Lab sheet 04

Title: Understand and Implement JSON

What is JSON?

JSON stands for **JavaScript Object Notation**. It is a lightweight **data-interchange format** that is easy for humans to read and write, and easy for machines to parse and generate.

Why Use JSON?

- **Simple and Readable**: Uses key/value pairs that resemble JavaScript object syntax.
- **Language Independent**: Although it originates from JavaScript, JSON is supported by most modern programming languages.
- **Widely Used**: Ideal for transmitting data between a server and a client (such as in APIs and web apps).

JSON Syntax Basics

1. **Data is in name/value pairs**:
2. `"name": "Alice"`
3. **Data is separated by commas**:
4. `"name": "Alice", "age": 25`
5. **Curly braces `{}` hold objects**:
6. `{`
7. `  "name": "Alice",`
8. `  "age": 25`
9. `}`
10. **Square brackets `[]` hold arrays**:
11. `{`
12. `  "employees": ["John", "Jane", "Mark"]`
13. `}`

Example of a JSON Object

```
{
  "name": "John Doe",
  "age": 30,
  "isStudent": false,
  "skills": ["JavaScript", "Python", "HTML"],
  "address": {
    "street": "123 Main St",
    "city": "New York",
    "zip": "10001"
  }
}
```

Implementing JSON in Code

JavaScript Example

```
// Creating a JSON object
const jsonString = '{"name":"John", "age":30}';

// Parsing JSON string into JavaScript object
const obj = JSON.parse(jsonString);
console.log(obj.name); // Output: John

// Converting JavaScript object to JSON string
const newJsonString = JSON.stringify(obj);
console.log(newJsonString); // Output: {"name":"John","age":30}
```

Python Example

```
import json

# JSON string
json_str = '{"name": "Alice", "age": 25}'

# Parse JSON string to dictionary
data = json.loads(json_str)
print(data["name"])  # Output: Alice

# Convert dictionary to JSON string
new_json = json.dumps(data)
print(new_json)  # Output: {"name": "Alice", "age": 25}
```

Conclusion

JSON is a simple, human-readable, and language-independent format widely used for **storing and exchanging data**, especially in **web applications** and **APIs**. Understanding JSON is essential for modern programming.

<u>JavaScript Variable vs JavaScript Object</u>

1. JavaScript Variable

- A **JavaScript variable** is a container that holds a **single value**.
- This value can be of different data types such as **string, number, boolean**, etc.

Example:

```
var car = "Vitz";
```

- In this example, the variable `car` holds a simple string value: `"Vitz"`.

---

2. JavaScript Object

- A **JavaScript object** is a more complex type of variable that can hold **multiple values** in the form of **key:value pairs**.
- Objects are used to represent real-world entities with properties.

Example:

```
var car = {
  type: "Vitz",
  color: "White",
  model: "2018"
};
```

- In this example, the variable `car` is an object with three properties:
  - `type`: "Vitz"
  - `color`: "White"
  - `model`: "2018"

Key Differences

Feature	JavaScript Variable	JavaScript Object
Holds	A single simple value	Multiple values as key:value pairs
Data Types	String, Number, Boolean, etc.	Object
Structure	Simple	Complex (with properties)
Use Case	Storing a single piece of data	Representing an entity with multiple attributes
Example	`var name = "John";`	`var person = {name: "John", age: 30};`

Conclusion

- Use **simple variables** when you only need to store one value.
- Use **objects** when you need to store and organize **related data** under one name.

Exchanging Data
When exchanging data between a browser and server, the data can only be text. JSON is text so can convert any JavavScript object into JSON, and send JSON to the server. And also convert any JSON received from the server into JavaScript object. No need complicate parsing and translation process.

Sending Data in JavaScript

JavaScript Object to JSON

- Before sending data to a **server**, a JavaScript object must be **converted into JSON**.
- This is because servers typically accept data in **JSON format** for APIs and web communication.

How to Convert a JavaScript Object to JSON

You can use the `JSON.stringify()` method to convert a JavaScript object into a JSON string.

Example 01: Convert Object to JSON and Display

```
<!DOCTYPE html>
<html>
<head>
  <title>JSON Example</title>
</head>
<body>

<p id="demo"></p>

<script>
  // Step 1: Create a JavaScript object
  var car = {
    type: "Vitz",
    color: "White",
    model: "2018"
  };

  // Step 2: Convert the object to a JSON string
  var carDetails = JSON.stringify(car);

  // Step 3: Display the JSON string in an HTML element
  document.getElementById('demo').innerHTML = carDetails;
</script>

</body>
</html>
```

Output:

```
{"type":"Vitz","color":"White","model":"2018"}
```

Why This Is Useful

- The `carDetails` variable now holds the object as a JSON string.
- This string can be **sent to a server using AJAX, Fetch API, or XMLHttpRequest**.

Receiving Data

When data is **received from a server** in **JSON format**, it needs to be **converted into a JavaScript object** to be used in your code.

This is done using the `JSON.parse()` method.

How to Convert JSON to a JavaScript Object

You use `JSON.parse()` to convert a **JSON string** into a **JavaScript object**.

Example 02: Convert Received JSON into JS Object

```
<!DOCTYPE html>
<html>
<head>
  <title>Receive JSON Data</title>
</head>
<body>

<p id="demo"></p>

<script>
  // Step 1: Received JSON data as a string (simulated here)
  var student = '{"name":"John", "age":21, "city":"Colombo"}';

  // Step 2: Convert JSON string to JavaScript object
  var JSObject = JSON.parse(student);

  // Step 3: Access and display object properties
  document.getElementById('demo').innerHTML =
    "Student Name: " + JSObject.name + "<br>" +
    "Age: " + JSObject.age + "<br>" +
    "City: " + JSObject.city;
</script>

</body>
</html>
```

Output:

```
Student Name: John
Age: 21
City: Colombo
```

Why This Is Useful

- Most data from APIs and servers are returned as **JSON strings**.
- You must **parse** the data into an object using `JSON.parse()` to **use it in your code**.

Storing Data

When storing data in the browser, it must be in a **string format**. JSON is the ideal format for this purpose because it can represent **structured data as text**.

You don’t need to worry **where** to store it—JavaScript provides built-in storage like **`localStorage`** for this.

