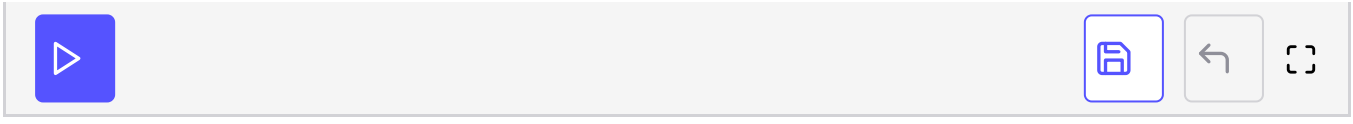
Iterables with Sets and Maps

using the entries, keys and values methods to play with iterables of keys or values

The `entries` method is defined on sets and maps. You can also use the `keys` and `values` method on a set or map to create an iterator/iterable of the keys or values. For example:

```
let colors = new Set( [ 'red', 'yellow', 'green' ] );
let horses = new Map( [
    [5, 'QuickBucks'],
    [8, 'Chocolate'],
    [3, 'Filippone']
] );

console.log( colors.entries() );
//> SetIterator {["red", "red"], ["yellow", "yellow"], ["green", "green"]}

console.log('\n')
console.log( colors.keys() );
//> SetIterator {"red", "yellow", "green"}

console.log( colors.values() );
//> SetIterator {"red", "yellow", "green"}

console.log( horses.entries() );
//> MapIterator {[5, "QuickBucks"], [8, "Chocolate"], [3, "Filippone"]}

console.log( horses.keys() );
//> MapIterator {5, 8, 3}

console.log( horses.values() );
//> MapIterator {"QuickBucks", "Chocolate", "Filippone"}
```

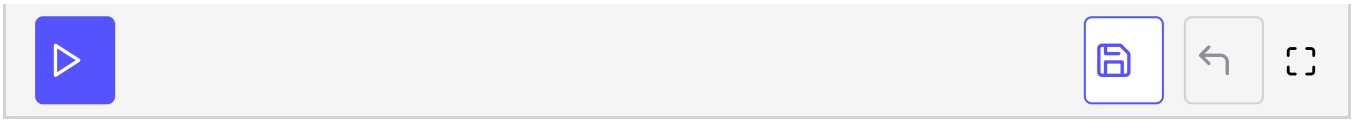


You don't need to create these iterators with the `keys`, `values`, or `entries` method though to perform an iteration on a set or a map. Sets and maps are iterable themselves. Therefore, they can be used in `for-of` loops.

> A common destructuring pattern is to iterate the keys and values of a map using destructuring in a for-of loop:

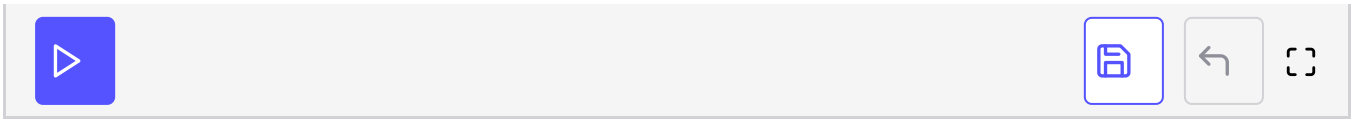
map using destructuring in a for-of loop.

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



When creating a set or a map, you can pass any iterable as an argument, provided that the results of the iteration can form a set or a map:

```
let nineToOne = new Set( countdownIterable );
console.log(nineToOne);

let horses = new Map( [
    [5, 'QuickBucks'],
    [8, 'Chocolate'],
    [3, 'Filippone']
] );
console.log(horses);
```



In the first example, we used a custom iterable, while in the second example, we used an array or key-value pairs.

Now, let's talk about the role of the iterable interface.