Map

WE'LL COVER THE FOLLOWING

ES6 introduces a few new ways to store data. Up until now we have had Objects and Arrays to store information. In ES6 we now have Map, WeakMap, Set and WeakSet. These Objects provide a new way to store data in a collection of either key value pairs, or collection of values.

Why new data structures

Most languages will have data structures that are purely for data; Ruby has Hashes, Python has Dictionaries. The goal of these structures is to create a

simple data store. In JavaScript we have always used Objects to do this.

Objects can be simple key value pairs, or include more features, such as methods.

However with Objects in JavaScript comes lots of added data that we might not want. When you create a simple object:

```
const person = {};
```

This will create an object that has a prototype chain, it comes with a `__proto__` object that contains inherited methods. We can create Objects that don't have this chain by using the `Object.create()` method to create a simple empty object.

```
const person = Object.create(null);
```

Doing so will create an object with no methods or properties at all. Let's look at how we can use `Map` and `Set` to work with data. This chapter will be about Map, and in the next chapter we will look at Set.

Map

In ES6 a Map is a key value object that we can instantiate to store data. Maps can use any value as a key in the object, and this includes other objects which we will see a little later.

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

In order to add values to our new Map we can use the `.set()` method.

```
person.set('name','Ryan Christiani');
person.set('age',31);
```

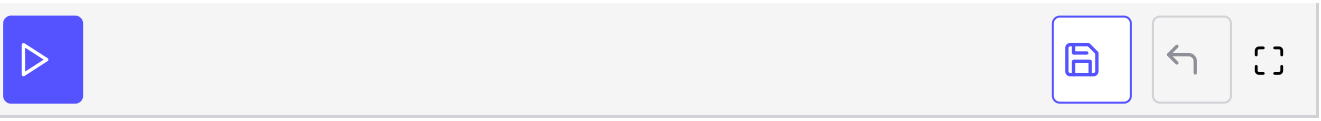
To retrieve values from our Map we use the `.get()` method.

```
console.log(person.get('name')); //Ryan Christiani
```

Size

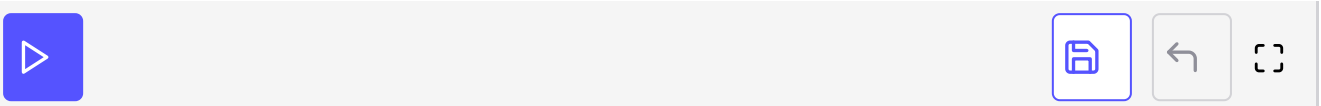
Something that is pretty nice about Maps is that they allow us to get the size of them by using the `.size` property. This is useful especially if you have ever tried to so something like this in the past on an Object.:

```
const person = {
  name: 'Ryan Christiani',
  age: 31
};

console.log(person.length); //undefined
```



Getting the size of an Object's keys in JavaScript requires us to write some code to count or keep track of the keys. The `.length` property is only for Arrays, but with Maps we can use the `.size` property to get the size back.

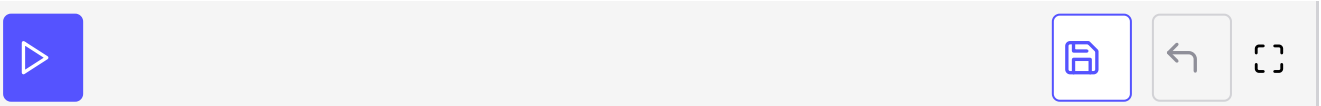
```
const person = new Map();
person.set('name', 'Ryan Christiani');
person.set('age', 31);

console.log(person.size); //2
```



Setting multiple properties

If you want to define a Map with multiple properties we need to use a multi dimensional array. The first element in the sub array is the key for your property, the second is the value.

```
const person = new Map([['name','Ryan'],['age',31]]);
console.log(person.get('name')); //Ryan
```