✅ Why Use JSON for Storage?

- JavaScript objects cannot be stored directly in `localStorage`.
- Use `JSON.stringify()` to **convert the object into a string** before storing.
- Use `JSON.parse()` to **convert it back into an object** when retrieving.

Example 03: Storing an Object in `localStorage`

```
// Step 1: Create a JavaScript object
var employee = {
  name: "John",
  age: 31,
  city: "New York"
};

// Step 2: Convert the object to a JSON string
var employeeJSON = JSON.stringify(employee);

// Step 3: Store the JSON string in localStorage
localStorage.setItem("testing", employeeJSON);
```

✅ What Happens Here?

- The `employee` object is converted to JSON:
- `{"name":"John","age":31,"city":"New York"}`
- It is stored in the browser's `localStorage` under the key `"testing"`.

🔄 Retrieving and Using the Stored Data

You can retrieve and parse it like this:

```
var storedData = localStorage.getItem("testing");
var employeeObj = JSON.parse(storedData);
console.log(employeeObj.name); // Output: John
```

📦 Where Is the Data Stored?

- `localStorage` stores data **in the user's browser**.
- Data persists even after the page is refreshed or the browser is closed.

Retrieving Data

After storing data in the browser (e.g., using `localStorage`), you often need to **retrieve** and **use** that data in your JavaScript code.

This requires two key steps:

1. Get the JSON string from storage.
2. Convert (parse) it back into a JavaScript object.

✅ Example: Retrieve and Display Stored Data

```
<!DOCTYPE html>
<html>
```

```
<head>
  <title>Retrieve JSON Data</title>
</head>
<body>

<p id="demo"></p>

<script>
  // Step 1: Retrieve the JSON string from localStorage
  var text = localStorage.getItem("testing");

  // Step 2: Convert the JSON string back into a JavaScript object
  var obj = JSON.parse(text);

  // Step 3: Access and display a property
  document.getElementById("demo").innerHTML = obj.name;
</script>

</body>
</html>
```

📝 Note 01: JSON is Language-Independent

- JSON uses **JavaScript syntax**, but it is just **text**.
- Because it's plain text, **any programming language** (like Python, Java, PHP, etc.) can read and use it.
- That makes JSON a universal data exchange format for APIs and web apps.

🛠 Note 02: `JSON.parse()` Method

- The `JSON.parse()` method converts **string data (JSON)** back into a **JavaScript object**.
- Use this when you retrieve data stored in JSON format and want to work with it like a normal object in JavaScript.

✅ Summary

Action	Method Used
Convert object to string	`JSON.stringify()`
Store in browser	`localStorage.setItem()`
Retrieve from browser	`localStorage.getItem()`
Convert string to object	`JSON.parse()`

Below is a **complete example** that shows how to:

1. **Store** a JavaScript object in `localStorage` (after converting it to JSON)
2. **Retrieve** and display the stored data
3. **Delete** the stored data

✅ Complete Example: Store, Retrieve, and Delete Data Using localStorage

```
<!DOCTYPE html>
<html>
<head>
  <title>LocalStorage JSON Example</title>
  <style>
    button {
      margin: 5px;
      padding: 8px 12px;
    }
  </style>
</head>
<body>

<h2>Employee Details</h2>
<p id="demo">Click a button to perform an action.</p>

<!-- Buttons to trigger actions -->
<button onclick="storeData()">Store Data</button>
<button onclick="retrieveData()">Retrieve Data</button>
<button onclick="deleteData()">Delete Data</button>

<script>
  // Function to store data
  function storeData() {
    var employee = {
      name: "John",
      age: 31,
      city: "New York"
    };

    var employeeJSON = JSON.stringify(employee);
    localStorage.setItem("employeeData", employeeJSON);

    document.getElementById("demo").innerHTML = "Data stored successfully!";
  }

  // Function to retrieve data
  function retrieveData() {
    var text = localStorage.getItem("employeeData");

    if (text) {
      var obj = JSON.parse(text);
      document.getElementById("demo").innerHTML =
        "Name: " + obj.name + "<br>" +
        "Age: " + obj.age + "<br>" +
        "City: " + obj.city;
    } else {
      document.getElementById("demo").innerHTML = "No data found in
localStorage.";
    }
  }

  // Function to delete data
  function deleteData() {
    localStorage.removeItem("employeeData");
    document.getElementById("demo").innerHTML = "Data deleted from
localStorage.";
  }
</script>

</body>
</html>
```

📝 Explanation

Action	Function Name	Description
Store data	`storeData()`	Converts an object to JSON and stores it in `localStorage`
Retrieve data	`retrieveData()`	Gets the JSON string from `localStorage`, parses it, and displays it
Delete data	`deleteData()`	Removes the item from `localStorage`

JSON Syntax

The **JSON (JavaScript Object Notation)** syntax is a **subset of JavaScript object syntax**. It is used to **store and exchange data** in a structured, text-based format.

🖊 JSON Syntax Rules

JSON syntax is derived from **JavaScript object notation**, but with stricter rules:

✅ Rule 01: Data is in name/value pairs

- A **name (key)** is always a string, followed by a colon `:`, and then a **value**.

Example:

```
"type": "Vitz"
```

✅ Rule 02: Data is separated by commas

- When you have multiple name/value pairs, separate them with commas `,`.

Example:

```
"type": "Vitz", "model": "2018", "color": "white"
```

✅ Rule 03: Curly braces `{}` hold objects

- An object is a collection of name/value pairs enclosed in curly braces.

Example (JavaScript syntax):

```
var car = {
  "type": "Vitz",
  "model": "2018",
  "color": "white"
};
```

✅ Rule 04: Square brackets `[]` hold arrays

- Arrays are ordered lists of values enclosed in square brackets.
- A key can hold an array of strings, numbers, or objects.

