25th November 2020

Map and Set

Till now, we've learned about the following complex data structures:

- Objects are used for storing keyed collections.
- Arrays are used for storing ordered collections.

But that's not enough for real life. That's why `Map` and `Set` also exist.

Map

Map is a collection of keyed data items, just like an `Object`. But the main difference is that `Map` allows keys of any type.

Methods and properties are:

- `new Map()` – creates the map.
- `map.set(key, value)` – stores the value by the key.
- `map.get(key)` – returns the value by the key, `undefined` if `key` doesn't exist in map.
- `map.has(key)` – returns `true` if the `key` exists, `false` otherwise.
- `map.delete(key)` – removes the value by the key.
- `map.clear()` – removes everything from the map.
- `map.size` – returns the current element count.

For instance:

```
let map = new Map();

map.set('1', 'str1');   // a string key
map.set(1, 'num1');     // a numeric key
map.set(true, 'bool1'); // a boolean key

// remember the regular Object? it would convert keys to string
// Map keeps the type, so these two are different:
alert( map.get(1)   ); // 'num1'
alert( map.get('1') ); // 'str1'

alert( map.size ); // 3
```

As we can see, unlike objects, keys are not converted to strings. Any type of key is possible.

`map[key]` isn't the right way to use a `Map`

Although `map[key]` also works, e.g. we can set `map[key] = 2`, this is treating `map` as a plain JavaScript object, so it implies all corresponding limitations (only string/symbol keys and so on).

So we should use `map` methods: `set`, `get` and so on.

Map can also use objects as keys.

For instance:

```
let john = { name: "John" };

// for every user, let's store their visits count
let visitsCountMap = new Map();

// john is the key for the map
visitsCountMap.set(john, 123);

alert( visitsCountMap.get(john) ); // 123
```

Using objects as keys is one of the most notable and important `Map` features. The same does not count for `Object`. String as a key in `Object` is fine, but we can't use another `Object` as a key in `Object`.

Let's try:

```
let john = { name: "John" };
let ben = { name: "Ben" };

let visitsCountObj = {}; // try to use an object

visitsCountObj[ben] = 234; // try to use ben object as the key
visitsCountObj[john] = 123; // try to use john object as the key, ben object w

// That's what got written!
alert( visitsCountObj["[object Object]"] ); // 123
```

As `visitsCountObj` is an object, it converts all `Object` keys, such as `john` and `ben` above, to same string `"[object Object]"`. Definitely not what we want.

How `Map` compares keys

To test keys for equivalence, `Map` uses the algorithm SameValueZero. It is roughly the same as strict equality `===`, but the difference is that `NaN` is considered equal to `NaN`. So `NaN` can be used as the key as well.

This algorithm can't be changed or customized.

Chaining

Every `map.set` call returns the map itself, so we can "chain" the calls:

```
map.set('1', 'str1')
  .set(1, 'num1')
  .set(true, 'bool1');
```

Iteration over Map

For looping over a `map`, there are 3 methods:

- `map.keys()` – returns an iterable for keys,
- `map.values()` – returns an iterable for values,
- `map.entries()` – returns an iterable for entries `[key, value]`, it's used by default in `for..of`.

For instance:

```
let recipeMap = new Map([
  ['cucumber', 500],
  ['tomatoes', 350],
  ['onion',    50]
]);

// iterate over keys (vegetables)
for (let vegetable of recipeMap.keys()) {
  alert(vegetable); // cucumber, tomatoes, onion
}

// iterate over values (amounts)
for (let amount of recipeMap.values()) {
  alert(amount); // 500, 350, 50
}

// iterate over [key, value] entries
for (let entry of recipeMap) { // the same as of recipeMap.entries()
  alert(entry); // cucumber,500 (and so on)
}
```

The insertion order is used

The iteration goes in the same order as the values were inserted. `Map` preserves this order, unlike a regular `Object`.

Besides that, `Map` has a built-in `forEach` method, similar to `Array`:

```
// runs the function for each (key, value) pair
recipeMap.forEach( (value, key, map) => {
  alert(`${key}: ${value}`); // cucumber: 500 etc
});
```

Object.entries: Map from Object

When a `Map` is created, we can pass an array (or another iterable) with key/value pairs for initialization, like this:

```
// array of [key, value] pairs
let map = new Map([
  ['1',  'str1'],
  [1,    'num1'],
  [true, 'bool1']
]);

alert( map.get('1') ); // str1
```

If we have a plain object, and we'd like to create a `Map` from it, then we can use built-in method Object.entries(obj) that returns an array of key/value pairs for an object exactly in that format.

