JS Przetwarzanie struktur danych

HTML / CSS

Prowadząca

Anna Rodziewicz
Senior Front-end Developer

Plan gry

- Tablice!
- funkcje wyższego rzędu
- przetwarzanie tablic

Tablice

```
let array = [1, 2, 3];

const array2 = [1, 2, 3];

const array3 = new Array(1,2,3);

const array4 = ['napis', 2, 3];
```

Tablica jest uporządkowanym zbiorem wartości przyporządkowanych ustalonej pojedynczej zmiennej!

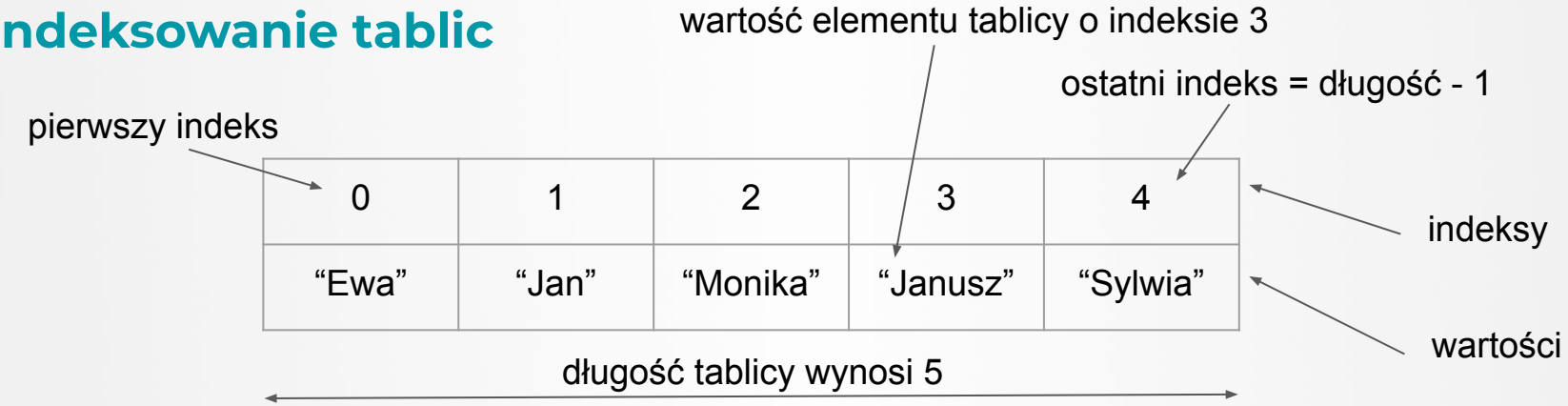
Tablice - odczyt wartości

// Standardowo odbywa się poprzez wskazanie indeksu.
// Tablice są indeksowane od zera.
Pierwszy element tablicy jest dostępny pod indeksem 0,
a ostatni pod indeksem [długość tablicy] - 1.
// Tablice przechowują swoją długość we właściwości length.

```
const array1 = [1, 2, 3, 4];
array1[1] = 100;
```

Tablice - odczyt wartości

Indeksowanie tablic




```
const names = [“Ewa”, “Jan”, “Monika”, “Janusz”, “Sylwia”];
const ewa = names[0]; // pierwszy element tablicy
const sylwia = names[names.length - 1]; // ostatni element tablicy
```

Tablice - push

Metoda dodaje element lub elementy, na **koniec** tablicy, jednocześnie modyfikując tablicę na której wywołujemy metodę push().

Do metody push możemy przekazać **jeden** lub **więcej elementów**:

```
const arr = [1, 2, 6];
arr.push(1);
arr.push(1,2,3);
```

Tablice - unshift

Metoda dodaje element lub elementy, na **początek** tablicy, jednocześnie modyfikując tablicę na której wywołujemy metodę unshift().

