**🚀 Day 3: Arrays (map, filter, reduce,**

**forEach, find, some, every)**

**🔹 1. map()**

👉 **Kya karta hai?**

* Har element pe ek function apply karta hai
* Aur ek naya array return karta hai (original array change nahi hota)

const arr = [1, 2, 3];

const doubled = arr.map(num => num \* 2);

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

console.log(arr); // [1, 2, 3] (original same)

✔️ Example use-cases:

* Numbers ko double/triple karna
* String ko uppercase banana
* Objects se sirf ek property nikalna

**🔹 2. filter()**

👉 **Kya karta hai?**

* Har element ko check karta hai ek condition ke basis pe
* Jo elements condition pass karte hain, unka **naya array** banata hai

const arr = [1, 2, 3, 4, 5];

const evens = arr.filter(num => num % 2 === 0);

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

✔️ Example use-cases:

* Even numbers filter karna
* Sirf positive numbers nikalna
* Array me se duplicate ya unwanted data hataana

**🔹 3. reduce()**

👉 **Kya karta hai?**

* Array ko **ek single value** me reduce karta hai.
* Har step pe accumulator (acc) update hota hai.

const arr = [1, 2, 3, 4, 5];

const sum = arr.reduce((acc, curr) => acc + curr, 0);

console.log(sum); // 15

➡️ Kaise chalta hai?

* Start: acc = 0
* Step1: 0 + 1 = 1
* Step2: 1 + 2 = 3
* Step3: 3 + 3 = 6
* Step4: 6 + 4 = 10
* Step5: 10 + 5 = 15

✔️ Example use-cases:

* Array ka sum / product
* Maximum ya minimum nikalna
* Object count karna

**🔹 4. forEach()**

👉 **Kya karta hai?**

* Har element pe ek function chalata hai
* Return kuch nahi karta (sirf iterate karta hai)

1]const arr = [10, 20, 30];

arr.forEach(num => console.log(num \* 2));

// Output:

// 20

// 40

// 60

2]const arr = [10,20,30];

const result = arr.forEach(num => num \* 2);

console.log(result); // ❌ undefined (no return)

✔️ Example use-cases:

* Printing/logging
* DOM manipulat
* ion
* Har element pe side-effect karna

### 🔑 Simple Rule:

* **Sirf loop chalana hai → forEach**
* **Naya array banana hai → map**

**🔹 5. find()**

👉 **Kya karta hai?**

* Array me se **pehla element** return karta hai jo condition pass kare
* Agar koi nahi mila → undefined return karta hai

const arr = [5, 12, 8, 130, 44];

const found = arr.find(num => num > 10);

console.log(found); // 12 (kyunki ye pehla hai jo >10 hai)

✔️ Example use-cases:

* Users list me se ek user dhoondhna
* Product ka pehla matching item dhoondhna

**🔹 6. some()**

👉 **Kya karta hai?**

* Check karta hai ki koi ek bhi element condition pass karta hai ya nahi
* Result → true ya false

const arr = [1, 2, 3, 4];

const hasEven = arr.some(num => num % 2 === 0);

console.log(hasEven); // true (kyunki 2,4 even hain)

✔️ Example use-cases:

* Check karna ki array me invalid value hai ya nahi
* User list me admin present hai ya nahi

**🔹 7. every()**

👉 **Kya karta hai?**

* Check karta hai ki saare elements condition pass karte hain ya nahi
* Result → true ya false

const arr = [2, 4, 6];

const allEven = arr.every(num => num % 2 === 0);

console.log(allEven); // true (sabhi even hain)

✔️ Example use-cases:

* Sabhi students pass huye ya nahi
* Sabhi products available hain ya nahi

**🎯 Practice Summary**

**Q1. Double numbers (map):**

[1, 2, 3].map(n => n \* 2);

// [2, 4, 6]

**Q2. Sum of array (reduce):**

[1, 2, 3, 4, 5].reduce((acc, curr) => acc + curr, 0);

// 15

# 📝 ****Arrays – Practice Set :****

### **Q1. map**

Array: [1, 2, 3, 4]  
👉 Har number ko square karke naya array banao.

Output:

const arr = [1,2,3,4];

const reduceee = arr.map(*n* => n\*n);

console.log(reduceee);

### **Q2. filter**

Array: [10, 25, 30, 45, 60]  
👉 Sirf 30 se bade numbers filter karo.

Output:

const arr = [10,25,30,45,60];

const filteryyy = arr.filter(*n* => n > 30);

console.log(filteryyy);

### **Q3. reduce**

Array: [5, 10, 15]  
👉 Sab numbers ka product (multiply) nikalna hai.

