

| Przetwarzanie Struktur Danych

Operatory porównania

20 == '20' → true

20 === '20' → false

20 != '20' → false

20 !== '20' → true

Wartości fałszywe

- False
- Undefined
- Null
- NaN
- 0
- ""

Typeof false ➡ 'boolean'

Co to znaczy

Przetwarzanie Stuktur Danych?

- Struktura danych (ang. data structure) - sposób uporządkowania informacji w komputerze. Na strukturach danych operują algorytmy.
- Podczas implementacji programu programista często staje przed wyborem między różnymi strukturami danych, aby uzyskać pożądaný efekt. Odpowiedni wybór może zmniejszyć złożoność obliczeniową, ale z drugiej strony trudność implementacji danej struktury może stanowić istotną przeszkodę.
- Próbą połączenia idei struktur danych i algorytmów jest pomysł programowania obiektowego.

TABLICE



Co to jest Tablica?

- Kontener uporządkowanych danych takiego samego typu, w którym poszczególne elementy dostępne są za pomocą kluczy (indeksu). Indeks najczęściej przyjmuje wartości numeryczne.
- Istnieje również tablica asocjacyjna, która przechowuje pary (unikatowy klucz, wartość) i umożliwia dostęp do wartości poprzez podanie tego klucza. Klucze nie muszą być wartościami numerycznymi

Arrays - tablice

```
var myTable = [ ];
```

```
var array = new Array('jeden', 'dwa', 3);
```

```
var liczby = [ 10 , 11, 12 ];
```

```
var cars = [ 'Saab', 'Volvo', 'BMW' ];
```

Arrays - indexy

Do elementów tablicy dostajemy się po indeksie

```
var cars = [ 'Saab', 'Volvo', 'BMW' ];
```

```
cars [ 0 ] === 'Saab,
```

```
cars [ 1 ] === 'Volvo,
```

```
cars [ 2 ] === 'BMW'
```


Arrays - zapis

```
var cars = [ 'Saab', 'Volvo', 'BMW' ];
```

```
cars [ 2 ] = 'Niemiecki wóz,
```

```
cars [ 3 ] = 'Trabant,
```

```
cars [ 4 ] = 'Czarna Wołga'
```

Array.length – długość tablicy

```
var a = [ ];
```

```
var b = [ "Freeze" ];
```

```
var c = [ "Batman", "Robin", "Freeze", "Riddler" ]
```

```
a.length == 0
```

```
b.length == 1
```

```
c.length == 4
```

array.forEach()

```
var heroes = ['Batman', 'Robin', 'Gordon'];
```

```
heroes.forEach(function(hero, index) {  
    console.log(hero, index);  
})
```

- wywołuje funkcję dla każdego elementu tablicy
- za każdym wywołaniem przekazuje do funkcji dwa parametry:
 - pierwszy - aktualny element tablicy
 - index - index aktualnego elementu tablicy

Array - pop, push, shift, unshift

unshift('Catwoman')

push('Catwoman')

["Batman", "Robin", "Freeze", "Riddler"];

shift()

pop()

Array – **shift()**

`arr.shift()` - Usuwa pierwszy element z tablicy zwracając go. Metoda ta zmienia długość tablicy.

```
var heroes = ['Batman', 'Robin', 'Gordon'];
```

```
var firstHero = heroes.shift();
```

```
firstHero    ==    'Batman'  
heroes       ==    ['Robin', 'Gordon'];
```

Array – pop()

`arr.pop()` - Usuwa ostatni element z tablicy zwracając go. Metoda ta zmienia długość tablicy.

```
var heroes = ['Batman', 'Robin', 'Gordon'];
```

```
var lastHero = heroes.pop();
```

```
lastHero    ==    'Gordon'  
heroes      ==    ['Batman', 'Robin'];
```

Array – **unshift()**

`arr.unshift()` - Dodaje jeden lub więcej elementów na początek tablicy i zwraca jej nową długość.

```
var heroes = ['Batman', 'Robin', 'Gordon'];  
var newHeroes = heroes.unshift('Rob', 'Gor');
```

```
heroes      == ['Rob', 'Gor', 'Batman', 'Robin', 'Gordon'];  
newHeroes   == 5
```

Array – push()

`arr.push ()` - Dodaje jeden lub więcej elementów na koniec tablicy i zwraca jej nową długość. Metoda ta zmienia długość tablicy.

```
var heroes = ['Batman', 'Robin', 'Gordon'];  
var newHeroes = heroes.push('Rob', 'Gor');
```

```
heroes      == ['Robin', 'Gordon', 'Batman', 'Rob', 'Gor'];  
newHeroes   == 5
```


Array.concat - sklejanie tablic

```
var a = [ "Batman", "Robin" ];  
var b = [ "Freeze", "Riddler" ];  
var c = a.concat(b)
```

```
a == [ "Batman", "Robin" ];  
b == [ "Freeze", "Riddler" ];  
c == [ "Batman", "Robin", "Freeze", "Riddler" ];
```

Tablice wielowymiarowe

```
var table = [ [11, 12], [21, 22], [31, 32] ];
```

table[1]  [21, 22]

table[1][0]  21

array.filter()

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

```
numbers.filter(function(num) {  
    return num > 9  
});
```

➡ [10]

```
numbers.filter(function(num) {  
    return num > 5 && num < 9  
});
```

➡ [6,7,8]

```
numbers.filter(function(num) {  
    return num < 3 || num > 8  
});
```

➡ [1,2,9,10]

Array - Kolejowanie funkcji - method chaining

```
['Batman', 'Robin', 'Gordon']  
  .map(function (hero) {  
    return 'Hero - ' + hero;  
  })  
  .forEach(function(hero) {  
    console.log(hero);  
  })
```

array.find()

Metoda **find()** zwraca pierwszy element tablicy, który spełnia warunek podanej funkcji testującej.

```
var numbers = [12, 5, 8, 130, 44];
```

```
function isBigEnough(element) {  
    return element >= 15;  
}
```

```
numbers.find(isBigEnough);
```



130

array.reduce()

Metoda **reduce()** wywołuje funkcję względem wartości przyrostowej z każdego wywołania i kolejnego elementu tablicy

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

```
numbers.reduce(function(result, number) {  
    return result += number; },  
0);
```

 55

```
numbers.reduce(function(result, number) {  
    return result -= number; },  
100);
```

 45

array.map()

```
var heroes = ['Batman', 'Robin', 'Gordon'];
```

```
var modifiedHeroes = heroes.map(function(hero) {  
    return 'Hero - ' + hero;  
})
```

- tworzy nową tablicę
- nowa tablica zawiera nowe elementy, stworzone w oparciu o elementy ze starej tablicy
- stara tablica pozostaje niezmienną