JavaScript

Četrnaesti dio

Pregled

- Funkcionalno programiranje
  - Uvod
  - Zašto FP
  - Kako pišemo kod FP
  - Zadaci
  - Function composition
- Lodash.js

Obnavljanje

- Fetch
  - Čemu služi
  - Koje argumente ima
- Postman
- APIs

Funkcionalno programiranje, uvod

- Stil/način kodiranja podržan od nekih programskih jezika
- Neki od programskih jezika, osim JSa, koji podržavaju funkcionalno programiranje
  - F#
  - Erlang
  - Haskell
  - Closure
  - Elm
  - Scala
- Neki od gore navedenih jezika podržavaju samo funkcionalno programiranje
- Za funkcionalno programiranje kaže se još i da je paradigma, tj. mindset

Ostale paradigme

Imperative ● Klasični vid programiranja ○ Prati moje instrukcije ○ Ovdje tačno na osnovu redosleda komadni definišemo šta će sledeće da se izvrši ○ Brzo postane komplikovano za čitanje	**Declarative** ● Na neki način apstraktno programiranje ○ Kažete programu šta da radi ali na neki način opisno ○ Uradi to, ne zanima me kako ○ To je npr. SQL
OOP ● Komunikacija sa objektima ○ Koristimo metode da komuniciramo sa objektima	**Funcional** ● Na neki način podskup od Declarative paradigme ○ Pa šta je onda tačno funkcionalno programiranje?

Funkcionalno programiranje, nastavak

- Funkcionalno programiranje koristi samo čiste funkcije, **pure functions!**
- **Pure functions** su funkcije koje koriste samo argumente kao input i ništa osim toga i vraćaju output

Pure?

```
var name = "Saint Petersburg";

function greet() {
   console.log("Hello, " + name + "!");
}

greet(); // Hello, Saint Petersburg!

name = "HolyJS";
greet(); // Hello, HolyJS!
```

Argumenti?
Name?
Return?
console.log() - side effect
Drugi poziv, output nije isti!

Funkcionalno programiranje, nastavak(2)

- Kako bi izgledala pure function za prethodni primjer?

```
function greet(name) {
   return "Hello, " + name + "!";
}

greet("Saint Petersburg");
// "Hello, Saint Petersburg!"

greet("HolyJS");
// "Hello, HolyJS!"
```

Zašto funkcionalno programiranje?

- Neki od razloga
  - Rezultati predvidljivi
  - Sigurniji smo
  - Lakše testiranje i debug
    - Znamo šta ćemo dobiti kao output za definisani input
    - Ako se desi greška znamo da je problem konkretna funkcija, a ne ništa van nje
  - Reusability

Kako pišemo kod FP

- Sav kod sastavljen je samo od funkcija!

Imperative style

```
var city = "St. Petersburg";
var greeting = "Hi";

console.log(greeting + ", " + city + "!");
// Hi, St. Petersburg!

greeting = "Привет";
console.log(greeting + ", " + city + "!");
// Привет, St. Petersburg!
```

Functional style

```
function greet(greeting, name) {
   return greeting + ", " + name + "!";
}

greet("Hi", "St. Petersburg");
// "Hi, St. Petersburg!"

greet("Привет", "HolyJS");
// "Привет, HolyJS!"
```

Kako pišemo kod FP, nastavak(1)

- Izbjegavati side-effects, tj. funkcija treba da obradi input-e(argumenti) i vrati output sa return
  - Npr. i console.log() je side-effect jer utiče na okruženje van funkcije

Primjer sa side-effects

```
var conf = {name: "SaintJS", date: 2017};

function renameConf(newName) {
   conf.name = newName;
   console.log("Renamed!");
}

renameConf("HolyJS"); // Renamed!
conf; // {name: "HolyJS", date: 2017}
```

Primjer bez side-effects

```
var conf = {name: "SaintJS", date: 2017};

function renameConf(oldConf, newName) {
   return {name: newName, date: oldConf.date}
}

var newConf = renameConf(conf, "HolyJS");
newConf; // {name: "HolyJS", date: 2017}
conf;    // {name: "SaintJS", date: 2017}
```