Output:

const arr = [5,10,15];

const filteryyy = arr.reduce((*s*,*n*) => s+n, 0);

console.log(filteryyy);

### **Q4. forEach**

Array: ['a', 'b', 'c']  
👉 Har element ko uppercase karke console.log karo.

Output:

const arr = ['a', 'b', 'c'];

const result = arr.forEach(*x* => console.log(x.toUpperCase()));

### **Q5. find**

Array: [3, 7, 12, 18, 21]  
👉 Pehla number jo 10 se bada hai, use find karo.

Output:

const arr = [3,7,12,18,21];

const result = arr.find(*x* => x > 10);

console.log(result);

### **Q6. some**

Array: [1, 3, 5, 7, 8]  
👉 Check karo kya array me koi even number hai?

Output:

const arr = [5,3,7,1,8];

const result = arr.some(*x* => x % 2 === 0);

console.log(result);

### **Q7. every**

Array: [2, 4, 6, 8]  
👉 Check karo kya saare numbers even hain?

Output:

const arr = [2,4,6,8];

const result = arr.every(*x* => x % 2 === 0);

console.log(result);

**🚀 (ADVANCED)**

🔹 **01.Basic Utility Array Methods**

**🔹 1. includes()**

👉 Check karta hai ki koi value array me present hai ya nahi.  
Return → true ya false.

const arr = [1, 2, 3, 4];

console.log(arr.includes(2)); // true

console.log(arr.includes(5)); // false

✔️ Use case:

* Search karna ki element hai ya nahi (like username list me).

**🔹 2. indexOf() / lastIndexOf()**

👉 Kisi element ka index (position) batata hai.

* indexOf() → pehli baar kaha mila uska index
* lastIndexOf() → last baar kaha mila uska index
* Agar element nahi mila → -1 return karta hai

const arr = [10, 20, 30, 20, 40];

console.log(arr.indexOf(20)); // 1 (pehla 20 yaha hai)

console.log(arr.lastIndexOf(20)); // 3 (last 20 yaha hai)

console.log(arr.indexOf(50)); // -1 (nahi mila)

✔️ Use case:

* Element ka position nikalna
* Check karna ki element duplicate hai ya nahi

**🔹 3. join()**

👉 Array ko ek **string** me convert karta hai.  
Default separator , hota hai, lekin custom separator de sakte ho.

const arr = ["Hello", "World", "JS"];

console.log(arr.join()); // "Hello,World,JS"

console.log(arr.join(" ")); // "Hello World JS"

console.log(arr.join("-")); // "Hello-World-JS"

✔️ Use case:

* Display karna
* CSV (comma separated values) banani ho

**🔹 4. concat()**

👉 Do ya zyada arrays ko jodne ke liye.  
Original arrays **change nahi hote**, naya array return hota hai.

const a = [1, 2];

const b = [3, 4];

const c = [5, 6];

const result = a.concat(b, c);

console.log(result); // [1, 2, 3, 4, 5, 6]

console.log(a); // [1, 2] (original same)

✔️ Use case:

* Arrays merge karna (old method, ab mostly spread operator use hota hai [...a, ...b]).

**🔹 5. slice()**

👉 Array ka **part copy** karta hai (original change nahi hota).  
Syntax: arr.slice(start, end) → start inclusive, end exclusive.

const arr = [10, 20, 30, 40, 50];

console.log(arr.slice(1, 3)); // [20, 30] (index 1 se 3 tak, 3 exclude)

console.log(arr.slice(2)); // [30, 40, 50] (2 se last tak)

console.log(arr.slice(-2)); // [40, 50] (last 2 elements)

✔️ Use case:

* Array ka subset nikalna
* Copy banani ho bina original ko change kiye

**🔹 6. splice()**

👉 Array me se **add/remove/update** karne ke liye (original array change hota hai).  
Syntax: arr.splice(start, deleteCount, item1, item2, ...)

const arr = [10, 20, 30, 40, 50];

// Remove

arr.splice(2, 1);

console.log(arr); // [10, 20, 40, 50] (30 delete hua)

// Add

arr.splice(2, 0, 99, 100);

console.log(arr); // [10, 20, 99, 100, 40, 50]

// Replace

arr.splice(1, 2, 200, 300);

console.log(arr); // [10, 200, 300, 100, 40, 50]

✔️ Use case:

* Delete karna
* Insert karna
* Replace karna

⚡ Trick:

* slice() safe hai (original same rehta hai)
* splice() destructive hai (original modify hota hai)

