Javascript Set - Theorie

Set

Ein `Set` ist ebenso wie eine `Map` oder ein `Array` eine Sammlung von Objekten. Ein `Set` hat allerdings **keinen Index** und **keinen Key**. Ein Set garantiert, dass **jedes Objekt im Set nur einmal vorkommt**.

Mengenlehre - ein `Set` entspricht einer Menge aus der Mengenlehre.

`new Set()`

```
class Person {
  constructor(name, eyeColor) {
    this.name = name;
    this.eyeColor = eyeColor;
  }
}

const peopleSet = new Set();

let marie = new Person('Marie', 'brown');
let peter = new Person('Peter', 'blue');
let anna = new Person('Anna', 'green');
let marie2 = new Person('Marie', 'brown'); // marie2 ist ein "neues Objekt" mit
den gleichen Werten wie marie
let marie3 = { ...marie2 }; // marie3 ist ein Clone von marie2
let marie4 = marie2; // marie4 ist eine Referenz auf marie2. marie2 und marie4
zeigen auf das gleiche Objekt (den gleichen Speicherbereich im Hauptspeicher)
```

`add(value)`

fügt ein neues Objekt in das `Set` ein. Das `Set` verhindert, dass einzelne Objekte doppelt im Set vorkommen. Man könnte dies mit einem Array auch selbst programmieren, das wäre aber aufwändiger und wahrscheinlich langsamer.

Hinweis: Sieh dir die Kommentare **genau** an, um zu verstehen, was bei den einzelnen Code-Zeilen passiert!

```
peopleSet.add(marie);
peopleSet.add(peter);
peopleSet.add(anna);
output2.innerHTML += '<br/>' + 'peopleSet.size:' + peopleSet.size; // 3

peopleSet.add(marie);
// add marie a second time: set keeps only unique values
output2.innerHTML += '<br/> marie second time' + 'peopleSet.size:' +
peopleSet.size; // bleibt 3

// for the interpreter marie2 is a new object.
```

```
peopleSet.add(marie2);
output2.innerHTML += '<br/> marie2: ' + 'peopleSet.size:' + peopleSet.size; // 4

// marie3 is a copy of marie2
peopleSet.add(marie3);
output2.innerHTML += '<br/> marie3: ' + 'peopleSet.size:' + peopleSet.size; // 5

// marie4 ist eine Referenz auf marie2. marie2 und marie4 zeigen auf das gleiche
Objekt
peopleSet.add(marie4);
output2.innerHTML += '<br/> marie4: ' + 'peopleSet.size:' + peopleSet.size; //
bleibt 5

for (let user of peopleSet) {
  output2.innerHTML += '<br/>' + user.name + ' - ' + user.eyeColor;
}
```

Ausgabe aller User:

```
for (let user of peopleSet) {
  output2.innerHTML += '<br/>' + user.name + ' - ' + user.eyeColor;
}
```

Output:

```
Marie - brown
Peter - blue
Anna - green
Marie - brown
Marie - brown
```

`delete(value)`

löscht ein Element aus dem `Set`.

```
peopleSet.delete(marie3);

for (let user of peopleSet) {
  output2.innerHTML += '<br/>' + user.name + ' - ' + user.eyeColor;
}
```

`has(value)`

prüft, ob ein Objekt im `Set` ist.

```
if (peopleSet.has(marie2)) {
  output2.innerHTML += '<br/>' + marie2.name + ' ist noch da';
}
```

--

Objekte mit dem gleichen Namen im Set

Wenn ihr Objekte mit dem gleichen Namen in das Set speichert müsst ihr in einer Schleife alle Objekte durchgehen um das Richtige zu löschen.

```
const peopleSet2 = new Set();

let person = new Person('Marie', 'brown');
peopleSet2.add(person);
person = new Person('Peter', 'blue');
peopleSet2.add(person);
person = new Person('Anna', 'green');
peopleSet2.add(person);

output2.innerHTML += '<br/>' + 'peopleSet2.size: ' + peopleSet2.size; // 3


for (let user of peopleSet2) {
  //output2.innerHTML += '<br/>' + user.name + ' - ' + user.eyeColor;
  if (user.name === 'Marie') {
    user.name = 'Hannah';  // Wenn ihr das Objekt ändern wollt
    peopleSet2.delete(user); // löscht das Objekt
  }
}

for (let user of peopleSet2) {
  output2.innerHTML += '<br/>' + user.name + ' - ' + user.eyeColor;
}
output2.innerHTML += '<br/>' + 'peopleSet2.size: ' + peopleSet2.size; // 2
```

A set is an unordered and mutable collection of unique elements :

- **A set is unordered:** users can't rely on the way the elements are stored within it. While sets are iterable, it is impossible to predict in which order their elements will appear.
- **Every element within a set is unique**
- **A set is mutable (veränderlich) but unindexed:** users can add and remove elements from it "in place", but since it is unindexed, users can't access its values without manipulating the whole collection, which makes it very uneasy.

--

Uniqueness: Array vs. Set

An `array` can hold duplicates of the same value. A `set` prohibits this!

Filter unique array members

Let `arr` be an array.

Create a function `unique(arr)` that should return an array with unique items of `arr`.

For instance:

```
// function that returns an array with unique values from array "arr"
function unique(arr) {
  /* your code */
}

//Main part of program
let values = ["Hare", "Krishna", "Hare", "Krishna",
  "Krishna", "Krishna", "Hare", "Hare", ":-O"
];

alert( unique(values) ); // Hare, Krishna, :-O
```

P.S. Here strings are used, but can be values of any type.

P.P.S. Use `Set` to store unique values!

Solution:

```
function unique(arr) {
  return Array.from(new Set(arr));
}
```

Autors

HUO
GAM
SOR

Sources

https://javascript.info/map-set

https://javascript.info

https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Set

https://medium.com/@macargnelutti/python-and-javascript-sets-are-cool-lets-use-them-5f4d6f6ad5cf

https://javascript.info/map-set#filter-unique-array-members