

Array hjælpe metoder

I Javascript findes der en række hjælpe metoder til Arrays. Nogle af metoderne fx **forEach** benyttes til at mutere (ændre/opdatere) det oprindelige Array andre returnere et nyt Array enten af samme størrelse som fx med **map** eller et nyt Array der kan være af anden størrelse fx med **filter** og **reduce**. Andre metoder returnere en enkelt værdi fx det første item der opfylder et kriterie – **find**, andre igen en boolsk værdi afhængigt om alle items opfylder et kriterie – **every**, eller minimum et item der opfylder kriteriet – **some**.

Fælles for alle metoderne er at de tager en CallBack-funktion som argument. CallBack funktionen tager 3 argumenter: currentValue (der er det aktuelle element i arrayet der bliver behandlet/processeret), index (der er indekset for currentValue) og array (der er det array som metoden er kald på). Index og array er optional.

Array Helper Methods

forEach

map

filter

find

every

some

reduce

forEach

Syntaks :

```
arr.forEach(callback[, thisArg]);
```

forEach() kalder (executes) callback-funktionen på alle elementerne i arrayet. Callback-funktionen bliver kaldt med de 3 argumenter: værdien af det aktuelle element, indekset af det aktuelle element og selve arrayet der bliver gennemløbet. *thisArg* (optional) vil blive benyttet som callback-funktionens *this* værdi.

Selve **forEach** metoden mutere (ændre) ikke arrayet, men det kan man med callback-funktionen (via 3. parameteren array, se nedenstående eksempel). **forEach** metoden returnere ikke noget (`undefined`).

Eksempel:

a) Gennemløber arrayet *numbers* vha. **forEach**-metoden og udskriver index og value for de enkelte elementer.

b) Gennemløber arrayet *numbers* vha. **forEach**-metoden og multiplicere elementerne med 2.

Bemærk, det er ikke selve *forEach*-metoden, men callback-funktionen der muterer arrayet, ved at opdatere indholdet med resultatet af multiplikationen:

```
var numbers = [1, 2, 3, 4, 5];

console.log(numbers);
numbers.forEach((item, index, array) => console.log(index, item));

numbers.forEach((item, index, array) => array[index]=item*2)
console.log(numbers);
```

```
▶ (5) [1, 2, 3, 4, 5]
  0 1
  1 2
  2 3
  3 4
  4 5
▶ (5) [2, 4, 6, 8, 10]
```

Eksempel:

Her erklæres funktionen *increaseSalary*. Funktionen benytter **forEach**-metoden til at gennemløbe arrayet *employees* og mutere arrayet ved at opdatere proprietien *salary* på den enkelte *employee* med værdien *amount*:

```
var employees = [
  {name: "Ib", salary: 45000},
  {name: "Bo", salary: 35000},
  {name: "Per", salary: 50000}
]

var increaseSalary = function(employees, amount){
  employees.forEach(employee => employee.salary+=amount);
}

increaseSalary(employees, 1000);
console.log(employees);
```

```
▼ (3) [{...}, {...}, {...}] ⓘ
  ▶ 0: {name: "Ib", salary: 46000}
  ▶ 1: {name: "Bo", salary: 36000}
  ▶ 2: {name: "Per", salary: 51000}
  length: 3
  ▶ __proto__: Array(0)
```

map

Syntaks:

```
var new_array = arr.map(function callback(currentValue[, index[, array]]) {  
    // Return element for new_array  
} [, thisArg])
```

map-metoden gennemløber det array metoden er kaldt på og den funktion der gives som argument til **map**-metoden, kaldes for hvert element i arrayet, med det enkelte elementet som argument til funktionskaldet. **map**-metoden mutere ikke (ændre ikke) det array den blev kaldt på, men opretter et nyt array af samme størrelse med resultaterne af funktionskaldene.

Bemærk at funktionen naturligvis ikke kaldes, hvis arrayet er tomt!

Eksempel: I nedenstående eksempel benyttes **map**-metoden til at gennemløbe arrayet *arr* og multiplitere alle elementer med 2. Bemærk **map** returnerer et nyt array der gemmes i *map1*

```
var arr = [1, 2, 9, 16];  
  
const map1 = arr.map(x => x * 2);  
  
console.log(map1); // expected output: Array [2, 4, 18, 32]
```

Eksempel: I nedenstående eksempel benyttes funktionen *Math.sqrt* som callback-funktion til **map**-metoden, for at beregne kvadratroden af alle tal i arrayet *numbers*:

```
var numbers = [1, 4, 9, 16];  
var roots = numbers.map(Math.sqrt);  
// expected output: [1, 2, 3, 4]  
// numbers is still [1, 4, 9, 16]
```

