Perfect. I'll now **rewrite "Global Objects in JavaScript"** in the **same refined, mastery-level PDF style** you want — **simple language, internal working, clear rules, edge cases, and interview traps**, fully **NoteLM-friendly**.

📘 Global Objects in JavaScript — Deep but Simple (PDF-Style)

🔹 1. What is a Global Object?

Definition

A **global object** is a special object that holds **all globally available variables, functions, and built-in APIs** of a JavaScript runtime.

👉 Anything in **global scope**
👉 Is attached to the **global object**

One-Line Meaning

> Global object = **top-level container of a JS environment**

🔎 Internal Perspective

- When JavaScript starts, the engine:
  - Creates a **global execution context**
  - Creates a **global object**
  - Binds global identifiers to it
- This object exists **before your code runs**

✅ Key Takeaway

Every JS environment has **exactly one global object**.

🔹 2. Names of Global Object in Different Environments

Environment	Global Object Name
Browser	`window`
Node.js	`global`
Web Workers	`self`
All modern JS	`globalThis`

Why `globalThis` Exists (ES2020)

Before ES2020:

```
js
```

```
window   // browser only
global   // Node.js only
```

After ES2020:

```
js

globalThis // works everywhere
```

🔎 Internal Reason

- JS runs in **multiple environments**
- A universal reference was needed
- `globalThis` points to the **current global object**

✅ Key Takeaway

`globalThis` is the **standard global reference**.

⚠️ Interview Trap

`window !== globalThis` in Node.js.

🔹 3. Global Variables & the Global Object

`var` vs `let` vs `const`

```
js

var a = 10;
let b = 20;
const c = 30;
```

Browser Behavior

```
js

window.a; // 10 ✅
window.b; // undefined ❌
window.c; // undefined ❌
```

🔎 Internal Explanation

Keyword	Attached to Global Object?
`var`	✅ Yes

Keyword	Attached to Global Object?
`let`	❌ No
`const`	❌ No

Why?

- `var` → becomes a **property of global object**
- `let` / `const` → stored in **global lexical environment**

✅ **Key Takeaway**

Only `var` creates global object properties.

⚠️ **Interview Trap**

Global scope ≠ global object.

🔹 4. Accessing Global Variables via Global Object

Example (Browser)

```
js

var name = "Tutorialspoint";
window.name; // "Tutorialspoint"
```

🔎 **Internal Flow**

1. `var name` declared in global scope
2. Engine attaches it to `window`
3. Accessible everywhere

⚠️ **Danger**

```
js

var name = "abc"; // conflicts with window.name
```

✅ **Key Takeaway**

Using `var` globally can cause **name collisions**.

🔹 5. Global Functions & the Global Object

Example

```
js
```

```
function sum(a, b) {
  return a + b;
}

window.sum(10, 20); // 30
```

🔎 Internal Explanation

- Function declarations in global scope:
  - Are hoisted
  - Become properties of the global object

Equivalent internally:

```
js

window.sum = function(a, b) { ... }
```

✅ Key Takeaway

Global functions live on the global object.

⚠️ Interview Trap

Arrow functions assigned to `const` do NOT attach to `window`.

🔹 6. Manually Creating Global Variables (⚠️ Dangerous)

Example

```
js

function add(a, b) {
  window.result = a + b;
}

add(20, 30);
window.result; // 50
```

🔎 Internal Explanation

- You explicitly attach a property to the global object
- It becomes accessible everywhere
- It **bypasses scope safety**

⚠️ Why This Is Dangerous

- Pollutes global namespace

- Causes hidden dependencies
- Hard to debug
- Memory leaks possible

✅ **Key Takeaway**

Avoid manually adding properties to the global object.

🔹 7. Implicit Globals (Very Important Trap)

```
js

function test() {
  x = 10; // ❌ no var/let/const
}

test();
window.x; // 10
```

🔎 **Internal Explanation**

- In **non-strict mode**:
  - Undeclared variables become global
- Engine attaches them to the global object

Strict Mode Fix

```
js

"use strict";
x = 10; // ❌ ReferenceError
```

✅ **Key Takeaway**

Always use `"use strict"` to avoid implicit globals.

⚠️ **Interview Trap**

Implicit globals are a common cause of bugs.

🔹 8. Built-in Methods on Global Object

The global object already contains many utilities.

Examples

```
js

isNaN("Hello");      // true
parseInt("42");     // 42
```

```
decodeURI("%20");    // " "
```

Browser:

```
js

window.isNaN("123");
```

Node:

```
js

global.isNaN("123");
```

🔎 Internal Explanation

- These functions are **native**
- Provided by the JS runtime
- Stored directly on the global object

✅ Key Takeaway

You usually call them **without referencing the global object explicitly**.

🔹 9. Using Global Object for Feature Detection (Polyfills)

Why Needed

- Different browsers support different features
- Need safe checks before usage

Example: Promise Support

```
js

if (globalThis.Promise) {
  // Promise supported
} else {
  // load polyfill
}
```

🔎 Internal Explanation

- Checking property existence
- No execution, just detection
- Prevents runtime crashes

✅ Key Takeaway

Global object is used for **capability detection**.

🔹 10. What Should NOT Be Stored on Global Object

❌ Application state
❌ Temporary variables
❌ User data
❌ Configuration objects

Why?

- Shared across entire app
- High risk of conflicts
- Breaks modularity

🔹 11. Global Object vs Global Scope (IMPORTANT)

Concept	Meaning
Global Scope	Where variables are visible
Global Object	Object holding some global bindings

Example

```
js


let x = 10;
globalThis.x; // undefined
```

🔎 **Why?**

- `let` lives in **lexical environment**
- Not added as a property

✅ **Key Takeaway**

Not everything global lives on the global object.

🔹 12. Best Practices (Real-World Rules)

✔️ Prefer `let` / `const`
✔️ Use modules (`import / export`)
✔️ Avoid `var` in global scope
✔️ Never create implicit globals
✔️ Use `globalThis` for cross-platform code

🧠 FINAL SUMMARY (EXAM + INTERVIEW)

- Global object is the top-level JS container
- `window` (browser), `global` (Node), `globalThis` (universal)

- Only `var` attaches to global object
- `let` & `const` are global-scoped but not global-object properties
- Functions declared globally attach to global object
- Avoid polluting global object
- Use for feature detection, not data storage

⏭️ NEXT (Choose One)

Reply with **one word**:

- `this-keyword`
- `execution-context`
- `scope-vs-global-object`
- `modules-vs-global`
- `js-engine-internals`

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