1. Why the need of Set?

In JavaScript, we often use arrays to store collections of values. But arrays allow **duplicates**, and checking for existence or ensuring uniqueness is **slow** (`includes()` has to loop through the array).

So, we need `Set` when:

- We want to **store unique values only**
- We need **fast lookups**
- We want to **remove duplicates from an array**
- We need to **perform mathematical set operations** like union, intersection, difference

2. What is the purpose of Set?

The purpose of a `Set` is to provide a **collection of values where each value is unique**. It helps with:

- **Enforcing uniqueness** (e.g., no duplicate user IDs)
- **Quick existence checks** (`set.has(value)` is faster than `array.includes(value)`)
- **Efficient data structures** for membership checking
- Performing **set-theory-like operations** (e.g., union, intersection)

3. What is a Set in JavaScript?

A `Set` is a **built-in object** (introduced in ES6) that lets you store **unique values** of any type — primitive values or object references.

Unlike arrays:

- `Set` does **not** allow duplicate values
- Elements are **not indexed**
- Insertion order is preserved

🛠️ 4. How can it be created, accessed, modified, and inserted?

✅ **Create a Set**

javascript Copy code

```javascript
const set1 = new Set();                    // empty set
const set2 = new Set([1, 2, 3, 3]);        // removes duplicates → Set {1, 2, 3}
```

✅ **Insert elements — `.add(value)`**

javascript Copy code

```javascript
set1.add(10);
set1.add("apple");
set1.add(10); // ignored, already exists
```

✅ Access elements — `.has(value)`

javascript Copy code

```javascript
set1.has(10);       // true
set1.has("banana") // false
```

✅ Modify / Delete elements — `.delete(value)`

javascript Copy code

```javascript
set1.delete(10);  // removes 10 if exists
```

✅ Get all values — iteration

javascript Copy code

```javascript
for (let item of set1) {
  console.log(item);
}
```

✅ Clear all elements — `.clear()`

javascript Copy code

```javascript
set1.clear();  // removes all values
```

✅ Get size — `.size`

javascript Copy code

```javascript
console.log(set1.size);  // number of elements
```

---

5. How it actually works (under the hood)?

Internals of a Set:

- A `Set` internally stores values using a **hash-based structure** (similar to a hash table).
- When you **add a value**, it checks if it already exists:
  - If **not present**, it stores it.
  - If **already present**, it ignores it.
- Unlike arrays, values in Sets are **compared by reference** (for objects) and **by value** (for primitives).
- The order of insertion is maintained, which allows predictable iteration.

🔍 **Examples:**

🧪 **Primitive values**

```javascript
const set = new Set();
set.add(1);
set.add(1); // duplicate, won't be added
console.log(set); // Set { 1 }
```

🧪 **Object references**

```javascript
const obj1 = { a: 1 };
const obj2 = { a: 1 };

const set = new Set();
set.add(obj1);
set.add(obj2); // different reference → added
console.log(set.size); // 2
```

Even though `obj1` and `obj2` look the same, they are **different objects** in memory.

✅ Summary Table

Action	Method / Property	Example
Create	`new Set([values])`	`new Set([1, 2, 3])`
Add	`.add(value)`	`set.add(5)`
Check Exists	`.has(value)`	`set.has(5)`
Remove	`.delete(value)`	`set.delete(5)`
Clear All	`.clear()`	`set.clear()`
Size	`.size`	`set.size`
Iterate	`for...of`, `forEach`	`for (let v of set) {}`

🧠 Use Cases

- Remove duplicates from an array:

```javascript
const arr = [1, 2, 2, 3];
const unique = [...new Set(arr)];
console.log(unique); // [1, 2, 3]
```

- Fast lookups:

```javascript
const visited = new Set();
visited.add("home");
if (!visited.has("about")) {
  visited.add("about");
}
```

- Set operations (manual):

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

const union = new Set([...a, ...b]);
const intersection = new Set([...a].filter(x => b.has(x)));
const difference = new Set([...a].filter(x => !b.has(x)));
```

📌 Summary Table

Method / Property	Description	Returns
`new Set(iterable)`	Creates a new Set	New Set object
`.add(value)`	Adds a new unique value	The Set itself (for chaining)
`.delete(value)`	Removes specified value	`true` if removed, else `false`
`.has(value)`	Checks if value exists	`true` or `false`
`.clear()`	Removes all elements	`undefined`
`.size`	Number of unique elements	Number
`.values()`	Iterator over values	Iterator
`.keys()`	Same as `.values()`	Iterator
`.entries()`	Iterator over `[value, value]` pairs	Iterator
`.forEach(callback, thisArg?)`	Calls callback for each value	`undefined`