Perfect 👍

I'll start with **OBJECTS** and I'll follow **exactly the same format, depth, and simplicity** you want (PDF-style, but clearer and easier).

📘 JavaScript Objects — Internal & Practical View

🔹 1. What is an Object in JavaScript?

Definition

An **object** is a collection of **key–value pairs**.

- Keys → strings or symbols
- Values → any data type (number, string, function, object, etc.)

👉 In JavaScript, **objects are reference types**.

Syntax

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

Output

```js
user.name  // "Anoop"
user.age   // 22
```

🔎 Internal Behavior

- Object is created in **Heap Memory**
- Variable `user` is stored in **Stack**
- `user` contains a **reference (address)** to the object in heap

```pgsql
Stack          Heap
------         -------
user ————————▶ { name: "Anoop", age: 22 }
```

✅ Key Takeaway

Objects live in heap; variables store only references.

⚠️ Interview Trap

Objects are **not copied by value**.

🔹 2. Object Creation Methods

1️⃣ Object Literal (Most Common)

```
js

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

2️⃣ Using `new Object()`

```
js

const obj = new Object();
obj.x = 10;
```

3️⃣ Using Constructor Function

```
js

function User(name) {
  this.name = name;
}

const u1 = new User("Anoop");
```

🔎 Internal Behavior

- `new` does:
  1. Creates empty object
  2. Sets prototype
  3. Binds `this`
  4. Returns object

✅ Key Takeaway

All creation methods ultimately produce an object in heap.

⚠️ Interview Trap

`class` and constructor functions are **not different internally**.

🔹 3. Accessing Object Properties

Dot Notation

```js
user.name;
```

Bracket Notation

```js
user["name"];
```

🔎 Internal Behavior

- JS converts key to string internally
- `user[name]` → looks for property `"name"`

When Bracket Notation is REQUIRED

```js
const obj = {
  "full name": "Anoop"
};

obj["full name"]; // ✅
```

✅ Key Takeaway

Bracket notation allows **dynamic keys**.

⚠️ Interview Trap

`obj.key` ≠ `obj[key]`

🔹 4. Object Reference Behavior (MOST IMPORTANT)

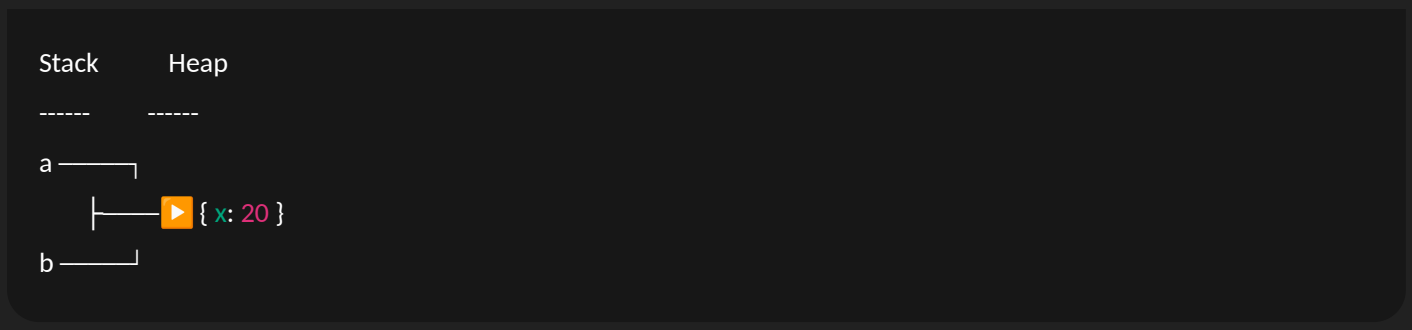
```js
const a = { x: 10 };
const b = a;

b.x = 20;

a.x; // 20
```

🔎 Internal Behavior

```css
```



- Both `a` and `b` point to **same object**
- Change via one affects the other

✅ **Key Takeaway**

Objects are passed **by reference**, not by value.

⚠️ **Interview Trap**

`const` does NOT make object immutable.

🔹 5. Comparing Objects

```
js


{} === {};  // false
```

```
js


const a = {};
const b = a;

a === b;    // true
```

🔎 **Internal Behavior**

- Comparison checks **reference**
- Not structure or content

✅ **Key Takeaway**

Objects are equal only if references are same.

⚠️ **Interview Trap**

Two identical objects are **never equal**.

🔹 6. Adding, Updating, Deleting Properties

```
js


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

```
user.age = 22;      // add
user.name = "AJ";   // update
delete user.age;    // delete
```

🔎 Internal Behavior

- Object remains same
- Only heap data is mutated
- Reference does not change

✅ Key Takeaway

Objects are mutable by default.

⚠️ Interview Trap

Deleting affects all references.

🔹 7. Shallow Copy vs Deep Copy

Shallow Copy

```
js

const obj1 = { a: 1, b: { c: 2 } };
const obj2 = { ...obj1 };

obj2.b.c = 99;
obj1.b.c; // 99
```

🔎 Internal Behavior

- Outer object copied
- Inner object reference shared

Deep Copy (Modern)

```
js

const copy = structuredClone(obj1);
```

✅ Key Takeaway

Shallow copy copies references, not nested objects.

⚠️ Interview Trap

Spread operator does NOT deep copy.

🔹 8. Object Methods & `this`

```js
const user = {
  name: "Anoop",
  greet() {
    console.log(this.name);
  }
};

user.greet(); // Anoop
```

🔎 Internal Behavior

- `this` points to object **before the dot**
- Determined at **call time**

⚠️ Arrow Function Trap

```js
const user = {
  name: "Anoop",
  greet: () => {
    console.log(this.name);
  }
};

user.greet(); // undefined
```

✅ Key Takeaway

Arrow functions do not have their own `this`.

🔹 9. Iterating Over Objects

```js
for (let key in user) {
  console.log(key, user[key]);
}
```

```js
Object.keys(user);
Object.values(user);
Object.entries(user);
```

🔎 Internal Behavior

- `for...in` iterates enumerable properties
- Includes inherited properties

⚠️ Interview Trap

Use `hasOwnProperty` when needed.

🧠 FINAL SUMMARY (EXAM + INTERVIEW)

- Objects are stored in heap
- Variables store references
- Comparison is by reference
- Mutations affect all references
- `this` depends on call-site
- Shallow copy ≠ deep copy

⏭️ NEXT (Same Format)

Reply with **one word**:

- `prototype`
- `object-destructuring`
- `array-object-difference`
- `object-methods-internal`
- `json`

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