Zadanie 1

const currentUser = {

name: "John",

surname: "Doe",

email: "john.doe@example.com",

www: "www.example.com",

 userType: "standard",

isActive: true,

show: function() {

console.log("Name:", this.name);

console.log("Surname:", this.surname);

console.log("Email:", this.email);

console.log("Website:", this.www);

console.log("User Type:", this.userType);

console.log("Active:", this.isActive);

},

setActive: function(active) {

this.isActive = active;

}

};

currentUser.show();

currentUser.setActive(false);

console.log("After setting isActive to false:");

currentUser.show();

Zadanie 2

class Book {

constructor() {

this.users = [];

}

addUser(name, age, phone) {

const newUser = { name, age, phone };

this.users.push(newUser);

}

showUsers() {

console.log("Wszyscy użytkownicy w książce:");

this.users.forEach(user => {

console.log(`Name: ${user.name}, Age: ${user.age}, Phone: ${user.phone}`);

});

}

findByName(name) {

const foundUser = this.users.find(user => user.name === name);

if (foundUser) {

console.log(foundUser);

} else {

console.log(false);

}

}

findByPhone(phone) {

const foundUser = this.users.find(user => user.phone === phone);

if (foundUser) {

console.log(foundUser);

} else {

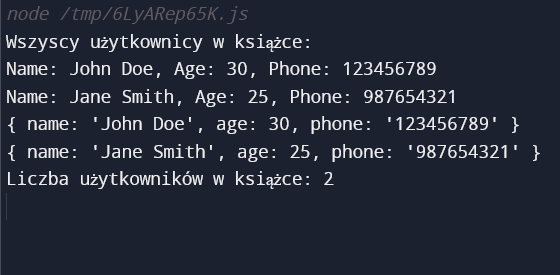
console.log(false);

}

}

getCount() {

console.log("Liczba użytkowników w książce:", this.users.length);

 }

}

const myBook = new Book();

myBook.addUser("John Doe", 30, "123456789");

myBook.addUser("Jane Smith", 25, "987654321");

myBook.showUsers();

myBook.findByName("John Doe");

myBook.findByPhone("987654321");

myBook.getCount();

Zadanie 3

const text = {

check: function(txt, word) {

return txt.includes(word);

},

getCount: function(txt) {

return txt.length;

},

 getWordsCount: function(txt) {

return txt.split(/\s+/).length;

},

setCapitalize: function(txt) {

return txt.replace(/\b\w/g, char => char.toUpperCase());

},

setMix: function(txt) {

let mixedText = '';

for (let i = 0; i < txt.length; i++) {

if (i % 2 === 0) {

mixedText += txt[i].toLowerCase();

} else {

mixedText += txt[i].toUpperCase();

}

}

return mixedText;

},

generateRandom: function(lng) {

const alphabet = 'abcdefghijklmnopqrstuvwxyz';

let randomText = '';

for (let i = 0; i < lng; i++) {

const randomIndex = Math.floor(Math.random() \* alphabet.length);

randomText += alphabet[randomIndex];

}

return randomText;

}

};

console.log(text.check("ala ma kota", "kota"));

console.log(text.getCount("ala ma kota"));

console.log(text.getWordsCount("Ala ma kota"));

console.log(text.setCapitalize("ala ma kota"));

console.log(text.setMix("ala ma kota"));

console.log(text.generateRandom(10));

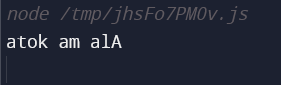
Zadanie 4

1. Dziedziczenie prototypowe w JavaScript odnosi się do sposobu, w jaki obiekty dziedziczą właściwości i metody od innych obiektów poprzez ich prototypy. Każdy obiekt w JavaScript ma prototyp, który jest obiektem również, i dziedziczy właściwości i metody z tego prototypu. W przypadku "klas" w JavaScript, mówimy, że są one tzw. "syntax sugar", ponieważ w rzeczywistości JavaScript nie ma klas w tradycyjnym sensie.

Mechanizm dziedziczenia prototypowego w JavaScript polega na tym, że gdy dostęp do danej właściwości lub metody jest próbowany na danym obiekcie, interpreter najpierw sprawdza, czy ten obiekt posiada tę właściwość lub metodę. Jeśli nie, to szuka jej w prototypie tego obiektu. Jeśli nie znajdzie tam, szuka dalej w prototypie prototypu, i tak dalej, aż dotrze do obiektu Object, który jest korzeniem łańcucha prototypów.

b)

Definiujemy nową metodę mirror() dla obiektów typu String poprzez rozszerzenie prototypu String.prototype. Ta metoda rozbija łańcuch na tablicę pojedynczych znaków za pomocą metody split(''), odwraca kolejność elementów tablicy za pomocą metody reverse() i łączy je z powrotem w łańcuch za pomocą metody join('').

String.prototype.mirror = function() {

return this.split('').reverse().join('');

};

console.log("Ala ma kota".mirror());