Example (Array inside object):

```
var car = {
  "type": ["Vitz", "Fit", "Aqua"],
  "model": "2018",
  "color": "white"
};
```

⚠ Important Notes About JSON

- All **keys must be in double quotes** `" "` (unlike JavaScript where quotes are optional).
- JSON values can be:
  - String (in double quotes)
  - Number

    - Boolean (`true`/`false`)
    - `null`
    - Array
    - Another JSON object

✕ Invalid JSON (JavaScript-style, not JSON):

```
var car = { type: 'Vitz' }; // ✕ Invalid in strict JSON (no quotes around
key or single quotes used)
```

✅ Valid JSON:

```
{ "type": "Vitz" }
```

Let's expand your understanding with:

1. ✅ **Recommended JSON Validator Tools**
2. □ **Examples of Nested JSON**
3. 📦 **JSON Arrays of Objects**

✅ 1. JSON Validator Tools

These tools help you **validate**, **format**, and **visualize** JSON data. Very useful when working with APIs or debugging.

🔍 Online JSON Validator Tools:

Tool	URL
JSONLint	https://jsonlint.com
JSON Formatter & Validator by JSONFormatter.org	https://jsonformatter.org
Code Beautify - JSON Validator	https://codebeautify.org/jsonvalidator
DevTools (Browser)	In Chrome or Firefox → Right click → Inspect → Console

Just paste your JSON in these tools and check if it's **valid**, **pretty**, or has **syntax errors**.

□ 2. Nested JSON Example

Nested JSON is when a value of a key is another object.

```
{
  "name": "John",
  "age": 30,
  "address": {
    "street": "123 Main St",
    "city": "Colombo",
    "postalCode": "10000"
  }
}
```

✅ Here, `"address"` is a **nested object**.

Access in JavaScript:

```
console.log(obj.address.city); // Output: Colombo
```

📦 3. JSON Arrays of Objects

Sometimes, you want to store multiple objects inside an array — commonly used in APIs.

```
{
  "students": [
    { "name": "Alice", "age": 22 },
    { "name": "Bob", "age": 23 },
    { "name": "Charlie", "age": 24 }
  ]
}
```

✅ Here, `"students"` is an **array** containing multiple **student objects**.

Access in JavaScript:

```
console.log(obj.students[1].name); // Output: Bob
```

Here's a **full HTML + JavaScript demo** that:

✅ Loads a JSON object
✅ Displays:

- Nested values (like `address.city`)
- Array of objects (like a list of `students`)

🔧 Full HTML + JavaScript Demo: Nested JSON + Arrays

```
<!DOCTYPE html>
<html>
<head>
  <title>Nested JSON & Arrays Demo</title>
  <style>
    body {
      font-family: Arial, sans-serif;
    }
    .section {
      margin-bottom: 20px;
    }
    .section h3 {
      color: #006699;
    }
  </style>
</head>
<body>

<h2>📄 JSON Demo: Nested Data & Arrays</h2>

<div class="section">
  <h3>🏠 Address Details (Nested Object)</h3>
  <p id="address"></p>
</div>

<div class="section">
  <h3>🎓 Student List (Array of Objects)</h3>
  <ul id="students"></ul>
</div>

<script>
  // Sample JSON object with nested and array data
  var data = {
    "name": "John",
    "age": 30,
    "address": {
      "street": "123 Main St",
      "city": "Colombo",
      "postalCode": "10000"
    },
    "students": [
      { "name": "Alice", "age": 22 },
      { "name": "Bob", "age": 23 },
      { "name": "Charlie", "age": 24 }
    ]
  };

  // Accessing and displaying nested object (address)
  var addressOutput = "Street: " + data.address.street + "<br>" +
                      "City: " + data.address.city + "<br>" +
                      "Postal Code: " + data.address.postalCode;
  document.getElementById("address").innerHTML = addressOutput;

  // Accessing and displaying array of student objects
  var studentList = document.getElementById("students");

  data.students.forEach(function(student) {
    var listItem = document.createElement("li");
    listItem.textContent = "Name: " + student.name + ", Age: " + student.age;
    studentList.appendChild(listItem);
  });
</script>

</body>
</html>
```

✓ What This Code Does:

- Shows the `address` info from a **nested object**
- Iterates over the `students` array and creates `<li>` elements for each student

---

How to Run:

1. Copy the code above
2. Paste it into a `.html` file (e.g., `json-demo.html`)
3. Open it in your browser

What is JSON?

JSON stands for **JavaScript Object Notation**. It is a way to store and exchange data in a structured format that is easy for both humans and machines to read.

JSON is made of:

- **Name/value pairs**
- The **name** (also called a key) is always a string, written inside double quotes `" "`.
- The **value** can be different types of data, like a string, number, object, array, boolean, or null.

How does a JSON name/value pair look?

```
"name": "John"
```

- Here, `"name"` is the key (or field name).
- `"John"` is the value, which is a string.

What types of values can JSON have?

JSON values can be:

- A **string** (text in quotes)
- A **number** (like 10, 5.5)
- An **object** (another set of key/value pairs inside curly braces `{}`)
- An **array** (a list of values inside square brackets `[]`)
- A **boolean** (`true` or `false`)
- **null** (empty value)

Note: JSON does NOT support functions, dates, or undefined values. Even though JavaScript supports those, JSON does not.

Example of JSON data:

```
{
  "employees": [
    { "firstName": "John", "lastName": "Doe" },
    { "firstName": "Anna", "lastName": "Smith" },
    { "firstName": "Peter", "lastName": "Jones" }
  ]
}
```

- This JSON has one key `"employees"`, which holds an array `[]` of objects `{}`.
- Each object represents an employee with two fields: `"firstName"` and `"lastName"`.

JSON vs XML:

Both JSON and XML are used to send data from a web server to a client, but JSON is simpler and easier to read.

The XML you posted already looks like a proper XML representation of a JSON structure with an array of employees, each having firstName and lastName.

