Tip 14: Iterating over Key-Value Data with Map & Spread Operator

In this tip, you'll learn how to iterate directly over key-value data in maps with either loops or the spread operator.

We'll cover the following

In the previous tip, you saw how maps are an improved key-value collection when you're regularly adding or deleting items. As you saw, objects are very useful, but there are times when a map has distinct advantages. You can see those advantages on the Mozilla Developer Network.

You've already explored several advantages pertaining to when keys are set. Now you're going to explore another suggested usage for maps: *collections that are iterated*.

Looping over objects

Objects are very frustrating to iterate over. In fact, there used to be no way to directly iterate over them. You were always forced to transform them before you could loop over the data. Things are a little better. You can now use a `for...in` loop to iterate over objects, but you'll have access only to the object key. In a way, it's not much different from looping over an array of keys. Check out Tip 27, Reduce Loop Clutter with for...in and for...each, for more about the `for...in` loop.

As you can see. Looping over objects is complicated. Conversely, you can iterate over maps directly.

Start by returning to your filters. Suppose you have an object of `filters` and you want to list the applied filters. After all, you want your users to remember they're seeing a subset of information. How would you write code that translates the objects to a string?

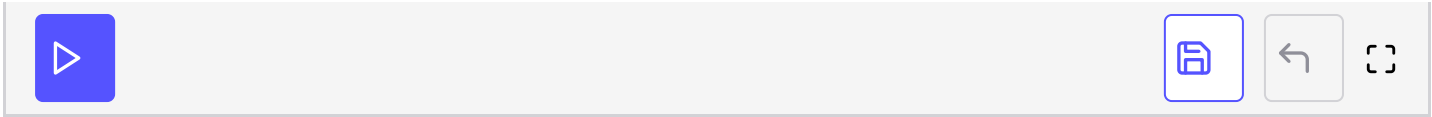
How would you, for example, transform all the key-values to be a string of the form *"key:value"*?

The odd thing is you won't iterate over the `filters` object. Instead, you'll pull out other information and then iterate over that.

```
const filters = {
    color: 'black',
    breed: 'labrador',
};

function getAppliedFilters(filters) {
    const keys = Object.keys(filters);
    const applied = [];
    for (const key of keys) {
        applied.push(`${key}:${filters[key]}`);
    }
    return `Your filters are: ${applied.join(', ')}.`;
}

console.log(getAppliedFilters(filters));
```



Looking at the code, you see that the first step is pulling out a section of the object into an array with `Object.keys()` and then you iterate over the keys with a `for` loop. And during that `for` loop, you have to pull the value out by referencing the object again.

Plus, there's no guarantee of order in an object. That means an object *can't be sorted*. If you wanted to get the `filters` in sorted order, you'd first need to sort the *keys*.

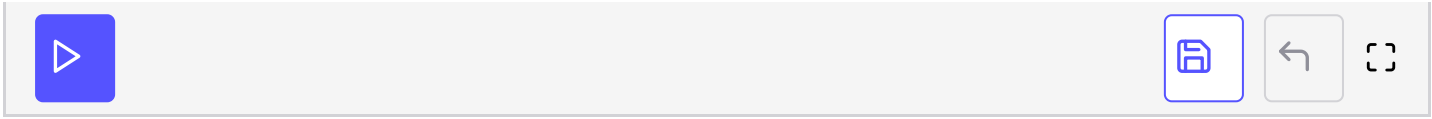
```
const filters = {
    color: 'black',
    breed: 'labrador',
};

function getSortedAppliedFilters(filters) {
    const keys = Object.keys(filters);
    keys.sort();
    const applied = [];
    for (const key of keys) {
```

```
        applied.push(`${key}:${filters[key]}`);
    }
    return `Your filters are: ${applied.join(', ')}.`;

}

console.log(getSortedAppliedFilters(filters));
```



That's a lot to keep track of when you want to do a simple iteration. A `Map`, by contrast, has everything you need to sort and iterate built in as part of the *MapIterator* you saw at the end of the previous tip.