Do metody unshift możemy przekazać **jeden** lub **więcej elementów**:

```
const arr = [1, 2, 6];
arr.unshift(1);
arr.unshift(1,2,3);
```

// Zadanie

```
const names = ['Janek', 'Wiola',
'Mateusz', 'Kamila', 'Olaf',
'Sylwia'];
```

// dodaj przycisk w index.html, który po wciśnięciu wyświetli okno dialogowe

// w oknie dialogowym zapytaj użytkownika o imię

// dodaj imię do tablicy names

// *wyconsologuj* wszystkie imiona po kolei

Tablice - pop

Metoda usuwa i zwraca **ostatni** element.

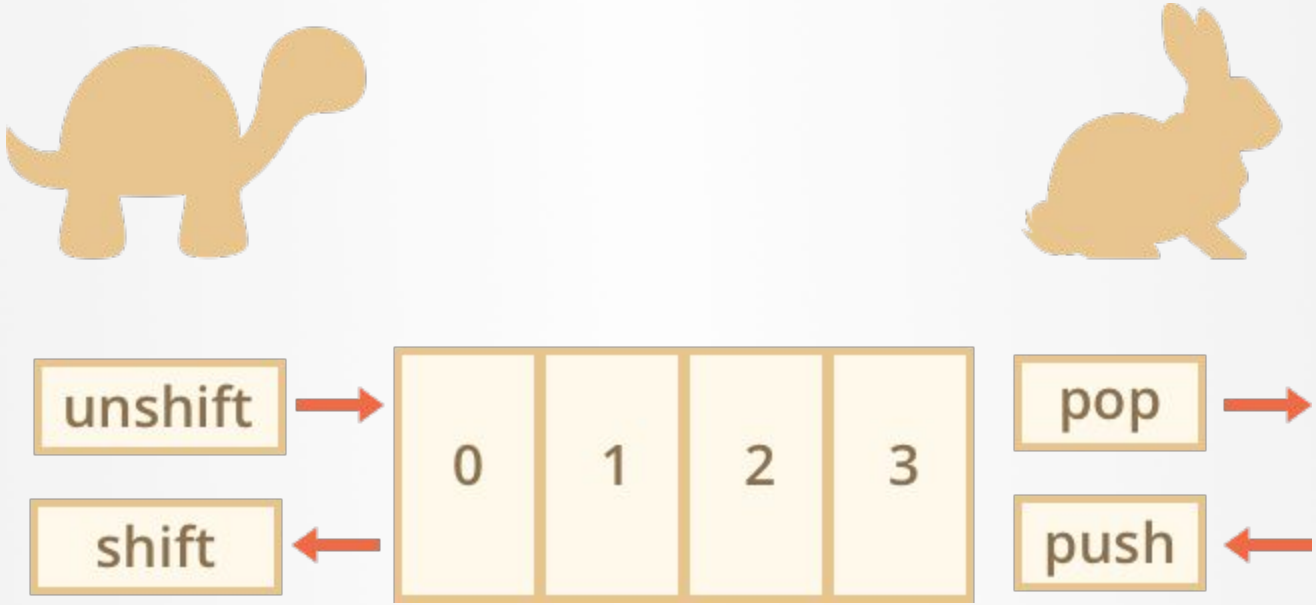
```
const arr = [1, 2, 3];

const lastElement = arr.pop();

console.log(arr); // [1, 2]

console.log(lastElement); // 3
```

Tablice - shift

Metoda usuwa i zwraca **pierwszy** element.

```
const arr = [1, 2, 3];

const firstElement = arr.shift();

console.log(arr); // [2, 3]

console.log(firstElement); // 1
```

Tablice - operacje



Tablice - splice

Metoda usuwa i zwraca element o określonym indeksie.

```
const arr = [1, 2, 3];

const element = arr.splice(1, 1);

console.log(arr); // [1, 3]

console.log(element); // 2
```

pierwszy argument - indeks,
drugi argument - ile usuwamy

Tablice - szukanie indeksu elementu

Array.indexOf(element)

```
const cars = [`BMW`,`Mercedes`,`Audi`];
const indexOfAudi = cars.indexOf('Audi');
console.log(indexOfAudi);     // 2

const indexOfVolvo = cars.indexOf('Volvo');
console.log(indexOfVolvo); // -1
```

Element nie został znaleziony

// Zadanie

// dodaj nowy element na końcu tablicy names

// usuń element z końca

// usuń element z początku

// usuń element z początku

// usuń z tablicy imię ‘Mateusz’

