Małe przypomnienie

Istotne elementy JS

Funkcja strzałkowa

```
const fn = (item) => {
    console.log("Podany argument to " + item)
}

fn("Hej!")


----
const fn = function() {
    console.log("Podany argument to " + item)
}
```

Funkcja strzałkowa

```
const fn = item => console.log("Podany argument to " + item)

fn("Hej!") // "Podany argument to Hej"
```

---------- to samo

```
const fn = (item) => {
      return console.log("Podany argument to " + item)
}
```

Funkcja strzałkowa

```
const fn = (item) => {
      return console.log("Podane argument to " + item)
}
```

---------- to samo ----------

```
const fn = (item) => console.log("Podany argument to " + item)
```

---------- to samo ----------

```
const fn = (item) => (
      console.log("Podany argument to " + item)
)
```

Funkcja strzałkowa

```
const fn = (item, item2) => {
    return `Podany argument to: ${item} i ${item2}`
}
const result = fn("Hej!", "Ho!")
// result zawiera   "Podany argument to: Hej! i Ho!"

---------- to samo poniżej
const fn = (item, item2) => (
    `Podany argument to: ${item} i ${item2}`
)
const result = fn("Hej!", "Ho!")
```

Funkcja strzałkowa

```
const btn = document.querySelector('button');
btn.addEventListener('click', () => console.log(this));


//this będzie wskazywało na window (obiekt globalny). Musisz pamiętać, że funkcja strzałkowa nie tworzy własnego przypisania wiązania this, a zamiast tego je dziedziczy (czyli przejmuje this, który istnieje we wcześniejszym kontekście)


const btn = document.querySelector('button')
btn.addEventListener('click', function() {
console.log(this)
})


//this będzie wskazywał na obiekt, na którym wywołane zostało zdarzenie
```

Metody tablic i iteratory tablic

.join()
.concat()

.map()
.forEach()
.filter()
.find() i findIndex()

Metoda join()

```
const users = ["adam", "bogdan", "czarek", "darek"];

// Metoda join - zwraca stringa z tablicy
const usersString = users.join(" ");
console.log(usersString); //"adam bogdan czarek darek"
```

Metoda concat()

```
const users = ["adam", "bogdan", "czarek", "darek"];

// Metoda concat - łączymy tablicę z innym elementem (czy
inną tablicą) i zwracamy nową tablicę.


const newUser = "edyta"
const allUsers = users.concat(newUser)
console.log(allUsers);
//["adam", "bogdan", "czarek", "darek", "edyta"]
```

Operator spread - alternatywa dla metody concat()

```
const users = ["adam", "bogdan", "czarek", "darek"];

const allUsers = [...users, "edyta"]
console.log(allUsers);
//["adam", "bogdan", "czarek", "darek", "edyta"]
```

Metody iterujące po tablicach

map()
forEach()
filter()
find()
findIndex()

map() - przykład 1

```
// Metoda map zwraca nową tablicę o tej samej długości

const users = ["adam", "bogdan", "czarek", "darek"];

const usersFirstLetterUpperCase = users.map(user =>
user[0].toUpperCase())

console.log(usersFirstLetterUpperCase);
// ["A", "B", "C", "D"]
```

map() - przykład 2

```
// Metoda map zwraca nową tablicę o tej samej długości

const numbers = [2, 3, 4]
const doubleNumber = numbers.map(number => number * 2)
console.log(doubleNumber);
```

forEach() - przykład 1

// forEach - pracuje na tablicy, nie zwraca nowej (zwraca undefined)

