94fa9c845.mockapi.io/students")
// Your MockAPI URL
      .then(response => response.json())
      .then(data => setStudents(data))
      .catch(error => console.error("Error fetching students:",
error));
  }, []);

  return (
    <div>
      <h2>Student List</h2>
      <ul>
        {students.map(student => (
          <li key={student.id}>
            {student.name} - Age: {student.age}
          </li>
        ))}
      </ul>
    </div>
  );
}

export default StudentList;
```

3. Sending JSON Data to an API

```
import React, { useState } from "react";
```

```
function AddStudent() {
  const [studentName, setStudentName] = useState("");
  const [age, setAge] = useState("");

  const handleSubmit = (e) => {
    e.preventDefault();

    const newStudent = { name: studentName, age };

    fetch("https://67b43473392f4aa94fa9c845.mockapi.io/students",
{ // Your MockAPI URL
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(newStudent)
    })
      .then(response => response.json())
      .then(data => console.log("Student Added:", data))
      .catch(error => console.error("Error adding student:",
error));
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        placeholder="Student Name"
        value={studentName}
        onChange={e => setStudentName(e.target.value)}
        required
      />
      <input
        type="number"
        placeholder="Age"
        value={age}
        onChange={e => setAge(e.target.value)}
        required
      />
      <button type="submit">Add Student</button>
    </form>
  );
}
```

```
export default AddStudent;
```

Final Assignment for Students

1. Create a **React app** that:
    - Fetches student data from a JSON file.
    - Displays the student list.
    - Adds a new student to the list.
2. Use **Postman** to send and receive JSON data.
3. Convert a **JavaScript object to JSON** and store it in localStorage.

Validating JSON – Using JSON.parse() with Error Handling

Why Validate JSON?

- If JSON data is **malformed or incorrect**, JSON.parse() will throw an error.
- We use try...catch to handle errors gracefully and prevent our code from breaking.

Example: Handling Invalid JSON

```
let invalidJson = '{"name": "Rahul", "age": 21,}'; // Incorrect
JSON (extra comma)

try {
  let parsedData = JSON.parse(invalidJson);
  console.log(parsedData);
} catch (error) {
  console.error("Error parsing JSON:", error.message);
}
```

Output:

Error parsing JSON: Unexpected token } in JSON at position 28

Why Error? The extra , at the end of "age": 21, makes it invalid.

Example: Valid JSON Parsing

```
let validJson = '{"name": "Rahul", "age": 21}';

try {
  let parsedData = JSON.parse(validJson);
  console.log("Valid JSON Parsed:", parsedData);
} catch (error) {
  console.error("Error parsing JSON:", error.message);
}
```

Output:

```
Valid JSON Parsed: { name: 'Rahul', age: 21 }

Using JSON.parse() Safely in a Function

function safeJsonParse(jsonString) {
  try {
    return JSON.parse(jsonString);
  } catch (error) {
    console.error("Invalid JSON:", error.message);
    return null;
  }
}

let result = safeJsonParse('{"name": "Ved"}');  // Valid JSON
console.log(result);  // Output: { name: "Ved" }

let result2 = safeJsonParse('{"name": "Ved",}');  // Invalid JSON
console.log(result2);  // Output: null
```

Working with localStorage – Saving and Retrieving JSON

Why Use localStorage?

- localStorage lets us store **data in the browser**.

- Data **persists** even after page refresh.
- We store data as **JSON strings**.

Example: Storing and Retrieving JSON in localStorage

```
let student = { name: "Rahul", age: 21 };

// Convert object to JSON and save it
localStorage.setItem("studentData", JSON.stringify(student));

// Retrieve JSON string and parse it
let storedData = localStorage.getItem("studentData");
let parsedData = JSON.parse(storedData);

console.log(parsedData.name); // Output: Rahul
console.log(parsedData.age);  // Output: 21
 Example: Adding More Data to localStorage
// Get stored students or set empty array if null
let students = JSON.parse(localStorage.getItem("students")) || [];

// Add a new student
students.push({ name: "Anjali", age: 22 });

// Save updated list
localStorage.setItem("students", JSON.stringify(students));

console.log(localStorage.getItem("students")); // JSON string

Example: Removing Data from localStorage

// Remove a single key
localStorage.removeItem("studentData");

// Clear all localStorage data
localStorage.clear();

Real-World Scenario: Save User Preferences

function saveTheme(theme) {
  localStorage.setItem("userTheme", JSON.stringify({ theme }));
}
```

```
function loadTheme() {
  let data = JSON.parse(localStorage.getItem("userTheme"));
  return data ? data.theme : "light"; // Default theme: light
}

saveTheme("dark");
console.log(loadTheme()); // Output: dark
```

Assignment for Students

1. **Validate JSON Input**: Write a function that takes a string and checks if it's valid JSON.
2. **Save & Retrieve User Data**: Create a form where users enter their name and age, and store it in localStorage. Retrieve and display the saved data on page reload.
3. **Create a Theme Switcher**: Save the selected **light/dark theme** in localStorage, and apply it when the page reloads.