**🎯 Summary Table**

| **Method** | **Kya karta hai** | **Original array badalta hai?** |
| --- | --- | --- |
| includes() | Check element | ❌ |
| indexOf() | Pehla index | ❌ |
| lastIndexOf() | Last index | ❌ |
| join() | String banata | ❌ |
| concat() | Arrays jodta | ❌ |
| slice() | Part copy | ❌ |
| splice() | Add/Remove/Update | ✅ |

## 🔹 02. ****Sorting & Reversing****

### ✅ sort()

#### 👉 Kya karta hai?

* Array ke elements ko **sort** karta hai.
* Default: **lexicographic (dictionary order)**, chahe numbers ho ya strings.
* Matlab "100" "20" se pehle aa jaata hai kyunki wo **string compare** hota hai.

#### 👉 Syntax:

array.sort([compareFunction]);

* compareFunction optional hai.
* Agar compareFunction diya → custom sorting possible.

#### 👉 Example 1 (Default sort):

const arr = [100, 2, 30, 4];

console.log(arr.sort());

// [100, 2, 30, 4] (wrong lag raha, string ke hisaab se sorted hai)

#### 👉 Example 2 (Custom numeric sort):

const numbers = [100, 2, 30, 4];

// Ascending

console.log(numbers.sort((a, b) => a - b));

// [2, 4, 30, 100]

// Descending

console.log(numbers.sort((a, b) => b - a));

// [100, 30, 4, 2]

#### 👉 Example 3 (Strings):

const fruits = ["banana", "apple", "cherry"];

console.log(fruits.sort());

// ["apple", "banana", "cherry"]

✔️ **Real Life Use**:

* Price low → high sort karna.
* Name list alphabetically arrange karna.
* Leaderboard points sort karna.

### ✅ reverse()

#### 👉 Kya karta hai?

* Array ke elements ko ulta kar deta hai.
* Original array modify hota hai.

#### 👉 Example:

const arr = [1, 2, 3, 4];

console.log(arr.reverse()); // [4, 3, 2, 1]

#### 👉 sort() + reverse() combo:

const names = ["John", "Zara", "Alex"];

console.log(names.sort().reverse());

// ["Zara", "John", "Alex"] (Z → A order)

✔️ **Real Life Use**:

* News articles ko latest → oldest show karna.
* Top scores ko descending order me lagana.

## 🔹 03. ****Flat & Nested Arrays****

### ✅ flat()

#### 👉 Kya karta hai?

* Nested arrays ko **flatten (ek level bana deta hai)**.
* Depth specify kar sakte ho kitna andar tak flatten karna hai.

#### 👉 Syntax:

array.flat([depth]);

* Default depth = 1
* Agar tumhe infinite depth chahiye → Infinity use karo.

#### 👉 Example 1:

const arr = [1, [2, 3], [4, [5, 6]]];

console.log(arr.flat());

// [1, 2, 3, 4, [5, 6]] (sirf ek level flat hua)

#### 👉 Example 2 (Custom depth):

const arr = [1, [2, [3, [4]]]];

console.log(arr.flat(2));

// [1, 2, 3, [4]]

console.log(arr.flat(Infinity));

// [1, 2, 3, 4]

✔️ **Real Life Use**:

* Server se nested JSON data aa jata hai → flatten karke process karna easy ho jata hai.
* 2D/3D arrays ko ek hi array me convert karna.

### ✅ flatMap()

#### 👉 Kya karta hai?

* Ye do kaam ek sath karta hai:
  1. Har element pe map() apply karega.
  2. Result ko **flat(1)** kar dega.

#### 👉 Example 1:

const arr = [1, 2, 3];

const result = arr.flatMap(x => [x, x \* 2]);

console.log(result);

// [1, 2, 2, 4, 3, 6]

👉 Yaha pehle har number ka [num, num\*2] banaya, fir flatten ho gaya.

#### 👉 Example 2 (String split):

const words = ["hello world", "hi js"];

const result = words.flatMap(str => str.split(" "));

console.log(result);

// ["hello", "world", "hi", "js"]

✔️ **Real Life Use**:

* Sentences ko words me todna.
* Array of objects se nested data extract karna.

# 📊 Quick Summary Table

| **Method** | **Work (Kya karta hai?)** | **Original Array change hota hai?** |
| --- | --- | --- |
| sort() | Lexicographic sort (ya custom comparator) | ✅ |
| reverse() | Array ulta karna | ✅ |
| flat() | Nested arrays ko flatten karna | ❌ |
| flatMap() | Map + Flat (1 level) | ❌ |

⚡ Trick to Remember:

* sort() = Default dictionary order, numbers ke liye comparator chahiye.
* reverse() = Sirf ulta karna.
* flat(depth) = Nested → simple.
* flatMap() = Shortcut of map() + flat(1).

