Objects in depth in javascript - 2

🔷 1. Object Destructuring

What it is:

Destructuring is a shortcut syntax that lets you extract values from objects into individual variables.

Basic Example:

```
const user = { name: 'Geeta', age: 25 };
const { name, age } = user;

console.log(name); // 'Geeta'
```

Why it's useful:

- Makes code cleaner and more readable.
- Reduces repetition.
- Works with deeply nested objects too.

Nested Destructuring:

```
const user = {
  name: 'Anita',
  address: { city: 'Delhi', zip: 110001 }
};
const { name, address: { city, zip } } = user;

console.log(city); // 'Delhi'
```

Other features:

- You can rename variables:

```
const { name: userName } = user;
```

- You can add default values:

```
const { score = 0 } = player;
```

🔷 2. Introduction to JSON

What is JSON?

- JSON (JavaScript Object Notation) is a lightweight data format used to **store** and **exchange** data.
- Commonly used in APIs and when sending data between frontend and backend.

Looks like an object, but it's just a string:

```
// JavaScript object:
{ name: 'Raju', age: 30 }

// JSON string:
'{"name":"Raju","age":30}'
```

Main methods:

- `JSON.stringify(obj)` → Converts a JS object to JSON string.
- `JSON.parse(string)` → Converts a JSON string back to a JS object.

🔷 3. 🔥 Why do we need `JSON.stringify()`?

> ⚠️ A key concept not always obvious to beginners.

You **can't send a JavaScript object directly** in an HTTP request body (like when using `fetch`). The body must be a **string**, and that's why you need to `stringify` it.

✅ Example:

```
const userData = { name: 'Amit', age: 22 };

fetch('/api/user', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify(userData) // Must be string
});
```

If you try to send the object directly (without stringifying), it won't work — the browser doesn't know how to send it, and the server won't understand it.

✅ Analogy:

Sending a JS object is like trying to send a live animal 🦙 through email. Not possible.

But taking a photo 📸 (i.e., `JSON.stringify`) and sending that — *works perfectly*!

🔷 4. Practical Use Cases

- Destructuring:
  - Reduces code length when pulling values from props, API data, or configs.
  - Great for cleaner function arguments.
- JSON:
  - Used when storing data in localStorage.
  - Required for sending or receiving data from APIs.

💡 Tips & Best Practices

- ✅ Always `JSON.stringify()` before sending data via `fetch` or `axios`.
- ✅ Always `JSON.parse()` the response if you're expecting a JSON object.
- ✅ Use `Content-Type: application/json` header when sending JSON.
- ✅ Use destructuring to avoid repetitive code when accessing object properties.

📝 Final Revision Notes (Copy-Paste Friendly)

```
# JavaScript Objects – Part 2

## 🔹 Object Destructuring
- Extract values:
  const { name, age } = user;

- Nested:
  const { address: { city } } = user;

- Renaming:
  const { name: userName } = user;

- Default values:
  const { score = 0 } = player;

## 🔹 JSON
- JSON.stringify(obj) → JS object to JSON string
- JSON.parse(str) → JSON string to JS object

## 🔹 Why JSON.stringify() is needed:
- HTTP requests (like fetch/axios) require string data.
- JS objects must be converted to strings before sending.
- Without stringify, the request will fail or be ignored.

✅ Example:
```js
const data = { user: 'Amit', age: 22 };
fetch('/api/save', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify(data)
});
```
```

🧠 Analogy

- JS object = live pet 🐶 (can’t ship)
- JSON string = photo of pet 📸 (can send!)

🔷 Best Practices

- Always set `Content-Type: application/json` when sending JSON.
- Use destructuring to write cleaner code and extract data easily.