Kako pišemo kod FP, nastavak(2)

- Izbjegavati mijenjanje podataka, uvijek kreirati nove kopije
  - To smo u prethodnom primjeru odradili sa **conf.name**
  - Ne raditi change in-place

Opasno, mutation

```
var numSysts = ["Roman", "Arabic", "Chinese"];

numSysts[1] = "Hindu-" + numSysts[1];

numSysts;
// ["Roman", "Hindu-Arabic", "Chinese"]
```

Sigurnije, kreiranje novog niza

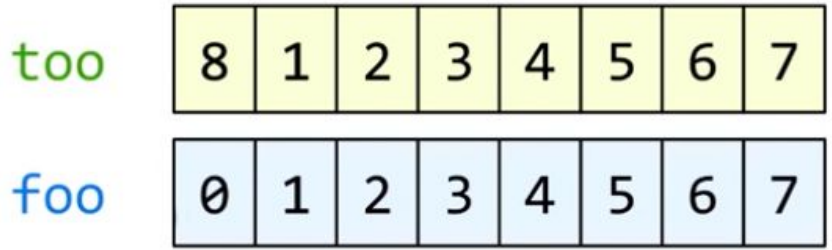
```
const numSysts = ["Roman", "Arabic", "Chinese"];

const newNumSysts = numSysts.map((num) => {
  if (num === "Arabic") { return "Hindu-" + num; }
  else { return num; }
});

newNumSysts;//["Roman", "Hindu-Arabic", "Chinese"]
numSysts;   //["Roman", "Arabic", "Chinese"]
```

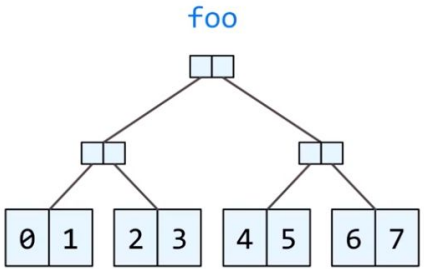
Mane za prethodni pristup

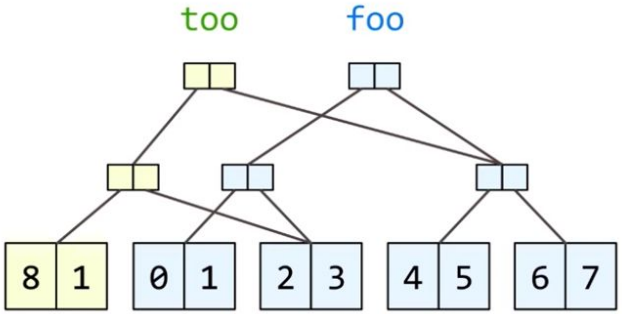
- Svaki put alociramo novi prostor u memoriji
- Ovo nije efikasno !



Rješenje za prethodni primjer

- Koristiti persistent strukture podataka





Paketi za persistent data structures

- Mori.js
- Immutable.js

Mori
https://swannodette.github.io/mori

```
var f = mori.vector(1,2);
var t = mori.conj(f, 3);
```

Immutable.js
https://facebook.github.io/immutable-js

```
var f = Immutable.List.of(1,2);
var t = f.push(3);
```

Kako pišemo kod FP, nastavak(3)

- Ne koristiti iteracije, koristite rekurziju

Iteracija

```
function sum (numbers) {
  let total = 0;
  for (i = 0; i < numbers.length; i++) {
    total += numbers[i];
  }
  return total;
}

sum([0,1,2,3,4]); // 10
```