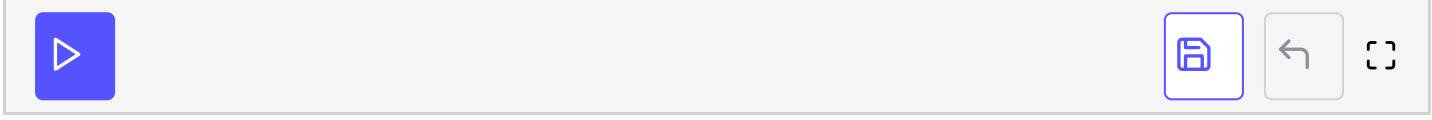
MapIterator: Looping over maps

To begin exploring the MapIterator, look at a simple `for` loop on your `filters` map. The `for ... of` syntax is also fairly new, but it's very simple. It returns each value in the collection one at a time. You'll explore it a little more in Tip 27, Reduce Loop Clutter with for...in and for...each.

```
const filters = new Map()
    .set('color', 'black')
    .set('breed', 'labrador');

function checkFilters(filters) {
    for (const entry of filters) {
        console.log(entry);
    }
}

checkFilters(filters);
```



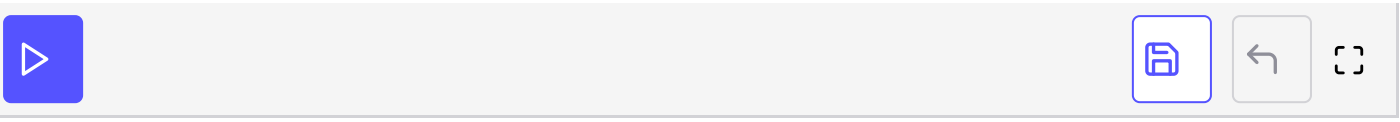
A few things should have immediately jumped out to you. The item you get from the iterator is neither the key nor the value. It's not even another `Map`. It's a pair of the *key-value*.

Even though you created this map using the `set()` method, it still translated the information back to an array. You also used a specific *variable name*, `entries()`, because `Map` has a special method that will give you a *MapIterator* of the key-values of a map as a group of pairs:

```
const filters = new Map()
    .set('color', 'black')
```

```
    .set('breed', 'labrador');

console.log(filters.entries());
```



Keep that in mind—you'll return to it in a moment. For now, just understand that a simple loop on a map will give you both the keys and the values in a pair. In fact, the ability to get entries is so convenient that it's being added to a method on objects in the next version of the JavaScript spec.

Of course, that means you'll be able to apply all the ideas you learn here to objects directly in the near future. That's another good reason to experiment with maps even if you don't adopt them often.

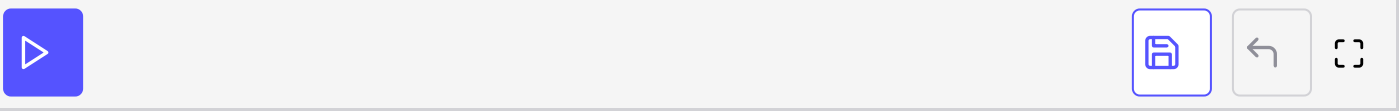
Return to your original method for turning key-values into a string using a `for` loop. Because you can iterate directly over the map, you don't need to pull out the keys first. Plus, when you loop over the entries in a map, you get a pair of the key-values, which you can immediately assign to variables using destructuring. You'll explore this more in Tip 29, Access Object Properties with Destructuring.

The result is more simple, and it helps you avoid breaking apart your data structure.

```
const filters = new Map()
    .set('color', 'black')
    .set('breed', 'labrador');

function getAppliedFilters(filters) {
    const applied = [];
    for (const [key, value] of filters) {
        applied.push(`${key}:${value}`);
    }
    return `Your filters are: ${applied.join(', ')}.`;
}

console.log(getAppliedFilters(filters));
```



Sorting problems

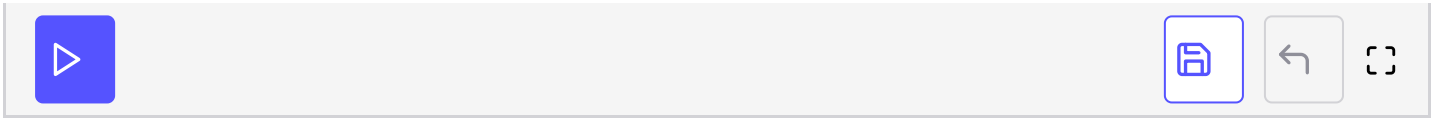
Of course, you quickly realize that you have the same sorting problem as you did earlier. Well, there is good news and bad news for you: The good news is that `Map` does preserve order. The first item you have will always be the first item in the

map. The bad news is that there *isn't* a built-in `sort` method as there is for an array.

