Chapter 38: Map

Parameter	Details
`iterable`	Any iterable object (for example an array) containing `[key, value]` pairs.
`key`	The key of an element.
`value`	The value assigned to the key.
`callback`	Callback function called with three parameters: value, key, and the map.
`thisArg`	Value which will be used as **`this`** when executing `callback`.

Section 38.1: Creating a Map

A Map is a basic mapping of keys to values. Maps are different from objects in that their keys can be anything (primitive values as well as objects), not just strings and symbols. Iteration over Maps is also always done in the order the items were inserted into the Map, whereas the order is undefined when iterating over keys in an object.

To create a Map, use the Map constructor:

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

It has an optional parameter, which can be any iterable object (for example an array) which contains arrays of two elements – first is the key, the seconds is the value. For example:

```
const map = new Map([[new Date(), {foo: "bar"}], [document.body, "body"]]);
//                     ^key          ^value          ^key        ^value
```

Section 38.2: Clearing a Map

To remove all elements from a Map, use the `.clear()` method:

```
map.clear();
```

Example:

```
const map = new Map([[1, 2], [3, 4]]);
console.log(map.size); // 2
map.clear();
console.log(map.size); // 0
console.log(map.get(1)); // undefined
```

Section 38.3: Removing an element from a Map

To remove an element from a map use the **`.delete()`** method.

```
map.delete(key);
```

Example:

```
const map = new Map([[1, 2], [3, 4]]);
console.log(map.get(3)); // 4
map.delete(3);
console.log(map.get(3)); // undefined
```

This method returns `true` if the element existed and has been removed, otherwise `false`:

```
const map = new Map([[1, 2], [3, 4]]);
console.log(map.delete(1)); // true
console.log(map.delete(7)); // false
```

Section 38.4: Checking if a key exists in a Map

To check if a key exists in a Map, use the `.has()` method:

```
map.has(key);
```

Example:

```
const map = new Map([[1, 2], [3, 4]]);
console.log(map.has(1)); // true
console.log(map.has(2)); // false
```

Section 38.5: Iterating Maps

Map has three methods which returns iterators: `.keys()`, `.values()` and `.entries()`. `.entries()` is the default Map iterator, and contains `[key, value]` pairs.

```
const map = new Map([[1, 2], [3, 4]]);

for (const [key, value] of map) {
  console.log(`key: ${key}, value: ${value}`);
  // logs:
  // key: 1, value: 2
  // key: 3, value: 4
}

for (const key of map.keys()) {
  console.log(key); // logs 1 and 3
}

for (const value of map.values()) {
  console.log(value); // logs 2 and 4
}
```

Map also has `.forEach()` method. The first parameter is a callback function, which will be called for each element in the map, and the second parameter is the value which will be used as `this` when executing the callback function.

The callback function has three arguments: value, key, and the map object.

```
const map = new Map([[1, 2], [3, 4]]);
map.forEach((value, key, theMap) => console.log(`key: ${key}, value: ${value}`));
// logs:
// key: 1, value: 2
// key: 3, value: 4
```

Section 38.6: Getting and setting elements

Use `.get(key)` to get value by key and `.set(key, value)` to assign a value to a key.

If the element with the specified key doesn't exist in the map, `.get()` returns `undefined`.

`.set()` method returns the map object, so you can chain `.set()` calls.

```
const map = new Map();
console.log(map.get(1)); // undefined
map.set(1, 2).set(3, 4);
console.log(map.get(1)); // 2
```

Section 38.7: Getting the number of elements of a Map

To get the numbers of elements of a Map, use the `.size` property:

```
const map = new Map([[1, 2], [3, 4]]);
console.log(map.size); // 2
```