Maps

Learn how to use a JavaScript map, a new data structure added in ES2015. Master its uses for more powerful and elegant code.

Map

Another new data structure introduced in ES2015 is the map. A map is essentially an enhanced object. It stores key-value pairs, just like an object. However, while an object's key can only be a string, a map's key can be any data type. We can use an object as a key if we like.

Again, when we insert an object as a key, only that unique object is inserted. Testing for an identical but unique object will fail.

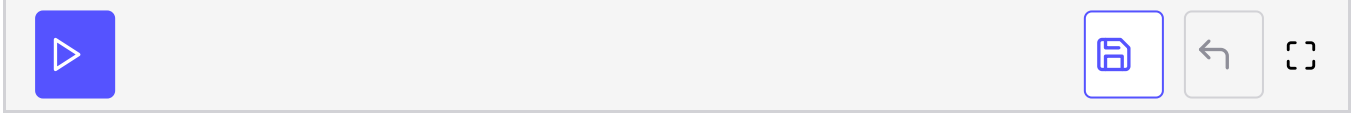
Constructing a Map

```
const map = new Map();
```

We can pass in a series of key-value pairs to the map constructor.

```
const map = new Map([
    [1, 'abc'],
    [null, 'def'],
    [{}, 'ghi']
]);

console.log(map);
// -> Map { 1 => 'abc', null => 'def', {} => 'ghi' }
```



An array of arrays is the most common way to call the map constructor. The first item in each inner array will be the key and the second will be the value.

Basic Methods & Properties

- `Map.set` takes in two paramets and inserts them as a key-value pair. Can

`Map.set` takes in two parameters and inserts them as a key-value pair. Can be chained.

- `Map.delete` takes in a key and removes the key-value pair.
- `Map.get` takes in a key and returns the associated value.
- `Map.size` is a property equal to the number of items in the set, similar to `Set.size`.

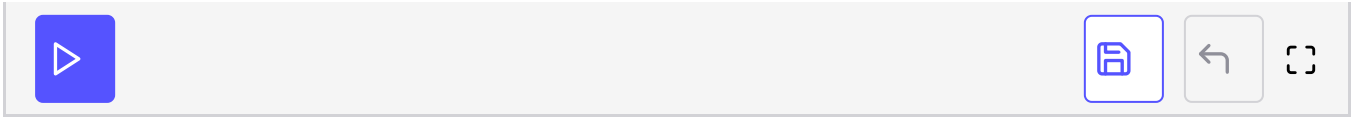
```
const map = new Map();

map.set(1, 'abc')
  .set(2, 'def')
  .set(3, 'ghi');

console.log(map); // -> Map { 1 => 'abc', 2 => 'def', 3 => 'ghi' }
console.log(map.size); // -> 3
console.log(map.get(3)); // -> ghi

map.delete(2);
console.log(map); // -> Map { 1 => 'abc', 3 => 'ghi' }

console.log(map.has(2)); // -> false
```

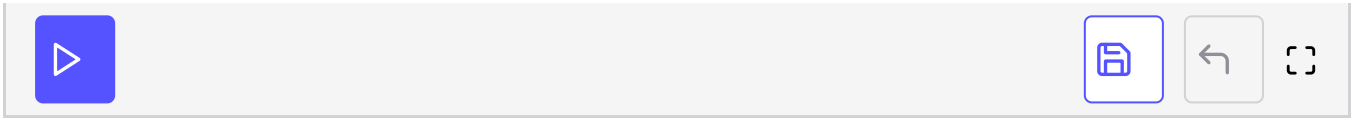


- `Map.clear` removes all items from the map.

```
const map = new Map();

map.set(1, 'abc')
  .set(2, 'def')
  .set(3, 'ghi');

map.clear();
console.log(map); // -> Map {}
```



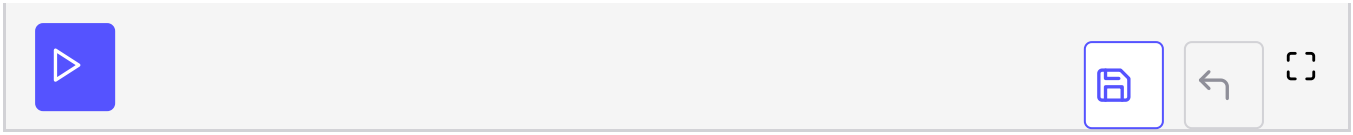
Duplicate Insertions

Duplicate insertion of a key replaces the value associated with that key.

```
const map = new Map();

map.set(1, 'abc')
  .set(2, 'def')
  .set(1, 17);
```

```
console.log(map.get(1)); // -> 17
```



Iterating

for-of loops

Maps can be iterated through using a for-of loop.