Tablice - łączenie elementów

`Array.prototype.join(separator)`

```
const arr = ['Ala', 'ma', 'kota'];

let mergedArray = arr.join('!'); // Ala!ma!kota

mergedArray = arr.join();  // Alamakota
```

// Zadanie

// znajdź indeks imienia ‘Ania’

// wyświetl wszystkie imiona rozdzielone przecinkiem

// * sprawdź czy imię podane przez użytkownika istnieje w tablicy, jeśli nie - dodaj je na końcu, jeśli istnieje - wyświetl jego index

Tablice - scalanie tablic

Array.prototype.concat(*array2*)

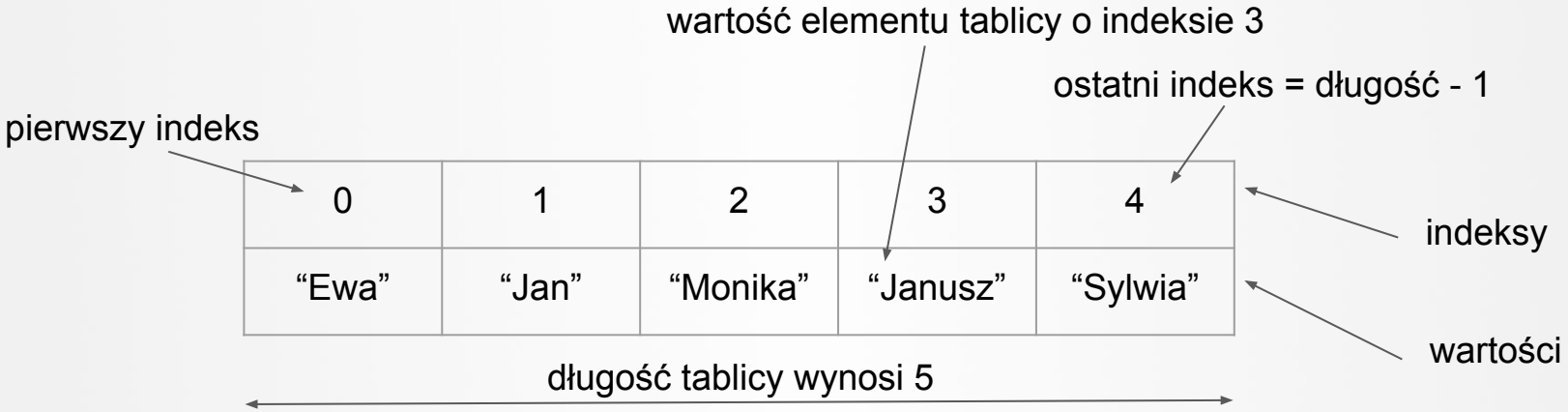
```
const cars = [`BMW`,`Mercedes`,`Audi`];
const planes = [`Boeing`,`Airbus`];

const carsAndPlanes = cars.concat(planes);

[`BMW`,`Mercedes`,`Audi`, `Boeing`,`Airbus`]
```

// Zadanie

```
const germanNames = ['Arnold',
'Carl', 'Heinrich', 'Klaus',
'Hannah'];
```

// do tablicy names dołącz germanNames

Tablice - iterowanie




```
const names = [“Ewa”, “Jan”, “Monika”, “Janusz”, “Sylwia”];
const ewa = names[0]; // pierwszy element tablicy
const sylwia = names[names.length - 1]; // ostatni element tablicy
```

Tablice - iterowanie

```
const numbers = [1, 2, 4, 9, 9];
for (let i = 0; i < numbers.length; i++) {
    if (numbers[i - 1]) {
        numbers[i] = numbers[i-1] + 2
    }
}
```

// Zadanie

```
const names = ['Janek',
'Wiola', 'Mateusz', 'Kamila',
'Olaf', 'Sylwia'];
```

// za pomocą pętli **for** przejdź przez całą tablicę i zaloguj na ekranie wartość każdego elementu

Tablice - iterowanie

Array.prototype.forEach()

```
const numbers = [1, 2, 4, 9, 9];

numbers.forEach(function(n, index) {
    console.log(n, index);
});
```

