Højere ordens funktioner

HOF

> En højere orden funktion er en funktion der enten tager en(eller flere) funktion(er) som argument og eller returnerer en funktion

Funktioner som argumenter

```
isEven = (num) => num % 2 === 0;
result = [1, 2, 3, 4].filter(isEven);

console.log(result); // [2, 4]
```

Returnerende funktioner

```
add = (x) => (y) => x + y;

// på en gang
result = add(10)(20);

// eller hver for sig
add10 = add(10);
result = add10(20);

console.log(result); // 30
```

Eksempler på indbyggede HOF

Vi bruger følgende array af objekter som udgangspunkt:

```
users = [
  {
    name: 'Yazeed',
    age: 25
  },
  {
    name: 'Sam',
    age: 30
  },
  {
    name: 'Bill',
    age: 20
  }
];
```

Map

Vi kan gøre det her:

```
getName = (user) => user.name;
usernames = [];

for (let i = 0; i < users.length; i++) {
  const name = getName(users[i]);

  usernames.push(name);
}

console.log(usernames);
// ["Yazeed", "Sam", "Bill"]
```

Map

Eller vi kunne gøre det her:

```
usernames = users.map(getName);

console.log(usernames);
// ["Yazeed", "Sam", "Bill"]
```

Så map kalder en function på hvert element i et array og returnerer et nyt array med resultaterne

`Filter`

Vi kan gøre det her:

```
startsWithB = (string) => string.toLowerCase().startsWith('b');

namesStartingWithB = [];

for (let i = 0; i < users.length; i++) {
  if (startsWithB(users[i].name)) {
    namesStartingWithB.push(users[i]);
  }
}

console.log(namesStartingWithB);
// [{ "name": "Bill", "age": 20 }]
```

`Filter`

Eller vi kunne gøre det her:

```
namesStartingWithB = users.filter((user) => startsWithB(user.name));

console.log(namesStartingWithB);
// [{ "name": "Bill", "age": 20 }]
```

Så filter returnerer et array med de elementer, der tilfredsstiller filteret

Reduce

Vi kan gøre det her:

```
totalAge = 0;

for (let i = 0; i < users.length; i++) {
  totalAge += users[i].age;
}
let averageAge = totalAge/users.length;

console.log(averageAge);
// 75
```

Reduce

Eller det her:

I (total, user) => user.age + total er første argument en akkumulator der til sidst returneres.
Functionen har ansvaret for at anvende akkumulatoren som ønsket.

```
totalAge = users.reduce((total, user) => user.age + total, 0);

console.log(totalAge/users.length);
// 75
```

Andre højere ordens funktioner på Array

- sort()
- foreach()
- find()
- findIndex()
- every()
- some()

sort

```
// sort.js
let tal = [30, 4, 10, 2, 0];

console.log(tal.sort()); // => [ 0, 10, 2, 30, 4 ]
console.log(tal); // => [ 0, 10, 2, 30, 4 ]

tal.sort((a, b) => a-b);
console.log(tal); // => [ 0, 2, 4, 10, 30 ]

tal.sort((a, b) => b-a);
console.log(tal); // => [ 30, 10, 4, 2, 0 ]
```

foreach

```
// foreach.js
let tal = [1, 2, 3, 4, 5];

let sum = 0;
tal.forEach(element => sum += element);
console.log(sum); // => 15

tal.forEach((element) => element++);
console.log(tal); // => [1, 2, 3, 4, 5]
//tæller ikke element op, bruger kopi af element

tal.forEach((element, index, array) => array[index]++);
console.log(tal); // => [ 2, 3, 4, 5, 6]
```

find

```
// find.js
let tal = [1, 2, 3, 4, 5];

console.log(tal.find(element => element % 2 === 0)); // 2
console.log(tal.find(element => element === 0)); // undefined
```

Chaining

De fleste hof i js returnerer et array så det er muligt at chaine brugen af hof

```
// chaining.js
let personer = [{navn: 'Åge', alder: 32}, {navn: 'Ida', alder: 23}];

console.log(personer.map(p => p.alder).reduce((a, e) => a + e)); // => 55

console.log(personer.sort((p1, p2) => p1.alder-p2.alder).map(p => p.navn)); // => [ 'Ida', 'Åge' ]

console.log(personer.find(p => p.navn === 'Ida').alder); // => 23
```