javascript notatki

lista wbudowanych funkcji

WBM = **Wersja Bez Modyfikacji** (Istniejącej Tablicy)

array.at()
Przyjmuje liczbę i zwraca wartość pod tym indeksem.

array.concat()
Łączy dwie tablice.

array.every()
Sprawdza czy wszystkie elementy spełniają predykat.

array.fill()
Zmienia wszystkie elementy na stałą wartość

array.filter()
Tworzy nową tablicę z elementami spełniającymi predykat.

array.find()/array.findLast()
Zwraca pierwszy/ostatni element spełniający predykat.

array.findIndex()/array.findLastIndex()
Zwraca indeks pierwszego/ostatniego elementu spełniającego predykat.

array.flat()
Tworzy nową rekursywnie spłaszczoną tablicę.

array.flatMap()
Przechodzi po elementach i je modyfikuje, potem spłaszcza tablicę o jeden poziom.

array.forEach()
Przechodzi po elementach tablicy.

array.includes()
Sprawdza czy tablica zawiera daną wartość.

array.indexOf()/array.lastIndexOf()
Zwraca pierwszy/ostatni indeks na którym pojawia się dana wartość.

Array.isArray(array)
Sprawdza czy podana wartość to tablica

array.join()
Zwraca stringa złożonego z elementów tablicy rozdzielonych podanym separatorem.

array.map()
Przechodzi po elementach i je modyfikuje za pomocą przekazanej funkcji.

Array.of()
Tworzy tablicę z podanych elementów

array.pop()/array.shift()
Usuwa ostatni/pierwszy element z tablicy i go zwraca.

array.push()/array.unshift()
Dodaje element na koniec/początek tablicy i zwraca nową długość tablicy

array.reduce()
No jeżeli tego nie wiesz to wydaje mi się, że te notatki niewiele pomogą.

array.reduceRight()
Reduce od prawej.

array.reverse()
Odwraca tablicę. WBM: array.toReversed()

array.slice()
Tworzy nową tablicę zawierającą elementy pomiędzy podanymi indeksami.

array.some()
Sprawdza czy co najmniej jeden element spełnia predykat.

array.sort()
Sortuje tablicę według podanej funkcji. Domyślnie (bez podania argumentu) sortuje alfabetycznie. Bardziej dokładny opis niżej. WBM: array.toSorted()

array.splice()
Usuwa/dodaje/podmienia istniejące elementy tablicy. WBM: array.toSpliced()

wartości falsey

- `false`
- `0`
- `-0`
- `0n`
- `“”`
- `null`
- `undefined`
- `NaN` => (Not a Number)
  reszta wartości to wartości truthy

scope

Zmienne są dostępne tylko na poziome zagnieżdżenia na którym zostały zadeklarowane i na poziomach niżej.

```
let number = 3;
console.log(number); {
        let number = 4;
        console.log(number);
}
console.log(number);

// na ekranie zostanie wyświetlone 3, 4, 3
// wynika to z zagnieżdżenia nadpisania zmiennej number
```

spread operator (...)

- przyjęcie nieograniczonej ilości argumentów do funkcji (zachowują się jakbyśmy przekazali tablicę)

```
function speak(…args){
        console.log(args)
}
```

- funkcyjne dodanie elementu do tablicy poprzez rozłożenie tablicy spread operator

```
// na koniec
array = [...array, elem]
// na początek
array = [elem, ...array]
```

var

Var deklaruje zmienną globalną, a let/const nie (są one zależne od poziomu zagnieżdżenia deklaracji). Var również pozwala na ponowną deklarację zmiennej o tej samej nazwie co inna zmienna.

hoisting

Przenoszenie deklaracji funkcji (tylko o tym mówił na wykładzie), zmiennych i klas na górę ich scope, przed wykonaniem kodu.

```
console.log(helloWorld());

function helloWorld(){
        return 'Hello World!'
} // dzięki hoistingowi ten kod się wykona
```

currying

Zamiana funkcji z formy `func(a, b, c)` na formę “curried” `func(a)(b)(c)`. Pozwala to np. na tworzenie “częściowych funkcji” i zwiększa elastyczność użycia. Biblioteka lodash zawiera funkcję zmieniającą formę funkcji podanej jako argument na formę curried.

```
func = _.curry(func); // wywołanie w lodashu
```

