STEP 1.2 — Objects, Arrays & Memory
Learn:

- Reference vs value
- Shallow vs deep copy
- Array methods
- Object methods
- Destructuring
- Spread/rest

explain these concepts of js in depth theory + code in detail

Great 🔥 This is one of the **most important JavaScript foundations**. If you master this, your JS understanding will become **next-level**. Let's go step by step in **simple but deep** way.

🚀 STEP 1.2 — Objects, Arrays & Memory in JavaScript

✅ 1. Reference vs Value (Very Important)

🔹 What is this?

In JavaScript, variables store data in **two different ways**:

👉 **By Value**
👉 **By Reference**

This depends on **data type**.

🔸 Primitive Types → Stored by VALUE

Examples:

```javascript
Number, String, Boolean, null, undefined, Symbol, BigInt
```

🔹 How it works?

When you assign a primitive, a **copy of the value** is created.

✅ Example:

```js
let a = 10;
let b = a;

b = 20;

console.log(a); // 10
console.log(b); // 20
```

🔹 Explanation:

1. `a` stores 10.
2. `b = a` → copy of 10 is stored.
3. Changing `b` does NOT affect `a`.

👉 Because they are **independent copies**.

🔸 Non-Primitive Types → Stored by REFERENCE

Examples:

```mathematica
Objects, Arrays, Functions
```

These are stored in **heap memory**, and the variable stores a **reference (address)**.

✅ Example:

```js
let obj1 = { name: "Ghost" };
let obj2 = obj1;

obj2.name = "Byte";

console.log(obj1.name); // Byte
```

🔹 Why?

Because:

```css
obj1 → memory address → { name: "Ghost" }
obj2 → same address
```

Both variables point to the **same object**.

👉 So changing one affects the other.

⚡ Memory Visualization (Important)

```css
Stack:
a = 10

Heap:
Object stored here

obj1 → address → {name: "Ghost"}
obj2 → same address
```

🎯 Interview Tip:

👉 Always remember:

> Primitives → Copy
> Objects/Arrays → Reference

✅ 2. Shallow vs Deep Copy

This is directly connected to **reference vs value**.

🔸 Shallow Copy

A shallow copy copies **only the first level**.

Nested objects are still referenced.

✅ Example:

```js
let user = {
  name: "Ghost",
  address: {
    city: "Pune"
  }
};

let copy = { ...user };

copy.address.city = "Mumbai";

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

🔹 Why?

Because:

👉 Spread operator copies only top-level.

👉 Nested object is still **shared reference**.

🔸 Deep Copy

A deep copy creates a **completely new object**, including nested objects.

✅ Method 1: JSON (simple but limited)

```js
let deepCopy = JSON.parse(JSON.stringify(user));
```

⚠️ Limitation:

• Doesn't work with:

- Functions
- Dates
- undefined
- Map, Set

✅ Method 2: structuredClone (Modern & best)

```js
let deep = structuredClone(user);
```

Copy code

This is the **modern deep copy**.

🎯 Interview Tip:

👉 Spread operator = shallow

👉 structuredClone = deep

✅ 3. Array Methods (Core of JavaScript)

Arrays are heavily used in real-world apps and frameworks like **React**.

Let's see important ones.

🔸 map() → Transform array

Returns a new array.

```js
let nums = [1, 2, 3];

let doubled = nums.map(n => n * 2);

console.log(doubled); // [2,4,6]
```

Copy code

🔸 filter() → Select elements

```js
let nums = [1, 2, 3, 4];

let even = nums.filter(n => n % 2 === 0);
```

```
console.log(even); // [2,4]
```

Copy code

🔸 reduce() → Powerful

Used for:

- Sum
- Counting
- Complex logic

```js
let nums = [1, 2, 3];

let sum = nums.reduce((acc, curr) => acc + curr, 0);

console.log(sum); // 6
```

Copy code

🔸 find()

```js
let users = [{id:1}, {id:2}];

let user = users.find(u => u.id === 2);
```

Copy code

🔸 some() & every()

```js
nums.some(n => n > 2); // true
nums.every(n => n > 0); // true
```

Copy code

🔸 sort() (Important)

```js
[3,1,2].sort((a,b) => a - b);
```

Copy code

🎯 Interview Tip:

👉 map = transform
👉 filter = condition
👉 reduce = everything

✅ 4. Object Methods

Objects are the backbone of JavaScript.

🔸 Object.keys()

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

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

🔸 Object.values()

```js
Object.values(obj); // [1,2]
```

🔸 Object.entries()

```js
Object.entries(obj); // [['a',1],['b',2]]
```

🔸 Object.assign()

Used for shallow copy.

```js
let newObj = Object.assign({}, obj);
```

◆ hasOwnProperty()

js Copy code

```js
obj.hasOwnProperty("a");
```

✅ 5. Destructuring (Very Important in modern JS)

Used everywhere in Node.js, React, APIs.

◆ Object Destructuring

js Copy code

```js
let user = { name: "Ghost", age: 25 };

let { name, age } = user;

console.log(name);
```

◆ Rename variable

js Copy code

```js
let { name: username } = user;
```

◆ Default value

js Copy code

```js
let { city = "Pune" } = user;
```

◆ Nested destructuring

js Copy code

```js
let user = {
  address: { city: "Pune" }
};

let { address: { city } } = user;
```

🔸 Array Destructuring

```js
let [a, b] = [10, 20];
```

✅ 6. Spread & Rest Operator (Modern JS)

🔸 Spread → Expand values

Used for:

• Copy

• Merge

• Function arguments

✅ Arrays:

```js
let arr1 = [1,2];
let arr2 = [...arr1, 3, 4];
```

✅ Objects:

```js
let user = {name:"Ghost"};
let updated = {...user, age:25};
```

🔸 Rest → Collect values

Used in:

- Functions
- Destructuring

✅ Functions:

```js
function sum(...nums){
  return nums.reduce((a,b) => a+b);
}
```

✅ Destructuring:

```js
let [a, ...rest] = [1,2,3,4];

console.log(rest); // [2,3,4]
```

🔥 Difference:

Spread	Rest
Expands	Collects

🎯 Final Pro-Level Summary

👉 JavaScript memory model:

- Primitives → value
- Objects → reference

👉 Shallow copy vs deep copy:

- Spread = shallow
- structuredClone = deep

👉 Arrays:
• map, filter, reduce → must master

👉 Objects:
• keys, values, entries → used in APIs

👉 Destructuring:
• Cleaner, modern code

👉 Spread & Rest:
• Very common in modern frameworks.

If you want 🔥 I can also give you:

- 🚀 Advanced interview questions on these
- 💻 Real-world coding problems
- 🎯 Mini project to master these concepts
- 🧠 Memory tricks to never forget
- ⚡ Common mistakes & edge cases
- 🔥 Deep dive into JavaScript memory & garbage collection

Just tell me 👍