So we can create a map from an object like this:

```
let obj = {
  name: "John",
  age: 30
};

let map = new Map(Object.entries(obj));

alert( map.get('name') ); // John
```

Here, `Object.entries` returns the array of key/value pairs: `[ ["name","John"], ["age", 30] ]`. That's what `Map` needs.

Object.fromEntries: Object from Map

We've just seen how to create `Map` from a plain object with `Object.entries(obj)`.

There's `Object.fromEntries` method that does the reverse: given an array of `[key, value]` pairs, it creates an object from them:

```
let prices = Object.fromEntries([
  ['banana', 1],
  ['orange', 2],
  ['meat', 4]
```

```
]);

// now prices = { banana: 1, orange: 2, meat: 4 }

alert(prices.orange); // 2
```

We can use `Object.fromEntries` to get a plain object from `Map`.

E.g. we store the data in a `Map`, but we need to pass it to a 3rd-party code that expects a plain object.

Here we go:

```
let map = new Map();
map.set('banana', 1);
map.set('orange', 2);
map.set('meat', 4);

let obj = Object.fromEntries(map.entries()); // make a plain object (*)

// done!
// obj = { banana: 1, orange: 2, meat: 4 }

alert(obj.orange); // 2
```

A call to `map.entries()` returns an iterable of key/value pairs, exactly in the right format for `Object.fromEntries`.

We could also make line `(*)` shorter:

```
let obj = Object.fromEntries(map); // omit .entries()
```

That's the same, because `Object.fromEntries` expects an iterable object as the argument. Not necessarily an array. And the standard iteration for `map` returns same key/value pairs as `map.entries()`. So we get a plain object with same key/values as the `map`.

Set

A `Set` is a special type collection – "set of values" (without keys), where each value may occur only once.

Its main methods are:

- `new Set(iterable)` – creates the set, and if an `iterable` object is provided (usually an array), copies values from it into the set.
- `set.add(value)` – adds a value, returns the set itself.
- `set.delete(value)` – removes the value, returns `true` if `value` existed at the moment of the call, otherwise `false`.
- `set.has(value)` – returns `true` if the value exists in the set, otherwise `false`.
- `set.clear()` – removes everything from the set.

- `set.size` – is the elements count.

The main feature is that repeated calls of `set.add(value)` with the same value don't do anything. That's the reason why each value appears in a `Set` only once.

For example, we have visitors coming, and we'd like to remember everyone. But repeated visits should not lead to duplicates. A visitor must be "counted" only once.

`Set` is just the right thing for that:

```
let set = new Set();

let john = { name: "John" };
let pete = { name: "Pete" };
let mary = { name: "Mary" };

// visits, some users come multiple times
set.add(john);
set.add(pete);
set.add(mary);
set.add(john);
set.add(mary);

// set keeps only unique values
alert( set.size ); // 3

for (let user of set) {
  alert(user.name); // John (then Pete and Mary)
}
```

The alternative to `Set` could be an array of users, and the code to check for duplicates on every insertion using arr.find. But the performance would be much worse, because this method walks through the whole array checking every element. `Set` is much better optimized internally for uniqueness checks.

Iteration over Set

We can loop over a set either with `for..of` or using `forEach`:

```
let set = new Set(["oranges", "apples", "bananas"]);

for (let value of set) alert(value);

// the same with forEach:
set.forEach((value, valueAgain, set) => {
  alert(value);
});
```

Note the funny thing. The callback function passed in `forEach` has 3 arguments: a `value`, then *the same value* `valueAgain`, and then the target object. Indeed, the same value appears in the arguments twice.

That's for compatibility with `Map` where the callback passed `forEach` has three arguments. Looks a bit strange, for sure. But may help to replace `Map` with `Set` in certain cases with ease, and vice versa.

The same methods `Map` has for iterators are also supported:

- `set.keys()` – returns an iterable object for values,
- `set.values()` – same as `set.keys()`, for compatibility with `Map`,
- `set.entries()` – returns an iterable object for entries `[value, value]`, exists for compatibility with `Map`.

Summary

`Map` – is a collection of keyed values.