IIFE (Immediately Invoked Function Expression)

Funkcje które uruchamiamy od razu po zdefiniowaniu umieszczając otwarte nawiasy na końcu deklaracji.

```
(function () {
        //...
})();  // <= nawiasy na końcu
```

```
(() => {
        //...
})(); // <= nawiasy na końcu
```

```
const longestWord = (function(napis) {
    return napis.split(' ').sort((a, b) => b.length - a.length)[0]
})('Ala ma kota'); // wynik: 'kota'
```

tworzenie projektu + config

1. `yarn init`
2. `yarn add eslint --dev`
3. `yarn run eslint --init`
4. stworzenie folderu src
5. .gitignore (node_modules)
6. `yarn add --dev nodemon`
7. edycja package.json np.

```
"scripts": {
  "start": "nodemon src/index.js",
  "eslint": "eslint '**/*.js'",
  "babel": "babel src -d es5out"
}
```

potencjalnie dodatkowe:

- `yarn add --dev @babel/core @babel/cli @babel/preset-env @babel/node`
- tworzymy `.babelrc`

```
{
  "presets": ["@babel/preset-env"]
}
```

- `yarn add lodash`

sortowanie

zwracana wartość	sortowanie
`> 0`	sortuje `a` po `b` => `[b, a]`
`< 0`	sortuje `a` przed `b` => `[a, b]`
`=== 0`	zostawia oryginalny porządek => `[a, b]`

```
function porównaj(a, b) {
  if (a jest mniejsze od b) {
    return -1;
  }
  if (a jest większe b) {
    return 1;
  }
  // a musi być równe b
  return 0;
}
```

```
function porównajLiczby(a, b) {
  return a - b;
}
```

dziedziczenie

```
class Person { // <= definiowanie klasy

        species = 'Human'; // <= ustawienie wartości

    constructor(name){
        this.name = name; // <= ustawienie wartości za pomocą konstruktora
    }

    sayHello(){ // definiowanie metody
        console.log('Hello');
    }
}

class Child extends Person { // <= klasa dziedziczy po innej
    constructor(name){
        super(name); // <= użycie super by uniknąć duplikacji kodu
```

```
    }

    sayHello(){ // <= nadpisanie metody
        console.log('Hi!');
    }

    saySuperHello(){
        super.sayHello(); // <= odwołanie się do nadpisanej funkcji rodzica
    }
}

const jaś = new Child('Jaś');
jaś.sayHello(); // wynik: 'Hi!'
jaś.saySuperHello(); // wynik: 'Hello'
console.log(jaś.species); // wynik: 'Human'
```

tablica kluczy obiektu

```
const keysOfElemArr = Object.keys(elem)
```

destrukturyzacja

```
const arr = ["Pies", "Kot", "Królik", "Papuga", "Małpa"];
const [pies, kot, krolik, papuga, malpa] = arr;

const book = {
  title: "Mistrz i Małgorzata",
  author: "Michaił Bułhakow",
  year: 1967
};

const { title, author, year } = book;

const {
  title: titleOfBook,
  author: authorOfBook,
  year: yearOfBook
} = book;

const { title: onlyTitle, ...bookWithoutTitle } = book;
```

rodzaje funkcji

“zwykłe” funkcje

```
function pierwiastek(n) {
  return n**(1/2);
}
```

funkcje anonimowe

```
function plusNumber(a){
    return function(b) { // <= funkcja anonimowa
        return a + b;
    };
};
const plusTen = plusNumber(10)
const suma = plusTen(5)
const oneLine = plusNumber(10)(5)
console.log(suma) //wynik: 15
console.log(oneLine) //wynik: 15
```

funkcje strzałkowe

```
const pierwiastek = (n) => n**(1/2);
// UWAGA!! Nie działają przy definiowaniu metod
```

importowanie

```
const nazwaZmiennej = require('./nazwaPliku').nazwaZmiennejWPliku;
const { nazwaZmiennej } = require('./nazwaPliku');
const nazwa = require('nazwaImportowanejBiblioteki');
```

generowanie tablicy o określonej długości funkcyjnie

```
Array.from(Array(ilośćElementów).keys()
```