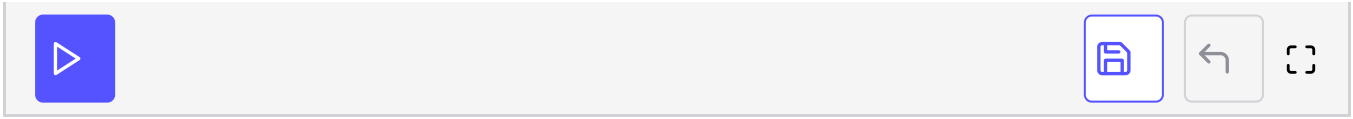


Map Keys

Map Keys

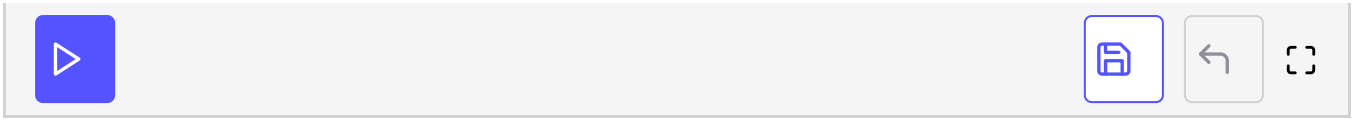
One neat feature of a Map is that you can use just about anything as a key, that means any of the primitive type as well as functions and objects. Meaning you could use something like this as a key.

```
const student = {
  id: '17UsJ290'
};

const person = new Map();
person.set(student,'Ryan Christiani');
console.log(person.get(student)); //Ryan Christiani
```



This can also be done with a function. As a quick example, check out the following.

```
const myFunction = () => {};

const person = new Map();
person.set(myFunction, 'A function used as a key');
console.log(person.get(myFunction)); //A function used as a key
```



I am interested to see how the community ends up using the ability to store values with a function or object as a key; perhaps some work around memoization for functions.

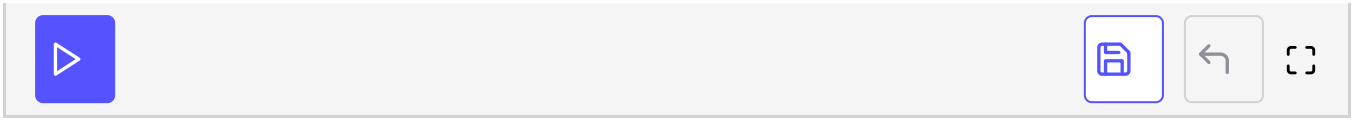
Map Methods

There are quite a few handy methods on Maps that we can use. We will look at `.has()`, `.forEach()`, `.keys()`, `.values()`, `.entries()`, `.delete()`, and `.clear()`.

has()

The `.has` method is used to check for the existence of a value in a key. Using the `person` example from above, let's check to see if it has `age`.

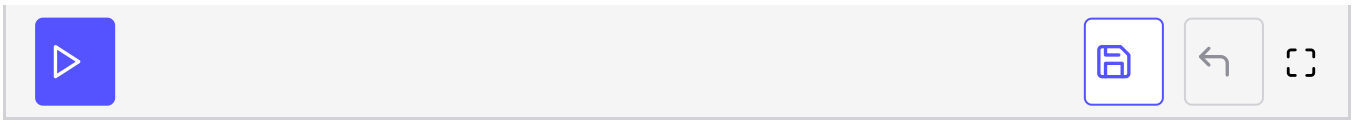
```
console.log(person.has('age')); //true
console.log(person.has('height')); //false
```



forEach()

We can also use the `.forEach()` method to enumerate over the values in our Map.

```
person.forEach((value,key) => {
    console.log(value);
    console.log(key);
});
```

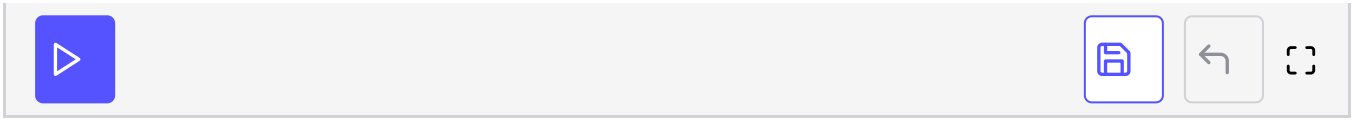


The `.forEach()` method takes a callback that is provided with the `value` and `key` for each element in the Map. It can also accept a second argument that will be used as the `this` context.