Eksempel: I nedenstående eksempel benytter vi **map-metoden** til udtrække *firstName* fra et array af *Person*-objekter samt til at mappe property navne (keys) fra engelsk til dansk for alle *Person* objekterne i arrayet *persons*:

```

class Person {
  constructor(firstName, lastName, age){
    this._firstName=firstName;
    this._lastName=lastName;
    this._age=age;
  }

  get firstName(){return this._firstName;}
  set firstName(value){this._firstName=value;}
  get lastName(){return this._lastName;}
  set lastName(value){this._lastName=value;}
  get age(){return this._age;}
  set age(value){this._age=value;}
}

const persons = [new Person("Peter", "Pan", 27), new Person("Donald", "Duck", 35), new Person("Robin", "Hood", 46)];

let personsFirstName = persons.map(p=>p.firstName);
console.log(personsFirstName);

let personsJSON = persons.map(p=>{return {Fornavn: p.firstName, Efternavn: p.lastName, Alder: p.age }});
console.log(JSON.stringify(personsJSON));

```

```

▼ (3) ["Peter", "Donald", "Robin"] ⓘ
  0: "Peter"
  1: "Donald"
  2: "Robin"
  length: 3
  ► __proto__: Array(0)

[{"Fornavn": "Peter", "Efternavn": "Pan", "Alder": 27},
 {"Fornavn": "Donald", "Efternavn": "Duck", "Alder": 35},
 {"Fornavn": "Robin", "Efternavn": "Hood", "Alder": 46}]
  >

```

filter

Syntaks:

```
var newArray = arr.filter(callback(element[, index[, array]])[, thisArg])
```

filter-metoden opretter et nyt array og indsætter alle de elementer der opfylder kriteriet givet ved callback-funktionen. Callback-funktionen til **filter**, er det "predicate" der tester elementet, hvis funktionen returnere true gemmes elementet i det nye array, ellers ikke.

Eksempel: find alle navne i arrayet navne, der har en længde større end 5:

```

var navne = ['Henrik', 'Per', 'Anders', 'Peter', 'Poul', 'Ebbe', 'Michael'];
const res = navne.filter(navn => navn.length > 5);

console.log(res); // expected output: Array ["Henrik", "Anders", "Michael"]

```

Eksempel: find alle diesel biler i arrayet cars (fra opgaven forEach-1)

```
//cars fra opgaven med forEach
let cars = [
  {brand: 'VW', model: 'Passat', fuel: 'diesel', owner_tax: 5550 },
  {brand: 'VW', model: 'Passat', fuel: 'gasoline', owner_tax: 460},
  {brand: 'VW', model: 'Passat', fuel: 'hybrid', owner_tax: 150},
  {brand: 'BMW', model: '320i', fuel: 'diesel', owner_tax: 4280},
  {brand: 'BMW', model: '320i', fuel: 'gasoline', owner_tax: 430},
  {brand: 'BMW', model: '320i', fuel: 'hybrid', owner_tax: 210},
  {brand: 'Tesla', model: 'S', fuel: 'electric', owner_tax: 0 }
]

let dieselCars = cars.filter(car => car.fuel === 'diesel');
console.log(dieselCars);
```

▼ (2) [{...}, {...}] ⓘ
▶ 0: {brand: "VW", model: "Passat", fuel: "diesel", owner_tax: 8325}
▶ 1: {brand: "BMW", model: "320i", fuel: "diesel", owner_tax: 6420}
length: 2
► __proto__: Array(0)

find

Syntaks:

```
arr.find(callback[, thisArg])
```

find-metoden returnerer værdien af det første element der opfylder kriteriet der testes af callback-funktionen. Callback-funktionen skal være en "predicate" function der returnere true/false.

Bemærk: **findIndex** returnere indekset for første element der opfylder kriteriet.

Eksempel: find i arrayet cars, en benzin bil med en grøn afgift på under 450:

```
let gasolineCarLowTax = cars.find(car => car.fuel === 'gasoline' && car.owner_tax < 450); console.log(gasolineCarLowTax);

ArrayHelp.js:221
▼{brand: "BMW", model: "320i", fuel: "gasoline", owner_tax: 430} ⓘ
  brand: "BMW"
  fuel: "gasoline"
  model: "320i"
  owner_tax: 430
  ► __proto__: Object
```

every

Syntaks:

```
arr.every(callback[, thisArg])
```

every-metoden gennemløber arrayet og returnere true, hvis alle elementer i arrayet opfylder kriteriet ellers false. Callback-funktionen skal være en "predicate" der returnerer true/false.

