

Excellent topic. I'll now rewrite "Proxy in JavaScript" in the same refined, mastery-level PDF style you've been building — concept-first, internal mechanics, clean code mental models, edge cases, interview traps, and NoteLM-friendly.

## Proxy in JavaScript — Deep but Simple (PDF-Style)

### ◆ 1. What is a Proxy in JavaScript?

#### Definition

A **Proxy** is a special object that **wraps another object (target)** and **intercepts fundamental operations** performed on it (like read, write, call, delete).

👉 Acts as a **middleman**

👉 Can **customize default behavior**

👉 Enables meta-programming

#### One-Line Meaning

Proxy = **intercept and redefine object behavior**

#### 🔍 Internal Perspective

- Every JS object has **internal operations** (get, set, call, etc.)
- A Proxy allows you to **hook into those operations**
- If a trap exists → custom logic runs
- If not → operation forwards to target

#### ✅ Key Takeaway

Proxy does not change the object — it **controls access to it**.

#### ⚠️ Interview Trap

Proxy ≠ wrapper copy. It operates on the **same target object**.

### ◆ 2. Creating a Proxy

#### Syntax

```
js  
  
const proxy = new Proxy(target, handler);
```

#### Parameters

Parameter	Meaning
<code>target</code>	Original object or function
<code>handler</code>	Object defining traps (hooks)

### Internal Flow

1. JS sees an operation on proxy
2. Checks handler for a matching trap
3. If found → execute trap
4. Else → forward to target

### ◆ 3. Basic Proxy Example (Property Access)

```
js

const person = {
  name: "Sam",
  age: 32
};

const handler = {
  get(target, prop) {
    return prop in target
      ? target[prop]
      : "Property does not exist";
  }
};

const proxyPerson = new Proxy(person, handler);

proxyPerson.name; // "Sam"
proxyPerson.height; // "Property does not exist"
```

### Internal Behavior

- `proxyPerson.height`
- JS triggers `get` trap
- Trap decides return value
- No `TypeError` is thrown

### ✓ Key Takeaway

Proxy lets you **override default** `undefined` behavior.

### ◆ 4. What Happens Without a Handler?

js

```
const proxy = new Proxy(person, {});  
proxy.name; // Same as person.name
```

### Internal Rule

- Empty handler → transparent proxy
- Behaves exactly like target

### Key Takeaway

A Proxy is inactive until traps are defined.

## ♦ 5. `get()` Trap — Intercept Property Reads

### Syntax

js

```
get(target, property, receiver)
```

### Example

js

```
const watch = {  
  brand: "Casio",  
  price: null  
};  
  
const proxy = new Proxy(watch, {  
  get(obj, prop) {  
    return obj[prop] ?? "Property is null";  
  }  
});  
  
proxy.brand; // "Casio"  
proxy.price; // "Property is null"
```

### Internal Behavior

- Triggered on:
  - `proxy.prop`
  - `proxy[prop]`
- `receiver` is usually the proxy itself

## ⚠ Interview Trap

Returning wrong types may break invariants.

### ◆ 6. `set()` Trap — Intercept Property Writes

#### Syntax

```
js

set(target, property, value, receiver)
```

#### Example

```
js

const watch = {
  price: null
};

const proxy = new Proxy(watch, {
  set(obj, prop, value) {
    if (prop === "price") {
      obj[prop] = value;
      return true;
    }
    obj[prop] = "Not Available";
    return true;
  }
});

proxy.price = 2000;
proxy.color = "Blue";
```

#### 🔍 Internal Rules

- Must return `true` for successful assignment
- Returning `false` throws `TypeError` in strict mode

#### ✅ Key Takeaway

`set()` traps can validate, restrict, or transform data.

