**Day 4 : Objects**

***A property is what an object has, while a method is what an object does.***

**🔹 1. Object Creation:**

Object = ek dabba (container) jisme data ko key:value pairs ke form me rakhte hain.

Example:

let person = {

name: "Amit",

age: 25,

city: "Mumbai"

};

Here:

* name, age, city → keys
* "Amit", 25, "Mumbai" → values

**🔹 2. Property Access**

Object ke andar se data nikalne ke 2 tarike hote hain:

1. **Dot notation**

console.log(person.name); // Amit

1. **Bracket notation**

console.log(person["age"]); // 25

👉 Dot notation tab use karte hain jab key normal ho.  
👉 Bracket notation tab use karte hain jab key me space, dash (-), ya special character ho.

## ✅ Bracket notation (special keys)

### 1. Key with space

let obj = {

"first name": "Amit"

};

console.log(obj["first name"]); // Amit

// console.log(obj.first name); ❌ Error

### 2. Key with dash (-)

let car = {

"car-model": "BMW"

};

console.log(car["car-model"]); // BMW

// console.log(car.car-model); ❌ Error (JavaScript confuse ho jayega)

### 3. Key starting with number

let marks = {

"1stRank": "Ravi"

};

console.log(marks["1stRank"]); // Ravi

// console.log(marks.1stRank); ❌ Error

### 4. Key stored in variable

let key = "city";

let address = {

city: "Delhi"

};

console.log(address[key]); // Delhi

// console.log(address.key); ❌ Undefined (yaha literal "key" ban jayega)

**🔹 3. Looping Through Object**

#### JavaScript me agar hume object ke andar ke **sabhi keys aur unki values** dekhni ho, to hum for...in loop ka use karte hain.

Object:

let obj = { a: 1, b: 2 };

**For...in loop**:

for (let key in obj) {

console.log(key, obj[key]);

}

Output:

a 1

b 2

👉 key → object ki keys = (a)  
👉 obj[key] → uski values = obj[“a”]

## Important Points to Note:

1. key hamesha **string** hoti hai, chahe tum number hi kyu na rakho.

let test = { 1: "one" };

for (let k in test) {

console.log(typeof k, k); // string "1"

}

#### obj.key likhne se undefined milega, kyunki loop ke andar key ek variable hai. Isliye hamesha **bracket notation** use karo → obj[key].

console.log(obj.key); // ❌ undefined

console.log(obj[key]); // ✅ correct

1. for...in sirf **keys iterate karta hai**. Agar tum values chahte ho direct, to Object.values() use karna padega.

let values = Object.values(obj);

console.log(values); // [1, 2]

👉 Toh easy line me:

* for...in loop → har key deta hai
* obj[key] → us key ki value deta hai.

### 🔚 Summary (ekdum short me):

| **Case** | **Kya hua** | **Result** |
| --- | --- | --- |
| obj = {…} | Naya object bana | Original nahi badla |

|  |  |  |
| --- | --- | --- |
| obj.name = 'x' | Same object me change | Original badla |

Bhai simple line me bolo toh 👉

Function me object bhejne se uska “copy” nahi jaata, uska “address” jaata hai.  
Par agar tu function ke andar **poora naya object assign karega**,  
to original variable (spaceship) par koi effect nahi hoga 🚫

**🔹 4. Merge Two Objects**

Example:

let obj1 = { a: 1, b: 2 };

let obj2 = { c: 3, d: 4 };

**Spread operator (...)**:

#### Spread operator ... ka matlab hota hai → **object ki sari properties ko "phaila dena"** (expand kar dena).

#### Agar dono objects me **same key** ho, to **baad wala object overwrite kar dega**.

let merged = { ...obj1, ...obj2 };

console.log(merged);

Output:

{ a: 1, b: 2, c: 3, d: 4 }

👉 Spread operator ek object ki properties ko dusre me dal deta hai.

**🎯 Final Practice Recap**

1. How to create object.
2. How to access properties.
3. How to loop through object.
4. How to merge two objects with spread operator.

.