Methods and properties:

- `new Map([iterable])` – creates the map, with optional `iterable` (e.g. array) of `[key,value]` pairs for initialization.
- `map.set(key, value)` – stores the value by the key, returns the map itself.
- `map.get(key)` – returns the value by the key, `undefined` if `key` doesn't exist in map.
- `map.has(key)` – returns `true` if the `key` exists, `false` otherwise.
- `map.delete(key)` – removes the value by the key, returns `true` if `key` existed at the moment of the call, otherwise `false`.
- `map.clear()` – removes everything from the map.
- `map.size` – returns the current element count.

The differences from a regular `Object`:

- Any keys, objects can be keys.
- Additional convenient methods, the `size` property.

`Set` – is a collection of unique values.

Methods and properties:

- `new Set([iterable])` – creates the set, with optional `iterable` (e.g. array) of values for initialization.
- `set.add(value)` – adds a value (does nothing if `value` exists), returns the set itself.
- `set.delete(value)` – removes the value, returns `true` if `value` existed at the moment of the call, otherwise `false`.
- `set.has(value)` – returns `true` if the value exists in the set, otherwise `false`.
- `set.clear()` – removes everything from the set.
- `set.size` – is the elements count.

Iteration over `Map` and `Set` is always in the insertion order, so we can't say that these collections are unordered, but we can't reorder elements or directly get an element by its number.

Tasks

Filter unique array members

importance: 5

Let `arr` be an array.

Create a function `unique(arr)` that should return an array with unique items of `arr`.

For instance:

```
function unique(arr) {
  /* your code */
}

let values = ["Hare", "Krishna", "Hare", "Krishna",
  "Krishna", "Krishna", "Hare", "Hare", ":-O"
];

alert( unique(values) ); // Hare, Krishna, :-O
```

P.S. Here strings are used, but can be values of any type.

P.P.S. Use `Set` to store unique values.

Open a sandbox with tests.

solution

Filter anagrams

importance: 4

Anagrams are words that have the same number of same letters, but in different order.

For instance:

```
nap - pan
ear - are - era
cheaters - hectares - teachers
```

Write a function `aclean(arr)` that returns an array cleaned from anagrams.

For instance:

```
let arr = ["nap", "teachers", "cheaters", "PAN", "ear", "era", "hectares"];
```

```
2
3  alert( aclean(arr) ); // "nap,teachers,ear" or "PAN,cheaters,era"
```

From every anagram group should remain only one word, no matter which one.

Open a sandbox with tests.

solution

Iterable keys

importance: 5

We'd like to get an array of `map.keys()` in a variable and then apply array-specific methods to it, e.g. `.push`.

But that doesn't work:

```
1  let map = new Map();
2
3  map.set("name", "John");
4
5  let keys = map.keys();
6
7  // Error: keys.push is not a function
8  keys.push("more");
```

Why? How can we fix the code to make `keys.push` work?

solution

Previous lesson Next lesson

Share

Tutorial map

Comments

- If you have suggestions what to improve - please submit a GitHub issue or a pull request instead of commenting.
- If you can't understand something in the article – please elaborate.
- To insert few words of code, use the `<code>` tag, for several lines – wrap them in `<pre>` tag, for more than 10 lines – use a sandbox (plnkr, jsbin, codepen...)

Ed Pfromer • 2 years ago • edited

Another way to create array of keys, using the spread operator:

```
let keys = [...map.keys()]
```


Ahmed Mokhtar • 2 years ago

Iterable keys:

```
let map = new Map();

map.set('name', 'ahmed');

let keys = [...map.keys()];

keys.push('more');

console.log(keys);  // ["name", "more"]
```


disqus_JLaTLhDhQL ➚ Ahmed Mokhtar • 9 months ago

one more shortcut

let keys = [...map.keys(), 'more'];

console.log(keys); // ["name", "more"];

pushmusicpro ➚ Ahmed Mokhtar • 2 months ago

I will use that when I am in a small team, but at the enterprise level with the interns and entry level developers, I use Array.from() explicitly. Just so they get familiar with the method...

Mahmut ERDEM ➚ Ahmed Mokhtar • a year ago

why does it show "name" and "more" but not "john"? I mean, did we delete "john"

somehow and I'm missing it? Shouldn't it show "name", "john", "more"? after all, we didn't 'pop' john, we pushed more in it.