Eksempel:

```
// har alle biler en grøn afgift?  
res = cars.every(car => car.owner_tax > 0);
```

some

Syntaks:

```
arr.some(callback[, thisArg])
```

some-metoden gennemløber arrayet og returnere true, hvis blot et af elementer i arrayet opfylder kriteriet ellers false. Callback-funktionen skal være en "predicate" der returnerer true/false.

Eksempel:

```
// er der biler med en grøn afgift under 200?  
res = cars.some(car => car.owner_tax < 200);
```

reduce

```
arr.reduce(callback[, initialValue])
```

reduce-metoden kalder en "reducer-function" / callback-funktion, på hvert element i arrayet og returnere en enkelt værdi. "Reducer" funktionen tager 4 argumenter: *accumulator*, *currentValue*, *currentIndex* og *array*. **Reduce**-metoden selv tager foruden callback-funktionen en *initialValue* (optional) som argument.

Hvis **reducer**-metoden bliver kaldt med en *initialValue* vil *accumulator* blive tildele denne startværdi og

currentValue vil blive tildelt den første værdi i arrayet. Såfremt der ikke er en *initialValue* bliver *accumulator* og *currentValue* tildelt de 2 første værdier i arrayet.

accumulator argumentet bliver løbende tildelt returværdien fra "reducer"/callback funktionen og bliver overført gennem hver iteration af arrayet. *accumulator* vil til slut indeholde resultatet i form af en enkelt akkumuleret værdi.

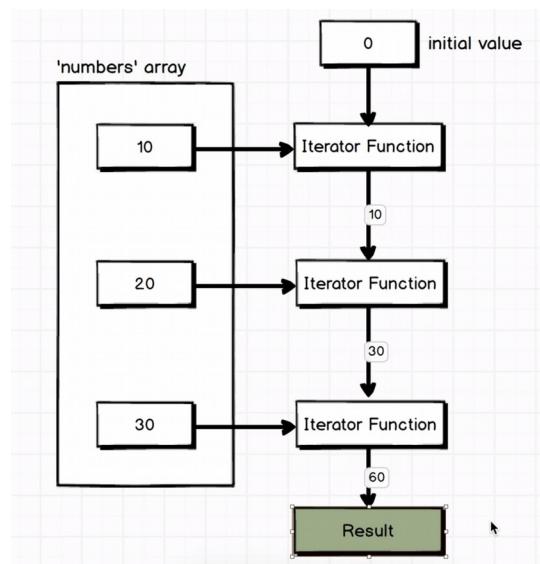
Eksæmplet: I eksemplet benyttes **reduce**-metoden til at gennemløbe arrayet *numbers*. *sum* er vores *accumulator* og *initialValue* er 0. Callback-funktionen summere tallene i *numbers*. Der er vist 2 versioner, en "traditionel" og en med den "nye" arrow-function:

```
var numbers = [10, 20, 30];

var res = numbers.reduce(function(sum, number) {
    sum += number;
    return sum
}, 0);

var res = numbers.reduce((sum, number) => sum += number, 0); //60
```

Reduce virker som illustreret i nedenstående figur:



Eksæmplet: I nedenstående eksempel gennemløbes arrayet *primaryColors*, *previous* er vores *accumulator* og *initialValue* er et tomt array (`[]`). Callback-funktionen push'er (indsætter) valuen fra proprietien *color* i *previous* og returnere *previous*.

```

1 var primaryColors = [
2   { color: 'red' },
3   { color: 'yellow' },
4   { color: 'blue' }
5 ];
6
7 primaryColors.reduce(function(previous, primaryColor) {
8   previous.push(primaryColor.color);
9
10  return previous;
11 }, []);
12

```

Eksempel: I nedenstående eksempel vises hvordan man kan benytte flere array hjælpe metoder ved i kæde at kalde dem efter hinanden (først kaldes **filter**-metoden på arrayet, dernæst kaldes **map**-metoden på returværdien fra **filter** og til slut kaldes **reduce**-metoden på returværdien fra **map**)

```

var personnel = [
  {
    id: 5,
    name: "Luke Skywalker",
    pilotingScore: 98,
    shootingScore: 56,
    isForceUser: true,
  },
  {
    id: 82,
    name: "Sabine Wren",
    pilotingScore: 73,
    shootingScore: 99,
    isForceUser: false,
  },
  {
    id: 22,
    name: "Zeb Orellios",
    pilotingScore: 20,
    shootingScore: 59,
    isForceUser: false,
  },
  {
    id: 15,
    name: "Ezra Bridger",
    pilotingScore: 43,
  }
]

