WeakMap and WeakSet

🔐 `WeakMap` and `WeakSet` in JavaScript — Explained in Detail

`WeakMap` and `WeakSet` are special collections introduced in ES6 that allow **weak references** to their keys or values, enabling **better memory management** for certain use cases like caching, DOM node tracking, or private data storage.

🔷 `WeakMap`

✅ Key Characteristics:

- A `WeakMap` is similar to `Map` but **only allows objects as keys**.
- Keys are **weakly referenced**, meaning they do **not prevent garbage collection** if no other references to the object exist.
- **Not iterable** — you cannot loop over `WeakMap`.
- Methods: `.set()`, `.get()`, `.has()`, `.delete()`

🔶 Syntax:

```
const wm = new WeakMap();

let obj = { name: 'Abhi' };
wm.set(obj, 'Some metadata');

console.log(wm.get(obj)); // 'Some metadata'

obj = null; // `obj` can now be garbage collected
```

❗ Important:

- You **can't** use primitive values (e.g., strings, numbers) as keys.
- You **can't** inspect or loop over the keys or values in a `WeakMap`.

🔷 Use Case of `WeakMap`

🔐 Storing Private Data

```
const privateData = new WeakMap();

function User(name) {
  const data = { secret: '123abc' };
  privateData.set(this, data);
  this.name = name;
}

User.prototype.getSecret = function () {
  return privateData.get(this).secret;
};

const user1 = new User('Abhi');
console.log(user1.getSecret()); // 123abc
```

If `user1` is no longer referenced, both the `user1` object and its secret data will be **garbage collected**.

🔷 `WeakSet`

✅ Key Characteristics:

- A `WeakSet` is like a `Set` but **only stores objects**, not primitive values.
- Objects are **weakly held** — they do not prevent garbage collection.
- **Not iterable**, no `.size` property.
- Methods: `.add()`, `.has()`, `.delete()`

🔶 Syntax:

```
const ws = new WeakSet();
```

```
let user = { name: 'Abhi' };
ws.add(user);

console.log(ws.has(user)); // true

user = null; // user is garbage collected if no other reference exists
```

🔷 Use Case of `WeakSet`

🧍 Tracking Object Presence

```
const visited = new WeakSet();

function visit(place) {
  if (!visited.has(place)) {
    console.log('Visiting', place.name);
    visited.add(place);
  }
}

let room = { name: 'Room A' };
visit(room); // Visiting Room A
visit(room); // No output

room = null; // room can now be garbage collected
```

🧠 Summary Table

Feature	`WeakMap`	`WeakSet`
Keys/Values	Object keys only	Object values only
Weak References	Yes (keys)	Yes (values)
Iterability	Not iterable	Not iterable
Garbage Collected	Yes, if key/value is unreferenced	Yes, if object is unreferenced

Use Case	Private data, metadata, caching	Tracking objects

⚠️ Why Not Iterable?

Since objects can be garbage collected anytime, it's impossible to guarantee consistency in iteration. Hence, **no** `.forEach`, `.size`, **or spread** for `WeakMap` and `WeakSet`.

Let me know if you want visual diagrams or a real-world browser example (like DOM node caching).