keys()

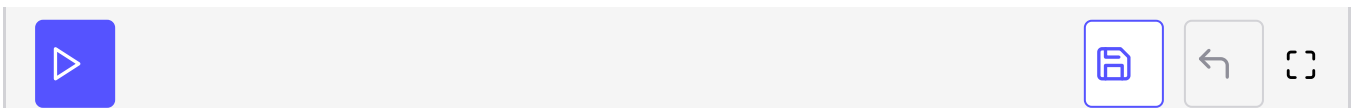
`.keys()` is a method that returns an Iterator Object which allows us to get the keys and use a `for...of` loop to iterate over them.

```
for(let key of person.keys()) {
  console.log(key);
}
```



Alternatively we can store the Iterator Object in a variable and use the `.next()` method to walk through it one at a time.

```
const personKeys = person.keys();
console.log(personKeys.next().value); //name
console.log(personKeys.next().value); //age
console.log(personKeys.next().value); //undefined
```

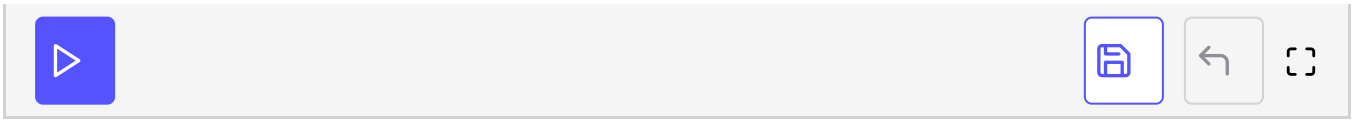


Once you exhaust the list it will return a value of `undefined`.

values()

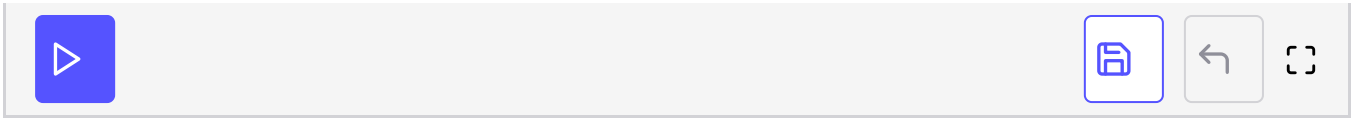
Similar to `.keys()`, the `.values()` method will allow us to create a new Iterator Object and either use the `for...of` or `.next()` methods to step through it. However, unlike `.keys()` this will instead return the value stored.

```
for(let value of person.values()) {
  console.log(value);
}
```



And just like `.keys()` we can use the `.next()` method as well.

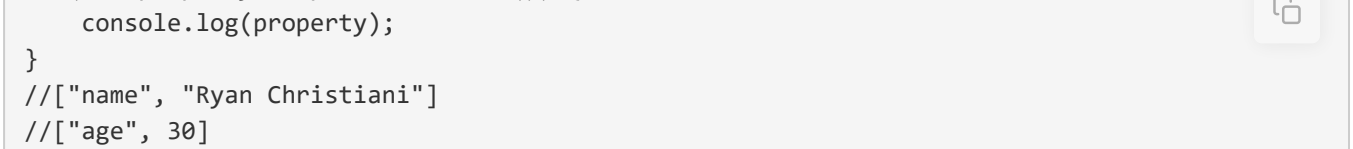
```
const personValues = person.values();
console.log(personValues.next().value); //Ryan Christiani
console.log(personValues.next().value); //31
console.log(personValues.next().value); //undefined
```



entries()

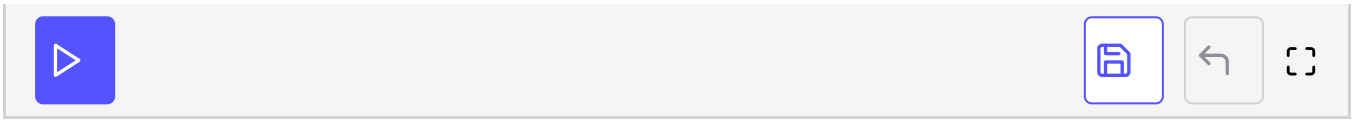
If you want to have easy access to both the key and value, the `.entries()` method is what you are looking for.

```
for(let property of person.entries()) {
    console.log(property);
}
//["name", "Ryan Christiani"]
//["age", 30]
```