Tablice - iterowanie

// `forEach` wywołuje przekazaną funkcję dla każdego elementu tablicy

// `forEach` zwraca `undefined`

// `forEach` najlepiej używać kiedy nie chcesz modyfikować tablicy, na której operujesz

// słówko `return` w `forEach` nie przerywa wykonywania pętli

Tablice - iterowanie

Array.prototype.map()

```
const numbers = [1, 2, 4, 9, 9];

const newNumbers = numbers.map(function(n, index) {
    return n + index;
});
```

Tablice - iterowanie

// `map()` wywołuje przekazaną funkcję dla każdego elementu tablicy

// `map()` nie modyfikuje tablice, na której operuje

// `map()` zwraca nową tablice, w której elementy są przetworzone przez funkcję, którą przekazaliśmy funkcji `map()`

// Zadanie

const items = ['Bulbasaur', 'Muk', 'Charizard', 'Metapod', 'Nidoqueen', 'Vulpix', 'Kadabra', 'Dewgong'];

// Użyj funkcji map(), aby do każdego elementu w tablicy dodać informacje o ilości liter w danym słowie: ['Bulbasaur - 9', 'Muk' - 3', ...];

Funkcje wyższego rzędu

Funkcja, która przyjmuje jako parametr inną funkcję lub zwraca funkcję

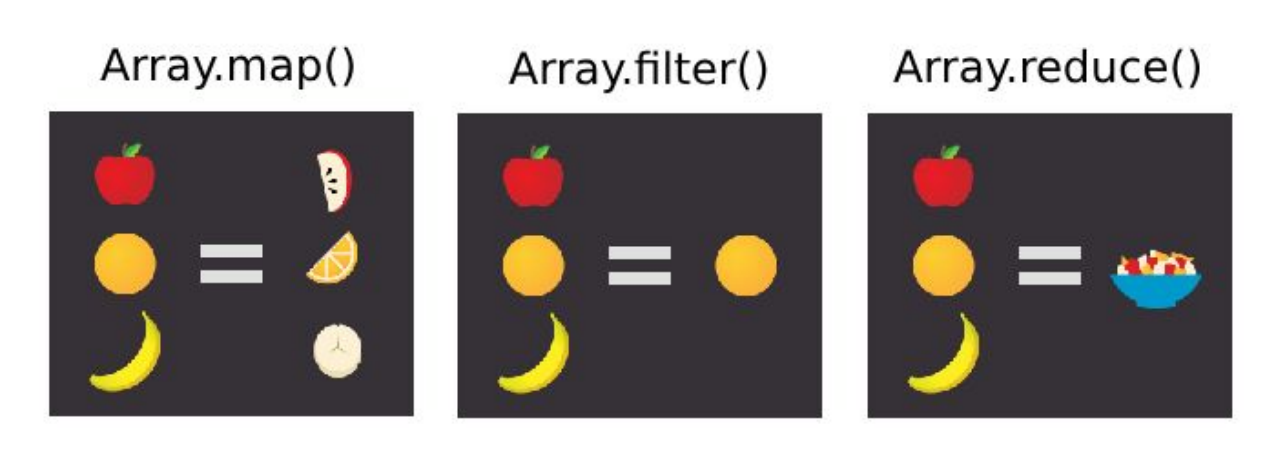
```
const sayMyName = function(sayingFunction) {
      sayingFunction();
};

const danger = function () {
      console.log("I am the danger")
};

sayMyName(danger);
```

Funkcje wyższego rzędu

Funkcja, która przyjmuje jako parametr inną funkcję lub zwraca funkcję

```
const numbers = [1, 2, 4, 9, 9];

numbers.forEach(function(n, index) {
    console.log(n, index);
});
```

Tablice - map | filter | reduce



Tablice - filtrowanie

`Array.prototype.filter()`

Metoda `filter()` tworzy nową tablicę z wszystkimi elementami, które przechodzą test określony w postaci funkcji.