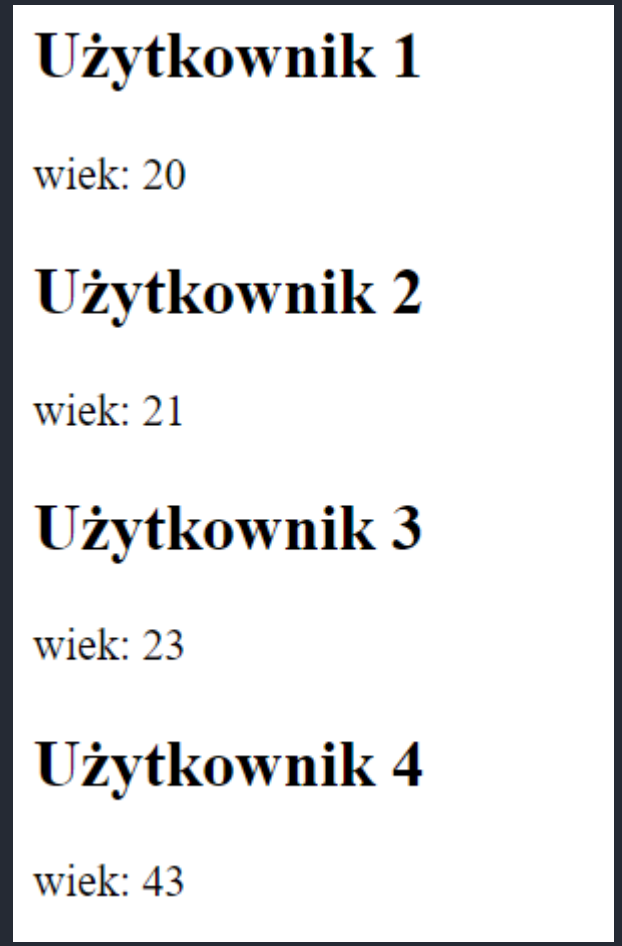
```
const usersAge = [20, 21, 23, 43];
usersAge.forEach(age => console.log(`W przyszłym roku użytkownik będzie miał ${age + 1} lat`))
```

forEach() - przykład 2

```
const usersAge = [20, 21, 23, 43];
let usersTotalAge = 0;

usersAge.forEach(age => usersTotalAge += age);
console.log(usersTotalAge);
//zmienna zawiera wartość 107
```

forEach() - przykład 3

```
const usersAge = [20, 21, 23, 43];

const section = document.createElement('section')

usersAge.forEach((age, index, array) => {
    section.innerHTML += (
        `<h1> Użytkownik ${index + 1}</h1>
        <p>wiek: ${age}</p>`
    )
    if (index === array.length - 1) {
        document.body.appendChild(section);
    }
})
```



filter() - przykład 1

//Zwraca nową tablicę złożoną z tych elementów, przy których iterator zwrócił true

```
const users = ["adam", "bogdan", "czarek", "darek"];

const NameWith6Letter = users.filter(user => user.length === 6)
console.log(NameWith6Letter);
//["bogdan", "czarek"]
```

filter() - przykład 2 (z wykorzystaniem indexOf)

//Zwraca nową tablicę złożoną z tych elementów, przy których iterator zwrócił true

```
const users = ["adam", "bogdan", "czarek", "darek"];

const NameWithLetterK = users.filter((user) => {
    return (
        user.indexOf("k") > -1
    )
})
console.log(NameWithLetterK);
//["czarek", "darek"]
```

filter() - przykład 2 (z wykorzystaniem indexOf)

//Zwraca nową tablicę złożoną z tych elementów, przy których iterator zwrócił true

```
const users = ["adam", "bogdan", "czarek", "darek"];

const NameWithLetterK = users.filter(user => user.indexOf("k") > -1)

console.log(NameWithLetterK);
//["czarek", "darek"]
```

findIndex()

// Metoda findIndex zwraca indeks elementu, który jako pierwszy zwróci true (spełnia warunek). Jeśli w żadnej iteracji nie będzie spełniony warunek, to zwróci -1

```
const customers = [
      { name: "Adam", age: 67 },
      { name: "Basia", age: 27 },
      { name: "Marta", age: 17 },
];

const isUsersAdult = customers.findIndex(customer => customer.age < 18)
console.log(isUsersAdult); //2
```

find()

