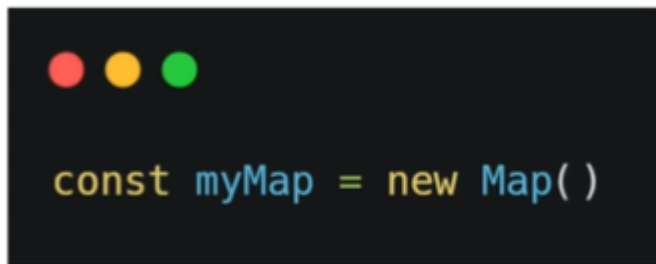


## Coleções chaveadas

- 1 - Apresentar a coleção Map
- 2 - Explicar sua aplicação
- 3 - Explicar a diferença entre Map e Objeto

### Estrutura

A estrutura do **map** é basicamente ele tem as entradas dele que são no formato [chave, valor], então uma coleção de arrays no formato [chave, valor] é o que representa um map e ele pode ser iterado no loop **for of** e a gente declara ele da seguinte maneira:

A screenshot of a code editor with a dark background. At the top left, there are three colored circles: red, yellow, and green. Below them, the code `const myMap = new Map( )` is displayed in a light blue font. The word `const` is in yellow, `myMap` is in light blue, `=` is in yellow, `new` is in light blue, `Map` is in light blue, and `( )` is in light blue.


O map tem uma série de argumentos se você criar uma variável map e depois visualizar os argumentos no console.

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

### Métodos

Métodos que podemos exercitar utilizando Map: Deletar, adicionar e ler

## Adicionar, ler e deletar



```
const myMap = new Map()  
  
myMap.set('apple', 'fruit');  
// Map(1) {"apple" => "fruit"}  
  
MyMap.get(apple);  
// "fruit"  
  
myMap.delete("apple")  
// true  
  
myMap.get("apple")  
// undefined
```

Para adicionar nós usamos o método **set**.

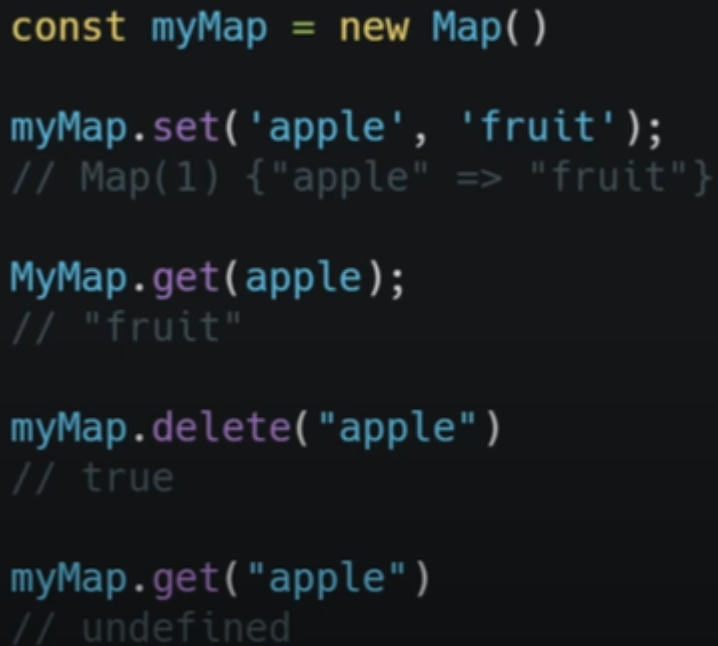
Para deletar usamos o **delete**

Para lermos usamos o **get**.

Apple é minha chave e fruit é o valor.