Reply • Share ›

ano ➚ Mahmut ERDEM • a year ago

its same as,
let obj = {name: john};

but in Map.set(this is the key, this is the value);

a key can be string, boolean or any other data type.

1 Reply • Share ›

NO NAME 7 ➚ Mahmut ERDEM • 4 months ago

@Mahmut ERDEM Selamün aleyküm, kolay gelsin

Reply • Share ›

Dickson Nyange ➚ Ahmed Mokhtar • a year ago

Awesome, I like this concept of spread operators. Thanks!

Reply • Share ›

Musaddiq manzoor ➚ Ahmed Mokhtar • 2 years ago

How does this work?

Reply • Share ›

toby ➚ Musaddiq manzoor • 2 years ago

Hi **@Musaddiq manzoor**, that is the JavaScript *spread operator*. It allows to expand iterables into arguments or individual elements. In the screenshot above, he basically used the spread operator to expand the iterable obtained from calling *map.keys()*.

You can read more about it here.

2 Reply • Share ›

Aykut Yararbas • 2 years ago • edited

Anagram solution does not produce only anagram words but also any unique word.
To verify it please add "lalala" to end of example and try it.
let arr = ["nap", "teachers", "cheaters", "PAN", "ear", "era", "hectares", "lalala"];

A solution to find only anagram words can be;

```
function aclean(arr) {
const aset=new Set();
const items=new Map();
arr.forEach(item=> {
let id = item.toLowerCase().split('').sort().join('');
if(aset.has(id) && ! items.has(id)) {
items.set(id, item);
} else {
aset.add(id)
```

```
}
})
return Array.from(items.values());
}
```

Cheers,

Aykut~

2 ^ | v • Reply • Share ›

Musaddiq manzoor ➚ Aykut Yararbas • 2 years ago • edited

This particular task considers the anagram(s) as elements of the array. "lalala" must have one too. (as "alalal".

^ | v • Reply • Share ›

Aykut Yararbas ➚ Musaddiq manzoor • 2 years ago

That's right , if you add that requirement.
But if the requirement is to find **only anagrams** then solution should be as above.

^ | v • Reply • Share ›

Milly ➚ Aykut Yararbas • a year ago

>return an array cleaned from anagrams.
the purpose of the task was to remove anagrams, keeping only one word from any set of words with the same combination of letters.

Although there isn't another word with the same combination of letters as the word you added 'lalala' (i.e set of words with the letters: ['l','a','l','a','l','a'] = 1)

'lalala' is still a unique combination of letters, so should be included on the array returned.

^ | v • Reply • Share ›

imran ali • 2 years ago

Where it is useful? i mean where we can use this map and set? However i've never seen it's usage often.

2 ^ | v • Reply • Share ›

Dhruvit ➚ imran ali • 2 years ago • edited

It is useful because it takes place of nested loop and reduces the time complexity of your code. You can show this trick to your interviewer that you care about the time complexity of your code because nested loop takes more time to execute your code and 2 separate loops are better than nested loops. However, you will not notice this performance in simple and small programs.

^ | v • Reply • Share ›

mustafa kemal tuna • 2 years ago

```
function unique(arr) {
  return [...(new Set(arr)).values()]
```

```
}
```

2 ^ | v 1 • Reply • Share ›

Time Forget • a year ago

```
function aclean(arr) {
       function getObj() {
          return arr.reduce((obj, item) => {
                let sorted = item.toLowerCase().split('').sort().join('')
                obj[sorted] = item
                return obj
          } ,{})
       }
       return Object.values(getObj())
}
```

1 ^ | v • Reply • Share ›

Diego Araujo • a year ago

3rd one was easy, haha♥. In contrast I think second one was a bit too hard for a beginner. We should be able to face a challenge though, it's good.

