**JSON Methods in JavaScript**

JavaScript provides two main global JSON methods:

1. **JSON.stringify(value, replacer, space):** Converts a JavaScript object/array/value into a **JSON string**.

**Example:**

let user = { name: "John", age: 25 };

console.log(JSON.stringify(user));

**Output:** {"name":"John","age":25}

**1.1 JSON.stringify can be applied to primitives as well.**

JSON supports following data types:

•   Objects { ... }

•   Arrays [ ... ]

•   Primitives:

o   strings,

o   numbers,

o   boolean values true/false,

o   null.

**Example:**

console.log( JSON.stringify(1) ) // 1

// a string in JSON is still a string, but double-quoted

console.log( JSON.stringify('test') ) // "test"

console.log( JSON.stringify(true) ); // true

console.log( JSON.stringify([1, 2, 3]) ); // [1,2,3]

**1.2. JSON is data-only language-independent specification,**

So some JavaScript-specific object properties are skipped by JSON.stringify.

Namely:

Function properties (methods).

Symbolic keys and values.

Properties that store undefined.

let user = {

  sayHi() { // ignored

    console.log("Hello");

  },

  [Symbol("id")]: 123, // ignored

  something: undefined // ignored

};

console.log( JSON.stringify(user) );

**Output:**

{} //(empty object)

**1.3 the nested objects are supported and converted automatically.**

let meetup = {

  title: "Conference",

  room: {

    number: 23,

    participants: ["john", "ann"]

  }

};

console.log( JSON.stringify(meetup) );

**Output:**

{"title":"Conference","room":{"number":23,"participants":["john","ann"]}}

1. **JSON.parse(text, reviver):** Converts a **JSON string back to a JavaScript object**.

**Example-1:**

let str = '{"name":" John ","age":25}';

let obj = JSON.parse(str);

console.log(obj);

console.log(obj.name); // John

**output:**

{name: ' John ', age: 25 }

 John

**Example-2: Round-trip conversion**

let user = { name: "John Smith", age: 35 };

let jsonStr = JSON.stringify(user);

let user2 = JSON.parse(jsonStr);

console.log(user);

console.log(jsonStr);

console.log(user2);

**output:**

{ name: 'John Smith', age: 35 }

{"name":"John Smith","age":35}

{ name: 'John Smith', age: 35 }

**Example-3:**

// stringified array

let numbers = "[0, 1, 2, 4]";

numbers = JSON.parse(numbers);

console.log( numbers[3] );

let userData = '{ "name": "John", "age": 35, "isAdmin": false, "friends": [0,1,2,3] }';

let user = JSON.parse(userData);

console.log( user)

console.log( user.friends[1] );

**output:**

**4**

**{ name: 'John', age: 35, isAdmin: false, friends: [ 0, 1, 2, 3 ] }**

**1**

1. **toJSON() Method**

* The **toJSON()** method is a special function you can define inside an object.
* When JSON.stringify() is called, **JavaScript checks for toJSON() in the object**.
* If found, it uses the **return value of toJSON()** instead of the original object.

**Example 1: Basic toJSON()**

let user = {

  name: "John",

  age: 25,

  password: "secret123",

  toJSON()

  {

    // Hide password when converting to JSON

    return { name: this.name, age: this.age };

  }

};

console.log(JSON.stringify(user));

// Output: {"name":"John","age":25}

**Note:** Here, the password field is excluded because toJSON() returned a custom object.

### **Example 2: Date.toJSON()**

JavaScript’s **Date object** already has a built-in toJSON() method.  
When stringified, it converts the date to **ISO format**.

**Example:**

let today = new Date("2025-08-26T06:30:00Z");

console.log(JSON.stringify(today));

// Output: "2025-08-26T06:30:00.000Z"

/\*

Internally, it works like:

Date.prototype.toJSON = function() {

  return this.toISOString();

};

\*/

## What is ISO Format?

* **ISO** stands for **International Organization for Standardization**.
* In JavaScript (and many programming languages), when we say **ISO format**, we usually mean **ISO 8601 date and time format**.
* It is a **standardized way to represent dates and times** across systems so that there’s no confusion.

## ISO 8601 Format (Common Form)

YYYY-MM-DDTHH:mm:ss.sssZ

* **YYYY** → Year (4 digits)
* **MM** → Month (2 digits)
* **DD** → Day (2 digits)
* **T** → Separator (indicates that time starts)
* **HH** → Hours (24-hour format)
* **mm** → Minutes
* **ss** → Seconds
* **sss** → Milliseconds (optional)
* **Z** → Timezone ("Z" means UTC, Coordinated Universal Time)

**Example:**

let now = new Date();

console.log(now.toISOString());

**Output (example):**

2025-08-26T06:45:12.345Z

### **Why is ISO format useful?**

* It is **universal** → works the same everywhere (no regional confusion like MM/DD/YYYY vs DD/MM/YYYY).
* Used in **APIs, databases, JSON, logs, and cloud services**.
* JavaScript’s Date.toJSON() and Date.toISOString() always return dates in **ISO format**.