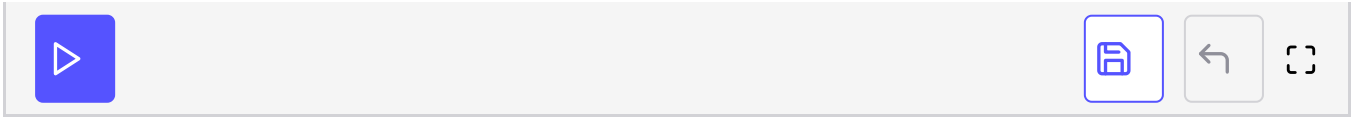
This will return an array where the first element is the key and the second is the value. We can use array destructuring to get the key and value separately if needed!

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



And just like the last two methods, we can create an Iterator Object that will allow us to step through the properties with the `.next()` method.

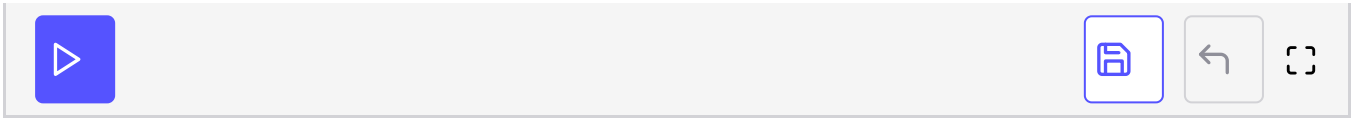
```
const personEntries = person.entries();
console.log(personEntries.next().value); //["name", "Ryan Christiani"]
console.log(personEntries.next().value); //["age", 30]
```



delete()

The `.delete()` method takes a key name and will delete this from the Map, and will also return `true` if it is deleted.

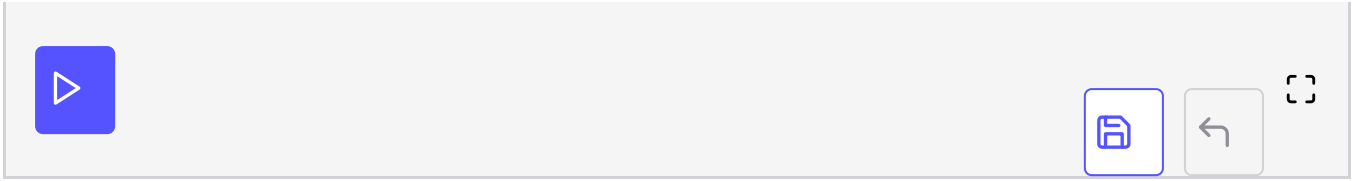
```
// Before
console.log(person.get('name'));

person.delete('name'); //true

// After
console.log(person.get('name'));
```



clear()

We can also clear out the entire Map using the `.clear()` method.

```
// Before
console.log(person.get('name'));

person.clear();

// After
console.log(person.get('name'));
```



WeakMap

There is a type of Map that is very similar to what we talked about above, however there is a very key difference. A `WeakMap` allows us to store the same sort of key value pairs as a regular map, but is stores them 'weakly'. This means that when it is not being referenced the garbage collector in the JavaScript engine will clean it up.

If you are not familiar with garbage collection, or GC, it is a process in with the JS engine will attempt to free up memory. When a variable or bit of data is being used and referenced throughout your program the GC can't do anything about it. However when you stop referencing it there is a certain time when it has no purpose, so a garbage collector will come along and clean it up for us. That is an overly simplistic explanation of GC, in the Addition Resources section you can find a link to an MDN article on Memory Management for a more in depth look.

No enumeration

One other thing to look at when it comes to a `WeakMap` is that is can not be enumerated, this means we can not used the `.entries`, `.keys`, or `.value` methods to get a list of information. If this is something you need, just use a Map!

Additional Resources

- http://www.ecma-international.org/ecma-262/6.0/#sec-map-objects
- https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Map
- http://www.2ality.com/2015/01/es6-maps-sets.html
- https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/WeakMap
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Memory_Management