JavaScript Arrays

An **array** is used to store **many values in one variable** instead of creating many variables.

Example

let fruits = ["Apple", "Banana", "Mango"];

Here:

- fruits is the array name
- "Apple", "Banana", "Mango" are elements
- Index starts from **0**

fruits[0] // Apple

fruits[1] // Banana

JS Array Const

Arrays are mostly declared using const.

const cars = ["BMW", "Audi", "Volvo"];

Why const?

- Because the array name should not change

You **can change elements**:

cars[0] = "Toyota";

You **cannot reassign**:

cars = ["Tesla"]; // Error

JS Array Methods (Basic Operations)

Array methods help us **add, remove, and change items**.

---

push() – Add element at end

```
let a = [1, 2];
a.push(3);
// [1, 2, 3]
```

---

pop() – Remove last element

```
a.pop();
// [1, 2]
```

---

unshift() – Add at start

```
a.unshift(0);
// [0, 1, 2]
```

---

shift() – Remove from start

```
a.shift();
// [1, 2]
```

---

length – Count elements

```
a.length; // 2
```

---

join() – Convert array to string

```
let fruits = ["Apple", "Banana"];
fruits.join(", ");
// "Apple, Banana"
```

JS Array Search

Used to **find items in array**.

---

indexOf()

Returns index number.

fruits.indexOf("Banana"); // 1

If not found → -1

---

includes()

Checks if value exists.

fruits.includes("Apple"); // true

---

find()

Returns **first value** matching condition.

let nums = [5, 10, 15];

nums.find(n => n > 8); // 10

---

findIndex()

Returns index of matched value.

nums.findIndex(n => n > 8); // 1

JS Array Sort (Arrange Data)

---

sort() – Alphabet order

let fruits = ["Banana", "Apple", "Mango"];

```
fruits.sort();
// ["Apple", "Banana", "Mango"]
```

---

reverse()

```
fruits.reverse();
```

Number Sorting

```
let nums = [40, 5, 100];
nums.sort((a, b) => a - b);
// [5, 40, 100]
```

JS Array Iterations (Looping)

Used to **run code for each element**.

---

forEach()

```
fruits.forEach(item => {
  console.log(item);
});
```

---

map()

Creates **new array**.

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

---

filter()

Returns values matching condition.

```
nums.filter(n => n > 1);

// [2, 3]
```

---

reduce()

Used to **calculate total**.

```
nums.reduce((sum, n) => sum + n, 0);

// 6
```

JavaScript Sets

A **Set** is a special JavaScript object used to **store unique values**.
It does **not allow duplicate elements**.

Example

```
let mySet = new Set([1, 2, 3, 3, 4]);

console.log(mySet);

// Output: Set {1, 2, 3, 4}
```

✓ Duplicate values are automatically removed.

Why Use Set?

- To remove duplicate values
- To store unique data
- Faster searching than arrays
- Useful in logic operations (union, intersection)

JS Set Methods

new Set()

Creates a set.

let s = new Set();

---

add()

Adds a value.

s.add(10);

s.add(20);

---

delete()

Removes a value.

s.delete(10);

---

has()

Checks if value exists.

s.has(20); // true

---

clear()

Removes all elements.

s.clear();

---

size

Returns total elements.

s.size; // 1

---

forEach()

Loop through set.

```
s.forEach(value => {
  console.log(value);
});
```

JS Set Logic (Important)

Set logic is used to perform **mathematical operations**.

Union (Combine values)

```
let a = new Set([1, 2, 3]);
let b = new Set([3, 4, 5]);

let union = new Set([...a, ...b]);
console.log(union);
// {1,2,3,4,5}
```

Intersection (Common values)

```
let intersection = new Set(
  [...a].filter(x => b.has(x))
);
console.log(intersection);
// {3}
```

Difference (A - B)

```
let difference = new Set(
  [...a].filter(x => !b.has(x))
);
console.log(difference);
// {1,2}
```

JS WeakSet

A **WeakSet** stores **only objects**, not primitive values.

Example

```
let obj1 = {name: "A"};
let obj2 = {name: "B"};

let ws = new WeakSet();
ws.add(obj1);
ws.add(obj2);
```

WeakSet Rules

- Only objects allowed
- No size property
- No forEach
- Values are weakly referenced (auto garbage collection)

WeakSet Use

Used for:

- Tracking objects
- Memory management
- Private object data

Set vs Array

Feature	Array	Set
Duplicate values	Allowed	Not allowed
Index based	Yes	No
Search speed	Slower	Faster
Unique values	No	Yes