## Meaning of Z

2025-08-26T06:45:12.345Z

* Z stands for **Zulu Time**, which is another name for **UTC (Coordinated Universal Time)**.
* It means the time is expressed in **UTC timezone (zero offset)**.
* No matter where you are in the world, the time "2025-08-26T06:30:00.000Z" refers to the **same absolute moment**.

**Example 3: Nested Objects:** the date used its own toJSON() method.

let meeting = {

  title: "Team Sync",

  date: new Date(),

  location: "Hyderabad"

};

console.log(JSON.stringify(meeting));

**output:**

{"title":"Team Sync","date":"2025-08-26T00:55:12.133Z","location":"Hyderabad"}

**Summary**

* JSON.stringify() converts objects → JSON string.
* If an object has a **toJSON()** method, it will be called automatically.
* Useful for hiding sensitive data or formatting values (like Date).

**Example 4: replacer function filters properties**

let user = { name: "John", age: 25, city: "Hyderabad" };

let json = JSON.stringify(user, (key, value) => {

  if (key === "age") return undefined; // remove age

  return value;

});

console.log(json);

**Output:**

{"name":" John ","city":"Hyderabad"}

**Example 5: toJSON() + replacer together**

let user = {

  name: "Rakesh",

  age: 25,

  password: "secret123",

  toJSON() {

    // Step 1: custom serialization (password removed here)

    return { name: this.name, age: this.age };

  }

};

// Step 2: replacer function runs on the result of toJSON()

let json = JSON.stringify(user, (key, value) => {

  if (key === "age") {

    return value + " years"; // modify age format

  }

  return value;

});

console.log(json);

**Output:** {"name":"Rakesh","age":"25 years"}

## Above How it works internally

1. When JSON.stringify() is called, it first checks if the object has toJSON().
2. If yes → toJSON() is executed, and the returned object is used.
3. Then, the replacer function runs on that result to filter/transform values.

## Step-by-step Explanation

### 1. JSON.stringify(object, replacer)

* The **second argument** in JSON.stringify() is called the **replacer**.
* If the replacer is a **function**, it will be called for **every key-value pair** in the object.
* Whatever the replacer returns becomes the **value in the final JSON**.
* If it returns undefined, that key is **excluded**.

**Example-1: Fetch data from JSON file (local file)**

**data.json------JSON File**

{

  "name": "John",

  "age": 25,

  "skills": ["JavaScript", "HTML", "CSS"]

}

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <title>Fetch JSON</title>

</head>

<body>

  <h2>JSON Data Output</h2>

  <div id="output"></div>

  <script>

    // Get the JSON file from same folder

    fetch("data.json").then(response => {

        if (!response.ok) {

          throw new Error("HTTP error " + response.status);

        }

        return response.json(); // Parse JSON from response

      }).then(data => {

        console.log(data); // Log in console

        document.getElementById("output").innerHTML = `

          Name: ${data.name} <br>

          Age: ${data.age} <br>

          Skills: ${data.skills.join(", ")}

        `;

      })

      .catch(error => {

        console.error("Fetch error:", error);

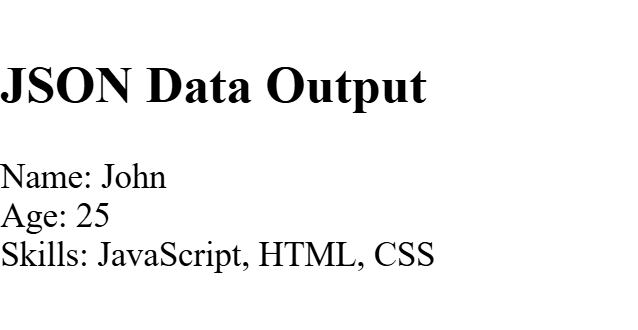
      });

  </script>

</body>

</html>

**Output:**

****

**Example-2: Embedded JSON data inside script tag:**

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <title>JSON without fetch</title>

</head>

<body>

  <h2>Student Information</h2>

  <div id="output"></div>

  <!-- Embedded JSON data inside script tag -->

  <script id="myData" type="application/json">

    {

      "name": "John",

      "age": 25,

      "course": "JavaScript",

      "skills": ["HTML", "CSS", "JavaScript"]

    }

  </script>

  <!-- Script to read and display JSON data -->

  <script>

    // Get JSON text from <script> tag and parse it

    //JSON.parse(...) takes a JSON string and converts it into a JavaScript object.

    //.textContent:This gets the raw text content inside the <script> tag.