1 ^ | v • Reply • Share ›

Joe Joe • a year ago

This is my solution for 2nd problem

let arr = ["nap", "teachers", "cheaters", "PAN", "ear", "era", "hectares"];

function aclean(anagrams){
let map = new Map();
let arrays = anagrams.forEach(word => {
if(!map.has([...word.toLowerCase()].sort().join().toString())){
map.set([...word.toLowerCase()].sort().join().toString(), word)
}
})
return [...map.values()];
}

console.log(aclean(arr))

1 ^ | v • Reply • Share ›

Bilikto Dashiev • a year ago • edited

#1

```
function unique(arr) {
  let newSet = new Set();
  for(let val of arr) {
    newSet.add(val);
  }
  return newSet;
}

let values = ["Hare", "Krishna", "Hare", "Krishna",
  "Krishna", "Krishna", "Hare", "Hare", ":-O"
```

```
];
console.log( unique(values) );
```

1 • Reply • Share ›

Joe Hint ➚ Bilikto Dashiev • 9 months ago

function unique(arr) {
let newSet = new Set();
for(let val of arr) {
newSet.add(val);
}
return Array.from(newSet);
}

• Reply • Share ›

samy • 2 years ago

iterable Keys

```
let map = new Map();

let arr = [];

map.set("name", "John");

for(let key of map.keys() ) {

 arr.push(key);

}

arr.push("more")



 alert(arr);// name,more
```

1 • Reply • Share ›

Karan Singh Negi ➚ samy • 2 years ago

That's exactly what I did lol

1 • Reply • Share ›

s • 2 years ago

filter unique array members

```
function unique(arr) {

let x = new Set();

for (let str of arr) {

x.add(str);

}

 return Array.from(x);

}
```

1 • Reply • Share ›

Nahuel Rabey • 2 years ago

I've got some doubts about Symbols and Objects. In the begining of this article, you say that the difference between an object and a map is that the map can takes any kind of data as a key. So, the objects transform the value placed as key into string. If the Symbols can't be represented as Strings (in other forms as "Symbol()"), so, how a Symbol can be used as keys in objects? and, exists a difference in how works in maps?

1 • Reply • Share ›

Sreyas Suresh ➚ Nahuel Rabey • 2 years ago

Original objects support both and ONLY strings and symbols, other primitive types are converted to strings. It is stated in the chapter introducing objects.
I hope this helps.

1 • 1 • Reply • Share ›

Victor Nikliaiev • 19 days ago • edited

In case if need to save first occurrence from task # 2 then this case works:

```
function aclean(arr) {
  let set = new Set();
  let cleanArr = [];

  for (let item of arr) {
    let sortItem = item.split("").sort().join("").toLowerCase();
    if (!set.has(sortItem)) {
      set.add(sortItem);
      cleanArr.push(item);
    }
  }
```

```
    }
    return cleanArr;
}
```

Reply • Share ›

Virat • 2 months ago

Please enable dark mode..

Reply • Share ›

TeeEi • 3 months ago

My version of aclean

```
function aclean(arr) {
const entries = arr.map(str => [str.toLowerCase().split('').sort().join(''), str]);
const obj = Object.fromEntries(entries);
return Object.values(obj);
}
```

Reply • Share ›

Prateek Mathur • 4 months ago

I decided to play around a little bit with Maps, and can't seem to understand this code -

```
let john = { name: "John" };

let visitsCountMap = new Map();
visitsCountMap.set(john, 123);

console.log(visitsCountMap.get(john)); // 123

john.name = 'John Wick';
console.log(john);

console.log(visitsCountMap.has(john)); // False. Or True?
```

The result of the last console log is true, and I can't understand how is that possible, when I have effectively changed the key.

Reply • Share ›

Aniket Somwanshi ➚ Prateek Mathur • 4 months ago

You've changed the **value** of the key right?

Reply • Share ›

Jackey • 5 months ago • edited

Another unique string is:

`[...new Set(values).values()]` or `[...new Set(values).keys()]`

Reply • Share ›

Jordan Sadim • 7 months ago

I really don't see the advantage of this over an array aside from the accessibility of keys and values. Anyone care to explain?

• Reply • Share ›

Daniel ➚ Jordan Sadim • 5 months ago

Set makes a list of unique values which cuts code lines that would check for duplicates.

Map is useful for those handy map.keys(), map.values() & map.entries() which give you a full array of whatever side you want with just 1 command. Working with arrays requires playing with numbers which you have to keep track of, these methods create a human-readable code.

Also **Map** preserves insertion order which can also be useful, no more handling the order yourself.

The names **Map** & **Set** alone can give you an idea of what this code is about, it would be harder if there were only arrays everywhere.

Map & Set are not better than arrays, they're some handy alternatives for specific situations.

It is just my opinion though, just what I understood. Anyone feel free to correct me.

