Built-in Iterables

built-in iterables explained using examples

You have seen four examples for built-in iterables among the previous lessons:

- Arrays are iterables, and work well with the `for-of` loop.
- Strings are iterables as arrays of 2 to 4-byte characters.
- DOM data structures are also iterables. If you want proof, just open a random website, and execute `[...document.querySelectorAll('p')]` in the console
- Maps and Sets are iterables.

Let's experiment with built-in iterables a bit:

```
let message = 'ok';

let stringIterator = message[Symbol.iterator]();
let secondStringIterator = message[Symbol.iterator]();

stringIterator.next();
//> Object {value: "o", done: false}

secondStringIterator.next();
//> Object {value: "o", done: false}

stringIterator.next();
//> Object {value: "k", done: false}

stringIterator.next();
//> Object {value: undefined, done: true}

secondStringIterator.next();
//> Object {value: "k", done: false}
```

Before you think how cool it is to use `Symbol.iterator` to get the iterator of built-in datatypes, I would like to emphasize that using `Symbol.iterator` is generally not cool. There is an easier way to get the iterator of built-in data structures using the public interface of built-in iterables.

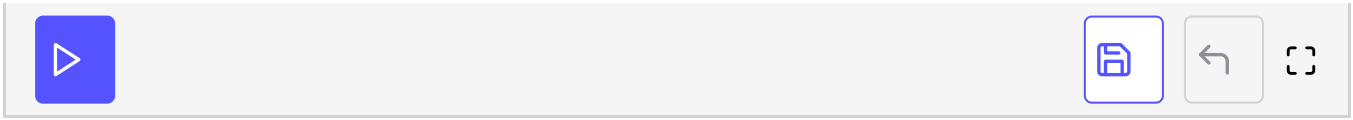
You can create an `ArrayIterator` by calling the `entries` method of an array. `ArrayIterator` objects yield an array of `[key, value]` in each iteration.

Strings can be handled as arrays using the spread operator:

```
let message = [...'ok'];

let pairs = message.entries();

for( let pair of pairs ) {
    console.log( pair );
}
```



Now, let's talk about iterables with sets and maps.