Perfect. Continuing in **the SAME PDF-style format, depth, and simplicity**.

📘 JavaScript Destructuring — Internal & Practical View

(Object & Array Destructuring • Memory • Defaults • Traps)

🔹 1. What is Destructuring?

Definition

Destructuring allows you to **extract values from objects or arrays** into variables **in one statement**.

👉 It's **syntax sugar** over property/index access.

Simple Meaning

Instead of:

```
js


const name = user.name;
const age = user.age;
```

You write:

```
js


const { name, age } = user;
```

🔎 Internal Behavior

- No new data structure is created
- JS performs **property lookup** (objects) or **index access** (arrays)
- Values are **assigned to new variables**
- References remain the same for objects

✅ Key Takeaway

Destructuring does **not copy objects**—it only assigns references.

⚠️ Interview Trap

Destructuring looks like cloning, but it is **not**.

🔹 2. Object Destructuring (Basics)

Syntax

```
js
```

```
const user = { name: "Anoop", age: 22 };

const { name, age } = user;
```

Output

```
js

name // "Anoop"
age  // 22
```

🔎 Internal Behavior

- JS looks for keys `"name"` and `"age"` in `user`
- Assigns found values to variables with same names

✅ Key Takeaway

Variable names must **match object keys**.

⚠️ Interview Trap

Missing keys result in `undefined`.

🔹 3. Renaming Variables (Very Common)

```
js

const user = { name: "Anoop" };

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

🔎 Internal Behavior

- Key is still `"name"`
- Variable name becomes `userName`

✅ Key Takeaway

`key: variableName` → rename during destructuring.

⚠️ Interview Trap

`userName` is NOT a property name.

🔹 4. Default Values in Object Destructuring

```
js
```

```
const user = { name: "Anoop" };

const { age = 18 } = user;
```

Output

```
js

age // 18
```

🔎 Internal Behavior

- Default applies **only if value is** `undefined`
- Does NOT apply for `null`

```
js

const obj = { x: null };
const { x = 10 } = obj;

x; // null
```

✅ Key Takeaway

Defaults trigger only for `undefined`.

⚠️ Interview Trap

`null` does NOT trigger default values.

🔹 5. Nested Object Destructuring

```
js

const user = {
  name: "Anoop",
  address: {
    city: "Delhi",
    pin: 110001
  }
};

const { address: { city } } = user;
```

🔎 Internal Behavior

- JS navigates object step by step

- No variable called `address` is created here

```
js

address; // ❌ ReferenceError
```

✅ Key Takeaway

Nested destructuring does NOT expose parent object automatically.

⚠️ Interview Trap

You must destructure parent separately if needed.

🔹 6. Array Destructuring (Basics)

```
js

const arr = [10, 20, 30];

const [a, b] = arr;
```

Output

```
js

a // 10
b // 20
```

🔎 Internal Behavior

- Index-based assignment
- Same as:

```
js

a = arr[0];
b = arr[1];
```

✅ Key Takeaway

Array destructuring depends on **order**, not names.

⚠️ Interview Trap

Skipping order gives wrong values.

🔹 7. Skipping Elements

```
js
```

```
const arr = [10, 20, 30];

const [, , c] = arr;
c; // 30
```

🔎 **Internal Behavior**

- Commas skip indexes
- No temporary variables created

✅ **Key Takeaway**

Commas control position.

⚠️ **Interview Trap**

Too many commas cause confusion.

🔹 8. Default Values in Array Destructuring

```
js

const arr = [10];

const [a, b = 20] = arr;
```

🔎 **Internal Behavior**

- `b` gets default because `arr[1]` is `undefined`

✅ **Key Takeaway**

Same default rule as objects: only for `undefined`.

🔹 9. Swapping Variables (Classic Use Case)

```
js

let x = 10;
let y = 20;

[x, y] = [y, x];
```

🔎 **Internal Behavior**

- Temporary array created
- Values reassigned in one step

✅ **Key Takeaway**

Cleaner than temp variable swapping.

⚠️ Interview Trap

Still creates a small temporary array.

🔹 10. Destructuring Function Parameters

```
js

function print({ name, age }) {
  console.log(name, age);
}

print({ name: "Anoop", age: 22 });
```

🔎 Internal Behavior

- Destructuring happens at **function entry**
- Same rules as normal destructuring

⚠️ Danger Case

```
js

print(); // ❌ TypeError
```

Safe Version

```
js

function print({ name, age } = {}) {}
```

✅ Key Takeaway

Always default destructured parameters.

🧠 FINAL SUMMARY (EXAM + INTERVIEW)

- Destructuring is syntax sugar
- Objects → key-based
- Arrays → index-based
- Defaults apply only for `undefined`
- No deep copy happens
- References remain intact

⏭️ NEXT (Same Format)

Reply with **one word**:

- `json`

- `map-set`
- `array-vs-object`
- `object-methods`
- `spread-rest`

I'll continue in **exact same simplified PDF-style**.