JSON

The format data travels in: JSON rules, JSON.parse and JSON.stringify live, reading a real API response, and the mistakes that break it.

Introduction

A JavaScript object lives inside your program — and only there. When data has to **travel** — to a server, from an API, or into storage — it needs a text uniform that every language and every machine can read. That uniform is **JSON** (JavaScript Object Notation). It *looks* like a JavaScript object, but it is plain **text**, and two built-in tools convert between the two worlds: `JSON.stringify` and `JSON.parse`. *(Roman Urdu: JSON data ka safar wala libaas hai — object ko text bana kar bhejo, text ko object bana kar istemal karo.)*

Key points

- **JSON is text, not an object:** the moment data leaves your program it travels as a string. That is why an API response arrives as text your code must convert before use.
- **Rule 1 — keys in double quotes:** every key is wrapped in `"..."`, even simple ones like `"price"`. Unquoted keys are invalid JSON.
- **Rule 2 — strings in double quotes only:** single quotes are not allowed anywhere in JSON. Numbers, `true`, `false` and `null` need no quotes.
  (Roman Urdu: JSON mein har key aur har string par double quotes lazmi hain — single quote bilkul nahi.)
- **Rule 3 — no trailing comma:** a comma after the last item breaks JSON, even though JavaScript objects forgive it.
- **Rule 4 — only data travels:** functions and `undefined` values cannot live in JSON — `stringify` simply leaves them out. JSON carries data, never behaviour.
- **`JSON.stringify` packs:** it turns a live object into JSON text — watch the double quotes appear around every key.
- **`JSON.parse` unpacks:** it turns JSON text back into a real object, so dot notation works again — `order.price` reads the value.
- **APIs speak JSON:** a real API response is JSON text, which is exactly what `fetch` will hand your code in lesson 3.22 — you will `parse` it and then read it like any object.
  (Roman Urdu: API se jo jawab aata hai woh JSON text hota hai — parse karo, phir object ki tarah parho.)

Code examples

Example 1 — stringify: watch the quotes appear

```
const chai = { name: "Masala Chai", price: 250, inStock: true };

const text = JSON.stringify(chai);

console.log(text);
console.log(typeof text);
console.log(chai.price);
```

What it does: `JSON.stringify` packs the live object into its travelling uniform. Every key gains double quotes, the string value keeps its double quotes, and the whole result is one piece of **text** — `typeof` proves it. The original object, of course, still works as an object.

What you will see: the console prints `{"name":"Masala Chai","price":250,"inStock":true}`, then `string`, then `250`.

Example 2 — parse: the text becomes an object again

```
const orderText = '{"name":"Masala Chai","price":250,"qty":2}';

const order = JSON.parse(orderText);

console.log(typeof order);
console.log(order.name);
console.log(order.price * order.qty);
```

What it does: This is the trip in reverse. `orderText` is plain text — you cannot multiply with text. `JSON.parse` reads the JSON rules, rebuilds a real object from it, and from that line on, dot notation works again: read the name, and do real maths with the price and quantity.

What you will see: the console prints `object`, then `Masala Chai`, then `500`.

Example 3 — Reading a real API response shape

```
const apiText = '{"shop":"Code Scope Cafe","products":[{"name":"Chai","price":250},
{"name":"Coffee","price":200}],"open":true}';

const data = JSON.parse(apiText);

console.log(data.shop);
console.log(data.products[0].name);
console.log(data.products[1].price);
console.log(data.open);
```

What it does: This text is shaped exactly like a real API response: an object holding text, a number-free boolean, and an **array of objects** — JSON allows arrays and objects inside each other, as deep as the data needs. After one `parse`, you walk into it with normal dot and index notation: into `products`, to index `0` for the first product, to its `name`.

What you will see: the console prints `Code Scope Cafe`, then `Chai`, then `200`, then `true`.

Quick recap

- JSON is the **text uniform** data travels in: double quotes on keys and strings, no trailing comma, no functions or `undefined`.
- `JSON.stringify` packs an object into JSON text; `JSON.parse` unpacks the text back into a usable object.
- Every API response is JSON text — parse first, then read it with dot notation.

Common mistakes

1. **Single quotes in JSON** — writing the text with single-quoted keys or values, like a JavaScript object, makes `JSON.parse` throw an error and the app stops on that line. The fix: double quotes only, for every key and every string.
2. **A trailing comma (or an unquoted key)** — the last item followed by a comma, or a key written without quotes, is forgiven in a JavaScript object but is invalid JSON, and `parse` refuses it. The fix: last item gets no comma, and every key wears double quotes. (Catching this error properly is lesson 3.23.)
3. **Using the text without parsing it** — reading a property straight from the JSON string gives `undefined`, because a string has no `price` key. The fix: always `JSON.parse` first, then use the object — pack with `stringify` when sending, unpack with `parse` when receiving.

Practice

1. Make an object `book` with keys `title`, `pages` and `free`, holding a book name, `320` and `false`. Turn it into JSON text with `stringify`, print the text, then parse it back and print the title and the page count from the parsed object.

2. Here is JSON text from an imaginary weather API: a string holding an object with keys `city` (the text `Sahiwal`), `temp` (the number `31`) and `rain` (the value `false`), all written by the JSON rules. Write it correctly in your code, parse it, and print one full sentence using a template literal: the city, its temperature and whether it is raining.

Practice quiz

1. What quotes do JSON keys and strings wear?

- Single quotes only
- Double quotes only, single quotes are invalid JSON
- Either single or double, it makes no difference
- No quotes at all

Answer: Double quotes only, single quotes are invalid JSON

2. Is a trailing comma after the last item valid JSON?

- Yes, JavaScript objects forgive it and so does JSON
- No, it breaks JSON, even though JavaScript objects forgive it
- Yes, but only for arrays
- Only when the last item is a number

Answer: No, it breaks JSON, even though JavaScript objects forgive it

3. What do JSON.stringify and JSON.parse do?

- stringify unpacks text into an object, parse packs an object into text
- stringify packs a live object into JSON text, parse unpacks JSON text back into a real object
- Both do the same thing
- Neither exists, you write JSON by hand

Answer: stringify packs a live object into JSON text, parse unpacks JSON text back into a real object

---

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

Read online: https://scopetreasure.com/notes/js-json · More free tools and notes at https://scopetreasure.com