## 🔹 04. Array Creation Tricks

## ✅ 1. Array.from()

### 👉 Kya karta hai?

* **String ya iterable** (like Set, Map, arguments, NodeList) ko array me convert karta hai.
* Optionally, ek **map function** bhi le sakta hai (like .map()).

### 👉 Syntax:

Array.from(arrayLike, mapFn, thisArg);

arrayLike → jo bhi array-like ya iterable object tum pass karte ho.

mapFn (optional) → ek function jo har element pe apply hota hai (exactly like Array.prototype.map).

thisArg (optional) → mapFn ke andar this ko set karne ke liye.

👉 Short me bole toh:  
Array.from() ek universal converter hai jo kisi bhi iterable ya array-like ko ek real Array me convert karta hai, aur uske elements ko ek hi step me transform karne ki flexibility deta hai.

### 👉 Example 1 (String → Array):

const str = "hello";

const arr = Array.from(str);

console.log(arr); // ["h", "e", "l", "l", "o"]

### 👉 Example 2 (Set → Array):

const set = new Set([1, 2, 3]);

const arr = Array.from(set);

console.log(arr); // [1, 2, 3]

### 👉 Example 3 (With map function):

const arr = Array.from([1, 2, 3], x => x \* 2);

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

✔️ **Real Life Use**:

* DOM NodeList ko array me convert karna.

*Ex: <!DOCTYPE html>*

*<html>*

*<head>*

*<title>NodeList Example</title>*

*</head>*

*<body>*

*<div class="box">Box 1</div>*

*<div class="box">Box 2</div>*

*<div class="box">Box 3</div>*

*<script>*

*// Ye NodeList milega*

*const nodeList = document.querySelectorAll(".box");*

*console.log(nodeList); // NodeList(3) [div.box, div.box, div.box]*

*// NodeList ko real Array me convert karo*

*const divArray = Array.from(nodeList);*

*console.log(divArray); // Array(3) [div.box, div.box, div.box]*

*// Ab array methods use kar sakte ho*

*const texts = divArray.map(div => div.textContent);*

*console.log(texts); // ["Box 1", "Box 2", "Box 3"]*

*</script>*

*</body>*

*</html>*

*🔹 Isme kya hua?*

*querySelectorAll(".box") → NodeList return karta hai.*

*Array.from(nodeList) → us NodeList ko ek real array bana deta hai.*

*Ab hum map() use karke sab div ka textContent nikal sakte hain.*

*👉 Yahi hota hai “DOM NodeList ko array me convert karna”.*

* Duplicate remove karke Set ko normal array me laana.
* Strings ko characters ke array me todna.

## ✅ 2. Array.of()

### 👉 Kya karta hai?

* Ye bas **arguments se ek naya array banata hai**.
* Jo values tum Array.of() me dete ho, unhi ko lekar ek array return karega.
* Isme **array-like ya iterable object ka conversion nahi hota**.

### 👉 Example 1:

console.log(Array.of(5));

// [5] (ek element wala array)

console.log(Array(5));

// [empty × 5] (5 length ka khali array)

### 👉 Example 2:

const arr = Array.of(1, 2, 3);

console.log(arr); // [1, 2, 3]

✔️ **Real Life Use**:

* Single value se bhi array banana safe ho jata hai.

## 🔥 ****Main Difference Summary****

| **Feature** | **Array.of()** | **Array.from()** |
| --- | --- | --- |
| Input | Arguments (numbers, strings, etc.) | Array-like objects or iterables |
| Output | New array with those arguments | Converted real array |
| Use | Simple array creation | Conversion + optional mapping |
| Special | Array.of(5) → [5] | Array.from("abc") → ["a","b","c"] |

## ✅ 3. new Array(length).fill(value)

### 👉 Kya karta hai?

* Naya array banata hai ek fixed **length ke sath**.
* Usko kisi ek **value se fill** kar deta hai.

### 🔹 Pehle samajh: new Array(length)

Agar tum likhte ho:

const arr = new Array(5);

console.log(arr); // [empty × 5]

Ye ek array banata hai jiska **length 5** hai, lekin values khali (undefined slots) hoti hain.  
Problem ye hai ki isme tum directly map() ya forEach() use nahi kar sakte, kyunki technically values exist nahi karti (sirf “holes” hote hain).

### 🔹 Ab kaam aata hai .fill(value)

.fill(value) ka role hai un sab “holes” ko kisi ek **default value** se bhar dena.