Truly yours,
a Daniel guy from the internet

• Reply • Share ›

Ray Zen • 10 months ago • edited

My attempt to run

```
let map = new Map();
map.set('banana', 1);
map.set('orange', 2);
map.set('meat', 4);
let obj = Object.fromEntries(map.entries())
```

errs with

```
let obj = Object.fromEntries(map.entries());
TypeError: Object.fromEntries is not a function
```

What do I need to correct this error?

• Reply • Share ›

Io Kirchen • a year ago

Hey! Could you precise that map uses the object reference, not the object value itself? As a beginner I found it ambiguous :)

• Reply • Share ›

Justas Kuizinas • a year ago

You have a bug in second test, script returns "PAN,hectares,era" it should return

"nap,teachers,ear

Reply • Share ›

M. Abdullah ➚ Justas Kuizinas • a year ago

Thank you for sharing your feedback. However, there is no bug in the second solution. We have the following array:

let arr = ["nap", "teachers", "cheaters", "PAN", "ear", "era", "hectares"];

If you look at the given array, you notice that "PAN", "era" and "hectares" are located towards the end of the given array. As we move through the for..of loop, map.set() replaces earlier words. For example, "nap" is replaced by "PAN". Similarly, "ear" is replaced by "era". In the same manner, "teachers" is first replaced with "cheaters" and then in the next loop, "cheaters" is replaced by "hectares". This is also fairly explained in the solution:

> “If we ever meet a word the same letter-sorted form again, then it would overwrite the previous value with the same key in the map. So we’ll always have at maximum one word per letter-form.”

Hope this will clarify.

Reply • Share ›

Karina_2020 • a year ago

The answer to the task 1 is amazing:p

Reply • Share ›

KhalOOP77 • a year ago

****

first task

****

```
function uniqueArray(arr) {
let newArray = [];
let seted = new Set();

for (let i = 0; i < arr.length; i++) {
seted.add(values[i]);
}
newArray.push(Array.from(seted.keys()));
return newArray[0];
}
console.log(uniqueArray(values));
```

Reply • Share ›

KhalOOP77 • a year ago

****

Filter anagrams

****

```
function aclean(arr) {
```

```
let maped = arr.map((item) => item.toLowerCase().split('').sort().join(''));// identique item are anagram
let noduplicationArr = [...new Set(maped)]; //to select only one anagrame
let resultIndex = [];
for (let i = 0; i < noduplicationArr.length; i++) {
let foundIndex = maped.indexOf(noduplicationArr[i], 0);//tis is to look for th index of the founded anagram in the outer array(maped array), the index will be the same in the input array
resultIndex.push(foundIndex);
}
let finalResult = [];
for (let j = 0; j < resultIndex.length; j++) {
finalResult.push(rp[resultIndex[j]]);//look anagram by theire index founded indified previously in the maped array
}
return finalResult;
}
console.log(aclean(rp));
```

^ | v • Reply • Share ›

Trinh Tu Lam • a year ago • edited

Really did enjoy your tutorials each and every time. Great job!

^ | v • Reply • Share ›

Don Lawn • a year ago

a Set datatype with no intersection, union capabilities? Why bother?

^ | v • Reply • Share ›

Omar Mo'men • 2 years ago

what is array form Mean ?
this code Work without it

```
return new Set(arr)
```

^ | v • Reply • Share ›

vivek pal ➚ Omar Mo'men • a year ago

this returns a Set but in task it's required to return an array.
So Array.from() is used to convert Set values to array.

^ | v • Reply • Share ›

Vivek Varma ➚ Omar Mo'men • a year ago

Array.from converts a set/map into an array.
The code you have written converts an array into a set

^ | v • Reply • Share ›

harsh mahajan • 2 years ago

How are we able to use methods of Object Constructor like Object.assign or Object.entries without creating any instance from Object Constructor?

^ | v • Reply • Share ›

Vinnie Watson ➚ harsh mahajan • 6 months ago

Object functions are already defined type Object into browser console, same with Array. when you do new Object you are now making a new version of Object contructor

 • Reply • Share ›

بهروز ورمزیار • 2 years ago

I think there is a mistake here:
A call to map.entries() returns an array of key/value pairs, exactly in the right format for Object.fromEntries

I think you should replace this line with this one:
A call to map.entries() returns an iterable object of key/value pairs , exactly in the right format for Object.fromEntries. In fact Object.fromEntries takes any iterable in which at every iteration, value is an array of [key,value].
is it not true?

for example: