Osnove imperativnog programiranja uz Python i C

Imperativni jezici

Java, C++, C, Python, C#, JavaScript...

Imperativni jezici se zasnivaju na sporednim efektima (mutaciji stanja) kroz niz instrukcija

```
a = 5          # 'a' menja vrednost u 5

a = 8          # 'a' ima novu vrednost 8. Vrednost 5 je
               # izgubljena (destructive update)

print(5)       # IO sporedni efekat
```

```
for (int i = 1; i <= 10; i++)
    printf("%d\n", i);
```

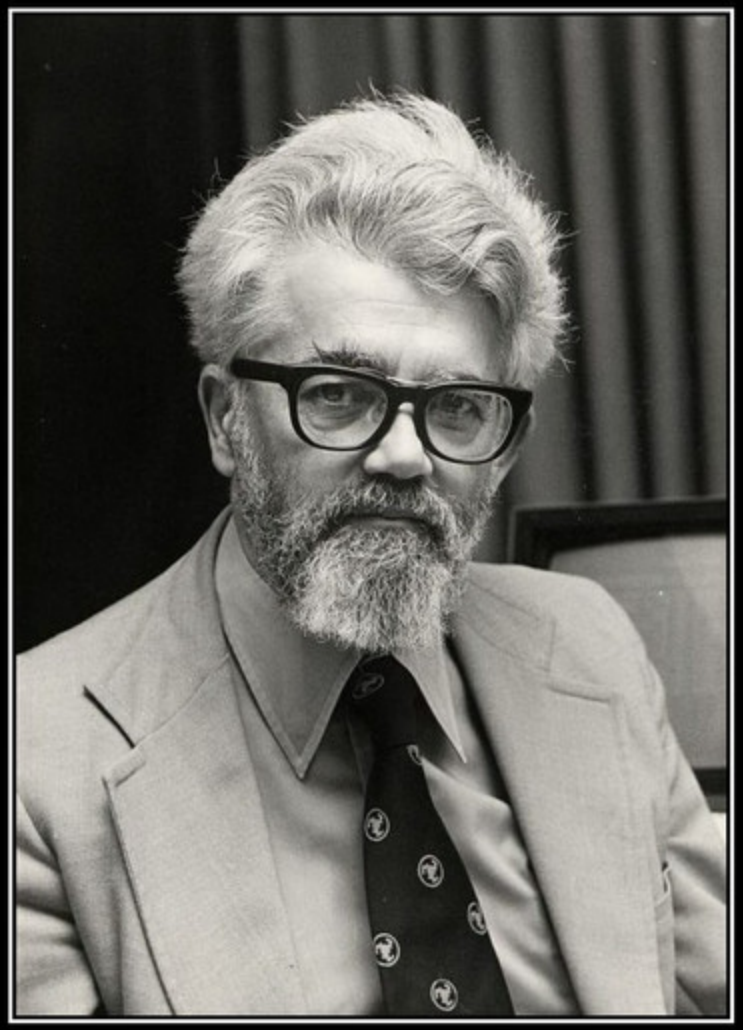
State nije problem ako je ogranicen na lokalni scope (enkapsulacija), zar ne?

Implikacije promenljivog stanja

- Otežano testiranje
  - Kod sa sporednim efektima se mora testirati u širem mock okruzenju (npr. ne možemo testirati metodu u izolaciji bez inicializacije klase)
- Kod je obično usko vezan za kontekst
  - free() je jedino validan ako je pre njega pozvan malloc()
  - Kod se može pozvati samo u određenom stanju sistema

```
( Obj.show_panel(true),  set_nesto(10) )
```

- Paralelizaciju koda je komplikovanija nego sto bi trebala biti
  - Threadovi nedeterministično pristupaju zajedničkim resursima (race conditions)



PROGRAMMING

You're Doing It Completely Wrong.

Osnove funckionalnog programiranja uz Clojure (kloužr)

Funkcionalni jezici

- Haskell, Scala, Erlang (Elixir), Lisp (Scheme, Clojure…), itd.

Zašto funkcionalni jezici?

- Nemaju neke od problema koje imaju imperativni jezici
- Pružaju drugačiji pogled na programiranje i rešavanje problema
- Programiranje u funkcionalnim jezicima je zabavno

Funkcionalni jezici se zasnivaju na vrednostima i vezama između vrednosti

Funkcionalni jezici su deklarativni

Vrednosti

```
5

“string”

3.8

true
```

Vrednosti

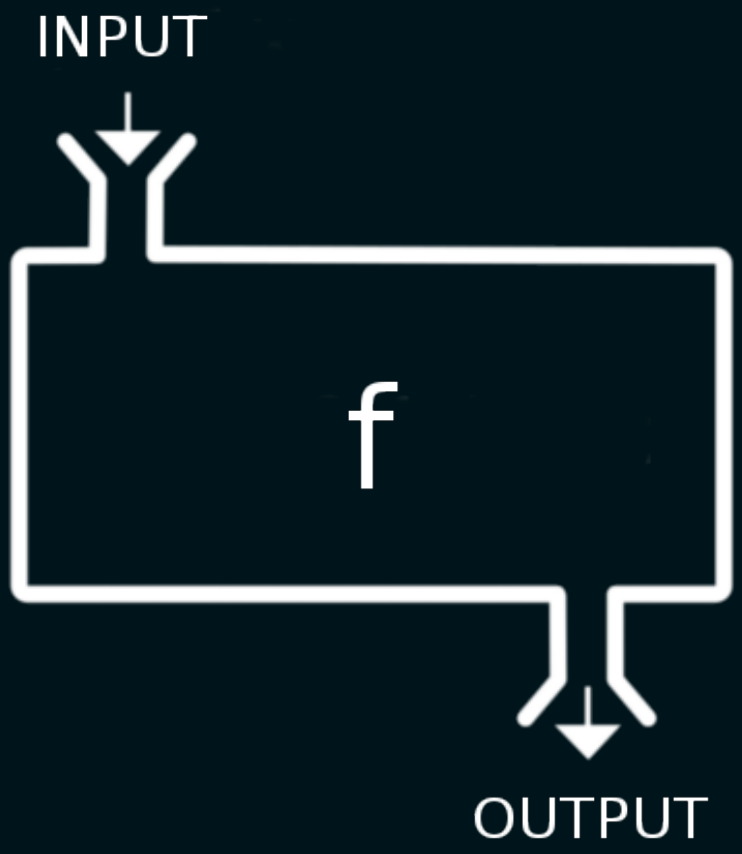
- Vrednosti predstavljaju same sebe
- Ne mogu se menjati (npr. 5 ne moze postati 3)
- Vrednosti se mogu deliti slobodno bez bojazi da ce biti promenjene

Funkcije

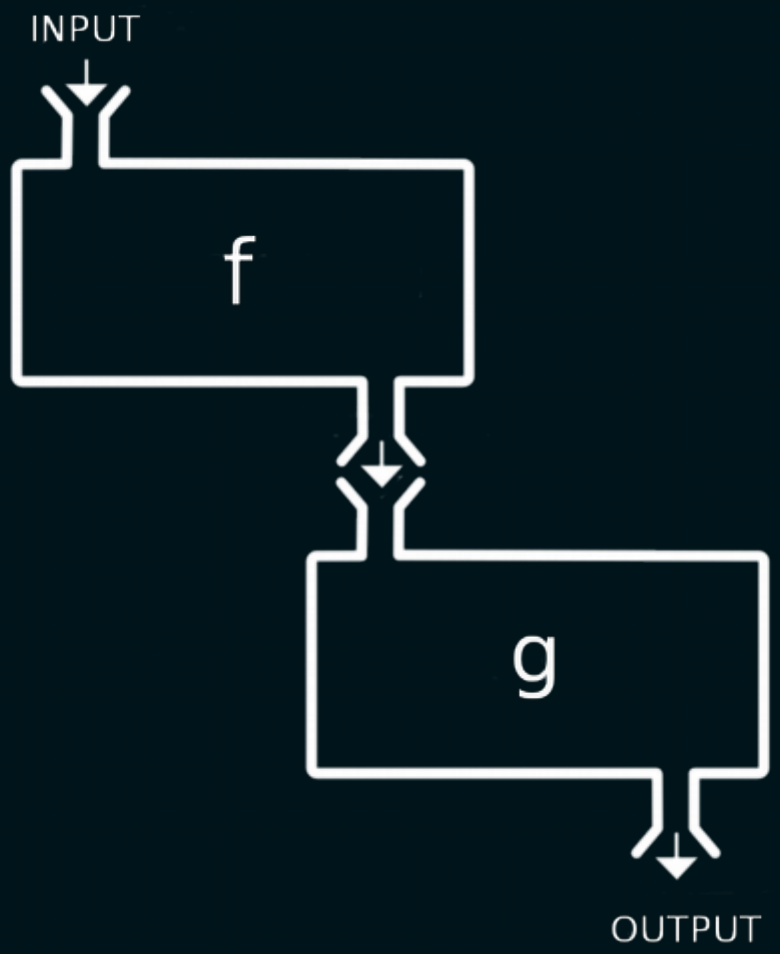
Funkcije mapiraju jednu vrednost na neku drugu vrednost

$$f(x) \rightarrow y$$

Funkcija



Kompozicija funkcija



Funkcije

`sqrt(9)` → `3`

`add2(5)` → `7`

Referenta transparentnost

- Izraz koji se može zameniti svojom vrednošću
  - 3 + (2 * 4) – 5 → 3 + 8 - 5
-
  - 3 + sqrt(16) – 10 → 3 + 4 - 10

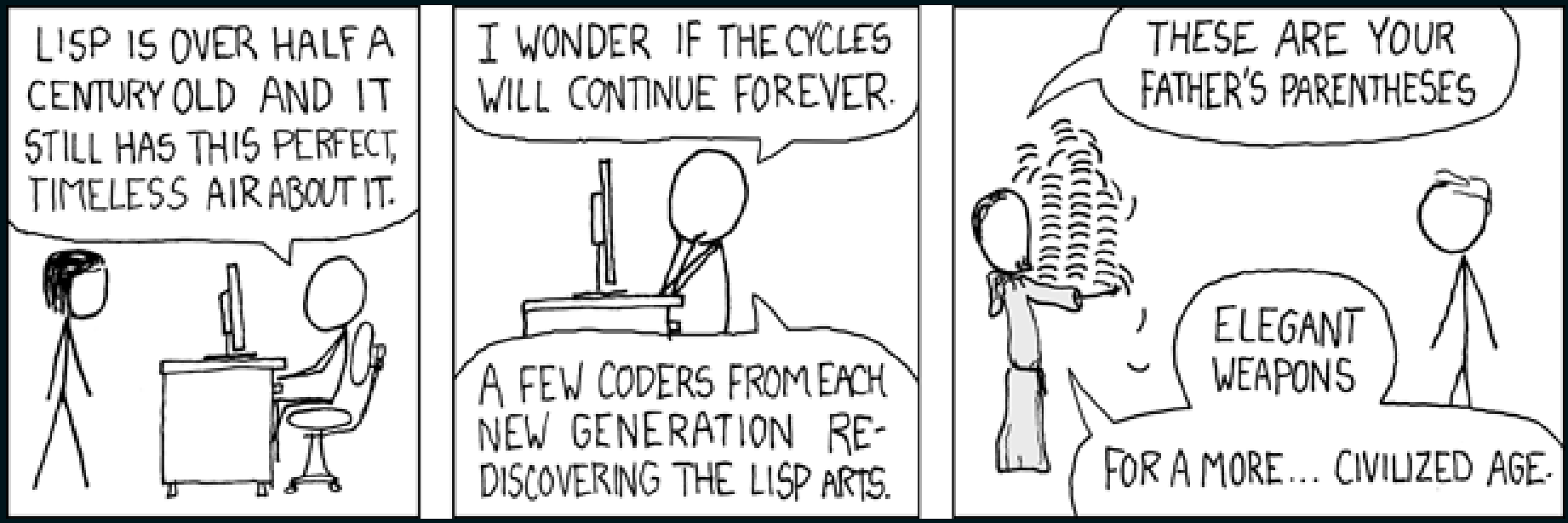
Šta nam funkcionalan kod pruža?

- Nema destruktivnih update-ova i samim tim se vrednosti ne gube
  - Program se može lako vratiti u prethodno stanje
- O programima je lakse razmisljati jer su funkcije izolovane i nezavisne od ostatka koda
  - Funkcije su kao podprogrami
- Kod sačinjen od referentno transparentnih funkcija je lako testirati
- Bezbedan multithreading
  - Paralelan kod bez “race condition-a”

Clojure

- Funkcionalan jezik (sa ograničenom mogućnošću pisanja imeprativnog koda)
- Dinamičan (tipovi vrednosti se odredjuju u runtime-u)
- Radi na JVM-u, .net platformi, JavaScript
- Interoperabilnost sa host platformom
- Moderni Lisp

