Programowanie w JavaScript

WSEI

#szkoła programowania

Funkcje czasu - setTimeout

Dzięki setTimeout możemy wykonać kawałek kodu po określonym czasie podanym w milisekundach.

```
setTimeout(() => {
    console.log('Wywołanie po 2 sekundach');
}, 2000)
```

Funkcje czasu - clearTimeout

clearTimeout służy do wyczyszczenia z pamięci wcześniej ustawionego timeoutu. Aby to zrobić należy funkcji setTimeout przypisać ID a następnie podać je do funkcji clearTimeout

```
let myTimer = setTimeout(() => {
    console.log('Wywołanie po 2 sekundach');
}, 2000)

clearTimeout(myTimer);
```

Funkcje czasu - setInterval

Uruchamia kawałek kodu w określonym w milisekundach przedziale czasowym.

```
setInterval(() => {
    console.log('Wywołanie co 2 sekundy');
}, 2000);
```

Funkcje czasu - clearInterval

clearInterval() czyści z pamięci wcześniej ustawiony interwał. Aby to zrobić należy funkcji setInterval przypisać ID a następnie podać je do funkcji clearInterval

```
let myInterval = setInterval(() => {
    console.log('Wywołanie co 2 sukundy');
}, 2000)

clearInterval(myInterval);
```

Funkcje są podstawowym elementem JS. Pozwalają na zamknięcie oraz reużywanie kawałka kodu w dowolnym miejscu kodu.

```
function myFunc() {
    console.log('Moja nowa funkcja');
}

myFunc = () => {
    console.log('Moja nowa funkcja');
}
```

Parametry to zmienne przekazywane do funkcji z miejsca wywołania

```
function myFunc(param1, otherParam, hmmmm) {
    console.log('Moje parametry: ', param1, otherParam, hmmmm);
}

myFunc = (param1, otherParam, hmmmm) => {
    console.log('Moje parametry: ', param1, otherParam, hmmmm);
}

oneParam = param1 => {
    console.log('Mój parametr: ', param1);
}

myFunc('aaaa', { a: 1 }, [1, 2]);
oneParam(true);
```

Funkcja oprócz operacji zawartych wewnątrz niej może też zwracać wynik na zewnątrz. Aby zwrócić coś z funkcji używamy słówka **return**

```
function myFunc(param1, otherParam, hmmmm) {
    console.log('Moje parametry: ', param1, otherParam, hmmmm);
    return param1 + 'bbb';
}

myFunc = (param1, otherParam, hmmmm) => {
    console.log('Moje parametry: ', param1, otherParam, hmmmm);
    return param1 + 'bbb';
}

oneParam = param1 => param1 + 'bbb';

myFunc('aaaa', { a: 1 }, [1, 2]);
oneParam('aaaa');
```

function() vs () =>

- Arrow function oraz function przypisane do zmiennych traktowane są przez JS jako funkcje
- Arrow function zmniejsza ilość linijek kodu
- Arrow function nie można użyć jako konstruktora
- Przy pomocy Arrow function nie tworzymy prototypów
- Arrow function nie zamyka nam kontektu tak jak function()

Tablice

- Wartości w tablicy umieszczamy w nawiasach kwadratowych
- Wartości oddzielamy przecinkiem
- Tablica przechowuje różne typy danych włącznie z tablicami
- Długość tablicy liczymy od 1 i pobieramy ją przy pomocy właściwości **length**
- **Indeks tablicy numerowany jest od 0 a nie od 1**

```
let myArray = [1, 2, 3, 4, 'string', { a: 1 }, [1, 2]];

let empty = [];

console.log(myArray[0]) // 1
console.log(myArray[1]) // 2
console.log(myArray[4]) // string
console.log(myArray[5]) // { a: 1 }
console.log(myArray[6]) // [1, 2]
console.log(myArray.length) // 7
```

Aby móc zrobić taką samą operację na każdym elemencie tablicy należy użyć pętli.

```
let myArray = [1, 2, 3, 4, 'string', { a: 1 }, [1, 2]];


for (let i=0;i<myArray.length; i++){
    console.log(myArray[i] * 2)
}
```

Tablice - dwuwymiarowe

W JavaScript nie ma tablic wielowymiarowych. Są za to tablice zagnieżdżone, często mylone z tablicami wielowymiarowymi. Aby móc dostać się do konkretnego elementu podajemy kolejne nawiasy kwadratowe.

```
let mutiArray = [[[[1, 2, 3], [1, 2, 3]], [[1, 2, 3], [1, 2, 3]]]]

console.log(multiArray[0]) // [[[1, 2, 3], [1, 2, 3]], [[1, 2, 3], [1, 2, 3]]]
console.log(multiArray[0][0]) // [[1, 2, 3], [1, 2, 3]]
console.log(multiArray[0][0][0]) // [1, 2, 3]
console.log(multiArray[0][0][0][0]) // 1
```

Tablice – dostępne metody (Mutacyjne)

