Perfect. I'll now **rewrite Nested Destructuring** in the **same refined, clean, deep-but-simple PDF-style** you've been following — with **clear mental models, internals, edge cases, and interview clarity**, and **no HTML noise**.

📘 Nested Destructuring in JavaScript — Deep but Simple (PDF-Style)

🔹 1. What is Nested Destructuring? (Core Idea)

Definition

Nested destructuring allows you to **extract values from nested objects and/or arrays** using a structure that **mirrors the original data shape**.

👉 Objects can contain objects
👉 Arrays can contain arrays
👉 Destructuring must **match the structure**

One-Line Meaning

> Nested destructuring = **destructuring inside destructuring**

🔎 Internal Perspective

- JavaScript walks the structure **step by step**
- Each level performs:
  - property lookup (objects)
  - index lookup (arrays)
- No data is copied (references stay intact)

✅ Key Takeaway

Nested destructuring follows the **shape of the data**, not variable names.

⚠️ Interview Trap

Wrong structure → `undefined` or runtime error.

🔹 2. Nested Object Destructuring (Basics)

Syntax

```
js


const { prop, nested: { a, b } } = obj;
```

Example

```
js
```

```
const car = {
  brand: "Hyundai",
  model: "Verna",
  info: {
    price: 1200000,
    color: "white"
  }
};

const { brand, model, info: { price, color } } = car;
```

🔎 Internal Behavior

- JS finds `info`
- Then looks inside `info` for `price` and `color`
- No variable named `info` is created here

```
js

info; // ❌ ReferenceError
```

✅ Key Takeaway

Nested destructuring **does not expose parent objects automatically**.

🔹 3. Nested Object Destructuring with Renaming

Syntax

```
js

const { prop: newName, nested: { a: x } } = obj;
```

Example

```
js

const car = {
  brand: "Hyundai",
  model: "Verna",
  info: {
    price: 1200000,
    color: "white"
  }
};
```

```
const {
  brand: company,
  model: name,
  info: { price: cost, color: carColor }
} = car;
```

🔎 **Internal Behavior**

- Lookup uses original keys
- Renaming affects only **variable names**
- Object structure remains unchanged

✅ **Key Takeaway**

Renaming applies to **variables**, not object properties.

⚠️ **Interview Trap**

Renamed variables do NOT exist on the object.

🔹 4. Nested Object Destructuring with Default Values

Why Needed

- Nested properties may be missing
- Values may be `undefined`

Example

```
js

const student = {
  firstName: "Sam",
  lastName: "Raina",
  grades: {
    English: 75
  }
};

const {
  firstName: name = "Jay",
  lastName: surname = "Shah",
  grades: {
    English: eng = 0,
    Science: sci = 0
  }
} = student;
```

🔎 Internal Behavior

- Defaults apply **only when value is** `undefined`
- Defaults do NOT apply for `null`

```
js

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

x; // null
```

✅ Key Takeaway

Defaults are triggered only by `undefined`.

🔹 5. Nested Object Destructuring with Rest Operator

Purpose

Collect remaining nested properties into a new object.

Example

```
js

const student = {
  firstName: "Kunal",
  lastName: "Karma",
  grades: {
    English: 75,
    Maths: 87,
    SocialScience: 90,
    Science: 80
  }
};

const {
  firstName,
  lastName,
  grades: { Maths, ...allGrades }
} = student;
```

🔎 Internal Behavior

- `Maths` extracted first
- Remaining properties copied into `allGrades`
- Copy is **shallow**

```js
allGrades;
// { English: 75, SocialScience: 90, Science: 80 }
```

✅ Key Takeaway

Rest creates a **new object** at that level only.

⚠️ Interview Trap

Nested rest does NOT deep-copy inner objects.

🔹 6. Nested Array Destructuring (Basics)

Syntax

```js
const [a, [b, c], d] = arr;
```

Example

```js
const arr = [10, [15, 20], 30];
const [a, [b, c], d] = arr;
```

🔎 Internal Behavior

- `a` → `arr[0]`
- `[b, c]` → `arr[1]`
- `d` → `arr[2]`

✅ Key Takeaway

Nested arrays are destructured by **position**.

🔹 7. Skipping Elements in Nested Arrays

```js
const arr = [2, [3, 4], [9, 10]];
const [a, [, b], [, c]] = arr;
```

🔎 Internal Behavior

- Commas skip unwanted indexes
- No temporary variables created

```js
```

```
a; // 2
b; // 4
c; // 10
```

✅ **Key Takeaway**

Commas control skipping at **any nesting level**.

🔹 8. Nested Array Destructuring with Default Values

```
js

const arr = [2, [3, 4], [9, 10]];

const [, [, , p = 29], [, q]] = arr;
```

🔎 **Internal Behavior**

- Missing index → `undefined`
- Default applied (`29`)
- `q` extracted normally

✅ **Key Takeaway**

Defaults work the same way as normal destructuring.

🔹 9. Nested Array Destructuring with Rest Operator

```
js

const arr = [[6, 7, 8, 9], [10, 11, 12, 13], 14];
const [[a, ...b], ...c] = arr;
```

🔎 **Internal Behavior**

- `a` → first element of first nested array
- `b` → remaining elements of first nested array
- `c` → remaining elements of parent array

```
js

a; // 6
b; // [7, 8, 9]
c; // [[10,11,12,13], 14]
```

✅ **Key Takeaway**

Rest works **independently at each nesting level**.

🔹 10. Array Inside Object (Nested Destructuring)

```js
const numbers = {
  num1: 10,
  num2: [40, 6, 5]
};

const { num1, num2: [a, b, c] } = numbers;
```

🔎 Internal Behavior

- Object destructuring first
- Array destructuring applied inside

✅ Key Takeaway

You can mix object & array destructuring freely.

🔹 11. Object Inside Array (Nested Destructuring)

```js
const numbers = [10, { p: 20, q: 30 }];
const [a, { p, q }] = numbers;
```

🔎 Internal Behavior

- Array index lookup
- Then object property lookup

✅ Key Takeaway

Structure must match exactly.

🔹 12. Common Errors & Interview Traps

❌ Accessing parent object accidentally

```js
const { info: { price } } = car;
info; // ❌ ReferenceError
```

❌ Destructuring undefined

```js
const { x: { y } } = {}; // ❌ TypeError
```

✔️ Safe version:

```js
const { x: { y } = {} } = {};
```

🧠 FINAL SUMMARY (EXAM + INTERVIEW)

- Nested destructuring follows data structure
- Objects → key-based, arrays → index-based
- Renaming and defaults work at all levels
- Rest works per nesting level
- No deep copy occurs
- Incorrect structure leads to runtime errors
- Extremely useful for APIs & configs

⏭️ NEXT (Choose One)

Reply with **one word**:

- `destructuring-edge-cases`
- `optional-chaining-vs-destructuring`
- `spread-vs-destructuring`
- `real-world-api-destructuring`
- `advanced-patterns`

I'll continue in the **same refined PDF-style**.