Tablice - filtrowanie

```
const numbers = [1, 2, 3, 4, 5, 6, 7, 8];

const biggerThanFive = numbers.filter(function(number) {
    return number > 5;
});

console.log(biggerThanFive); // [ 6, 7, 8 ]
```

// Zadanie

const items = ['Bulbasaur', 'Muk', 'Charizard', 'Metapod', 'Nidoqueen', 'Vulpix', 'Kadabra', 'Dewgong'];

// Użyj funkcji filter(), aby stworzyć tablicę zawierającą elementy dłuższe niż 5 znaków

Tablice - redukowanie

`Array.prototype.reduce()`

Metoda **reduce()** wywołuje funkcję względem wartości przyrostowej z każdego wywołania i kolejnego elementu tablicy (od lewej do prawej) w celu sprowadzenia tej tablicy do pojedynczej wartości.

Tablice - redukowanie

```
const numbers = [1, 2, 3, 4, 5, 6, 7, 8];

const sum =
    numbers.reduce(function(result, nextNumber) {
        return result + nextNumber;
});

console.log(sum); // 36
```

Tablice - redukowanie

```
const numbers = [1, 2, 3, 4, 5, 6, 7, 8];

const sum =
    numbers.reduce(function(result, nextNumber) {
        return result + nextNumber;
}, 100);

console.log(sum); // 136
```

// Zadanie

const items = ['Bulbasaur', 'Muk', 'Charizard', 'Metapod', 'Nidoqueen', 'Vulpix', 'Kadabra', 'Dewgong'];

// Użyj funkcji reduce(), aby od liczby 100 odejmować długość każdego kolejnego słowa z tablicy items.

Kolejkowanie metod

Ponieważ funkcje takie jak map, filter, reduce działają na tablicy, to możemy wykonywać kolejne operacje od razu po sobie.

```
Array.map(...).filter(...).reduce(...)
```

Kolejkowanie metod

```
const numbers = [1, 2, 3, 4, 5];
const doubledSum = numbers.map(function(number) {
    return number * 2}).reduce(function(result, number){
    return result + number})

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

Kolejkowanie metod

```
const doubledFilteredSum = numbers
.map((number) => {
 return number * 2;
})
.filter((number) => {
  return number > 6
})
.reduce((result, number) => {
 return result + number;
})
console.log(doubledFilteredSum); // 18
```

Zadanie | eng *

Write a function, gooseFilter/goose_filter/GooseFilter, that takes an array of strings as an argument and returns a filtered array containing the same elements but with the 'geese' removed.

The geese are any strings in the following array, which is pre-populated in your solution:

```
geese = ["African", "Roman Tufted", "Toulouse", "Pilgrim", "Steinbacher"];
```

For example, if this array were passed as an argument:

```
["Mallard", "Hook Bill", "African", "Crested", "Pilgrim", "Toulouse", "Blue Swedish"]
```

Your function would return the following array:

```
["Mallard", "Hook Bill", "Crested", "Blue Swedish"]
```

The elements in the returned array should be in the same order as in the initial array passed to your function, albeit with the 'geese' removed. Note that all of the strings will be in the same case as those provided, and some elements may be repeated.

zadanie z codewars.com

Zadanie | pl *

Napisz funkcję, gooseFilter/goose_filter/GooseFilter, która przyjmuje tablicę stringów (gatunki ptaków) jako argument i zwraca przefiltrowaną tablicę bez gatunków gęsi
Gatunki gęsi są przechowywane w postaci tablicy stringów w Twojej funkcji:

```
geese = ["African", "Roman Tufted", "Toulouse", "Pilgrim", "Steinbacher"];
```

Dla przykładu, przekazujemy metodę:

```
["Mallard", "Hook Bill", "African", "Crested", "Pilgrim", "Toulouse", "Blue Swedish"]
```

Twoja funkcja powinna zwrócić:

```
["Mallard", "Hook Bill", "Crested", "Blue Swedish"]
```

Elementy w zwróconej funkcji powinny być w tej samej kolejności, co w oryginalnej tablicy, zwróć uwagę, że niektóre elementy w tablicy mogą się powtórzyć.

zadanie z codewars.com

Kontakt

Dziękuję!

Ania Rodziewicz
aerodziewicz@gmail.com