- array.pop() – usuwa i zwraca ostatni element tablicy
- array.push() – dodaje element do końca tablicu
- array.reverse() – odwraca tablicę
- array.shift() – usuwa i zwraca pierwszy element tablicy
- array.sort() – sortuje tablice na podstawie przekazanej funkcji – domyślnie od njamniejszejgo do największego elementu
- array.splice() – usuwa i zwraca kawałek tablicy (indexPoczątkowy, liczbaElementów, elementDoWstawienia)
- array.unshift() – dodaje element na początek tablicy i zwraca nową długość

```
let foo = [2, 5, 1, 6, 25, 3];

foo.sort() // [1,2,25,3,5,6]
foo.sort(function (a, b) {
    return a - b
}) // [1,2,3,5,6,25]
foo.sort(function (a, b) {
    return b - a
}) // [25,6,5,3,2,1]
```

```
let foo = [1, 2, 3, 4];

console.log(foo); // [1,2,3,4]
console.log(foo.pop()); // 4
foo.push(5);
console.log(foo); // [1,2,3,4,5]
foo.reverse();
console.log(foo); // [5,4,3,2,1]
foo.shift();
console.log(foo); // [4,3,2,1]
console.log(foo.shift()); // 4
foo.unshift(5); // [5,4,3,2,1]
console.log(foo.unshift(5)); // 5
foo.splice(2, 1, 'ala', 'ala2')
// [5,4,'ala','ala2',2,1]
```

Tablice – dostępne metody (Dostępowe)

- array.concat() – łączy dwie tablice w jedną
- array.join() – łączy elementy tablicy w ciąg znaków
- array.slice() – zwraca kawałek tablicy
- array.indexOf() – pierwsza pozycja szukanego elementu
- array.lastIndexOf() – ostatnia pozycja szukanego elementu

```
let foo = [1, 2, 3];
let bar = [4, 5, 5];
let textArray = ['ala', 'ma', 'kota'];
console.log(foo.concat(bar)); // [1,2,3,4,5, 5]
console.log(textArray.join()); // ala,ma,kota
console.log(textArray.join('-')); // ala-ma-kota
console.log(foo.slice(0,2)); // [1,2]
console.log(foo.indexOf(2)); // 1
console.log(bar.lastIndexOf(5)); // 2
```

Tablice – dostępne metody (Iteracyjne)

- array.forEach() – taki for na tablicy – wywołuje kod na każdym elemencie tablicy
- array.filter() – zwraca nową tablicę zawierające elementy które spełniają warunek
- array.map() – zwraca nową tablicę ze zmodyfikowanymi elementami

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

foo.forEach(function(element, index, array) {
    console.log(element, index, array);
});

let bar = foo.filter(function(element, index, array) {
    return element % 2 === 0;
});
console.log(bar); // [2]

let bar = foo.filter(function(element, index, array) {
    return element * 2
});
console.log(bar); // [2, 4, 6]
```

Obiekty tworzymy przy pomocy **{ }**. Dane wewnątrz obiektu tworzą parę **key: value.** Obiekt może przechowywać wszystkie typy danych włącznie funkcjami, które nazywamy metodami.

```
let car = {
    name: 'BMW',
    age: 12,
    mileage: 125000,
    addMileage: function() {
        this.mileage += 500;
    }
}
```

Aby dostać się do żądanego pola w obiekcie należy użyć kropki. Aby wywołać metodę obiektu należy dodać ()

```
let car = {
    name: 'BMW',
    age: 12,
    mileage: 125000,
    addMileage: function(mileage) {
        this.mileage += mileage;
    }
}

console.log(car.name); // BMW
car.addMileage(mileage);
```

```
let car = {
    name: 'BMW',
    age: 12,
    mileage: 125000,
    addMileage: function(mileage) {
        this.mileage += mileage;
    }
}

console.log(car);
car.model = 'E46';
console.log(car);
```

Jeśli chcielibyśmy dostać się kluczy obiektu powinniśmy użyć pętli for...in

```
let car = {
    name: 'BMW',
    age: 12,
    mileage: 125000,
    addMileage: function (mileage) {
        this.mileage += mileage;
    }
}

for (let key in car) {
    console.log(key); // keys
    console.log(car[key]); //values
}
```

```
let foo = 50;
let bar = foo;
```

Jest to kopiowanie zmiennych. Będziemy mieć dwie niezależne zmienne które nie będą miały ze sobą nic wspólnego. Z obiektami jest nieco inaczej.

```
let car = {
    name: 'BMW',
    age: 12,
    mileage: 125000,
    addMileage: function (mileage) {
        this.mileage += mileage;
    }
}
```

Obiekt przypisany do zmiennej nie jest przechowywany w tej zmiennej. Przechowywany jest **adres (referencja)** do niego. Całość obiektu przechowywane jest w pamięci.

Kopiując obiekt kopiujemy jego referencję a nie dane. Mamy zatem dwie zmienne które przechowują tą samą referencję do tego samego miejsca w pamięci. Dlatego zmieniając coś w jednej zmiennej będzie to widoczne również w drugiej zmiennej.

```
let car = {
    name: 'BMW',
    age: 12,
    mileage: 125000,
    addMileage: function (mileage) {
        this.mileage += mileage;
    }
}

let mercedes = car;
```

Obiekty – porównywanie

Porównywanie obiektów działa na tej samej zasadzie.

```
let foo = {};
let bar = foo;

foo == bar; //true
foo === bar; //true

let foo = {};
let bar = {};

foo == bar; //false
foo === bar; //false
```