Map and Set data structures

🗺️ `Map` and `Set` Data Structures in JavaScript — Explained in Detail

---

JavaScript introduced two powerful built-in data structures with **ES6**:

- `Map` – For storing key-value pairs
- `Set` – For storing unique values

They are alternatives to traditional objects and arrays when certain requirements like order, uniqueness, or performance come into play.

🔷 `Map` — Key-Value Store

✅ Features of `Map`:

- Keys can be **any type** (object, function, primitive).
- Maintains the **insertion order** of entries.
- Has built-in methods like `set()`, `get()`, `has()`, `delete()`, `clear()`.

🔶 Creating and Using a Map:

```
const map = new Map();

map.set('name', 'Abhi');
map.set(100, 'Number key');
map.set(true, 'Boolean key');

console.log(map.get('name'));     // Abhi
console.log(map.get(100));        // Number key
console.log(map.has(true));       // true
console.log(map.size);            // 3
```

```
map.delete('name');               // Removes the key 'name'
map.clear();                      // Empties the map
```

🔄 Looping through a Map:

```
const map = new Map([
  ['name', 'Abhi'],
  ['age', 21],
  ['language', 'JavaScript']
]);

for (let [key, value] of map) {
  console.log(`${key}: ${value}`);
}
```

🔷 `Set` — Unique Value Collection

✅ Features of `Set`:

- Stores only **unique values** (no duplicates).
- Can store values of **any type**.
- Maintains **insertion order**.
- Has methods like `add()`, `has()`, `delete()`, `clear()`.

🔶 Creating and Using a Set:

```
const set = new Set();

set.add(1);
set.add(2);
set.add(2); // Duplicate, ignored
set.add('hello');
```

```
console.log(set.has(2));     // true
console.log(set.size);       // 3
set.delete(1);               // Removes 1
set.clear();                 // Empties the set
```

🔄 Looping through a Set:

```
const set = new Set([1, 2, 3, 4]);

for (let value of set) {
  console.log(value);
}
```

🔁 Conversion between Set/Map and Array

🔶 Set ↔ Array

```
const set = new Set([1, 2, 3]);
const arr = [...set]; // [1, 2, 3]

const newSet = new Set(arr);
```

🔶 Map ↔ Array

```
const map = new Map([
  ['a', 1],
  ['b', 2]
]);

const arr = [...map]; // [['a', 1], ['b', 2]]
```

🔍 Use Cases

`Map` Use Cases	`Set` Use Cases
Lookup tables	Storing unique values
Caching/memoization	Removing duplicates from arrays
Storing objects as keys	Set operations (union, intersection)

🧠 Summary Table

Feature	`Map`	`Set`
Stores	Key-value pairs	Unique values
Keys	Any type	Only values (no keys)
Duplicates	Allows duplicate values	No duplicates
Order	Maintains insertion order	Maintains insertion order
Useful for	Lookups, caching	Filtering, uniqueness

Would you like examples of `Set` operations like **union**, **intersection**, or converting between arrays and sets?