Quando eu uso o **get** eu estou vendo checando o valor de Apple.

```
> const myMap = new Map()
< undefined
> myMap
< ▼ Map(0) {} ⓘ
  ▼ [[Entries]]
    No properties
    size: 0
  ▼ __proto__: Map
    ▶ clear: f clear()
    ▶ constructor: f Map()
    ▶ delete: f delete()
    ▶ entries: f entries()
    ▶ forEach: f forEach()
    ▶ get: f ()
    ▶ has: f has()
    ▶ keys: f keys()
    ▶ set: f ()
    size: (...)
    ▶ values: f values()
    ▶ Symbol(Symbol.iterator): f entries()
    Symbol(Symbol.toStringTag): "Map"
    ▶ get size: f size()
    ▶ __proto__: Object
```



```
const myMap = new Map()

myMap.set('apple', 'fruit');
// Map(1) {"apple" => "fruit"}

MyMap.get(apple);
// "fruit"

myMap.delete("apple")
// true

myMap.get("apple")
// undefined
```

Para adicionar nós temos o **set** então a gente tem sempre seta uma chave e um valor, na 1ª linha eu estou sertando que apple é minha chave e o valor é fruit então isso vai logar no console que a minha chave é apple e o valor é fruit.

Se eu quiser checar o valor de apple eu coloco **get**('apple') e aí eu estou lendo.

Se eu quiser deletar eu só coloco **delete** sempre com a chave que no caso é apple.

Depois de deletado se eu quiser retornar esse valor ele vai retornar *undefined* porque ele não existe mais, ele foi deletado nessa coleção.

### Etapa 3: Map vs Objeto

Os dois são estruturas de dados no formato chave e valor



- Maps podem ter chaves de qualquer tipo;
- Maps possuem a propriedade length;
- Maps são mais fáceis de iterar;
- Utilizado quando o valor das chaves é desconhecido;
- Os valores tem o mesmo tipo.

- Um objeto ele sempre tem chaves no formato de strings, então as minhas chaves no Map podem ser outros tipos de valores;
- Enquanto no objeto você tem que terá por todas as propriedades pra ver quantas propriedades tem, o map já tem o length pronto pra você ver o tamanho desse map;
- Justamente porque é uma coleção que você já tem o tamanho e você já consegue saber quantos elementos tem;
- Quando você utiliza o valor das chaves elas são desconhecidas, então no objeto a gente utiliza um objeto quando a gente conhece o valor das chaves que a gente quer procurar aquele valor, então a gente já acessa ele pela chave, mas no map não quando a gente não conhece os valores e a gente quer iterar por eles então a gente utiliza o map;
- Os valores do map tem o mesmo tipo, no objeto não necessariamente você precisa ter valores do mesmo tipo;

## Set

- 1 - Apresentar a coleção Set
- 2 - Explicar sua aplicação
- 3 - Explicar a diferença entre Set e Array

### Estrutura

Sets são estruturas que armazenam **valores únicos**. Num array enquanto eu posso ter valores repetidos, no set eu só posso ter valores únicos e os sets tem eles tem algumas propriedades e métodos bem diferentes do array. Mas basicamente sets é uma coleção com valores únicos que não se repetem nunca.

```
const myArray = [1, 1, 2, 2, 3, 4, 5, 6, 7, 8, 2, 1];  
const mySet = new Set(myArray);
```

```
Set(8) {1, 2, 3, 4, 5, ...}  
  [[Entries]]  
    0: 1  
    1: 2  
    2: 3  
    3: 4  
    4: 5  
    5: 6  
    6: 7  
    7: 8  
  size: (...)  
  __proto__: Set
```

Sets são estruturas  
que armazenam  
apenas **valores únicos**.

## Métodos

Para adicionar, consultar e deletar que são os métodos básicos:

Para adicionar usamos o: **add**

Para consultar usamos o: **has**

Para deletar usamos o: **delete**

```
const mySet = new Set();

mySet.add(1);
mySet.add(5);

mySet.has(1);
// true

mySet.has(3);
// false

meuSet.delete(5);
```

### Set vs Array



- Possui valores únicos;
- Em vez da propriedade length, consulta-se o número de registros pela propriedade size;
- Não possui os métodos map, filter, reduce etc.

- Além de possuir valores únicos;
- Em vez da propriedade length, o número de registros que tem num set é dado pela propriedade size;
- São métodos de Arrays, o set possui outros métodos que você consulta olhando na documentação mas ele não possui uma série de métodos que um Array possui, um Array ele é muito mais flexível nesse sentido tem muito mais coisas que você pode

fazer, enquanto que num set o número de operações que você pode fazer é muito mais limitado;