👉 Example:

const arr = new Array(5).fill(0);

console.log(arr); // [0, 0, 0, 0, 0]

Ab array ke andar actual values aa gayi hain, to tum map(), filter(), etc. use kar sakte ho.

### 🔹 Role / Use-Cases

1. **Pre-filled array banana**
   * Jaise agar tumhe 100 length ka array chahiye jisme sab elements 1 ho.
2. const ones = new Array(100).fill(1);
3. **Initialize karna**
   * Default value ke sath initialize karna (e.g., 0 for numbers, false for booleans, "" for strings).
4. **Quick Testing / Placeholder**
   * Testing ke liye dummy arrays banana.
5. **Range generate karna (map ke sath)**
   * Tum isko map karke index ke basis pe alag values bhi bana sakte ho:
6. const range = new Array(5).fill(0).map((\_, i) => i + 1);
7. console.log(range); // [1, 2, 3, 4, 5]

### 🔥 Ek Line Role

👉 new Array(length).fill(value) ka role hai **“ek fixed length ka array banana jisme pehle se hi koi default value bhari ho”**.

# 📊 Quick Comparison

| **Method** | **Kya karta hai?** |
| --- | --- |
| Array.from(iterable) | Iterable (string, set, map, etc.) ko array banata hai |
| Array.of(...values) | Arguments se array banata hai (safe way) |
| new Array(n).fill(value) | Fixed size + prefilled array banata hai |

⚡ **Trick to Remember**:

* Array.from() → “convert” karne ke liye.
* Array.of() → “arguments se banana” ke liye.
* new Array().fill() → “ready-made array” banane ke liye.

## 🔹 05. Higher-Level Problems (Interview Special)

## ✅ 1. Unique Elements Nikalna

👉 Question: Array me se duplicate elements hatao.

### 🔹 Method 1: Set + spread

const arr = [1, 2, 2, 3, 4, 4, 5];

const unique = [...new Set(arr)];

console.log(unique); // [1, 2, 3, 4, 5]

*🔸 Kaise kaam karta hai?*

*new Set(arr)*

*Set ek data structure hai jo sirf unique values rakhta hai.*

*Agar tum duplicate daloge, to Set automatically usko ignore kar dega.*

*Yaha new Set([1,2,2,3,4,4,5]) → {1,2,3,4,5} ban jayega.*

*[...new Set(arr)]*

*Spread (...) ka matlab hai Set ke elements ko ek ek karke nikal kar array me dalna.*

*Result: [1, 2, 3, 4, 5]*

### 🔹 Method 2: filter + indexOf

const unique2 = arr.filter((val, i) => arr.indexOf(val) === i);

console.log(unique2); // [1, 2, 3, 4, 5]

✔️ Interview me mostly Set method ko expect karte hain, kyunki O(n) me ho jaata hai.

## ✅ 2. Array of Objects pe map, filter, reduce

👉 Array:

const users = [

{ name: "A", age: 20 },

{ name: "B", age: 25 },

{ name: "C", age: 30 }

];

### 🔹 Example 1: Sirf names nikalna (map)

const names = users.map(u => u.name);

console.log(names); // ["A", "B", "C"]

### 🔹 Example 2: Age > 21 wale (filter)

const adults = users.filter(u => u.age > 21);

console.log(adults); // [{ name: "B", age: 25 }, { name: "C", age: 30 }]

### 🔹 Example 3: Total age (reduce)

const totalAge = users.reduce((sum, u) => sum + u.age, 0);

console.log(totalAge); // 75

✔️ Ye bahut common hai — “array of objects me se kuch property nikalna, filter karna, ya total calculate karna.”

## ✅ 3. Grouping / Counting Frequency

👉 Question: Count karo ki har element kitni baar aaya hai.

### 🔹 Example:

const arr = ["apple", "banana", "apple", "orange", "banana", "apple"];

const freq = arr.reduce((acc, fruit) => {

acc[fruit] = (acc[fruit] || 0) + 1;

return acc;

}, {});

console.log(freq);

// { apple: 3, banana: 2, orange: 1 }

✔️ Interviewer aksar yeh puchte hain:

* “Word frequency count” *👉 Matlab ek sentence ya array of words me* ***kitni baar kaunsa word aaya*** *uska record banana. (The above example.)*
* “Most repeated element” *👉 Matlab array me jo element* ***sabse zyada baar aaya*** *usko find karna.*
* “Group objects by category” 👉 objects ko ek field ke basis pe group karna.

## ✅ 4. Deep Copy vs Shallow Copy

👉 Ye ek concept-based sawal hai. Arrays/objects copy karne me problem hoti hai.