If you wanted the equivalent JSON for that XML, it would look something like this:

```
{
  "root": {
    "employees": [
      {
        "firstName": "John",
        "lastName": "Doe"
      },
      {
        "firstName": "Anna",
        "lastName": "Smith"
      },
      {
        "firstName": "Peter",
        "lastName": "Jones"
      }
    ]
  }
}
```

But since you want the XML for the JSON, your posted XML is correct:

```
<root>
  <employees>
    <firstName>John</firstName>
    <lastName>Doe</lastName>
  </employees>
  <employees>
    <firstName>Anna</firstName>
    <lastName>Smith</lastName>
  </employees>
  <employees>
    <firstName>Peter</firstName>
    <lastName>Jones</lastName>
  </employees>
</root>
```

JSON vs XML — Quick Comparison

Feature	JSON	XML
Self-describing	✅ Human-readable and self-describing	✅ Human-readable and self-describing
Structure	Hierarchical (nested objects/arrays)	Hierarchical (nested elements)
Parsing	Built-in parsing in JavaScript (`JSON.parse()`)	Requires an XML parser
Tags	No end tags, uses braces and brackets	Uses start and end tags
Size	Generally shorter and more concise	Usually more verbose
Arrays support	Supports arrays natively	Arrays simulated via repeated elements
Speed	Faster to read/write	Slower compared to JSON

Data types	String, number, object, array, boolean, null	Everything is text, with attributes for metadata

---

JSON Data Types Recap

Values in JSON **must** be one of:

- **String** — `"hello"`
- **Number** — `42`, `3.14`
- **Object** — `{ "key": "value" }`
- **Array** — `[1, 2, 3]`
- **Boolean** — `true` or `false`
- **null** — `null`

JSON Strings

- Strings in JSON must be written in **double quotes**.

Example:

```
{"name": "John"}
```

JSON Numbers

- Numbers in JSON must be an **integer** or a **floating point** (i.e., decimal).

Example:

```
{"age": 30}
```

JSON Objects

- Values in JSON can be **objects** (key-value pairs enclosed in curly braces `{}`).

Example:

```
{
  "employee": {
    "name": "John",
    "age": 30,
    "city": "New York"
  }
}
```

JSON Arrays

- Values in JSON can be **arrays** (ordered lists enclosed in square brackets `[]`).

Example:

```
{
  "employees": ["John", "Anna", "Peter"]
}
```

JSON Booleans

- Values in JSON can be **true** or **false**.

Example:

```
{
  "Submission": true
}
```

JSON Null

- Values in JSON can be **null** (to represent no value).

Example:

```
{
  "middlename": null
}
```

✅ Explanation of `JSON.parse()`

- **Purpose**: `JSON.parse()` is used to **convert a JSON-formatted string into a JavaScript object**.
- **Common Use Case**: When data is received from a web server, it's usually in the form of a string. You use `JSON.parse()` to turn it into an object you can work with in JavaScript.

🧠 Example Breakdown

✅ HTML + JavaScript Example:

```
<!DOCTYPE html>
```

```
<html>
<body>

<h2>Create Object from JSON String</h2>
<p id="demo"></p>

<script>
  // A JSON-formatted string
  var txt = '{"name":"John", "age":30, "city":"New York"}';

  // Convert JSON string to JavaScript object
  var obj = JSON.parse(txt);

  // Output the values to the HTML element
  document.getElementById("demo").innerHTML = obj.name + ", " + obj.age;
</script>

</body>
</html>
```

🔍 Output:

This script will display the following in the browser:

```
John, 30
```

HTML + JavaScript snippet. I see what you're trying to do — you're making an **AJAX request** to fetch JSON data from a file (`json_demo.txt`) and then **parse** it using `JSON.parse()`.

However, your current code has **some syntax and logic issues**, such as:

1. **Invalid inline text** in the JavaScript block (e.g., `"JSON From the Server"`).
2. **Accessing `myObj.name`** before it's defined.
3. **Improper comment format** in JavaScript.
4. **Better structure needed for readability and execution.**

✅ Cleaned-up & Working Version of Your Code:

```
<!DOCTYPE html>
<html>
<body>

<h2>Load JSON Data from Server</h2>
<p id="demo"></p>

<script>
var xmlhttp = new XMLHttpRequest();
```

```
xmlhttp.onreadystatechange = function() {
  // Check if request is complete and successful
  if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
    // Parse JSON string into a JavaScript object
    var myObj = JSON.parse(xmlhttp.responseText);

    // Display values on the page
    document.getElementById("demo").innerHTML = myObj.name + ", " + myObj.age
+ ", " + myObj.city;
  }
};

// Send GET request to json_demo.txt
xmlhttp.open("GET", "json_demo.txt", true);
xmlhttp.send();
</script>

</body>
</html>
```

📝 Your `json_demo.txt` file should contain:

```
{
  "name": "John",
  "age": 30,
  "city": "New York"
}
```

□ How it works:

1. The browser sends an HTTP GET request to fetch `json_demo.txt`.
2. When the response is ready and status is OK (200), the text is parsed into a JS object.
3. The values (`name`, `age`, `city`) are then displayed inside the `<p>` element with ID `"demo"`.

trying to parse a **JSON array** using `JSON.parse()` in JavaScript, which is correct — it will return a **JavaScript array**.

However, your current code has several syntax and structural issues:

- Comments and text are written inside the JavaScript code (like `Array as JSON`, `Create a file...`).
- You're trying to access `myArr[0]` **before** it's defined.
- There's a misplaced closing brace `}` after the array.
- Missing a check for request completion before accessing data.

✅ Correct and Clean Version of Your Code:

```
<!DOCTYPE html>
<html>
<body>

<h2>Load JSON Array from Server</h2>
<p id="demo"></p>

<script>
var xmlhttp = new XMLHttpRequest();

xmlhttp.onreadystatechange = function() {
  // Check if the request is done and successful
  if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
    // Parse JSON array into JavaScript array
    var myArr = JSON.parse(xmlhttp.responseText);

    // Display the first item of the array
    document.getElementById("demo").innerHTML = myArr[0];
  }
};

// Request JSON file containing array
xmlhttp.open("GET", "json_array_demo.txt", true);
xmlhttp.send();
</script>

</body>
</html>
```

📝 Your `json_array_demo.txt` file should contain:

```
["Ford", "BMW", "Fiat"]
```

JSON does not support Date objects directly.

🔍 Why?

In JSON, data types are limited to:

- Strings
- Numbers
- Booleans
- Arrays
- Objects
- `null`

Dates are not one of the supported data types, so they are usually stored as **ISO-formatted strings**, like `"1986-12-14"`.

✅ How to Handle Dates in JSON

If you **need to include a date**:

1. Store it as a **string** in the JSON.
2. Convert it back to a **JavaScript `Date` object** using `new Date()` after parsing.