   //In this case, it returns the JSON string:

    const jsonData = JSON.parse(document.getElementById("myData").textContent);

    // Prepare output HTML

    const outputDiv = document.getElementById("output");

    outputDiv.innerHTML = `

      <p><strong>Name:</strong> ${jsonData.name}</p>

      <p><strong>Age:</strong> ${jsonData.age}</p>

      <p><strong>Course:</strong> ${jsonData.course}</p>

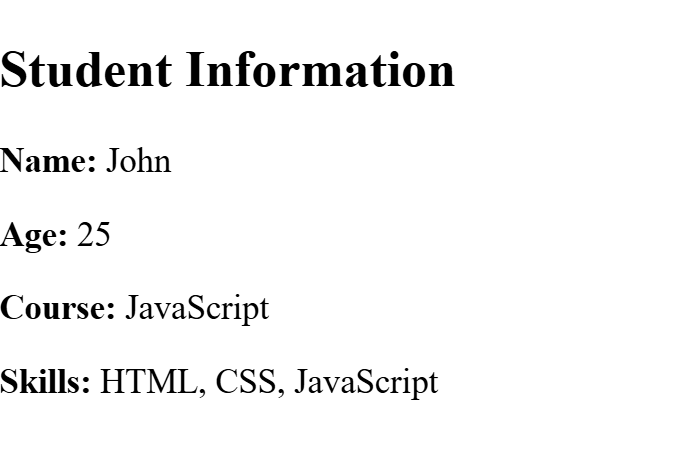
      <p><strong>Skills:</strong> ${jsonData.skills.join(", ")}</p>

    `;

  </script>

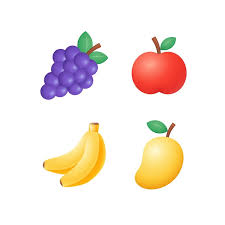
</body>

</html>

****

**Example-3: Fetch images form local JSON file**

**Images are available in images folder**

****

**data.json**

[

  { "id": 1, "name": "apple", "imageUrl": "images/1.jpeg" },

  { "id": 2, "name": "banana", "imageUrl": "images/2.jpeg" },

  { "id": 3, "name": "manago", "imageUrl": "images/3.jpeg" }

]

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <title>Load Image from JSON</title>

</head>

<body>

  <h2>Image from JSON</h2>

  <div id="imageContainer"></div>

  <script>

    async function loadImage() {

      try {

        const response = await fetch("data.json");

        if (!response.ok) {

          throw new Error("Failed to load JSON");

        }

        const data = await response.json();

        // create img element

        const img = document.createElement("img");

        img.src = data.imageUrl;

        document.getElementById("imageContainer").appendChild(img);

      } catch (error) {

        console.error("Error:", error);

      }

    }

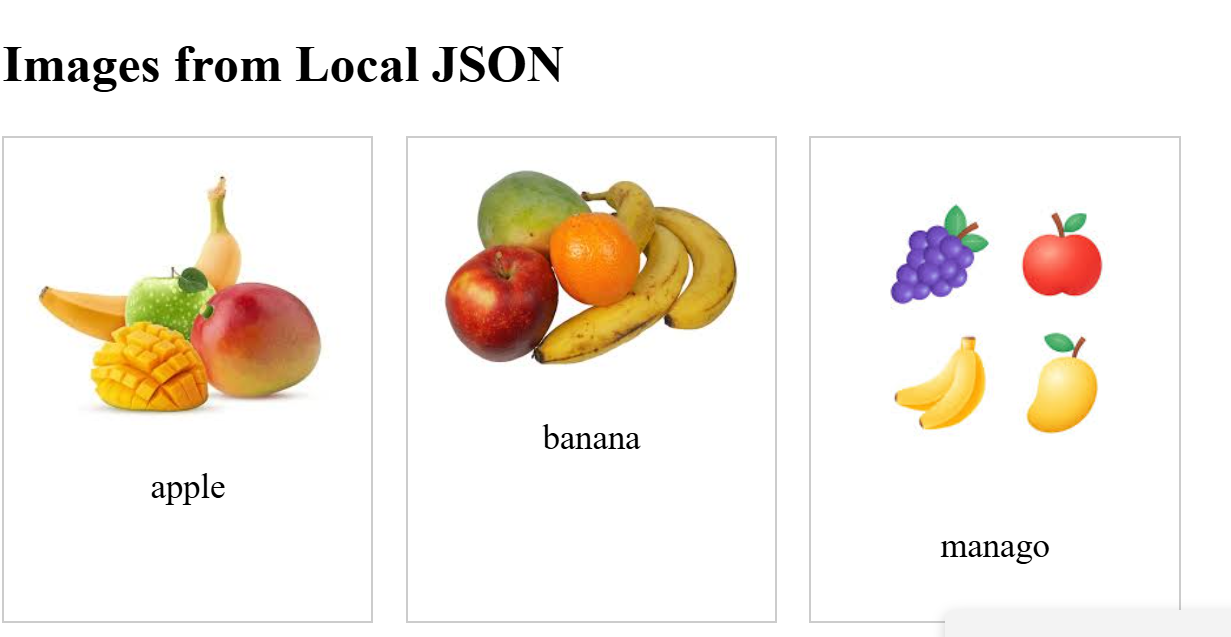
    loadImage();

  </script>

</body>

</html>

**Output:**

****