### 🔹 Shallow Copy (sirf ek level tak copy karta hai)

* slice()
* spread operator ([...])

const arr = [[1, 2], [3, 4]];

const shallow = [...arr];

shallow[0][0] = 99;

console.log(arr); // [[99, 2], [3, 4]] (original bhi badal gaya!)

👉 Kyunki inner arrays **reference se copy hue**.

### 🔹 Deep Copy (poora alag copy banata hai)

* JSON.parse(JSON.stringify(obj)) (simple objects ke liye best).

const arr = [[1, 2], [3, 4]];

const deep = JSON.parse(JSON.stringify(arr));

deep[0][0] = 99;

console.log(arr); // [[1, 2], [3, 4]] (original safe hai!)

console.log(deep); // [[99, 2], [3, 4]]

✔️ Interview me difference samjhna zaroori hai.

* Shallow copy: fast, but reference ke issues.
* Deep copy: safe, but thoda heavy (JSON method nested objects ke liye hi kaam karta hai).

# 📊 Quick Recap (Interview Special)

| **Problem** | **Technique** |
| --- | --- |
| Unique elements | Set + spread |
| Array of objects ops | map, filter, reduce |
| Frequency count | reduce |
| Copy array/object | Shallow → slice, spread  Deep → JSON.parse(JSON.stringify()) |

⚡ **Most asked interview tasks**:

* Duplicate remove

*const arr = [1, 2, 2, 3, 4, 4, 5];*

*// Method 1: Set*

*const unique = [...new Set(arr)];*

*console.log(unique); // [1, 2, 3, 4, 5]*

*// Method 2: filter + indexOf*

*const unique2 = arr.filter((val, i) => arr.indexOf(val) === i);*

*console.log(unique2); // [1, 2, 3, 4, 5].*

* Word frequency

*const words = ["apple", "banana", "apple", "orange", "banana", "apple"];*

*const freq = words.reduce((acc, w) => {*

*acc[w] = (acc[w] || 0) + 1;*

*return acc;*

*}, {});*

*console.log(freq);*

*// { apple: 3, banana: 2, orange: 1 }*

* Array of objects aggregation (e.g. total salary of employees)

*const employees = [*

*{ name: "A", salary: 1000 },*

*{ name: "B", salary: 2000 },*

*{ name: "C", salary: 3000 }*

*];*

*// Total salary*

*const total = employees.reduce((sum, emp) => sum + emp.salary, 0);*

*console.log(total); // 6000*

*// Sirf names nikalna*

*const names = employees.map(e => e.name);*

*console.log(names); // ["A", "B", "C"]*

*// Salary > 1500 wale*

*const highPaid = employees.filter(e => e.salary > 1500);*

*console.log(highPaid);*

*// [ { name: "B", salary: 2000 }, { name: "C", salary: 3000 } ]*

* Shallow vs Deep copy

*const obj = { name: "A", address: { city: "Pune" } };*

*// Shallow copy*

*const shallow = { ...obj };*

*shallow.address.city = "Mumbai";*

*console.log(obj.address.city); // "Mumbai" ❌ (original bhi change ho gaya)*

*// Deep copy*

*const deep = JSON.parse(JSON.stringify(obj));*

*deep.address.city = "Delhi";*

*console.log(obj.address.city); // "Mumbai" ✅ (original safe hai)*

**🚀 Day 3: JavaScript Arrays – Quick Guide**

**1️⃣ Iteration & Transformation**

| **Method** | **Kya karta hai** | **Return** | **Original Array** |
| --- | --- | --- | --- |
| map() | Har element pe function apply karta hai | New array | ❌ (unchanged) |
| filter() | Condition pass karne wale elements nikalta hai | New array | ❌ |
| reduce() | Array → single value | Value | ❌ |
| forEach() | Har element pe function chalata hai | undefined | ❌ |
| find() | Pehla matching element return | Element / undefined | ❌ |
| some() | Kya koi element match karta hai? | true/false | ❌ |
| every() | Kya sab elements match karte? | true/false | ❌ |

**Tip:**

* Loop → forEach
* Naya array chahiye → map
* Condition ke elements → filter
* Aggregate value → reduce

**2️⃣ Utility Methods**

| **Method** | **Kya karta hai** | **Original Array** |
| --- | --- | --- |
| includes(val) | Check element | ❌ |
| indexOf(val) | First index | ❌ |
| lastIndexOf(val) | Last index | ❌ |
| join(sep) | Array → string | ❌ |
| concat(arr1, arr2) | Arrays merge | ❌ |
| slice(start,end) | Part copy | ❌ |
| splice(start,delCount,items...) | Add/Remove/Replace | ✅ |