✅ Working Example – Parsing Date from JSON String

```
<!DOCTYPE html>
<html>
<body>

<h2>Convert a JSON string into a JavaScript Date object</h2>
<p id="demo"></p>

<script>
  // JSON string with a date represented as a string
  var text = '{"name":"John", "birth":"1986-12-14", "city":"New York"}';

  // Parse the JSON string into a JavaScript object
  var obj = JSON.parse(text);

  // Convert the birth string to a real Date object
  obj.birth = new Date(obj.birth);

  // Display the result
  document.getElementById("demo").innerHTML = obj.name + ", born on " +
obj.birth.toDateString();
</script>

</body>
</html>
```

Output:

```
John, born on Sun Dec 14 1986
```

📝 Summary:

- JSON stores dates as **strings**.

- Use `new Date(string)` in JavaScript to **convert to a Date object**.
- This is a standard approach when working with JSON and date values.

Functions are not allowed in JSON.

JSON is purely a **data format**, not a way to store behavior like functions. But if you absolutely need to include a function, you can:

1. **Store it as a string**, and then
2. **Convert it back into a real function** using `eval()` (⚠ with caution).

⚠ Why This Is Risky

Using `eval()` can lead to **security risks**, especially if the content is from an **untrusted source**. It executes **any code** inside the string, including malicious scripts. Always sanitize or avoid it when possible.

✅ Working Example – Converting a String to a Function

```
<!DOCTYPE html>
<html>
<body>

<h2>Parse a function from JSON</h2>
<p id="demo"></p>

<script>
  // JSON string with a function as a string
  var text = '{"name":"John", "age":"function() {return 30;}", "city":"New
York"}';

  // Parse JSON string into object
  var obj = JSON.parse(text);

  // Convert the string to a real function
  obj.age = eval("(" + obj.age + ")");

  // Call the function and display result
  document.getElementById("demo").innerHTML = obj.name + ", age: " +
obj.age();
</script>

</body>
</html>
```

□ Output:

```
John, age: 30
```

📝 Summary:

✅ Allowed in JSON	❌ Not Allowed
Strings	Functions
Numbers	Dates (as native type)
Booleans	Undefined
Null	Comments
Arrays	
Objects	

If you must store functions or dynamic logic, it's better to handle that **outside of your JSON**, or map logic to **function names** and reference them in your code.

demonstrating the use of `JSON.stringify()` — which is the correct way to convert a **JavaScript object into a JSON string**.

But your HTML needs some cleanup for best practice and clarity.

✅ Cleaned-Up & Working Version

```
<!DOCTYPE html>
<html>
<body>

<h2>Create JSON string from a JavaScript object</h2>
<p id="demo"></p>

<script>
  // JavaScript object
  var obj = { name: "John", age: 30, city: "New York" };

  // Convert object to JSON string
  var myJSON = JSON.stringify(obj);

  // Display the JSON string
  document.getElementById("demo").innerHTML = myJSON;
</script>

</body>
</html>
```

□ Output in browser:

```
{"name":"John","age":30,"city":"New York"}
```

📝 Explanation

- `JSON.stringify()` takes a JavaScript object and converts it into a **JSON-formatted string**.
- This is useful when:
  - Sending data to a server (`XMLHttpRequest`, `fetch`)
  - Storing data in localStorage
  - Saving data in a file or database

you can use `JSON.stringify()` to convert a **JavaScript array** into a **JSON-formatted string** just like you do with objects.

Here's a **clean and complete version** of your example:

✓ Stringify a JavaScript Array – Working Example

```
<!DOCTYPE html>
<html>
<body>

<h2>Convert JavaScript Array to JSON String</h2>
<p id="demo"></p>

<script>
  // JavaScript array
  var arr = ["John", "Peter", "Sally", "Jane"];

  // Convert array to JSON string
  var myJSON = JSON.stringify(arr);

  // Display the JSON string
  document.getElementById("demo").innerHTML = myJSON;
</script>

</body>
</html>
```

□ Output in Browser:

```
["John","Peter","Sally","Jane"]
```

📝 Notes:

- `JSON.stringify()` works on arrays, objects, and nested structures.
- The resulting string is valid JSON and can be sent to a server or stored locally.
- If you want it **pretty-printed** (with indentation), you can use:

  ```
  JSON.stringify(arr, null, 2)
  ```

When you use `JSON.stringify()` on an object containing a **Date** object, the date is automatically converted into a **string** in ISO format.

Here’s a clean, complete example with explanation:

✅ Stringify Dates Example

```
<!DOCTYPE html>
<html>
<body>

<h2>Stringify JavaScript Object with Date</h2>
<p id="demo"></p>

<script>
  // JavaScript object with a Date object
  var obj = {
    name: "John",
    today: new Date(),  // current date/time
    city: "New York"
  };

  // Convert object to JSON string
  var myJSON = JSON.stringify(obj);

  // Display the JSON string
  document.getElementById("demo").innerHTML = myJSON;
</script>

</body>
</html>
```

Sample output:

```
{"name":"John","today":"2025-09-29T15:20:00.000Z","city":"New York"}
```

📝 Explanation:

- The `today` property is a **Date object** in JavaScript.
- When stringified with `JSON.stringify()`, it becomes an **ISO 8601 string** (e.g., `"2025-09-29T15:20:00.000Z"`).
- This string format can later be parsed back to a Date with `new Date()`.

in **JSON**, functions are **not allowed**. And when you use `JSON.stringify()` on an object containing functions, those function properties are **removed** from the resulting JSON string.

✅ Here's a Clean & Working Example

```
<!DOCTYPE html>
<html>
<body>

<h2>Stringify Object with Function (Function Will Be Removed)</h2>
<p id="demo"></p>

<script>
  // JavaScript object with a function
  var obj = {
    name: "John",
    age: function () { return 30; },
    city: "New York"
  };

  // Convert object to JSON string
  var myJSON = JSON.stringify(obj);

  // Display the JSON string
  document.getElementById("demo").innerHTML = myJSON;
</script>

</body>
</html>
```

Output in browser:

```
{"name":"John","city":"New York"}
```

📝 Explanation:

- `JSON.stringify()` **ignores**:
  - Functions
  - `undefined` values
  - Symbol properties
- In the above case, the `age` function is completely removed from the resulting JSON.