**🔄 Alternative Methods (Interview point of view)**

Spread operator sabse easy hai, lekin aur bhi tarike hote hain:

1. **Object.assign()**

### let merged = Object.assign({}, obj1, obj2);

### console.log(merged);

### // { a: 1, b: 2, c: 3, d: 4 }

1. **Manual Looping (old way)**

### let merged = {};

### for (let key in obj1) {

### merged[key] = obj1[key];

### }

### for (let key in obj2) {

### merged[key] = obj2[key];

### }

### console.log(merged);

# 🔹 Additional things to add (Basic + Interview friendly)

### **1. Property Addition / Update:**

let person = { name: "Amit" };

person.age = 25; // add new property

person.name = "Ravi"; // update existing

### **2. Property Deletion:**

delete person.age; // removes age property

### **3. Check if property exists:**

console.log("name" in person); // true

console.log(person.hasOwnProperty("age")); // false

### **4. Object.keys(), Object.values(), Object.entries():**

let obj = { a: 1, b: 2 };

console.log(Object.keys(obj)); // ["a","b"]

console.log(Object.values(obj)); // [1,2]

console.log(Object.entries(obj)); // [["a",1], ["b",2]]

### **5. Nested Objects Access:**

let user = {

name: "Amit",

address: { city: "Mumbai", pin: 400001 }

};

console.log(user.address.city); // Mumbai

console.log(user["address"]["pin"]); // 400001

### **6. Computed Property Names:**

let key = "score";

let student = {

[key]: 90

};

console.log(student.score); // 90

### **7. Shallow Copy (Important for interview):**

let copy = { ...obj }; // spread operator

let copy2 = Object.assign({}, obj);

### **8. Object.freeze / Object.seal:**

### [Freeze] matlab **object ko completely lock kar dena**

### [Seal] matlab **object ki structure lock** kar dena.

### **Naye property add nahi kar sakte**, aur delete bhi nahi kar sakte.

### Lekin **existing property ka value change kar sakte ho**.

`

let obj = { a: 1 };

Object.freeze(obj);

obj.a = 2; // ❌ cannot modify

Object.seal(obj);

obj.a = 3; // ✅ can modify value

obj.b = 2; // ❌ cannot add new property

### **9. Optional Chaining (ES11+):**

console.log(user?.address?.city); // safe access, undefined if missing

### **Optional Chaining (?.)**

* Kabhi kabhi hum nested objects ke andar property access karte hain.
* Agar **koi intermediate object missing ho** → normal access se **error** aata hai.
* Optional chaining (?.) se ye **safe ho jata hai**, matlab agar koi property missing ho to **undefined return hota hai**, aur error nahi aata.

### **Example 1: Normal access → error**

let user = { name: "Amit" };

console.log(user.address.city);

// ❌ Error: Cannot read property 'city' of undefined

### **Example 2: Optional chaining → safe**

let user = { name: "Amit" };

console.log(user?.address?.city);

// ✅ undefined (no error)

**Explanation:**

* user?.address → check karta hai user.address exist karta hai ya nahi.
* Agar exist nahi karta → **undefined return**, loop/error nahi hota.
* Agar exist karta → fir .city access karta hai.

### **Example 3: Real nested object**

let user = {

name: "Amit",

address: { city: "Mumbai", pin: 400001 }

};

console.log(user?.address?.city); // Mumbai

console.log(user?.contact?.phone); // undefined (safe)

✅ **Summary:**

* ?. ka use **safe nested property access ke liye** hota hai.
* Error avoid karne ka shortcut hai, specially jab object ka structure uncertain ho.

**Index**

### **1. Object Creation**

* What is an object
* Example: let person = { name: "Amit", age: 25, city: "Mumbai" };
* Keys vs Values

### **2. Property Access**

* Dot notation
* Bracket notation
* Special Keys:
  + Key with space
  + Key with dash (-)
  + Key starting with number
  + Key stored in a variable

### **3. Looping Through Objects**

* for...in loop
* obj[key] to get value
* Object.values() for values
* Important notes (keys always string, bracket notation recommended)

### **4. Merging Objects**

* Spread operator (...)
* Object.assign()
* Manual looping

### **5. Additional Useful Operations (Basic + Interview)**

1. Property addition / update
2. Property deletion
3. Check if property exists (in, hasOwnProperty)
4. Object.keys(), Object.values(), Object.entries()
5. Nested objects access
6. Computed property names
7. Shallow copy (..., Object.assign)
8. Object.freeze / Object.seal
9. Optional chaining (?.)

### **6. Optional Chaining – Detailed**

* What it is
* Normal access → Error
* Optional chaining → Safe
* Examples
  + Missing intermediate object
  + Real nested object
* Summary / Key points