// Metoda find zwraca element, który jako pierwszy zwróci true (spełnia warunek). Jeśli w żadnej iteracji nie będzie spełniony warunek, to zwróci undefined.

```
const customers = [
      { name: "Adam", age: 67 },
      { name: "Basia", age: 27 },
      { name: "Marta", age: 17 },
];

const firstAdultUser = customers.find(customer => customer.age >= 18)
console.log(firstAdultUser); //{name: "Adam", age: 67}
```

Klasa i instancja

```
//deklaracja klasy
class City {
}


//tworzenie instancji klasy
const Warsaw = new City();
const NewYork = new City();


// Powstają dwa (różne, niepołączone) obiekty będące instancją City.
// City{}
```

Klasa - właściwości instancji i prototyp

```
class Country {
      // constructor() {}
}

const poland = new Country();
```

Klasa - właściwości instancji i prototyp

```
class Country {
   constructor(name, capital, population) {
      this.name = name;
      this.capital = capital;
      this.popultion = population;
   }
}

const poland = new Country('Polska', 'Warszawa', 38000000);

//Country {name: "Polska", capital: "Warszawa", popultion: 38000000}
```

Klasa - właściwości instancji i prototyp

```
class Country {
      ...
}

const poland = new Country('Polska', 'Warszawa', 38000000);

// poland (object)
1.capital: "Warsawa"
2.citizensNumber: 38000000
3.name: "Polska"
4.__proto__: Object
```

Klasa - dodawanie metod do prototypów i instancji

```
class Country {
        constructor(name) {
                this.name = name; //właściwość każdej instancji
                this.showName = () => console.log(this.name); //metoda każdej instancji
        }
        //Wszystkie metody tworzone w klasie znajdują się w prorotypie klasy, do której dostęp mają wszystkie instancje.
        showCountryName() {
                console.log(`Nazwa kraju to ${this.name}`);
        }
}

const Poland = new Country('Polska');
const Italy = new Country('Italia');

Poland.showCountryName(); // Nazwa kraju to Polska
Italy.showCountryName(); // Nazwa kraju to Italia
Poland.showName(); // Polska
Italy.showName(); // Italia
```

Klasa - dodawanie metod do prototypów i instancji

```
class Country {
        constructor(name) {
                this.name = name;
                this.showName = () => console.log(this.name);
        }
        showCountryName() {
                console.log(`Nazwa kraju to ${this.name}`);
        }
}

const Poland = new Country('Polska');

Poland.showCountryName(); // Nazwa kraju to Polska
Poland.showName(); // Polska
```

```
Country {name: "Polska", showName: ƒ}
name: "Polska"
showName: () => console.log(this.name)
__proto__:
        constructor: class Country
        showCountryName: ƒ showCountryName()

        __proto__:
                constructor: ƒ Object()
                hasOwnProperty: ƒ hasOwnProperty()
                isPrototypeOf: ƒ isPrototypeOf()
                propertyIsEnumerable: ƒ propertyIsEnumerable()
                toLocaleString: ƒ toLocaleString()
                toString: ƒ toString()
                ...
```

Klasa - metody prototypu czy instancji?

```
class Country {
   constructor(name) {
      this.name = name;
      this.showCountryName = function() {
         console.log("Metoda w instancji wskazuje: " + this.name);
      }
   }
   showCountryName() {
      console.log(`Metoda w prototypie wskazuje ${this.name}`);
   }
}

const Poland = new Country('Polska');

Poland.showCountryName(); // Metoda w instancji wskazuje Polska
```

Klasa - dziedziczenie

```
class Person {
   constructor(name) {
       this.name = name;
   }
   showName() {
       console.log(`Imię osoby to ${this.name}`);
   }
}

class Student extends Person {
   constructor(name = "", degrees = []) {
      super(name)
      this.degrees = degrees
   }

   showDegrees() {
      const completed = this.degrees.filter(degree => degree > 2)
      console.log(`Studen ${this.name} ma stopnie: ${this.degrees} i zaliczył już ${(completed.length} przedmiotów`);
   }
}

const Janek = new Student("Adam", [2, 3, 4, 5, 2, 3])
Janek.showDegrees() // Student Adam ma stopnie: 2,3,4,5,2,3 i zaliczył już 4 przedmiotów
```

Mechanizm this

Mechanizm this polega na wiązaniu słowa kluczowego this z obiektem. Wiązanie to jest tworzone w chwili wywołania funkcji.

