ES6 Maps

introduction to maps and their methods

Maps represent key-value pairs, similar to objects.

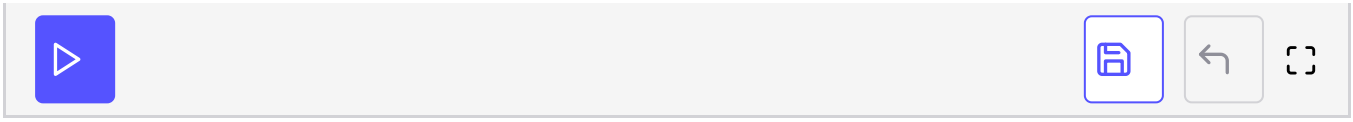
You may be wondering why we need a map data structure if objects can also be used as maps. The answer is simple: maps can have keys of any type, and the keys are not converted to strings. Therefore, `0` and `'0'` are two different, and valid keys.

Objects are also valid keys for maps. They are not converted to `'[object Object]'`. Note though, that objects are reference types. Therefore, using the `{}` object literal twice as keys will result in two different entries in the map.

We can build maps in multiple ways. We can either use the `set` method to add values to the map or pass an array of key-value pairs to the map constructor.

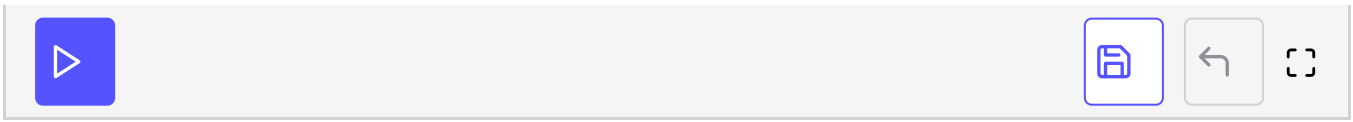
```
let horses = new Map();

horses.set( 8, 'Chocolate' );
horses.set( 3, 'Filippone' );
console.log(horses);
```



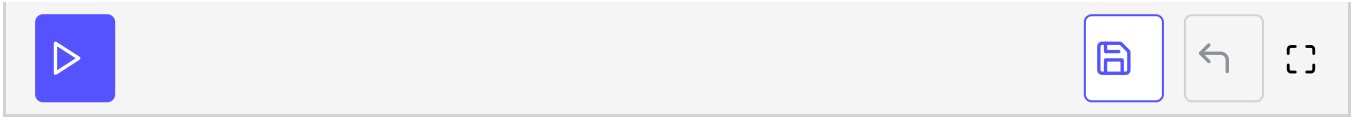
As the `set` method is chainable, the above code is equivalent to

```
let horses = new Map().set( 8, 'Chocolate' ).set( 3, 'Filippone' );
console.log(horses);
```



Using arrays of key-value pairs works as follows:

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



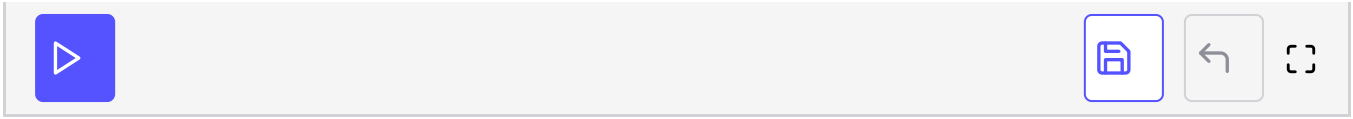
The property `size`, and the methods `has` and `delete` also work for maps. The `has` and `delete` methods expect a key in the map. Also, we can get a value from the map using the `get` method and providing a key.

```
console.log('Size:\t\t'+horses.size);
//> 2

console.log('has id=3:\t'+horses.has( 3 ));
//> true

console.log('value at key=3:\t'+horses.get( 3 ));
//> "Filippone"

horses.delete( 3 );
console.log('\nAfter deleting:\t');
console.log(horses);
//> true
```



In the next lesson, we will talk about traversing maps.