✅ Key Point:

JSON is meant to store **data only**, not behavior (like functions).

If you absolutely must include a function, you can store it as a **string** (like `"function() { return 30; }"`), and later convert it using `eval()` — but this is **not recommended** unless you're in full control of the data.

✅ JSON Objects – Explanation

- **JSON objects** are enclosed in **curly braces `{}`**.
- They consist of **key/value pairs**.
- **Keys** must be **strings** (wrapped in double quotes `" "`).
- **Values** can be:
  - String
  - Number
  - Boolean (`true` / `false`)
  - Array
  - Object
  - `null`
- Each **key and value** is separated by a **colon `:`**.
- Each **pair** is separated by a **comma `,`**.

✅ JSON Object Example:

```
{
  "name": "John",
  "age": 30,
  "car": null
}
```

This object contains:

- `"name"` with a string value `"John"`
- `"age"` with a number value `30`
- `"car"` with a null value `null`

📝 Notes:

- All keys must be **in double quotes** (`"key"`), unlike JavaScript object keys, which can sometimes be unquoted.
- JSON format is **language-independent** and used widely for data exchange (especially between frontend and backend systems).

✅ Accessing Object Values in JavaScript

1. Dot Notation

```
x = myObj.name;
```

- Simple and clean.
- Only works when the key is a valid identifier (no spaces, special characters, etc.).

2. Bracket Notation

```
x = myObj["name"];
```

- More flexible.
- Allows keys with spaces or special characters.
- Useful when accessing keys dynamically (e.g., from a variable).

✅ Complete Working Example:

```
<!DOCTYPE html>
<html>
<body>

<h2>Accessing Object Values</h2>
<p id="demo"></p>

<script>
  var myObj = {
    "name": "John",
    "age": 30,
    "car": null
  };

  // Access value using dot notation
  var x = myObj.name;

  // You could also use: var x = myObj["name"];

  document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

□ Output:

```
John
```

✅ Summary:

Notation	Example	Use When
Dot Notation	`obj.key`	Key is known and valid as a variable name
Bracket Notation	`obj["key"]`	Key is dynamic or contains special characters

the `**for...in**` loop in JavaScript is commonly used to iterate over the **properties of an object**.

✅ Looping Through a JSON Object Using `for...in`

Your example is almost correct, but to make it fully functional, you need to:

1. Include the `<p>` element with `id="demo"`.
2. Ensure that you use `+=` so that it appends each property name instead of replacing the content.

✅ Clean & Complete Example (EX11):

```
<!DOCTYPE html>
<html>
<body>

<h2>Looping Through Object Properties</h2>
<p id="demo"></p>

<script>
  var myObj = {
    "name": "John",
    "age": 30,
    "car": null
  };

  var x;
  for (x in myObj) {
    document.getElementById("demo").innerHTML += x + "<br>";
  }
</script>

</body>
</html>
```

□ Output:

```
name
age
car
```

📝 Explanation:

- `for (x in myObj)` loops through all **enumerable property keys** of `myObj`.
- Inside the loop, `x` is the **property name** (like `"name"`, `"age"`, etc.).
- To access the **value**, you could do `myObj[x]`.

✅ If You Want to Show Both Keys and Values:

```
for (x in myObj) {
  document.getElementById("demo").innerHTML += x + ": " + myObj[x] + "<br>";
}
```

□ Output:

```
name: John
age: 30
car: null
```

You want two things here:

1. Loop through the **values** of a JSON object using `for...in` and output them.
2. Modify a **nested value** in a JSON object using dot notation, then loop and display the updated values.

Here’s a cleaned-up, complete, and working example combining both parts:

✅ Looping Values & Modifying Nested Object Properties (EX13)

```
<!DOCTYPE html>
<html>
<body>

<h2>Modify Value in JSON and Loop through Object</h2>
<p id="demo"></p>

<script>
  var myObj = {
    "name": "John",
    "age": 30,
    "cars": {
      "car1": "Ford",
      "car2": "BMW",
      "car3": "Fiat"
    }
  };

  // Modify the nested value (car2)
  myObj.cars.car2 = "Mercedes";

  // Loop through the cars object values and build a string
  var x = "";
  for (var i in myObj.cars) {
    x += myObj.cars[i] + "<br>";
  }

  // Display the result
  document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

□ Output:

```
Ford
Mercedes
Fiat
```

📝 Explanation:

- You **modify** the nested property `car2` via dot notation:
  `myObj.cars.car2 = "Mercedes";`
- Then, loop through **all keys** of `myObj.cars` using `for...in`.
- Use **bracket notation** `myObj.cars[i]` inside the loop to get the corresponding values.
- Append each value to the string `x` with a line break.
- Finally, display the result inside the element with `id="demo"`.

✅ Deleting Object Properties (EX14)

```
<!DOCTYPE html>
<html>
<body>

<h2>Delete a Property from a JSON Object</h2>
<p id="demo"></p>

<script>
  var myObj = {
    "name": "John",
    "age": 30,
    "cars": {
      "car1": "Ford",
      "car2": "BMW",
      "car3": "Fiat"
    }
  };

  // Delete the property car2 from cars
  delete myObj.cars.car2;

  // Loop through the remaining car properties and build output string
  var x = "";
  for (var i in myObj.cars) {
    x += myObj.cars[i] + "<br>";
  }

  // Display the result
  document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

□ Output:

```
Ford
Fiat
```

📝 Explanation:

- The `delete` keyword removes the property `car2` from `myObj.cars`.
- The `for...in` loop iterates over the remaining properties (`car1` and `car3`).
- Their values are appended and displayed in the page.

✅ JSON Arrays Inside Objects & Accessing Array Values

- In JSON, **arrays** can be assigned as values to object properties.
- You can access elements inside the array by using **index numbers** (starting at 0).

Example JSON Object with Array:

```
{
  "name": "John",
  "age": 30,
  "cars": ["Ford", "BMW", "Fiat"]
}
```

Accessing Array Values in JavaScript

```
var myObj = {
  "name": "John",
  "age": 30,
  "cars": ["Ford", "BMW", "Fiat"]
};

// Access the first car
var x = myObj.cars[0];  // "Ford"
```

Complete HTML Example

```
<!DOCTYPE html>
<html>
<body>

<h2>Accessing JSON Array Values</h2>
<p id="demo"></p>

<script>
  var myObj = {
    "name": "John",
    "age": 30,
    "cars": ["Ford", "BMW", "Fiat"]
  };

  // Access the first car in the array
  var x = myObj.cars[0];

  document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

□ Output:

```
Ford
```

✅ Nested Arrays in JSON Objects — Explanation

- `cars` is an array of objects.
- Each car object has a `name` and a `models` array.
- The `models` array contains strings (car model names).