Wiązanie domyślnie następuje automatycznie, ale możemy je zmienić.

```
"use strict"
const fn = function() {
    console.log(this);
}
fn()
//undefined
```

```
const fn = function() {
    console.log(this);
}
fn()
//Window
```

Mechanizm this

```
const car = {
        brand: 'opel',
        age: 2018,
        showAge() {
                console.log(`samochód z ${this.age}`);
        },
        showBrand: () => {
                console.log(`samochód marki ${this.brand}`);
        }
}

car.showAge() //samochód z 2018
car.showBrand() //samochód marki undefined (dziedziczy po
zakresie wyższym a w zakresie wyższym jest window. Więc
window.brand zwraca nam undefined)
```

Mechanizm this - "problem 1"

```
const dog = {
   name: 'rocky',
   showName() {
   console.log("Imię psa to " + this.name);
   }
}

dog.showName() // Imię psa to rocky

const dogName = dog.showName
dogName() // Cannot read property 'name' of undefined
```

Mechanizm this - "problem 2"

```
const cat = {
        kids: ['lucek', 'łapciuch'],
        showKidsNames() {
                console.log(`kot ma potomstwo: ${this.kids}`);
                const showKidsNumber = function() {
                        console.log(this.kids.length);
                }
                showKidsNumber()
        }
}

cat.showKidsNames()
// showKidsNames - kot ma potomstwo: lucek, łapciuch
// showKidsNumber - Cannot read property 'kids' of undefined
```

bind() - trwałe przypisanie this

Jeśli potraktować tamte przykłady jako problem (nie jest to sam w sobie błąd jezyka, ale może to stanowić dla nas utrudnienie), to możemy zastosować trwałe, zdefiniowane wiązanie za pomocą metody bind.

cat.showName() //wykonane na obiekcie cat, wiązanie domyślnie będzie do obiektu cat.

cat.showName.bind(dog) //zwraca nową funkcję o tej samej nazwie (nie wywołuje jej), ale z przypisanym na stałe do this nowym obiektem. This nie będzie się już definiować w chwili wywołania metody, ono będzie wtedy miało przypisany do siebie obiekt dog.

Mechanizm this - "problem 1 - rozwiązanie"

```
const dog = {
   name: 'rocky',
   showName() {
   console.log("Imię psa to " + this.name);
   }
}

dog.showName() // Imię psa to rocky

const dogName = dog.showName.bind(dog)

dogName() // ~~Cannot read property 'name' of undefined~~ Imię psa to rocky
```

Mechanizm this - "problem 2 - rozwiązanie"

```
const cat = {
        kids: ['lucek', 'łapciuch'],
        showKidsNames() {
                console.log(`kot ma potomstwo: ${this.kids}`);
                const showKidsNumber = function() {
                        console.log(this.kids.length);
                }.bind(this)
                showKidsNumber()
        }
}

cat.showKidsNames()
// showKidsNames - kot ma potomstwo: lucek,łapciuch
// showKidsNumber - 2
```

Mechanizm this - "problem 2 - rozwiązanie (2)"

```
const cat = {
        kids: ['lucek', 'łapciuch'],
        showKidsNames() {
                console.log(`kot ma potomstwo: ${this.kids}`);
                const showKidsNumber = () => {
                        console.log(this.kids.length);
                }
                showKidsNumber()
        }
}

cat.showKidsNames()
// showKidsNames - kot ma potomstwo: lucek,łapciuch
// showKidsNumber - 2
```

To jesteśmy gotowi na start z React...

Przejdźmy do kodu