Funkcionalno programiranje

Vežbe

01 Uvod

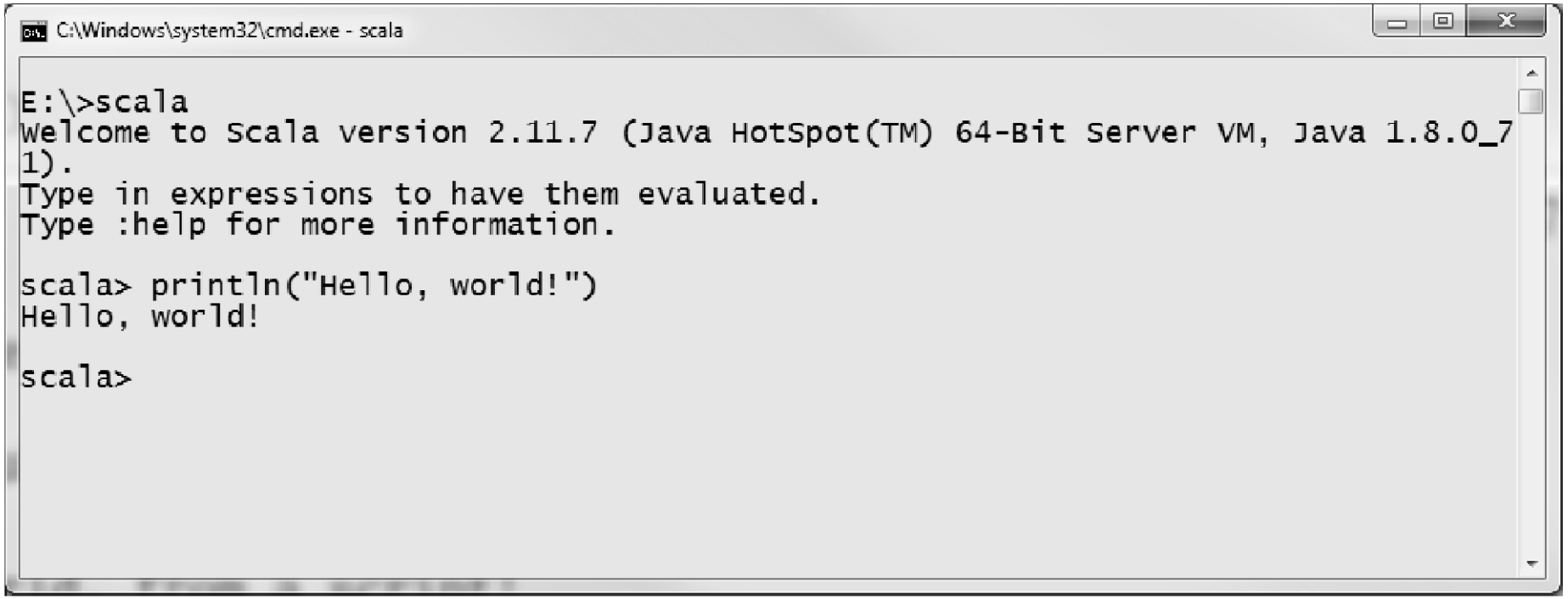
Uvod

- Par reci o raznim IDE, instalaciji + komandnoj liniji, skriptama, itd.

Zadatak 1.1 – "Hello, world"