Accessing Nested Arrays & Objects Example

```
var myObj = {
  "name": "John",
  "age": 30,
  "cars": [
    { "name": "Ford", "models": ["Fiesta", "Focus", "Mustang"] },
    { "name": "BMW", "models": ["320", "X3", "X5"] },
    { "name": "Fiat", "models": ["500", "Panda"] }
  ]
};

// Example: Access the first model of the first car (Ford Fiesta)
var firstModel = myObj.cars[0].models[0];  // "Fiesta"
```

Complete HTML Example to Display All Car Models

```
<!DOCTYPE html>
<html>
<body>

<h2>Nested Arrays in JSON Objects</h2>
<p id="demo"></p>

<script>
  var myObj = {
    "name": "John",
    "age": 30,
    "cars": [
      { "name": "Ford", "models": ["Fiesta", "Focus", "Mustang"] },
      { "name": "BMW", "models": ["320", "X3", "X5"] },
      { "name": "Fiat", "models": ["500", "Panda"] }
    ]
  };

  var output = "";
  for (var i = 0; i < myObj.cars.length; i++) {
    output += myObj.cars[i].name + ":<br>";
    for (var j = 0; j < myObj.cars[i].models.length; j++) {
      output += "- " + myObj.cars[i].models[j] + "<br>";
    }
  }

  document.getElementById("demo").innerHTML = output;
</script>

</body>
</html>
```

□ Output:

```
Ford:
- Fiesta
- Focus
- Mustang
BMW:
- 320
- X3
- X5
Fiat:
- 500
- Panda
```

If you want, I can help with:

- More complex nested JSON traversals
- Adding/removing items dynamically
- Converting nested JSON to other formats

Your **EX15** example loops through a nested JSON structure and displays the car names as headers, then lists their models underneath.

Here’s the example formatted clearly and ready to use:

✓ EX15 — Looping Nested JSON Objects and Arrays

```
<!DOCTYPE html>
<html>
<body>
```

```
<h2>Nested JSON Looping Example (EX15)</h2>
<p id="demo"></p>

<script>
  var myObj = {
    "name": "John",
    "age": 30,
    "cars": [
      { "name": "Ford", "models": ["Fiesta", "Focus", "Mustang"] },
      { "name": "BMW", "models": ["320", "X3", "X5"] },
      { "name": "Fiat", "models": ["500", "Panda"] }
    ]
  };

  var x = "";

  for (var i in myObj.cars) {
    x += "<h2>" + myObj.cars[i].name + "</h2>";
    for (var j in myObj.cars[i].models) {
      x += myObj.cars[i].models[j] + "<br>";
    }
  }

  document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

What it does:

- Outer loop: iterates over `myObj.cars` array.
- For each car, it adds a `<h2>` with the car brand name.
- Inner loop: iterates over the `models` array of each car and adds each model followed by a line break.
- The final HTML is set inside the paragraph with `id="demo"`.

□ Output example in browser:

Ford
Fiesta
Focus
Mustang

BMW
320
X3
X5

Fiat
500
Panda

Exercise: Create an HTML table from JSON data using `XMLHttpRequest`.

✅ Step-by-Step: Display JSON in HTML Table

□ Step 1: Create a JSON File

Create a file named `data.json` and save the following content:

```
[
  { "name": "John", "age": 30, "city": "New York" },
  { "name": "Anna", "age": 25, "city": "London" },
  { "name": "Peter", "age": 35, "city": "Paris" }
]
```

🖥 Step 2: Create an HTML File and Load JSON via XMLHttpRequest

Create another file named `table.html` with this content:

```
<!DOCTYPE html>
<html>
<head>
  <title>JSON to Table</title>
  <style>
    table {
      width: 50%;
      border-collapse: collapse;
    }
    th, td {
      border: 1px solid #000;
      padding: 8px;
      text-align: left;
    }
    th {
      background-color: #f2f2f2;
    }
  </style>
</head>
<body>

<h2>HTML Table from JSON File</h2>
```

```
<table id="dataTable">
  <tr>
    <th>Name</th>
    <th>Age</th>
    <th>City</th>
  </tr>
</table>

<script>
  var xmlhttp = new XMLHttpRequest();
  xmlhttp.onreadystatechange = function() {
    if (this.readyState === 4 && this.status === 200) {
      var data = JSON.parse(this.responseText);
      var table = document.getElementById("dataTable");

      for (var i = 0; i < data.length; i++) {
        var row = "<tr>";
        row += "<td>" + data[i].name + "</td>";
        row += "<td>" + data[i].age + "</td>";
        row += "<td>" + data[i].city + "</td>";
        row += "</tr>";
        table.innerHTML += row;
      }
    }
  };
  xmlhttp.open("GET", "data.json", true);
  xmlhttp.send();
</script>

</body>
</html>
```

□ Result:

Once both files are saved in the **same folder**, open `table.html` in a browser.

The page will display a table like:

Name	Age	City
John	30	New York
Anna	25	London
Peter	35	Paris

⚠ Note:

Due to browser security restrictions:

- You might need to use a local server (like **VSCode Live Server**, **XAMPP**, or **Python's HTTP server**) to load local JSON files via `XMLHttpRequest`.

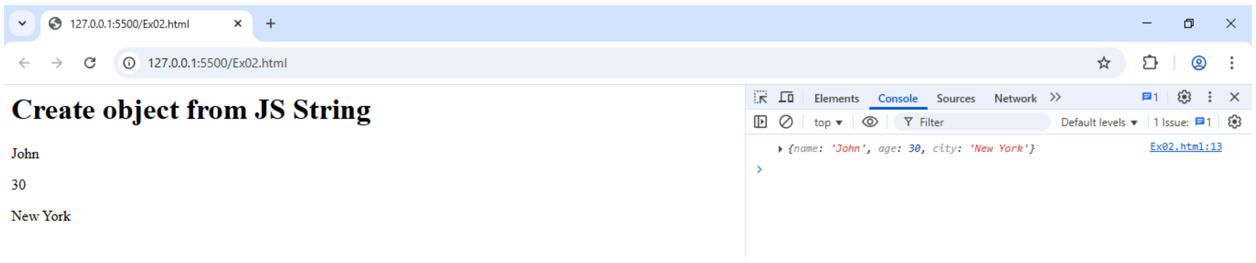
EX01