**3️⃣ Sorting & Reversing**

* sort() → default lexicographic, custom comparator for numbers
* reverse() → array ulta kar deta hai

**Example:**

[100,2,30,4].sort((a,b)=>a-b); // [2,4,30,100]

[1,2,3,4].reverse(); // [4,3,2,1]

**4️⃣ Nested Arrays**

| **Method** | **Kya karta hai** | **Original Array** |
| --- | --- | --- |
| flat(depth) | Nested → flat | ❌ |
| flatMap(fn) | map + flat(1) | ❌ |

**5️⃣ Array Creation Tricks**

| **Method** | **Kya karta hai** | **Example** |
| --- | --- | --- |
| Array.from(iterable,mapFn?) | Iterable → array | Array.from("abc") // ["a","b","c"] |
| Array.of(...values) | Arguments → array | Array.of(5,10) // [5,10] |
| new Array(n).fill(val) | Fixed length + fill | new Array(5).fill(0) // [0,0,0,0,0] |

**6️⃣ Higher-Level Problems (Interview)**

* **Unique elements:**

[...new Set(arr)]

* **Array of objects:** map, filter, reduce
* **Frequency count:**

arr.reduce((acc, x)=> (acc[x]=(acc[x]||0)+1, acc), {})

* **Shallow vs Deep copy:**

// Shallow

[...arr]

// Deep

JSON.parse(JSON.stringify(arr))

💡 **Pro Tips**

* slice() → safe, splice() → modifies original
* map() vs forEach() → naya array chahiye ya nahi
* flatMap() → map + flatten ek hi step me

## Arrays(built-in properties):

[.at()](https://www.codecademy.com/resources/docs/javascript/arrays/at)

Returns the element at a specified index in an array.

[.concat()](https://www.codecademy.com/resources/docs/javascript/arrays/concat)

Merges, or concatenates, two or more arrays.

[.copyWithin()](https://www.codecademy.com/resources/docs/javascript/arrays/copyWithin)

Returns a mutated array with part of it copied to another location in the same array, and its length unchanged.

[.entries()](https://www.codecademy.com/resources/docs/javascript/arrays/entries)

Returns an iterator with key/value pairs for each index in the array.

[.every()](https://www.codecademy.com/resources/docs/javascript/arrays/every)

Checks if all elements in an array satisfy the condition specified by the given function.

[.fill()](https://www.codecademy.com/resources/docs/javascript/arrays/fill)

Changes all elements within a range of indices in an array to a static value.

[.filter()](https://www.codecademy.com/resources/docs/javascript/arrays/filter)

Creates a new array containing the elements from the original array that pass a test implemented by a provided function.

[.find()](https://www.codecademy.com/resources/docs/javascript/arrays/find)

Returns the first element in the array that satisfies the given function.

[.findIndex()](https://www.codecademy.com/resources/docs/javascript/arrays/findIndex)

Returns the first index that passes the callback function's test. Returns -1 if no element passes the test.

[.findLast()](https://www.codecademy.com/resources/docs/javascript/arrays/findLast)

Returns the last instance of an element in an array that meets the specified condition.

[.findLastIndex()](https://www.codecademy.com/resources/docs/javascript/arrays/findLastIndex)

Iterates through the array in reverse order and returns the index that passes the provided testing function.

[.flatMap()](https://www.codecademy.com/resources/docs/javascript/arrays/flatMap)

Returns a new array formed by applying a callback function to each element of the original array, then flattening the result by one level.

[.forEach()](https://www.codecademy.com/resources/docs/javascript/arrays/forEach)

Loops over a given array, passing each item in the array into the callback function provided.

[.from()](https://www.codecademy.com/resources/docs/javascript/arrays/from)

Creates a new Array instance from an iterable or array-like object.

[.includes()](https://www.codecademy.com/resources/docs/javascript/arrays/includes)

Returns true if a given value is included in an array.

[.indexOf()](https://www.codecademy.com/resources/docs/javascript/arrays/indexOf)

Returns the first index at which a specified element can be found in an array, or -1 if not present.

[.isArray()](https://www.codecademy.com/resources/docs/javascript/arrays/isArray)

Returns true for arrays, otherwise false.

[.join()](https://www.codecademy.com/resources/docs/javascript/arrays/join)

Elements of an array are converted to strings and concatenated together, returning the resulting string.

[.keys()](https://www.codecademy.com/resources/docs/javascript/arrays/keys)

Returns a new array iterator object containing the keys for each index in the array.

