JavaScript Arrays — Complete Guide (Beginner → Advanced)

This is a full, structured, clear, practical guide to **every important JavaScript array concept and method**, starting from absolute basics and rising gently toward advanced usage. Examples included throughout.

---

1. What Is an Array?

An **array** is an ordered list of values. Each value has a position called an index.

```
const items = ["apple", "banana", "mango"]; // indexes: 0,1,2
```

You use arrays when you want to store multiple values together.

---

2. array.length

`length` tells you how many items are inside an array.

```
const names = ["A", "B", "C"];
names.length; // 3
```

3. push() — Add to End

```
array.push(value)
```

Adds a new item at the end.

```
const nums = [1, 2];
nums.push(3); // [1, 2, 3]
```

4. pop() — Remove From End

```
array.pop()
```

Removes the last item.

```
const nums = [1, 2, 3];
nums.pop(); // [1, 2]
```

5. unshift() — Add to Start

```
array.unshift(value)
```

Adds an item at the beginning.

```
const arr = [2, 3];
arr.unshift(1); // [1, 2, 3]
```

6. shift() — Remove From Start

```
array.shift()
```

Removes the first item.

```
const arr = [1, 2, 3];
arr.shift(); // [2, 3]
```

7. indexOf() — Find Index of Value

```
array.indexOf(value)
```

Returns index of a value or `-1` if not found.

```
const a = ["x", "y", "z"];
a.indexOf("y"); // 1
```

8. includes() — Check If Value Exists

```
array.includes(value)
```

Returns true/false.

```
const nums = [10, 20, 30];
nums.includes(20); // true
nums.includes(40); // false
```

9. slice() — Copy Part of Array (Non-Destructive)

```
array.slice(start, end)
```

- Doesn't modify original. - `start` inclusive - `end` exclusive

Examples:

```
const nums = [10, 20, 30, 40];
nums.slice();       // [10,20,30,40]
nums.slice(1);      // [20,30,40]
nums.slice(1, 3);   // [20,30]
```

---

10. splice() — Add/Remove Items (Destructive)

```
array.splice(start, deleteCount, ...itemsToAdd)
```

Changes the original.

Examples:

```
const colors = ["red", "green", "blue"];
colors.splice(1, 1); // remove 1 at index 1 → ["red", "blue"]
colors.splice(1, 0, "yellow"); // insert → ["red", "yellow", "blue"]
```

---

11. forEach() — Simple Iteration

```
array.forEach((element, index, array) => {})
```

Runs a function for each element. Does NOT return anything.

```
["a", "b", "c"].forEach((item, idx) => {
  console.log(idx, item);
});
```

12. map() — Transform Array

```
array.map((element, index, array) => newValue)
```

Creates a new array of transformed values.

```
const nums = [1, 2, 3];
const doubled = nums.map(n => n * 2);
// [2,4,6]
```

13. filter() — Keep Matching Values

```
array.filter((element, index, array) => condition)
```

Keeps only elements that return true.

```
const nums = [5, 10, 15, 2];
nums.filter(n => n > 5); // [10,15]
```

14. reduce() — Reduce to One Value

```
array.reduce((acc, element, index, array) => newAcc, initialValue)
```

Used to calculate totals, products, objects, etc.

```
const nums = [1, 2, 3, 4];
nums.reduce((sum, n) => sum + n, 0); // 10
```

15. reverse() — Reverse Array Order

```
array.reverse()
```

Modifies original.

```
const a = [1,2,3];
a.reverse(); // [3,2,1]
```

16. flat() — Flatten Nested Arrays

```
array.flat(depth)
```

```
const arr = [1, [2, [3, 4]]];
arr.flat();    // [1,2,[3,4]]
arr.flat(2);   // [1,2,3,4]
```

---

17. find() — Get First Matching Value

```
array.find(callback)
```

```
const users = [{id:1},{id:2},{id:3}];
users.find(u => u.id === 2); // {id:2}
```

18. findIndex() — Index of First Match

```
array.findIndex(callback)
```

```
const nums = [10, 20, 30];
nums.findIndex(n => n > 15); // 1
```

19. some() — At Least One Match?

```
array.some(callback)
```

```
[1,2,3].some(n => n > 2); // true
```

20. every() — All Must Match

```
array.every(callback)
```

```
[2,4,6].every(n => n % 2 === 0); // true
```

21. concat() — Join Arrays

```
array1.concat(array2)
```

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

22. join() — Convert Array to String

```
array.join(separator)
```

```
["a","b","c"].join("-"); // "a-b-c"
```

23. sort() — Sort Array

```
array.sort(compareFunction)
```

Warning: modifies original.

```
[3,1,2].sort(); // [1,2,3]
```

24. fill() — Replace All Values

```
array.fill(value, start, end)
```

```
[1,2,3].fill(0); // [0,0,0]
```

25. flatMap() — map() + flat()

```
array.flatMap(callback)
```

```
[1,2,3].flatMap(n => [n, n*2]); // [1,2,2,4,3,6]
```

26. Array.from() — Convert to Array

```
Array.from(iterable)
```

```
Array.from("ABC"); // ["A","B","C"]
```

27. Array.isArray()

Check if a value is an array.

```
Array.isArray([1,2,3]); // true
Array.isArray("hello"); // false
```

28. toReversed(), toSorted(), toSpliced() (ES2023)

Non-mutating versions of destructive methods.

```
const a = [3,1,2];
a.toSorted(); // [1,2,3] (original unchanged)
```

Final Summary Table

Method	Mutates?	Returns
push / pop	Yes	number / element
shift / unshift	Yes	element / number
indexOf / includes	No	index / boolean
slice	No	new array
splice	Yes	removed items
forEach	No	undefined
map	No	new array

Method	Mutates?	Returns
filter	No	new array
reduce	No	single value
reverse	Yes	same array reversed
flat	No	new flattened array
find / findIndex	No	element / index
some / every	No	boolean
sort	Yes	sorted array
concat	No	new array
join	No	string
fill	Yes	modified array
flatMap	No	new array
Array.from	No	new array
Array.isArray	No	boolean
toSorted / toReversed / toSpliced	No	new array

If you want, I can add: - Memory tricks for every method - Real-world project examples - Interview questions - Visual diagrams of slice vs splice, map vs filter, etc.