Technologie Internetu - wykład

Programowanie funkcyjne w JavaScript

Paradygmaty programowania

Czytelny i logiczny kod

- Intuicyjny.
- Uporządkowany.
- Bez **skutków ubocznych** i niespodzianek.

Najważniejsze paradygmaty

- **Programowanie obiektowe**. Hermetyzacja klas porządkuje i zapobiega niechcianym zmianom.
- **Programowanie funkcyjne**. Mało elementów podatnych na niechciane zmiany.

Programowanie funkcyjne

Obiekt pierwszej klasy.

- To taki obiekt, który można zapisać do zmiennej i np. przekazać do funkcji.
- Wiemy już, że w JavaScript funkcja może być takim obiektem.

```
var x = function(a) { return a * 2; }

x(4); //8
```

Callback

Funkcję przekazujemy jako parametr.

- Funkcje wyższego rzędu, jako parametr dostają inne funkcje.
- Dzięki temu cały kod możemy pozamykać w funkcje.
- Cel: jak najmniej danych i instrukcji “luzem”.

```
var mnozenie = function(a, b) {
  return a * b;
}
var dzielenie = function(a, b) {
  return a / b;
}

function kalkulator(fn, n1, n2)
{
  return fn(n1, n2);
}
kalkulator(mnozenie, 5, 3); //15
```

Deklaratywne, czy imperatywne

Programowanie funkcyjne jest deklaratywne

- W kodzie deklarujemy co chcemy osiągnąć…
- … a nie iterujemy aż to osiągniemy.
- Pętle są imperatywne, więc ich unikamy tam gdzie to możliwe.

```
var arr = [10, 20, 30];
for(let i = 0; i < arr.length; i++) {
   arr[i]++; // pętla iteruje po tablicy
}
```

Deklaratywne, czy imperatywne

Programowanie funkcyjne jest deklaratywne

- W kodzie deklarujemy co chcemy osiągnąć…
- … a nie iterujemy aż to osiągniemy.
- Pętle są imperatywne, więc ich unikamy tam gdzie to możliwe.

```
var arr = [10, 20, 30];
let plus1 = function(a) {return ++a;}
let newArr = arr.map(plus1);
// wykonujemy funkcję na każdym elemencie tablicy
```

Funkcje strzałkowe

Skoro wszystko chcemy obsłużyć funkcjami, przyda się prostszy zapis.

- Funkcja zapisana w jednej linijce.
- [Zmienna] = ([opcjonalne parametry]) => wynik.
- [Zmienna] = ([opcjonalne parametry]) => { instrukcje; return wynik; }.

```
var arr = [10, 20, 30];
let plus1 = (a) => ++a;

let newArr = arr.map(plus1);
```

Czystość funkcji

Nieczysta

```
let x = 10;

let f1 = () => {x = x * 2;}

f1();

//x = 20
```

Czysta

```
let x = 10;

let f1 = (n) => n * 2;

f1(x);

//f1() = 20; x = 10
```

Czystość funkcji

- Dla tych samych parametrów ten sam wynik.
- Nie modyfikuje wartości poza swoim **scope**.
- Otrzymuje parametry i zwraca wynik.
- Minimalizujemy skutki uboczne.

Niemutowalność danych

Czyli niezmienność.

- Nie chcemy by zdefiniowane struktury danych zmieniały się.
- Zamiast tego tworzymy nowe struktury na podstawie istniejących.
- Robimy to w ramach funkcji.

```
var arr = [10, 20, 30];
for(let i = 0; i < arr.length; i++) {
   arr[i]++; // globalna zmienna została zmieniona, to źle
}
```

Niemutowalność danych

Czyli niezmienność.

- Nie chcemy by zdefiniowane struktury danych zmieniały się.
- Zamiast tego tworzymy nowe struktury na podstawie istniejących.
- Robimy to w ramach funkcji.

```
var arr = [10, 20, 30];
let plus1 = (a) => ++a;
let newArr = arr.map(plus1);
// powstała nowa tablica, stara się nie zmieniła, to dobrze
```

Niemutowalność danych

- **Skutek uboczny** - gdy funkcja zmieni coś na zewnątrz od siebie.
- Zupełna eliminacja skutków ubocznych jest niemożliwa.
- W programowaniu funkcyjnym, skutkiem ubocznym jest np. console.log, zapis do pliku, czy wywołanie zapytania do serwera.
- Zaawansowane aplikacje muszą pozwalać na jakieś skutki uboczne, ale tylko jeśli to konieczne.

Zwracanie funkcji

Wynikiem działania funkcji może być... funkcja

```
function wizytowka(zwrot) {
    return (imie) => `${zwrot}, ${imie}`;
}

let formal = wizytowka("Dzien dobry");
let informal = wizytowka("Czesc");

let jan = formal("Jan");//Dzien dobry, Jan
```

Zwracanie funkcji

Wynikiem działania funkcji może być... funkcja

```
let fn = a => b => c => a * b * c;


fn(2)(3)(4);
```

Zwracanie funkcji

Wynikiem działania funkcji może być… funkcja

```
let fn = a => (b => (c => a * b * c));
//równoważny zapis

fn(2)(3)(4);
```

Funkcje na tablicach

… które mogą wykonać dowolną funkcję na każdym elemencie tablicy.

- Każda z nich zwraca wynik do zmiennej.
- Oryginalna tablica jest niezmieniona.
- map() - wykonuje funkcję na elementach,
- filter() - wyszuka elementy, dla których podana funkcja zwraca true,
- reduce() - połączy elementy zgodnie z regułą z podanej funkcji,
- sort() - posortuje elementy, porównując je ze sobą na podstawie podanej funkcji.

```
let a = [10, 21, 30];

let a1 = a.map(x => x + 5);
//15, 26, 35

let a2 = a.filter(x => x % 10);
//21

let a3 = a.reduce((suma, x) =>
`${suma}; ${x}`);
// "10; 21; 30"

let a4 = a.sort((a, b) => b-a);
// 30, 21, 10
```