Iz interpretera

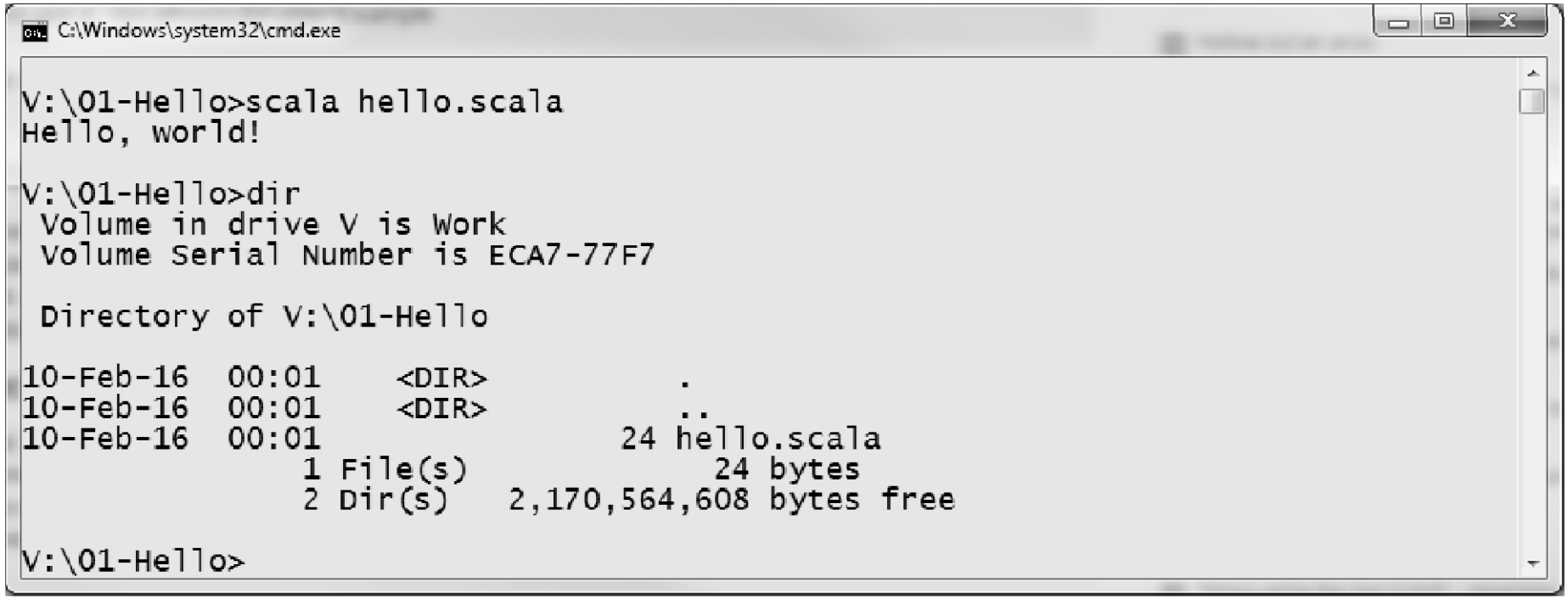
```
scala> println("Hello, world!")
Hello, world!
```



Zadatak 1.1 – "Hello, world"

U vidu skripte (hello.scala)

```
C:\> scala hello.scala
Hello, world!
```



Zadatak 1.2 - zbir

- Napisati skriptu koja iz komandne linije pročita dva cela broja i ispiše njihov zbir.

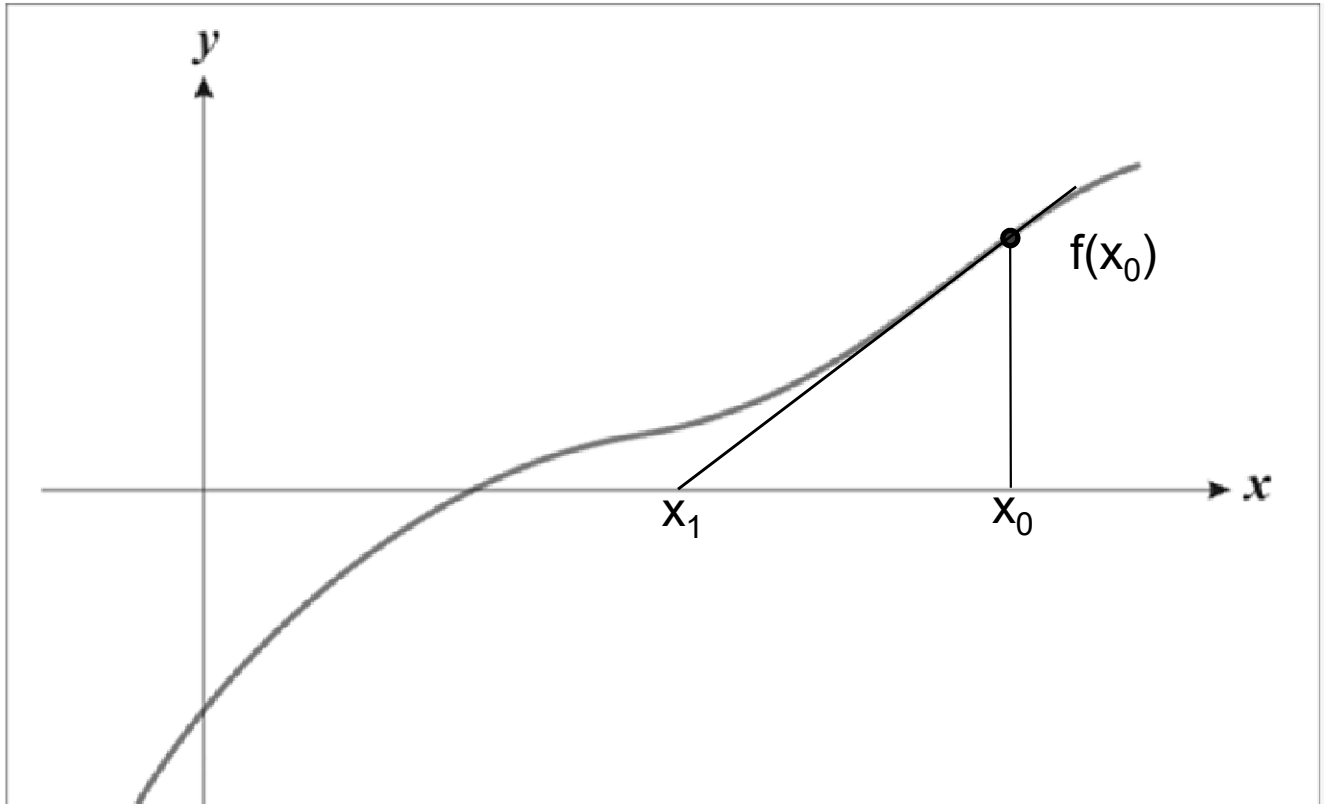
```
println(args(0) + " + " +args(1) + " = " +
        (
          Integer.parseInt(args(0))
            +
          Integer.parseInt(args(1))
        )
       )
```