Clojure… Novi stari jezik



Clojure sintaksa je minimalistična

Clojure sintaksa

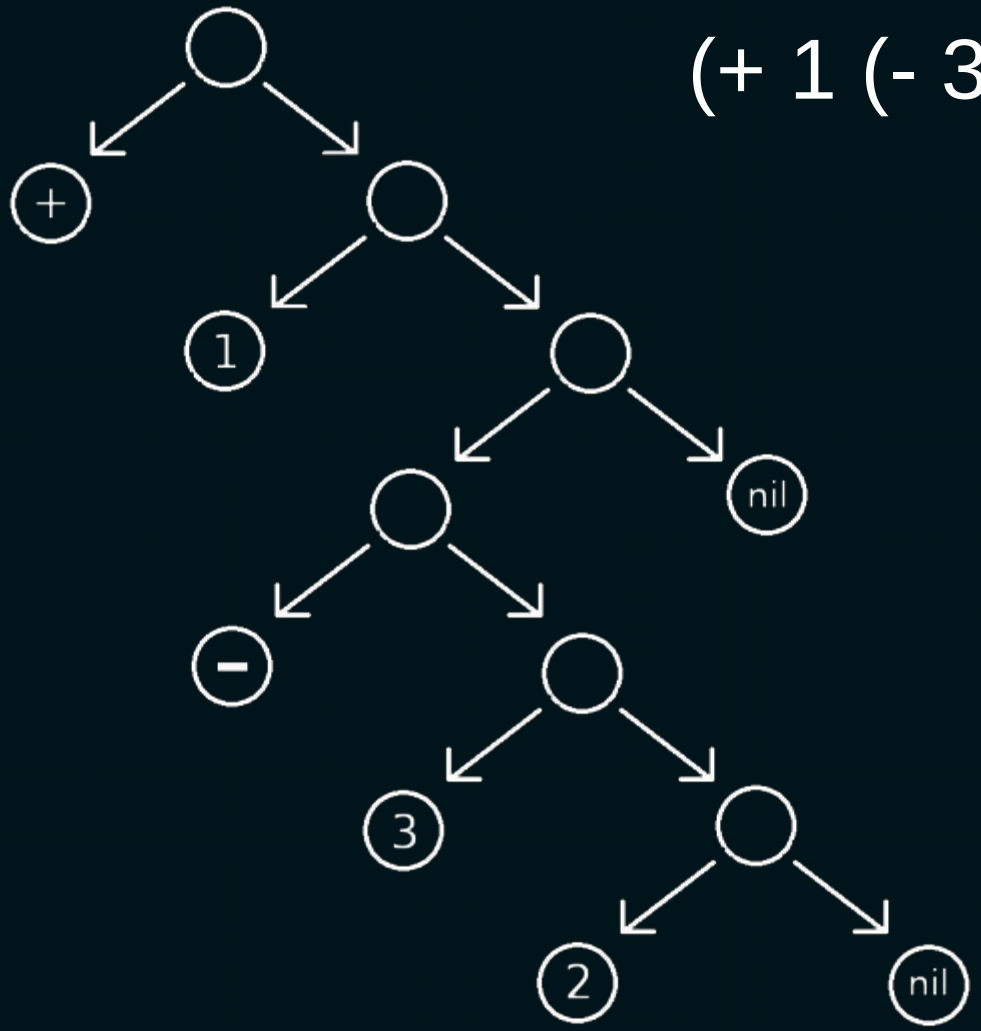
```
[+ 1 2 3]      ;; vektor sa elementima +, 1, 2 i 3

{+ 1 2 3}      ;; mapa (key val key val): + → 1, 2 → 3

#{+ 1 2 3}     ;; skup

'(+ 1 2 3)     ;; lista

(+ 1 2 3)      ;; poziv funkcije '+' sa parametrima 1, 2, 3
```

Clojure programi su 's-izrazi'

(uglavnom)

s - izraz (s - expression)

(+ 1 (- 3 2)



“Data is code and code is data”

- Ne postoji suštinska razlika između podataka i koda.
  Razlika je samo u interpretaciji.

program

```
(+ 2 (* 3 2) (- 5 2) 7 9)
```

lista

```
'(+ 2 (* 3 2) (- 5 2) 7 9)
```

Fokus je na jednostavnim nepromenljivim strukturama podataka

- Mape, liste, vektori, skupovi
- Složenije strukture se grade kompozicijom osnovih struktura
- Strukture su vrednosti (immutable)