```

```
        shootingScore: 67,
        isForceUser: true,
    },
{
    id: 11,
    name: "Caleb Dume",
    pilotingScore: 71,
    shootingScore: 85,
    isForceUser: true,
},
];
//Our objective: get the total score of force users only. Let's do it
step by step!

//First, filter out the personnel who can't use the force:
var jediPersonnel = personnel.filter(function (person) {
    return person.isForceUser;
});
// Result: [{...}, {...}, {...}] (Luke, Ezra and Caleb)

//Next, create an array containing the total score of each Jedi:
var jediScores = jediPersonnel.map(function (jedi) {
    return jedi.pilotingScore + jedi.shootingScore;
});
// Result: [154, 110, 156]

//Finally, use reduce to get the total:
var totalJediScore = jediScores.reduce(function (acc, score) {
    return acc + score;
}, 0);
// Result: 420

//we can chain all of this to get what we want in a single line,
//and look how pretty it is with arrow functions:
totalJediScore = personnel
    .filter(person => person.isForceUser)
    .map(jedi => jedi.pilotingScore + jedi.shootingScore)
    .reduce((acc, score) => acc + score, 0);
```

JSON.parse() og JSON.stringify()

Når man kalder en webservice modtager man ofte data som en tekst-streng der indeholder objekter i JSON-format. For at konvertere tekst-strengen til JavaScript objekter benyttes metoden JSON.parse() fx

```
var txtJSON = '{"name": "John", "age":30, "city": "New York"}' //string med et objekt i JSON-format
var objJS = JSON.parse(txtJSON);
console.log(objJS);
```

```
▼ {name: "John", age: 30, city: "New York"} ⓘ
  age: 30
  city: "New York"
  name: "John"
▶ __proto__: Object
```

Den modsatte vej, er når data i form af JavaScript objekter skal sendes til en webservice. Her benyttes JSON.stringify() til at oprette en tekst-streng med objekterne på JSON-format:

```
objJS = {name:"John", age:30, city:"New York"}; // JavaScript objekt
txtJSON = JSON.stringify(objJS);
console.log(txtJSON);
```

```
{"name": "John", "age": 30, "city": "New York"}
```

Weird Things in JavaScript

Logging objects

```
console.log(obj);
```

"Please be warned that if you log objects in the latest versions of Chrome and Firefox what you get logged on the console is a *reference to the object*, which is not necessarily the 'value' of the object at the moment in time you call console.log(), but it is the value of the object at the moment you click it open."

“Don't use `console.log(obj);`,
use `console.log(JSON.parse(JSON.stringify(obj)));`.
This way you are sure you are seeing the value of obj at the moment you log it.”
<https://developer.mozilla.org/en-US/docs/Web/API/Console/log>

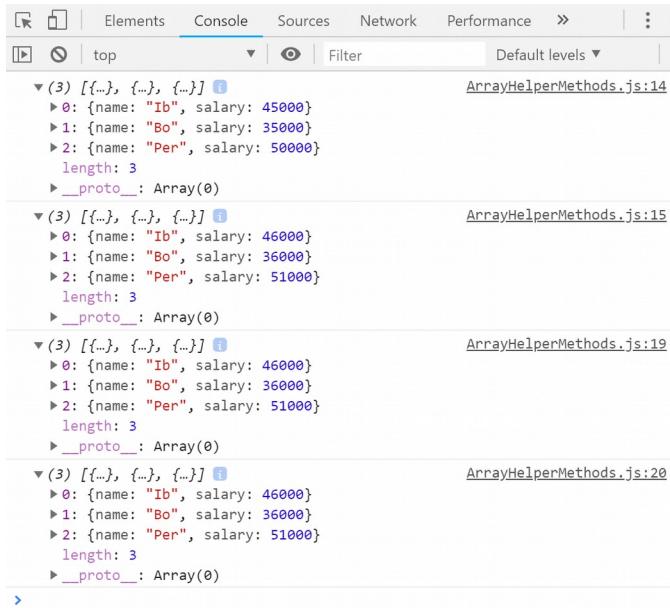
```
var employees = [
  {name: "Ib", salary: 45000},
  {name: "Bo", salary: 35000},
  {name: "Per", salary: 50000}
]

var increaseSalary = function(employees, amount){
  employees.forEach(employee => employee.salary+=amount);
}

console.log(JSON.parse(JSON.stringify(employees)));
console.log(employees);

increaseSalary(employees, 1000);

console.log(JSON.parse(JSON.stringify(employees)));
console.log(employees);
```



Bemærk:

`console.log(employees)` logger værdien af `employees` på det tidspunkt Chrome åbnes og ikke på det tidspunkt funktionen kaldes – derfor brug altid `console.log(JSON.parse(JSON.stringify(obj)))`; når der skal logges et objekt!