In other words, you can't do this:

```
const filters = new Map()
    .set('color', 'black')
    .set('breed', 'labrador');

filters.sort()
```

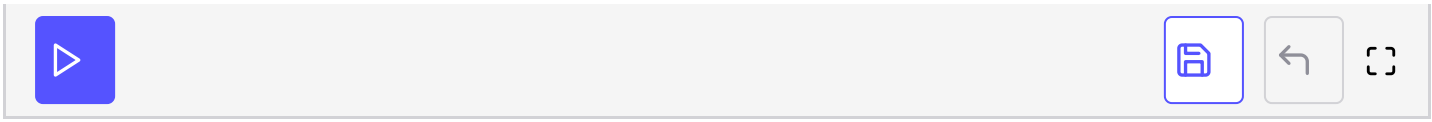


All of a sudden, your map is looking less helpful. Fortunately, there's a very simple solution: *the spread operator*.

Using the spread operator on map

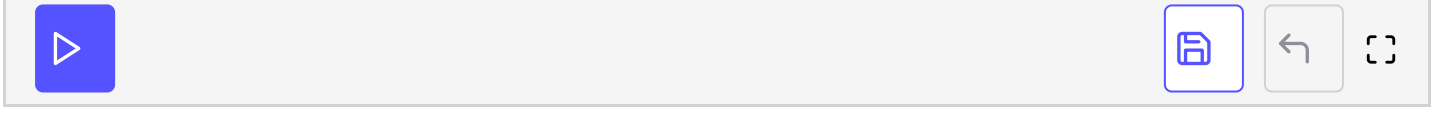
The spread operator works on a map the same way it does on an array. The main difference is that it returns a list of pairs instead of single values.

```
const filters = new Map()
    .set('color', 'black')
    .set('breed', 'labrador');
console.log(...filters);
```



And like the spread operator on arrays, you have to spread it into something, which means you can easily make an array of pairs:

```
const filters = new Map()
    .set('color', 'black')
    .set('breed', 'labrador');
console.log([...filters]);
```



I hope this has given you an idea about how to solve your sort problem. Try it out and see what you come up with. The only catch is that, because you're sorting an array of arrays, you should supply an explicit compare function. This isn't strictly necessary because the default compare function will convert the array of pairs to a string, but it's better to be clear in your intentions. Once you learn how to make arrow functions in Tip 20, Simplify Looping with Arrow Functions, the compare

function will be a one-liner.

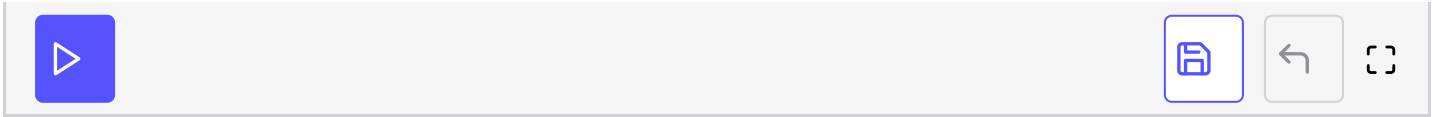
I bet you came up with something simple like this.

```
const filters = new Map()
    .set('color', 'black')
    .set('breed', 'labrador');

function sortByKey(a, b) {
    return a[0] > b[0] ? 1 : -1;
}

function getSortedAppliedFilters(filters) {
    const applied = [];
    for (const [key, value] of [...filters].sort(sortByKey)) {
        applied.push(`${key}:${value}`);
    }
    return `Your filters are: ${applied.join(', ')}.`;
}

console.log(getSortedAppliedFilters(filters));
```



Now look closely—it can be easy to miss. In the `for` loop initiator when you're assigning variables, you quickly spread the map out into an array and then sorted that array. Now you're getting the results you wanted.