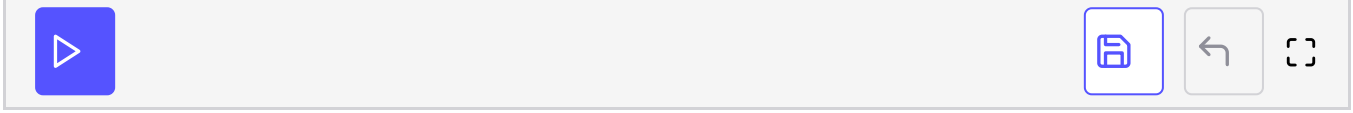
```
for(let items of set) {
}
```

At each iteration in the loop above, `items` is an array containing the key and value currently being processed. Each pair will be processed in the order in which it was inserted.

```
const map = new Map([
    [1, 'abc'],
    [2, 'def'],
    [3, 'ghi']
]);

for(let items of map) {
    console.log(items);
}

// -> [ 1, 'abc' ]
//    [ 2, 'def' ]
//    [ 3, 'ghi' ]
```



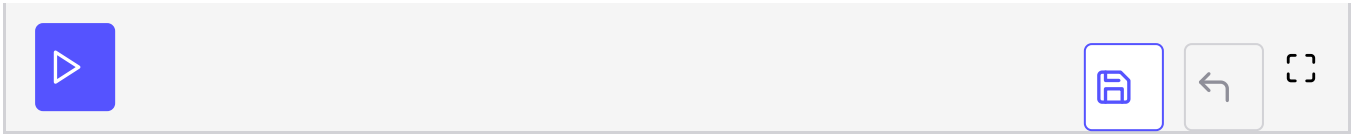
Destructuring comes in very handy here.

```
const map = new Map([
    [1, 'abc'],
    [2, 'def'],
    [3, 'ghi']
]);

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

// -> 1 'abc'
//    2 'def'
```

```
//    3 'ghi'
```



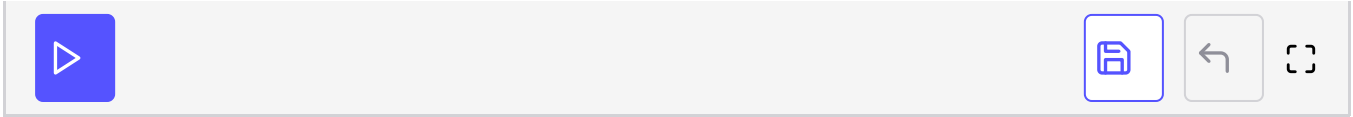
`Map.forEach`

Maps have their own `forEach` method that is almost identical to the parallel array method. The only difference is that instead of an index, we get the key that we originally provided.

```
const map = new Map([
    [1, 'abc'],
    [2, 'def'],
    [3, 'ghi']
]);

map.forEach((item, key, originalMap) => {
    console.log(item, key, originalMap);
});

// -> abc 1 Map { 1 => 'abc', 2 => 'def', 3 => 'ghi' }
//    def 2 Map { 1 => 'abc', 2 => 'def', 3 => 'ghi' }
//    ghi 3 Map { 1 => 'abc', 2 => 'def', 3 => 'ghi' }
```

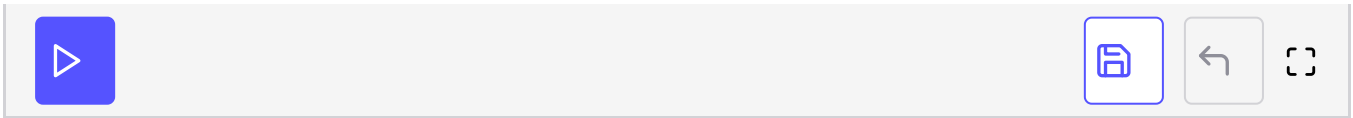


Map Operations

Maps have `keys`, `values`, and `entries` methods that function just like their object counterparts.

```
const map = new Map([
    [1, 'abc'],
    [2, 'def'],
    [3, 'ghi']
]);

console.log(map.keys()); // -> MapIterator { 1, 2, 3 }
console.log(map.values()); // -> MapIterator { 'abc', 'def', 'ghi' }
console.log(map.entries());
// -> MapIterator { [ 1, 'abc' ], [ 2, 'def' ], [ 3, 'ghi' ] }
```



That's it for maps.