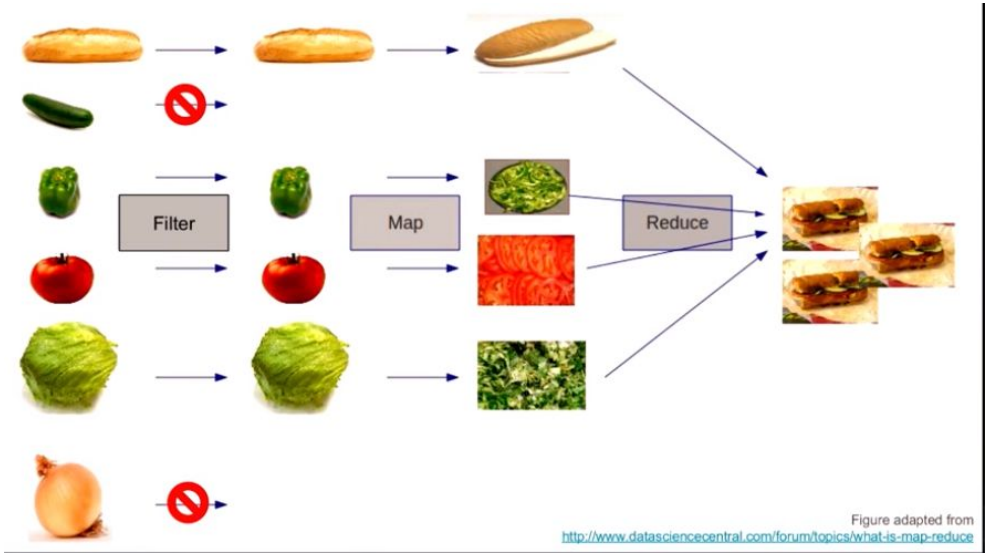
Rekurzija

```
function sum (numbers) {
  if (numbers.length === 1) {
    return numbers[0];
  } else {
    return numbers[0] + sum(numbers.slice(1));
  }
}

sum([0,1,2,3,4]); // 10
```

Iteracija bez petlji

- Uvedimo nove funkcije za iterabilne strukture
  - .filter()
  - .map()
  - .reduce



Practice time :)

- Zadaci (code-1)
  - forEach
  - Filter
  - Map
  - Sort
  - Reduce
  - Every
  - Some

```
function acronym(str,word) {
    return str + word.charAt(0);
}

["Functional","Light","JavaScript","Stuff"]
.reduce(acronym,"");
// FLJS
```

Primjer, function composition

```
var ender = (ending) => (input) => input + ending;

var adore = ender(" rocks");
var announce = ender(", y'all");
var exclaim = ender("!");

var hypeUp = (x) => exclaim(announce(adore(x)));
hypeUp("JS"); // "JS rocks, y'all!"
hypeUp("FP"); // "FP rocks, y'all!"
```

```
var r = require("ramda");

var rtlHype = r.compose(adore, announce, exclaim);
rtlHype("FP");  // "FP!, y'all rocks"

var ltrHype = r.pipe(adore, announce, exclaim);
ltrHype("FP");  // "FP rocks, y'all!"
```

Primjer-2, composition

```
1  function sum(x,y) {
2      return x + y;
3  }
4
5  function mult(x,y) {
6      return x * y;
7  }
8
9  // (3 * 4) + 5
10 sum( mult( 3, 4 ), 5 ); // 17
```

```
1  function increment(x) {
2      return x + 1;
3  }
4
5  function double(x) {
6      return x * 2;
7  }
8
9  var f = composeRight(increment,double);
10 var p = composeRight(double,increment);
11
12 f(3);   // 7
13 p(3);   // 8
```

Lodash

- https://lodash.com
- Modularan
  - Ne moramo da importujemo cijeli paket, možemo da koristimo samo ono što nam treba
- Zasnovan na funkcionalnom programiranju
- Koristan za manipulaciju nizovima i objektima
- Pogledajmo i testirajmo par funkcija

Šta nismo pomenuli

- Trampolines
- Monads and Functors
- Curry
- Memoization
- Fusion
- Transducing
- Observables

Resursi

- https://www.youtube.com/watch?v=11vZ80kGa3U
- Functional JavaScript v2 (Frontend-Masters)
- https://github.com/stoeffel/awesome-fp-js

Pitanja