JS - tablice cz. 2

Destrukturyzacja

```
1 const [number1, number2] = [20, 46, 18];
2
3
4 // ignorujemy drugi element
5 const [number1, , number3 = 0] = [20, 46, 18];
6
7
8 // pozostałe elementy utworzą nową tablicę
9 const [firstNumber, ...otherNumbers] = [20, 78, 11, 33];
```

Łączenie tablic

```
1 const numbers1 = [10, 45];
2 const numbers2 = [20, 49];
3
4
5 const allNumbers = [...numbers1, ...numbers2];
```

Klonowanie

```
1 const numbers = [45, 12, 16];
2 const numbersCopy = [...numbers];
```

Przekazywanie elementów tablicy jako argumenty funkcji

```
1 const numbers = [12, 18, 19, 44, 64, 81];
2
3 console.log(Math.min(...numbers)); // 12
```

Niezdefiniowana liczba argumentów funkcji

```
1 const myFunction = (firstParameter, ...otherParameters) => {
2   console.log(`Pierwszy argument: ${firstParameter}`);
3   console.log(`Liczba pozostałych argumentów: ${otherParameters.length}`);
4 };
5
6 myFunction(4, 8, 1, 9, 74, 51);
```

Wyciąganie fragmentu tablicy

```
1 const seasons = ["wiosna", "lato", "jesień", "zima"];
2
3 const last2Seasons = seasons.slice(2);      // ["jesień", "zima"]
4
5 const middle2Seasons = seasons.slice(1, 3); // ["lato", "jesień"]
6
7 const last3Seasons = seasons.slice(-3);     // ["lato", "jesień", "zima"]
8
9 const allButLast = seasons.slice(0, -1);    // ["wiosna", "lato", "jesień"]
```

tworzy nową tablicę

indexOf, lastIndexOf

```
1 const numbers = [20, 10, 45, 10];
2
3 console.log(numbers.indexOf(10));     // 1
4 console.log(numbers.lastIndexOf(10)); // 3
5 console.log(numbers.indexOf(5));      // -1
6 console.log(numbers.lastIndexOf(5));  // -1
```

find

```
1  const numbers = [-7, 0, 10, -6, 45];
2
3  const firstPositive = numbers.find(number => number > 0);  // 10
4  const first100 = numbers.find(number => number === 100);   // undefined
5
6  const persons = [
7      { name: "Krzysiek" },
8      { name: "Ania" },
9  ];
10
11 const chris = persons.find(({ name }) => name === "Krzysiek");
```

zwraca pierwszy element, który spełnia funkcję sprawdzającą

findIndex

```
 1 const numbers = [-7, 0, 10, -6, 45];
 2
 3 const firstPositiveIndex = numbers.findIndex(number => number > 0); //  2
 4 const first100Index = numbers.findIndex(number => number === 100);  // -1
 5
 6 const persons = [
 7     { name: "Krzysiek" },
 8     { name: "Ania" },
 9 ];
10
11 const chrisIndex = persons.findIndex(({ name }) => name === "Krzysiek"); // 0
```

includes

```
1 const guests = ["Włodek", "Irmina", "Melodia"];
2
3
4 console.log(guests.includes("Włodek")); // true
5 console.log(guests.includes("Zenek"));  // false
```

some

```
1  const tasks = [
2      { content: "odkurzyć całą piwnicę", done: false },
3      { content: "zrobić ciasto rabarbarowe", done: true },
4  ];
5
6  const isAnyTaskDone = tasks.some(({ done }) => done); // true
7
8  const numbers = [1, 3, 5, 7];
9
10 const isAnyNumberEven = numbers.some(number => number % 2 === 0); // false
```

sprawdza, czy przynajmniej jeden element spełnia funkcję sprawdzającą

every

```
1  const tasks = [
2      { content: "odkurzyć całą piwnicę", done: false },
3      { content: "zrobić ciasto rabarbarowe", done: true },
4  ];
5
6  const isEveryTaskDone = tasks.every(({ done }) => done); // false
7
8  const numbers = [1, 3, 5, 7];
9
10 const isEveryNumberEven = numbers.every(number => number % 2 === 0); // false
```

