Sets, WeakSets, Maps and WeakMaps

Learn how to store unique values in sets and much more.

WE'LL COVER THE FOLLOWING

What is a `Set`?

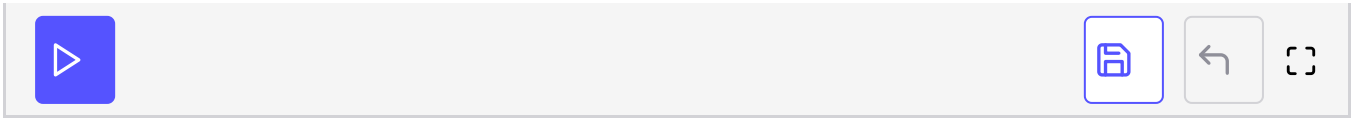
A `Set` is an `Object` where we can store **unique values** of any type.

```
// create our set
const family = new Set();

// add values to it
family.add("Dad");
console.log(family);
// Set [ "Dad" ]

family.add("Mom");
console.log(family);
// Set [ "Dad", "Mom" ]

family.add("Son");
console.log(family);
// Set [ "Dad", "Mom", "Son" ]

family.add("Dad");
console.log(family);
// Set [ "Dad", "Mom", "Son" ]
```

As you can see, at the end we tried to add “Dad” again at line 17, but the `Set` still remained the same because a `Set` can only take **unique values**.

Let’s continue using the same `Set` and see what methods we can use on it.

```
const family = new Set(["Dad", "Mom", "Son"]);

console.log(family.size);
// 3
console.log(family.keys());
// SetIterator {"Dad", "Mom", "Son"}
console.log(family.entries());
// SetIterator {"Dad", "Mom", "Son"}
console.log(family.values());
// SetIterator {"Dad", "Mom", "Son"}
family.delete("Dad");
console.log(family);
// Set [ "Mom", "Son" ]
family.clear();
console.log(family);
// Set []
```



As you can see, a `Set` has a `size` property and we can `delete` an item from it or use `clear` to delete all the items from it.

We can also notice that a `Set` does not have keys, so when we call `.keys()` we get the same result as calling `.values()` or `.entries()`.

Loop over a `Set`

We have two ways of iterating over a `Set`: using `.next()` or using a `for of` loop.

```
const family = new Set(["Dad", "Mom", "Son"]);
// using `.next()`
const iterator = family.values();
console.log(iterator.next());
// Object { value: "Dad", done: false }
console.log(iterator.next());
// Object { value: "Mom", done: false }


// using a `for of` loop
```

```
// using a `for of` loop
for(const person of family) {
  console.log(person);

}
// Dad
// Mom
// Son
```

The method `values()` will return an `Iterator` object on which we can call `next()` similarly to how we did when we discussed about `generator` function.

Remove duplicates from an array

We can use a `Set` to remove duplicates from an array since we know it can only hold unique values. As you can see, the new array contains only the unique values from the original array.

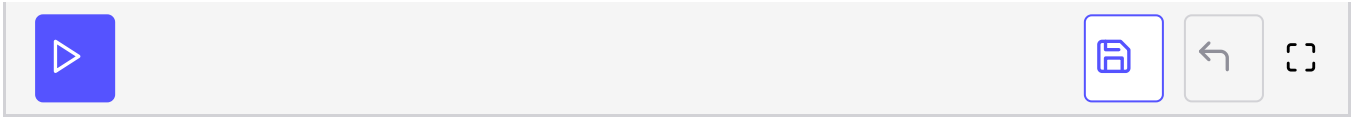
```
const myArray = ["dad", "mom", "son", "dad", "mom", "daughter"];

const set = new Set(myArray);
console.log(set);
// Set [ "dad", "mom", "son", "daughter" ]
// transform the `Set` into an Array
const uniqueArray = Array.from(set);
console.log(uniqueArray);
// Array [ "dad", "mom", "son", "daughter" ]

// write the same but in a single line
const uniqueArray2 = Array.from(new Set(myArray));
// Array [ "dad", "mom", "son", "daughter" ]
```

As you can see the new array only contains the unique values from the original array.