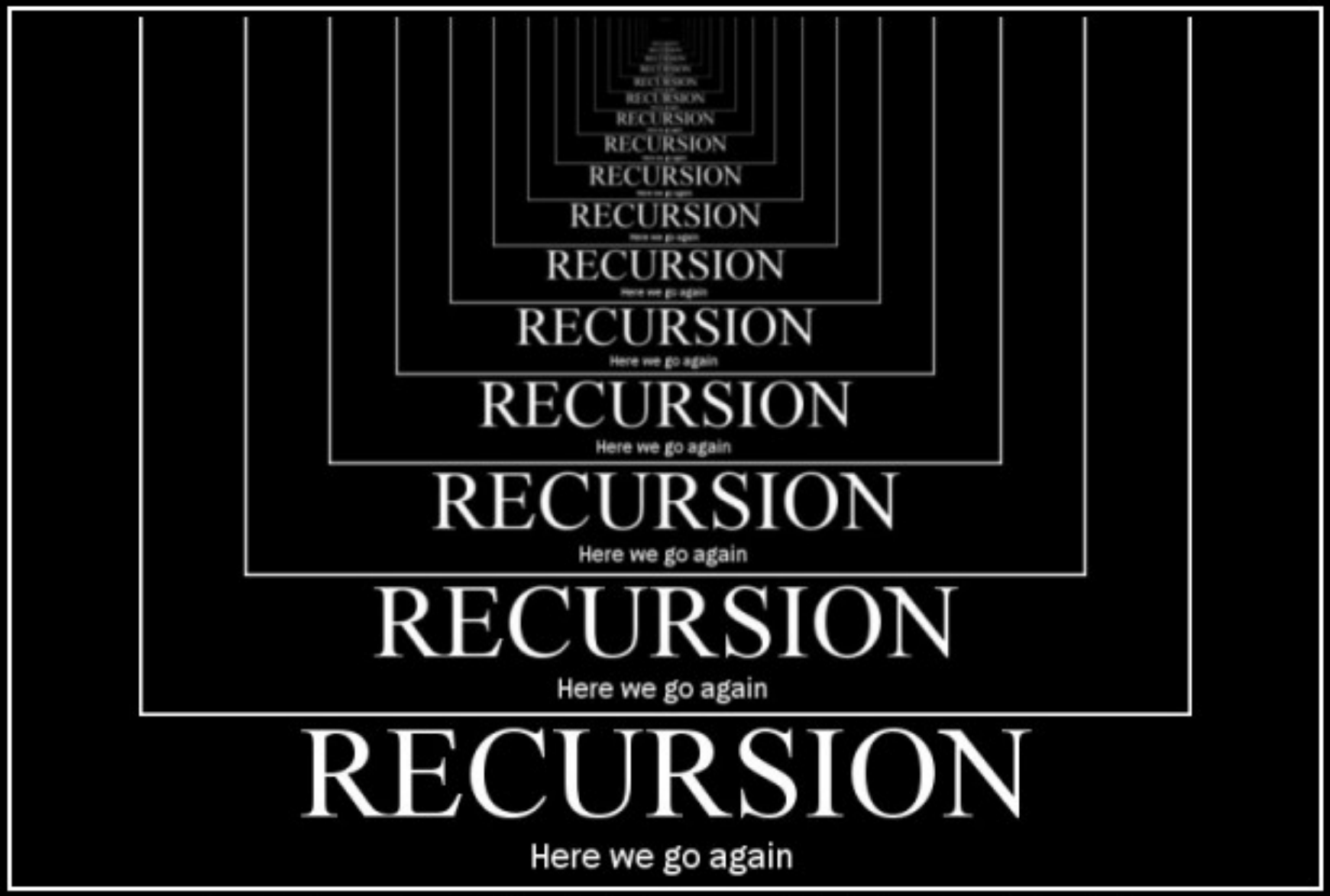
Clojure kod se moze kiristiti kao format podataka

```
{:naziv “Uvod u funkcionalno programiranje”
 :url “http://example.com”
 :jezik {:naziv “Clojure”
         :url “http://clojure.org”
         :verzija 1.8}
 :niz-brojeva [9 8 0 3 3 1]
 :neka-funkcija '(defn foo [n] (+ n 1))}
```

Rešavanje problema na funkcionalan način

Iteracija?

```
for (int i = 1; i <= 10; i++) {
    printf(“%d”, i);
}
```


RECURSION
Here we go again

Rekurzija

```clojure
(defn print-range
  [from to]
  (if (> from to)
    nil
    (do (println from)
        (print-range (inc from)
                     to))))


(print-range 1 10)
```

```clojure
(loop [from 1 to 10]
  (if (> from to)
   nil
   (do (println from)
       (recur (inc from) to))))
```

Funkcija “map”

```
(map println (range 1 11))
```

Lenja evaluacija (Lazy evaluation)

- Odložena evaluacija nekog izraza dok njegova vrednost ne postane potrebna
- Potrošnja memorije je umanjena

Beskonačne lenje liste

Primeri

Pouka cele ove priče?

Izbor jezika je bitan jer vas tera da kroz njega razmišljate.

Kraj