There's a slight problem. If you read the code carefully, you may have noticed that something changed. You started off with a map, but your `for` loop didn't actually iterate over the map. It iterated over a new array.

Honestly, this isn't much of a problem. There's nothing wrong with converting to an array. In fact, it gives you an opportunity to simplify your function even more.

Now that you can move easily to an array, you might as well use all of the array methods at your disposal. Because you're changing every value of the array in the same way, you don't need to create a new array to collect the results as you did with `let applied = []`. You can simply use the `map()` method. If that's new to you, jump ahead to Tip 22, Create Arrays of a Similar Size with map(). Try rewriting your initial function to use the `map()` method.

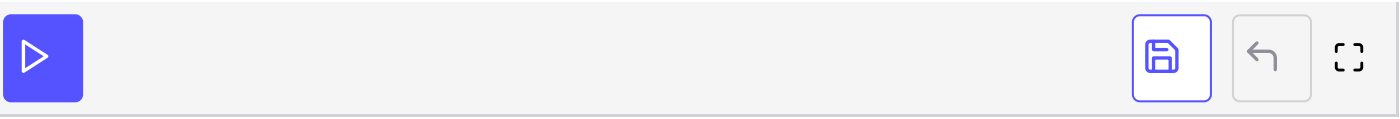
```
const filters = new Map()
    .set('color', 'black')
    .set('breed', 'labrador');

function getAppliedFilters(filters) {
```

```
    const applied = [...filters].map(([key, value]) => {
        return `${key}:${value}`;
    });

    return `Your filters are: ${applied.join(', ')}.`;
}

console.log(getAppliedFilters(filters));
```



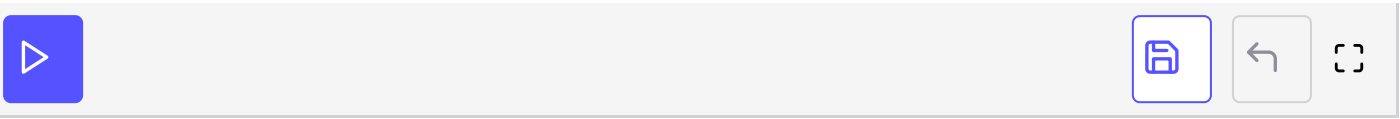
And because everything is now an array, you can combine your `sort()` function and your `join()` function using chaining to get everything nice and simple.

```
const filters = new Map()
    .set('color', 'black')
    .set('breed', 'labrador');

function sortByKey(a, b) {
    return a[0] > b[0] ? 1 : -1;
}

function getSortedAppliedFilters(filters) {
    const applied = [...filters]
        .sort(sortByKey)
        .map(([key, value]) => {
            return `${key}:${value}`;
        })
        .join(', ');
    return `Your filters are: ${applied}.`;
}

console.log(getSortedAppliedFilters(filters));
```



Looping over maps: Summary

If you're getting lost, here's a summary of the steps:

1. Convert your map to an array.
2. Sort the array.
3. Convert each pair to a string of the form *key:value*.
4. Join the array items creating a string.
5. Combine the string with other information using a template literal.

It's worth repeating that a strong knowledge of arrays can help you create very

simple and efficient code. Now that you know that you can move between a map and array with three simple dots, you've opened yourself up to many more opportunities for creatively using maps.

1

What will be the output of the code below?

```
let map = new Map([
    [1, 'Anne'],
    [2, 'Joe'],
    [3, 'Rachel'],
]);
let arr = [...map.keys()];
console.log(arr);
```

2

What will be the output of the code below?

```
let names = new Map()
.set(18, 'Anne')
.set(25, 'Rachel')
.set(35, 'Joe');

const answer = [...names]
                .filter(([k]) => k < 30)
```

```
console.log(answer);
```

Retake Quiz

In the next tip, you'll see how you can use the spread operator to avoid side effects and mutations.