What is a `WeakSet`?

A `WeakSet` is similar to a `Set` but it can **only** contain Objects.

```
let dad = {name: "Daddy", age: 50};
let mom = {name: "Mummy", age: 45};

const family = new WeakSet([dad,mom]);

for(const person of family){
  console.log(person);
}
// TypeError: family is not iterable
```



We created our new `WeakSet` but when we tried to use a `for of` loop it didn't work, we can't iterate over a `WeakSet`.

A `WeakSet` cleans itself up after we delete an element from it.

```
let dad = {name: "Daddy", age: 50};
let mom = {name: "Mummy", age: 45};

const family = new WeakSet([dad,mom]);

dad = null;
console.log(family);
// WeakSet [ {…}, {…} ]

// wait a few seconds
console.log(family);
// WeakSet [ {…} ]
```

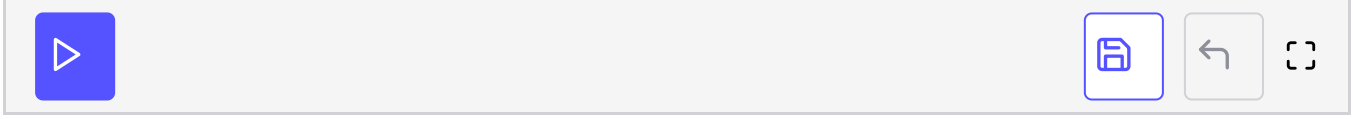
You can try running the example above in the Dev Tools of your browser. As you can see after a few seconds, **dad** was removed and *garbage collected*. That happened because the reference to it was lost when we set the value to `null`.

What is a `Map`?

A `Map` is similar to a `Set`, but they have key/value pairs.

```
const family = new Map();

family.set("Dad", 40);
family.set("Mom", 50);
family.set("Son", 20);

family;
// Map { Dad → 40, Mom → 50, Son → 20 }
family.size;
```

```
// 3

family.forEach((val,key) => console.log(key,val));

// Dad 40
// Mom 50
// Son 20

for(const [key,val] of family){
  console.log(key,val);
}
// Dad 40
// Mom 50
// Son 20
```



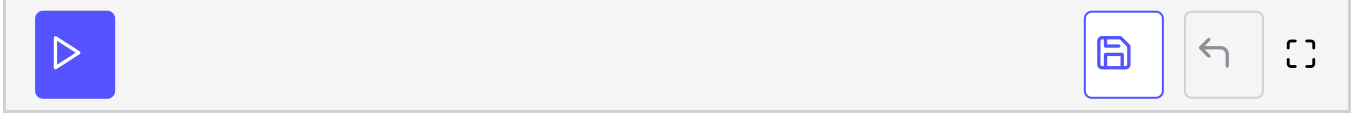
If you remember, we could iterate over a `Set` only with a `for of` loop, while we can iterate over a `Map` with both a `for of` and a `forEach` loop.

What is a `WeakMap`?

A `WeakMap` is a collection of key/value pairs and similarly to a `WeakSet`. Even in a `WeakMap`, the keys are *weakly* referenced, which means that when the reference is lost, the value will be removed from the `WeakMap` and *garbage collected*.

A `WeakMap` is **not** enumerable. Therefore we cannot loop over it.

```
let dad = { name: "Daddy" };
let mom = { name: "Mommy" };

const myMap = new Map();
const myWeakMap = new WeakMap();

myMap.set(dad);
myWeakMap.set(mom);

dad = null;
mom = null;

console.log(myMap);
// Map(1) {{…}}
console.log(myWeakMap);
// WeakMap {}
```



As you can see *mom* was garbage collected after we set its value to `null` while *dad* still remains inside our `Map`.

Let's take another quiz next.