### ◆ 7. `apply()` Trap — Intercept Function Calls

#### Works Only When Target Is a Function

#### Syntax

```
js
```

```
apply(target, thisArg, args)
```

## Example

```
js
```

```
function getDetails(watch) {  
  return `${watch.brand} ${watch.price}`;  
}
```

```
const proxyFn = new Proxy(getDetails, {  
  apply(target, thisArg, args) {  
    return target(...args).toUpperCase();  
  }  
});
```

```
proxyFn({ brand: "Casio", price: 2000 });  
// "CASIO 2000"
```

### Internal Behavior

- Triggered on:
  - `proxy()`
  - `Function.prototype.call/apply`

### Interview Trap

`apply` works only on callable targets.

## ◆ 8. Proxy for Validation (Very Common Use)

```
js
```

```
const numbers = { num1: 10 };
```

```
const proxy = new Proxy(numbers, {  
  set(obj, prop, value) {  
    if (value > obj[prop]) {  
      obj[prop] = value;  
    }  
    return true;  
  }  
});
```

```
proxy.num1 = 20; // allowed
proxy.num1 = 5; // ignored
```

### Internal Benefit

- Centralized validation logic
- No setters required on object

## ◆ 9. Proxy for Access Control (Read-Only Objects)

```
js

const proxy = new Proxy(numbers, {
  set() {
    console.log("Object is read-only");
    return false;
  }
});
```

### Internal Effect

- Prevents mutation
- Still allows reads

### Strict Mode Trap

```
js

"use strict";
proxy.num1 = 20; // ✖ TypeError
```

## ◆ 10. Side Effects with Proxy

```
js

const proxy = new Proxy(emails, {
  set(obj, prop, value) {
    if (value.includes("@")) {
      obj[prop] = value;
    }
    return true;
  }
});
```

### Internal Use

- Logging
- Validation

- Triggers
- Reactive systems

## ◆ 11. Full List of Proxy Traps (Interview Gold)

Trap	Purpose
<code>get</code>	Property read
<code>set</code>	Property write
<code>has</code>	<code>in</code> operator
<code>deleteProperty</code>	<code>delete obj.prop</code>
<code>ownKeys</code>	<code>Object.keys()</code>
<code>getPrototypeOf</code>	Prototype access
<code>setPrototypeOf</code>	Prototype mutation
<code>defineProperty</code>	<code>Object.defineProperty</code>
<code>isExtensible</code>	<code>Object.isExtensible</code>
<code>preventExtensions</code>	<code>Object.preventExtensions</code>
<code>construct</code>	<code>new proxy()</code>
<code>apply</code>	Function calls

## ◆ 12. `Proxy.revocable()` (Advanced)

js

```
const { proxy, revoke } = Proxy.revocable(target, handler);
revoke();
proxy.prop; // ❌ TypeError
```

### 🔍 Use Case

- Security
- Temporary access
- Sandbox environments

## ◆ 13. Proxy vs Object Methods

Feature	Proxy	Getters/Setters
Scope	Whole object	Per property
Dynamic	Yes	No
Central control	Yes	No
Meta-programming	Yes	Limited

## ◆ 14. Limitations & Pitfalls

- ✗ Performance overhead
- ✗ Harder debugging
- ✗ Can break engine optimizations
- ✗ Should not replace normal logic

## ◆ 15. When to Use Proxy (Real-World)

- ✓ Validation layers
- ✓ Reactive frameworks (Vue 3)
- ✓ Logging & monitoring
- ✓ Security wrappers
- ✓ Virtual objects

## 🧠 FINAL SUMMARY (EXAM + INTERVIEW)

- Proxy intercepts object operations
- Uses handler traps (`get`, `set`, `apply`, etc.)
- Enables meta-programming
- Operates on original object
- Powerful but dangerous if misused
- Backbone of modern frameworks

## ▶▶ NEXT (Choose One)

Reply with **one word**:

- `reflect-api`
- `proxy-vs-defineproperty`
- `this-keyword`
- `execution-context`
- `js-meta-programming-summary`

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