✨ LECTURE 14 — ADVANCED LOOPS & OBJECT PROPERTY BEHAVIOR (JS)

---

❶ 🔷 `for-in` Loop — Object Iteration

📖 Definition

👉 **`for-in` loop** JavaScript me **objects ke keys (properties)** ko iterate karne ke liye use hota hai.
👉 Ye **arrays ke liye design nahi** kiya gaya hai.

📌 Golden Rule

for-in = OBJECTS
for / for-of = ARRAYS

🧠 Basic Example

```
let obj = {
  name: "Rohan",
  age: 23,
  gender: "male",
  city: "Banaras"
};

for (let key in obj) {
  console.log(key);
}
```

🟢 **Output**

```
name
age
gender
city
```

---

❷ 🔷 Accessing Values using `for-in`

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

⚠️ Important Highlight

- **`key`** → property ka naam
- **`obj[key]`** → us property ki value

📌 **Dot notation (`obj.key`) yahan fail ho jaata hai**
kyunki `key` ek **variable** hai, literal name nahi.

---

❸ 🔷 Internal Working (Step-by-Step)

🧠 JavaScript internally aise kaam karta hai:

- 1️⃣ `key = "name"` → `obj["name"] = "Rohan"`
- 2️⃣ `key = "age"` → `obj["age"] = 23`
- 3️⃣ `key = "gender"` → `obj["gender"] = "male"`
- 4️⃣ `key = "city"` → `obj["city"] = "Banaras"`

➡ **Har iteration me JS ek key pick karta hai**

---

❹ 🔷 `Object.keys()` vs `for-in`

📖 Definition

- `Object.keys(obj)` → sirf **own properties**
- `for-in` → **own + inherited (prototype) enumerable properties**

🧪 Example

```
let obj1 = { name: "Harsh", age: 20 };

let obj2 = Object.create(obj1);


obj2.money = 420;

obj2.id = "Ron";


Object.keys(obj2);
```

Output

```
['money', 'id']
```

```
for (let key in obj2) {

  console.log(key);

}
```

Output

```
money

id

name

Age
```

🧠 Real-Life Analogy

- 🏠 **Object.keys()** → *sirf apna ghar*
- 👨‍👧 **for-in** → *apna + inherited ghar*

---

❺ 🔷 Why `Object.prototype` properties don't appear?

```
Object.getOwnPropertyDescriptor(Object.prototype, "toString");


{
  writable: true,
  enumerable: false,
  configurable: true
}
```

📌 Key Reason

👉 `enumerable: false`
➡ isliye **for-in usse skip karta hai**

---

❻ 🔷 Property Descriptors — Core JS Concept

📖 Definition

Har object property ke paas **3 hidden controls** hote hain:

Attribute	Meaning
✏️ **writable**	value change ho sakti hai ya nahi
👁 **enumerable**	loop me dikhegi ya nahi

🔒 **configurable**	delete / redefine allowed hai ya nahi

✒️ writable Example

```
Object.defineProperty(obj, "name", {
  value: "rohit",
  writable: false
});
```

📌 **Result**

Value change ❌

🔒 configurable Example

```
Object.defineProperty(obj, "name", {
  configurable: false
});
```

📌 **Rule**

- delete ❌
- redefine ❌
- writable **true** → **false** ✓ allowed

👁 enumerable Example

```
Object.defineProperty(customer, "name", {
  enumerable: false
});
```

📌 **Result**

- loop me ❌
- direct access ✓

---

❼ 🔷 All-in-One BEST Example

```
Object.defineProperty(person, "id", {
  value: 101,
  writable: false,
  enumerable: false,
  configurable: false
});
```

🧠 Real-Life Analogy

 Aadhaar Number

- Change ❌
- Public list ❌
- Delete ❌

---

❽ 🔷 Why `for-in` is BAD for Arrays

```
const arr = [10, 20, 30];
arr.name = "saurav";

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

❌ **Problem**

```
0 10

1 20

2 30

name saurav
```

📌 **Reason**
Array bhi object hai → extra properties bhi iterate ho jaati hain.

✅ Correct Options

✓ `for`
✓ `for-of`
✓ `forEach`

---

❾ 🔷 Inherited Properties Example

```
let customer = { name: "Saurav", age: 21 };

let customer2 = Object.create(customer);

customer2.city = "Haridwar";


for (let key in customer2) {

  console.log(key);

}
```

Output

```
city

name

age
```

📌 **for-in inherited enumerable properties ko bhi read karta hai**

🔁 FINAL QUICK REVISION

✓ **for-in** → object keys (own + inherited)
✓ **Object.keys()** → sirf own
✓ **Descriptors** → writable | enumerable | configurable
✓ **Arrays + for-in** ❌
✓ **Prototype methods hidden (enumerable:false)**