[.lastIndexOf()](https://www.codecademy.com/resources/docs/javascript/arrays/lastIndexOf)

Returns the last index at which an element can be found.

[.length](https://www.codecademy.com/resources/docs/javascript/arrays/length)

Returns the specific number of elements in the array.

[.map()](https://www.codecademy.com/resources/docs/javascript/arrays/map)

Creates a new array with the results of calling a function for every element in array.

[.pop()](https://www.codecademy.com/resources/docs/javascript/arrays/pop)

Removes the last element of an array, decrements the array length, and returns the value that it removed.

[.push()](https://www.codecademy.com/resources/docs/javascript/arrays/push)

Adds one or more elements to the end of the array and returns the new length.

[.reduce()](https://www.codecademy.com/resources/docs/javascript/arrays/reduce)

Executes a reducer function on each element of an array, resulting in a single output value.

[.reverse()](https://www.codecademy.com/resources/docs/javascript/arrays/reverse)

Reverses the order of the elements of an array in place and returns the reversed array.

[.shift()](https://www.codecademy.com/resources/docs/javascript/arrays/shift)

Removes and returns the first element of the array. All subsequent elements will shift down one place.

[.slice()](https://www.codecademy.com/resources/docs/javascript/arrays/slice)

Returns a shallow copy of a part of an array. It contains references to the sliced elements in the original array, not the elements themselves.

[.some()](https://www.codecademy.com/resources/docs/javascript/arrays/some)

Runs a conditional through an array and returns a boolean if any value fulfills the conditional.

[.sort()](https://www.codecademy.com/resources/docs/javascript/arrays/sort)

Sorts the elements of an array in place.

[.splice()](https://www.codecademy.com/resources/docs/javascript/arrays/splice)

Modifies an array by inserting, deleting, and/or replacing array elements then returns an array of deleted elements.

[.toLocaleString()](https://www.codecademy.com/resources/docs/javascript/arrays/toLocaleString)

Converts array elements to localized string representations and joins them with locale-specific separators.

[.toReversed()](https://www.codecademy.com/resources/docs/javascript/arrays/toReversed)

Reverses the elements within the array and returns a new copy of the array.

[.toSorted()](https://www.codecademy.com/resources/docs/javascript/arrays/toSorted)

Takes an array and returns a new array with all the elements sorted in ascending order.

[.toSpliced()](https://www.codecademy.com/resources/docs/javascript/arrays/toSpliced)

Returns a new array with deleted, replaced, or inserted values at the given index.

[.toString()](https://www.codecademy.com/resources/docs/javascript/arrays/toString)

Returns a string with each of the array values, separated by commas. Does not mutate the original array.

[.unshift()](https://www.codecademy.com/resources/docs/javascript/arrays/unshift)

Adds one or more elements to beginning of array and returns new length.

[.valueOf()](https://www.codecademy.com/resources/docs/javascript/arrays/valueOf)

Returns the value of all the elements of the original array.

[.values()](https://www.codecademy.com/resources/docs/javascript/arrays/values)

Returns a new array iterator object that contains the values of each element in the array.

[.with()](https://www.codecademy.com/resources/docs/javascript/arrays/with)

Returns a copy of an array with the given modification.

[flat()](https://www.codecademy.com/resources/docs/javascript/arrays/flat)

Creates a new array with all sub-array elements recursively concatenated into it up to the specified depth.

[reduceRight()](https://www.codecademy.com/resources/docs/javascript/arrays/reduceRight)

Applies a reducer function to array elements from right to left, accumulating a single output value.

💡 Conclusion:

1. **Iteration & Transformation methods**
   * map(), filter(), reduce(), forEach(), find(), some(), every() ✅
2. **Utility / Searching methods**
   * includes(), indexOf(), lastIndexOf(), join(), concat(), slice(), splice() ✅
3. **Sorting & Reversing**
   * sort(), reverse() ✅
4. **Nested arrays**
   * flat(), flatMap() ✅
5. **Array Creation Tricks**
   * Array.from(), Array.of(), new Array().fill() ✅
6. **Higher-Level / Interview problems**
   * Unique elements, Array of objects, Frequency count, Shallow vs Deep copy ✅
7. **Extra built-in array methods**
   * .at(), .copyWithin(), .entries(), .findIndex(), .findLast(), .findLastIndex(), .isArray(), .keys(), .pop(), .push(), .shift(), .toLocaleString(), .toReversed(), .toSorted(), .toSpliced(), .toString(), .unshift(), .valueOf(), .values(), .with(), .reduceRight() ✅