Hello World

Prowadzący
Tomasz Nastały

Operacje na tablicach

Array.prototype

JavaScript oferuje nam szereg metod ułatwiających operacje na tablicach:

- Mapowanie tablic (map)
- Filtrowanie tablic (filter)
- Spłaszczanie tablic do jednej wartości (reduce)
- Sprawdzenie, czy jakikolwiek element w kolekcji spełnił warunek (some)
- Sprawdzenie, czy wszystkie elementy kolekcji spełniły warunek (every)
- Konkatencja elementów do stringa (join)
- Odwrócenie kolejności elementów w tablicy (reverse)
- Zwrócenie indexu elementu (indexOf)
- Wykonanie operacji dla każdego elementu (forEach)
- Sortowanie kolekcji (sort)

2 typy metod

Rozróżniamy 2 typy metod:

Zmieniające tablicę wejściową (mutujące):

- Push
- Shift
- Pop
- Unshift
- Sort
- Reverse

…

Niezmieniające tablicę wejściową (niemutujące):

- Map
- Filter
- Reduce
- Some
- Every
- Join

…

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/prototype#Mutator_methods

Co zwracają metody?

Metoda	Return
Map	[ ] - nowa tablica
Filter	[ ] – nowa tablica
Reduce	Obiekt, tablica, liczba, string…
Some	Boolean – true lub false
Every	Boolean – true lub false
Sort	[ ] – referencja do tablicy
Reverse	[ ] – referencja do tablicy
Join	string
IndexOf	Liczba (-1 jeśli element nie istnieje)
ForEach	Undefined (nic nie zwraca)

Przykład użycia MAP

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

var powNumbers = numbers.map(function(number) {
    return Math.pow(number, 2);
});
```

```
function powNumber(number) {
    return Math.pow(number, 2);
}

var powNumbers = numbers.map(powNumber);
```

Brak nawiasów
() !

Przykład użycia MAP

```
var users = [
    { login: 'Mike', password: 'h4ix' },
    { login: 'Johnny', password: 'lubie placki' },
    { login: 'Romuald', password: 'qwerty' },
    { login: 'Thomas', password: 'asia123' }
];

var logins = users.map(function(user) {
    return user.login;
});
```

Nie tylko jeden parametr…

```
var numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

var oddOrEven = numbers.map(function(number, i, arr) {
    return i % 2 === 0 ? number + ' parzysty' : number + ' nieparzysty';
});
```

Pierwszy parametr – wskazuje na kolejne elementy tablicy
Drugi parametr – wskazuje na index elementu w danej iteracji
Trzeci parametr – tablica po której mapujemy

Parametry w JS zawsze możemy nazwać jak chcemy –
Więc wyżej może być np. (N, idx, self) lub (N, index, array)… etc

Kolejność parametrów ma znaczenie!

Zadania – map.js

Metody FILTER, SOME, EVERY

Filter, Some i Every, muszą przyjąć funkcję która zwraca true lub false.
Wykonują test.

Filter – jeśli element spełnił test, to zostawia go w kolekcji **–** jak nie, to usuwa, zwraca przefiltrowaną tablicę

Every – jeśli wszystkie elementy przeszły test **–** zwraca true, jak nie, to false.

Some – jeśli trafi na pierwszy element który spełnił test, od razu zwraca true, jeśli żaden nie spełnił, to zwraca false.

Przykład użycia FILTER, SOME, EVERY

```
var numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

var evenNumbers = numbers.filter(function(N, i) {
    return i % 2 === 0;
});  // [0, 2, 4, 6, 8, 10];

var isAnyNumberGreaterThanFor = numbers.some(function(N) {
   return N > 4;
}); // true

var areAllNumbersGreaterThanFor = numbers.every(function(N) {
   return N > 4;
}); // false
```

Filtrowanie z dodatkowym parametrem

```
var paintings = [
  { code: '4432', color: 'red' },
  { code: '7755', color: 'black' },
  { code: '1111', color: 'white' },
  { code: '4347', color: 'yellow' },
  { code: '8812', color: 'yellow' }
];

function filterByColor(color) {
  return function(painting) {
    return painting.color === color;
  }
}

var yellowPaintings = paintings.filter(filterByColor('yellow'));
var redPaintings = paintings.filter(filterByColor('red'));
```

Zadania – filter.js

Reduce

Reduce pozwala spłaszczyć tablicę do oczekiwanej wartości.

```
var products = [
    { name: 'Coffee', price: 30, inStock: false },
    { name: 'Milk', price: 50, inStock: true },
    { name: 'Chocolate', price: 12, inStock: true },
    { name: 'Bread', price: 15, inStock: false }
];

var sum = products.reduce(function(acc, next) {
    return acc + next.price;
}, 0);
```

WARTOŚĆ POCZĄTKOWA

Zadania – reduce.js

Chaining - łańcuchowanie

```
var products = [
    { name: 'Coffee', price: 30, inStock: false },
    { name: 'Milk', price: 50, inStock: true },
    { name: 'Chocolate', price: 12, inStock: true },
    { name: 'Bread', price: 15, inStock: false }
];

var productsList = products
    .map(function(product) { return product.name; })
    .filter(function(name) { return name.startsWith('C'); })
    .map(function(name, i) { return (i + 1) + '. ' + name; })
    .join(' ')
    .toUpperCase();
```

Kopiowanie bez referencji

```
var products = [
    { name: 'Coffee', price: 30, inStock: false },
    { name: 'Milk', price: 50, inStock: true },
    { name: 'Chocolate', price: 12, inStock: true },
    { name: 'Bread', price: 15, inStock: false }
];

var copiedProducts = [...products]; (spread operator Z ES6)
```

Zadania – other.js & modify-array.js