

1. JSON Overview

What is JSON?

JSON stands for **JavaScript Object Notation**.

It is a **text-based data representation format** used to store and exchange information between systems. Although its syntax is inspired by JavaScript, JSON is **language-independent** and supported by **all major programming languages** such as Java, Python, PHP, C#, Go, and many more.

Why JSON became popular?

Earlier, XML was widely used for data transfer. But JSON replaced XML because:

- JSON is **simpler and shorter**
- Processing JSON is **faster**
- JSON is **easier to read and write**
- JSON maps directly to most languages' data structures (objects, arrays, key–value pairs)

JSON plays a major role in **Modern Web Development**:

- REST APIs use JSON as the default format
- Database systems like **MongoDB** store JSON-like documents
- Mobile applications communicate using JSON
- Server-less platforms (Firebase, NoSQL DBs) use JSON for configuration

Advantages of JSON

1. **Lightweight format** → Less storage and faster network transfer
2. **Easy to read** → Human-friendly structure
3. **Structured data** → Nested objects and arrays possible
4. **Language-independent** → Support in almost every platform
5. **Easy conversion** → Built-in functions for parsing and stringifying

Where JSON is used?

- Client–server communication
 - REST and AJAX-based applications
 - Storing settings (config files)
 - NoSQL databases
 - Logging systems
 - Data interchange between microservices
-

2. JSON Syntax

JSON syntax is simple and follows strict formatting rules.

Core JSON Structure

JSON consists of two major components:

1. JSON Objects

- Represented using **curly braces** { }
- Contain **key–value pairs**
- Keys must be **strings**
- Each key is followed by a colon :
- Values can be any valid JSON type

Syntax:

```
{  
  "key1": value1,  
  "key2": value2  
}
```

2. JSON Arrays

- Represented using **square brackets** []
- Contain ordered list of values
- Values do not need to be of the same type

Syntax:

```
[  
  value1,  
  value2,  
  value3  
]
```

Rules of JSON Syntax (Explained Clearly)

1. **Keys must always be strings in double quotes**
Example: "name": "Nidhi"
Wrong: name: "Nidhi" (Not allowed)
2. **String values must also be in double quotes**
"city": "Ahmedabad"
3. **No trailing commas**
This is wrong:

```
4. {  
5.   "a": 10,  
6.   "b": 20,  
7. }
```

8. **Values can be nested objects or arrays**

9. Whitespace does not matter (indentation is optional), but used for readability.

10. JSON is strictly UTF-8 encoded, so it supports all languages.

3. JSON Data Types

JSON supports **six primary data types**, each explained in detail:

1. String

- Must always use **double quotes**
- Can contain unicode characters

Example:

```
"name": "Nidhi Panchal"
```

2. Number

- JSON numbers are not enclosed in quotes
- Can be integer or decimal
- No special formats (no NaN, no Infinity)

Example:

```
"marks": 89,  
"price": 199.50
```

3. Boolean

Only two values:

```
true  
false
```

Used to represent logical conditions.

4. Null

Represents **empty** or **missing** value.

```
"middlename": null
```

5. Object

A collection of key–value pairs enclosed in { }.

Example:

```
"address": {  
  "city": "Ahmedabad",  
  "state": "Gujarat"  
}
```

Objects can contain:

- strings
 - arrays
 - numbers
 - booleans
 - other nested objects
-

6. Array

An ordered list of values enclosed in [].

Example:

```
"skills": ["Java", "Python", "React"]
```

Arrays can contain mixed types:

```
"randomValues": ["Hello", 12, true, null, { "x": 10 }]
```

4. JSON Objects

JSON objects are similar to objects in many languages.

Structure

```
{  
  "key1": value1,  
  "key2": value2,  
  ...  
}
```

Key Rules

- Keys must be **unique**
- Keys must be **strings**
- Keys must be followed by a colon
- The order of key–value pairs does not matter

Example (Deep Explanation)

```
{  
  "studentId": 101,  
  "name": "Amit",  
  "marks": {  
    "math": 90,  
    "science": 88,  
    "english": 92  
  },  
  "hobbies": ["reading", "cricket"]  
}
```

Here:

- "studentId" is a number
- "marks" is another object inside the main object
- "hobbies" is an array
- JSON supports nesting → helps create a structured, hierarchical format

5. JSON Schema

JSON Schema is a **blueprint** or **rulebook** that defines how a JSON document should look.

Essentially, JSON Schema is used for:

- **Validation**
- **Documentation**
- **Automation**
- **Ensuring correctness of data**

What JSON Schema Can Define

- Type of data (`string`, `number`, `array`, etc.)
- Required fields
- Range of numbers
- String length limit
- Format (email, date)
- Allowed values

Detailed Example

Schema:

```
{
  "type": "object",
  "properties": {
    "name": { "type": "string" },
    "age": { "type": "number", "minimum": 1 },
    "email": { "type": "string", "format": "email" }
  },
  "required": ["name", "email"]
}
```

Explanation

- The JSON document must be an **object**
- It must contain `name` and `email` fields
- `age` must be a number and greater than 1
- `email` must be in email format

JSON Schema is widely used in:

- API testing
- Server-side input validation
- Microservices communication

6. Serializing into JSON

Serialization means converting a language-specific object (Java/Python/JS object) into a JSON-formatted string.

Why Serialization is needed?

- To send data from client to server
- To store object data in files
- To transmit information across networks
- To convert internal data structures to a standard format

In JavaScript Example

```
let student = { name: "Nidhi", age: 22 };
let jsonString = JSON.stringify(student);

console.log(jsonString);
```

Output:

```
{"name":"Nidhi","age":22}
```

Notes

- `JSON.stringify()` converts objects → JSON
 - It removes methods/functions because JSON only stores data, not behavior
-

7. Parsing JSON

Parsing means converting a JSON string into a language-specific object.

Why Parsing is necessary?

- Data received from APIs is always in **JSON string format**
- To manipulate JSON in programming, we must convert it into objects

In JavaScript

```
let jsonText = '{"name":"Nidhi","age":22}';
let obj = JSON.parse(jsonText);

console.log(obj.name); // Output: Nidhi
```

Important Notes

- If the JSON string is invalid, parsing will throw an error
- `JSON.parse` converts strings \rightarrow JavaScript objects