sprawdza, czy wszystkie elementy spełniają funkcję sprawdzającą

filtrowanie

```
1 const numbers = [1, 2, 3, 4];
2 const evenNumbers = numbers.filter(number => number % 2 === 0);
3
4 const tasks = [
5     { content: "odkurzyć całą piwnicę", done: false },
6     { content: "zrobić ciasto rabarbarowe", done: true },
7 ];
8
9 const doneTasks = tasks.filter(({ done }) => done);
```

zwraca nową tablicę, która zawiera elementy, spełniające funkcję sprawdzającą

Mapowanie / odwzorowanie

```
const numbers = [1, 5, 10];
const doubledNumbers = numbers.map(number => number * 2);

const persons = [
    { name: "Krzysiek", surname: "Dąbrowski" },
    { name: "Kalina", surname: "Jakubowska" },
];

const personFirstNames = persons.map(({ name }) => name);

const personHTMLTableRows = persons.map(({ name, surname }) => `
  <tr><td>${name}</td><td>${surname}</td></tr>
`);
```

map zwraca nową tablicę, której elementami są wartości zwrócone przez podaną funkcję dla każdego elementu

Sortowanie tablic

```
1 const strings = ["B", "a", 10, 2];
2
3 strings.sort();
4
5 console.log(strings); // ["10", "2", "B", "a"]
```

- **sort** sortuje tablicę i zwraca posortowaną
  - **uwaga**: modyfikuje tablicę
- domyślnie elementy konwertowane są na stringi i porównywane są kody znaków UTF-16
  - dlatego to się prawie do niczego nie nadaje

Sortowanie liczb

```
1 const numbers = [40, 8, 1, 0];
2
3 numbers.sort((a, b) => a - b); // [0, 1, 8, 40];
4 numbers.sort((a, b) => b - a); // [40, 8, 1, 0];
```

Sortowanie alfabetyczne

```
1 const surnames = ["Duda", "Dąbrowski"];
2
3 surnames.sort((a, b) => a.localeCompare(b)); // ["Dąbrowski", "Duda"]
4 surnames.sort((a, b) => b.localeCompare(a)); // ["Duda", "Dąbrowski"]
```

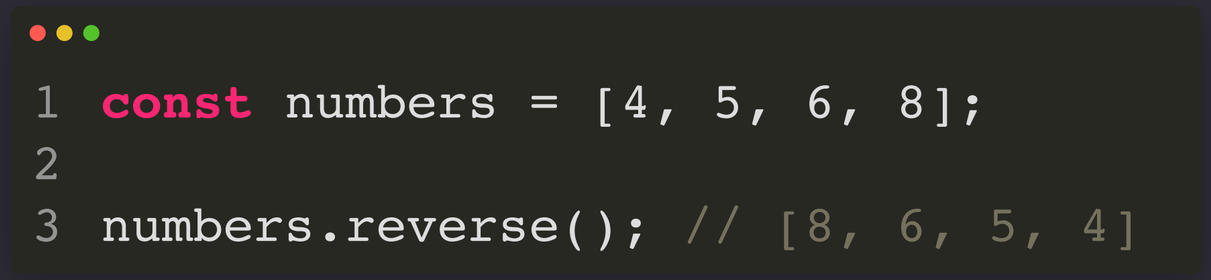
Sortowanie obiektów

```
const persons = [
    { name: "Rafał", surname: "Trzaskowski" },
    { name: "Andrzej", surname: "Duda" },
];

const getFullName = ({ name, surname }) => `${name} ${surname}`;

persons.sort((person1, person2) => (
    getFullName(person1).localeCompare(getFullName(person2))
));
```

Odwrócenie kolejności elementów



Linki

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array#Instance_methods
https://tc39.es/ecma262/#sec-properties-of-the-array-prototype-object