Array Functions

SPENGERGASSE

Vorbemerkung array-likes und iterables

Nicht alles was aufzählbar ist, ist gleich ein Array!

`Array.from(aufzählbar)` hilft oft, indem es ein Array daraus macht. Beispiel

```
let x = document.querySelectorAll("span")
x
NodeList(4) [span#fahrpreis, span#guthaben, span, span#einnahmen]
x instanceof NodeList
true
x.map(()=>{})
VM132:1 Uncaught TypeError: x.map is not a function
    at <anonymous>:1:3
y=Array.from(x)
y.map(()=>{})
(4) [undefined, undefined, undefined, undefined]
```

Bemerkung `in` und `of`

Objekte: `for (i in obj)`: `in` liefert alle *keys* von obj, `of` geht bei Objekten nicht.

Array - Iteratoren: Hier ist es genau umgekehrt: Man bekommt die Werte mit `of`:

```
> array = ["a", 7, true]
[ 'a', 7, true ]
> it = array.values()  // iterator
Object [Array Iterator] {}
> for (i of it) console.log(i)
a
7
true
```

Bei **Arrays** ist beides möglich:

- `for (let i in array)` iteriert über alle indices. S.u. `array.keys()`
- `for (let v of array)` iteriert über alle values. S.u. `array.values()`

Array Methods

Beispiele mit `array = ["a", 7, true]`

- array.at(index), gleich wie array[index], negative Werte erlaubt
- array.concat(array2), modifiziert nicht, returned das zusammengekettete
- array.constructor
- array.copyWithin
- array.entries(): iterator

```
> Array.from(array.entries())
[ [ 0, 'a' ], [ 1, 7 ], [ 2, true ] ]
```

- array.every(cb) callback wird für jedes element aufgerufen und returned boolean (ist es true for every element?)
- array.fill(what, istart, iend) Sachen in ein Array einfüllen. Nur fill() gibt lauter `undefined` hinein. **inplace!**
- array.filter(cb): Neues Array mit allen Elementen wo cb ”true” ergibt.
- array.find(cb): liefert das element oder undefined (cb: boolean)
- array.findIndex(cb): wie oben, aber Index
- array.findLast(cb): wie gehabt, aber von hinten
- array.findLastIndex(): ebenso.
- array.flat(): wenn das array sub-arrays hat, werden diese ”aufgemacht”, das returnte array hat keine sub-arrays mehr.
- array.flatMap()
- array.forEach(cb): cb wird für jedes Element aufgerufen
- array.includes(elem): boolean
- array.indexOf(elem): index
- array.join(sepStr): array wird zu einem String zusammengejoined
- array.keys(): Interator über Indexe (`for (let i of array.keys())`)
- array.lastIndexOf(elem)
- array.map(cb): Neues Array mit jedem return-wert von cb
- array.pop(): entfernt und returned das letzte Element
- array.push(elt): hängt elt an
- array.reduce(cb, initialValue): ermöglicht beliebig komplexe Operationen. Argumente für callback cb(accumumlator, currentValue, currentIndex, originalArray)
- array.reduceRight()
- array.reverse(): inplace!
- array.shift(): entfernt und returniert das elt mit index 0. (wie pop() von links)

- array.slice(): retourniert ein ”Scheiberl”. Ändert nicht.
- array.some(cb): neues array mit allen elementen wo cb true war.
- array.sort(cb): inplace sort. cb returned <0, 0, oder >0.
- array.splice(start, deleteCount, item1, item2, ...): in-place modification
- array.toLocaleString()
- array.toReversed(): returned reversed
- array.toSorted(cb): returned sortiert, siehe sort()
- array.toSpliced()
- array.toString(): String des arrays, siehe auch join()
- array.unshift(elt): fügt elt an den index 0 ein (wie push() von links)
- array.values(): iterator über values
- array.with(): allows you to create a new array with a modified value at a specific index, without mutating the original array.