```
<html>
    <body>
        <h1>Create object from JS String</h1>
        <p id="demo"></p>
        <script>
            var text =' {"name":"Kalhara","age":25,"city":"Apmara"}';
            var obj = JSON.parse(text);
            document.getElementById("demo").innerHTML = obj.name;
        </script>
    </body>
</html>
```

Ex02

```
<html>
    <body>
        <h1>Create object from JS String</h1>
        <p id="demo"></p>
        <p id="age"></p>
        <p id="city"></p>
        <script>

            var xmlhttpRequest =new XMLHttpRequest();
            xmlhttpRequest.onreadystatechange =()=>{
                if(xmlhttpRequest.readyState ==4 & xmlhttpRequest.status ==200){
                    var myObj =JSON.parse(xmlhttpRequest.responseText);
                    console.log(myObj);
                    document.getElementById("demo").innerHTML =myObj.name;
                    document.getElementById("age").innerHTML =myObj.age;
                    document.getElementById("city").innerHTML =myObj.city;
                }
            }
            xmlhttpRequest.open("GET","json_demo.text",true);
            xmlhttpRequest.send()
        </script>
    </body>
</html>
```



Create object from JS String

John

30

New York

```
{name: 'John', age: 30, city: 'New York'}    Ex02.html:13
```

✅ Step 1: Create a JSON file (data.json)

Create a file named `data.json` in the same directory as your HTML file:

```
[
  {
    "id": 1,
    "name": "Alice",
    "email": "alice@example.com"
  },
  {
    "id": 2,
    "name": "Bob",
    "email": "bob@example.com"
  },
  {
    "id": 3,
    "name": "Charlie",
    "email": "charlie@example.com"
  }
]
```

✅ Step 2: Create an HTML file with table and use `XMLHttpRequest`

Create a file named `index.html`:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>JSON to Table</title>
  <style>
    table {
      border-collapse: collapse;
      width: 60%;
      margin: 20px auto;
    }

    th, td {
      border: 1px solid #ccc;
      padding: 10px;
      text-align: left;
    }

    th {
      background-color: #f2f2f2;
    }
  </style>
</head>
<body>

  <h2 style="text-align: center;">User Table from JSON</h2>
  <table id="userTable">
    <thead>
      <tr>
        <th>ID</th>
        <th>Name</th>
        <th>Email</th>
      </tr>
    </thead>
    <tbody>
      <!-- Data will be inserted here -->
    </tbody>
  </table>

  <script>
    var xhr = new XMLHttpRequest();
    xhr.open('GET', 'data.json', true);
    xhr.onload = function () {
      if (xhr.status === 200) {
        var users = JSON.parse(xhr.responseText);
        var tableBody = document.querySelector('#userTable tbody');

        users.forEach(function(user) {
          var row = document.createElement('tr');

          var cellId = document.createElement('td');
          cellId.textContent = user.id;
          row.appendChild(cellId);

          var cellName = document.createElement('td');
          cellName.textContent = user.name;
          row.appendChild(cellName);

          var cellEmail = document.createElement('td');
          cellEmail.textContent = user.email;
          row.appendChild(cellEmail);

          tableBody.appendChild(row);
        });
      } else {
        console.error('Failed to load JSON data');
      }
    };
    xhr.send();
  </script>
```

```
</body>
</html>
```

✅ Step 3: Run the Project

To test locally:

1. Make sure both `index.html` and `data.json` are in the same folder.
2. You **must use a local server** to load the JSON file due to browser CORS restrictions.

Run a local server:

If you have Python installed, run this command in your project folder:

- For Python 3:
- `python -m http.server`
- Open your browser and go to:
- `http://localhost:8000/index.html`

data.json

```
[
    {
      "id": 1,
      "name": "Alice",
      "email": "alice@example.com"
    },
    {
      "id": 2,
      "name": "Bob",
      "email": "bob@example.com"
    },
    {
      "id": 3,
      "name": "Charlie",
      "email": "charlie@example.com"
    }
  ]
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>JSON to Table</title>
  <style>
    table {
      border-collapse: collapse;
      width: 60%;
      margin: 20px auto;
    }

    th, td {
      border: 1px solid #ccc;
      padding: 10px;
      text-align: left;
    }

    th {
      background-color: #f2f2f2;
    }
  </style>
</head>
<body>

  <h2 style="text-align: center;">User Table from JSON</h2>
  <table id="userTable">
    <thead>
      <tr>
        <th>ID</th>
        <th>Name</th>
        <th>Email</th>
      </tr>
    </thead>
    <tbody>
      <!-- Data will be inserted here -->
    </tbody>
  </table>

  <script>
    var xhr = new XMLHttpRequest();
    xhr.open('GET', 'data.json', true);
    xhr.onload = function () {
      if (xhr.status === 200) {
        var users = JSON.parse(xhr.responseText);
        var tableBody = document.querySelector('#userTable tbody');

        users.forEach(function(user) {
          var row = document.createElement('tr');
          var cellId = document.createElement('td');
          cellId.textContent = user.id;
          row.appendChild(cellId);

          var cellName = document.createElement('td');
          cellName.textContent = user.name;
          row.appendChild(cellName);

          var cellEmail = document.createElement('td');
          cellEmail.textContent = user.email;
          row.appendChild(cellEmail);

          tableBody.appendChild(row);
        });
      } else {
        console.error('Failed to load JSON data');
      }
    };
    xhr.send();
  </script>

</body>
</html>
```