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

📘 JavaScript Prototype — Internal & Practical View

🔹 1. What is a Prototype?

Definition

A **prototype** is an object from which **other objects inherit properties and methods**.

👉 JavaScript uses **prototypal inheritance**, not class-based inheritance.

Simple Meaning

If a property is **not found** in an object, JavaScript looks for it in its **prototype**.

🔎 Internal Behavior

- Every object has a hidden internal property:

```lua
[[Prototype]]
```

- This forms a **prototype chain**
- Lookup continues until:
  - property is found
  - or prototype is `null`

✅ Key Takeaway

Prototype is JavaScript's inheritance mechanism.

⚠️ Interview Trap

JavaScript does NOT copy properties during inheritance.

🔹 2. Prototype Chain (Property Lookup)

```js
const parent = { x: 10 };
const child = {};

Object.setPrototypeOf(child, parent);

child.x; // 10
```

🔎 Internal Behavior

Lookup steps:

1. Check `child`
2. Not found → check `parent`
3. Found → return value

```javascript
child → parent → Object.prototype → null
```

✅ **Key Takeaway**

Property lookup walks **up the prototype chain**.

⚠️ **Interview Trap**

Prototype lookup happens at **runtime**, not compile time.

🔹 3. `__proto__` vs `[[Prototype]]`

Important Distinction

Term	Meaning
`[[Prototype]]`	Internal hidden slot
`__proto__`	Getter/setter to access it

Example

```js
const obj = {};
obj.__proto__ === Object.prototype; // true
```

🔎 **Internal Behavior**

- `__proto__` is just an **accessor**
- Actual linkage is via `[[Prototype]]`

✅ **Key Takeaway**

`__proto__` exposes prototype, but is not the prototype itself.

⚠️ **Interview Trap**

Avoid using `__proto__` in production code.

🔹 4. `Object.prototype` (Root of All Objects)

```js
const obj = {};

obj.toString(); // works
```

🔎 Internal Behavior

- All normal objects inherit from `Object.prototype`
- Methods like:
  - `toString`
  - `hasOwnProperty`
  - `valueOf`

```js
Object.prototype.__proto__ === null; // true
```

✅ Key Takeaway

`Object.prototype` is the **end of the chain**.

⚠️ Interview Trap

Functions and arrays also inherit from `Object.prototype`.

🔹 5. Constructor Functions & `prototype`

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

User.prototype.greet = function () {
  console.log("Hi", this.name);
};

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

🔎 Internal Behavior

When `new User()` runs:

1. Empty object created
2. `[[Prototype]]` → `User.prototype`

3. `this` bound to new object
4. Constructor executed

```ini

u1.__proto__ === User.prototype // true
```

✅ Key Takeaway

Methods should be placed on `prototype`, not inside constructor.

⚠️ Interview Trap

Putting methods in constructor wastes memory.

🔹 6. `prototype` vs `__proto__` (VERY IMPORTANT)

Property	Belongs To	Used For
`prototype`	Constructor function	Inheritance
`__proto__`	Object instance	Prototype access

```js
u1.__proto__ === User.prototype; // true
```

✅ Key Takeaway

`prototype` builds objects, `__proto__` links them.

⚠️ Interview Trap

Only functions have `prototype`.

🔹 7. Prototypal Inheritance (Manual)

```js
const animal = {
  eats: true
};

const dog = Object.create(animal);
dog.barks = true;

dog.eats; // true
```

🔎 Internal Behavior

- `dog` has no `eats`
- JS looks in prototype → `animal`
- Found → returned

✅ **Key Takeaway**

`Object.create()` creates pure prototype linkage.

⚠️ **Interview Trap**

No constructor is called in `Object.create()`.

🔹 8. Method Shadowing

```js
const proto = { x: 10 };
const obj = Object.create(proto);

obj.x = 20;
obj.x; // 20
```

🔎 **Internal Behavior**

- Own property shadows prototype property
- Prototype property is NOT modified

✅ **Key Takeaway**

Own properties always win over prototype properties.

⚠️ **Interview Trap**

Shadowing ≠ overriding prototype value.

🔹 9. Prototype Performance Consideration

- Deep prototype chains = slower lookup
- JS optimizes common cases
- Prefer **composition over deep inheritance**

🧠 FINAL SUMMARY (EXAM + INTERVIEW)

- JavaScript uses prototypal inheritance
- Objects delegate to prototypes
- `[[Prototype]]` is internal
- `__proto__` accesses prototype
- `prototype` belongs to constructor
- Lookup is dynamic and runtime-based

⏭️ NEXT (Same Format)

Reply with **one word**:

- `destructuring`
- `object-methods`
- `array-vs-object`
- `json`
- `map-set`

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