Snimiti u fajl "suma.scala"

```
> scala suma.scala 5 8
> 5 + 8 = 13
```

Zadatak 1.3 – računanje kvadratnog korena

- Korišćenjem iterativne (rekurentne) formule prema Newton-Raphson-ovom metodu izračunati približno kvadratni koren datog broja



$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

$$f(x) = x^2 - a$$

$$f'(x) = 2x$$

$$\Downarrow$$

$$x_{n+1} = x_n - \frac{x_n^{\,2} - a}{2x_n}$$

Zadatak 1.3 – računanje kvadratnog korena

- Generalna ideja
  - metod koristi sukcesivne aproksimacije
  - u i-tom koraku treba utvrditi da li je aproksimacija dovoljno dobra
  - ako nije, proceniti novu aproksimaciju i ponoviti

```
def sqrtIter(guess: Double, x: Double): Double =
  if (isGoodEnough(guess, x)) guess
  else sqrtIter(improve(guess, x), x)
```

- Primetiti
  - koristi se (specijalna) rekurzija – **mora da se navede tip rezultata**
  - rekurzija na dnu (*tail recursion*) – rekurzivan poziv je poslednji izraz
  - u tom slučaju, ponaša se kao **iteracija**, ne kao rekurzija

Zadatak 1.3 – računanje kvadratnog korena

```
object sqrtProgram {
  def sqrt(x: Double) = {
    def approx(guess: Double) = (guess + x/guess) / 2
    def goodApprox(guess: Double) =
      abs(sqr(guess)-x)/x < 1e-15
    def abs(x: Double) = if(x>=0) x else -x
    def sqr(x: Double) = x*x
    def sqrtIter(guess: Double): Double =
      if( goodApprox(guess) )   guess
      else                      sqrtIter( approx(guess) )

    sqrtIter(1)
  }                  //> sqrt: (x: Double)Double
   sqrt(15)          //> res0: Double = 3.872983346207417
}
```

Zadatak 1.4 – linearna i terminalna rekurzija

Funkcija sum koristi linearnu rekurziju. Napisati funkciju tako da koristi terminalnu rekurziju.

- Podsetnik sa predavanja:

```
def sum(f: Int => Double)(a: Int, b: Int): Double = {
  if ( a>b ) 0 else f(a) + sum(f)(a+1, b)
}
```

Zadatak 1.4 – linearna i terminalna rekurzija

Ideja: parcijalni rezultat treba prenositi u rekurzivne pozive

```
def sum(f: Int => Double)(a: Int, b: Int) = {
  def sumTR(a: Int, r: Double): Double =
    if ( a>b )      r
    else            sumTR(a+1, r+f(a))

   sumTR(a, 0)
}
```

Zadatak 1.5 – generalizacija pristupa

- a) Na osnovu funkcije sum iz zadatka 1.4, napisati funkciju `product` koja računa proizvod vrednosti u zadatom intervalu
- b) Na osnovu funkcije iz tačke (a) napisati funkciju koja računa n!
- c) Generalizovati pristup

```
def sum(f: Int => Double)(a: Int, b: Int) = {
  def sumTR(a: Int, r: Double): Double =
    if ( a>b )      r
    else            sumTR(a+1, r+f(a))

  sumTR(a, 0)
}
```

Zadatak 1.5 – generalizacija pristupa

a)

```
def product(f: Int => Double)(a: Int, b: Int) = {
  def productTR(a: Int, r: Double): Double =
    if ( a>b )      r
    else            productTR(a+1, r*f(a))

  productTR(a, 1)
}

product(x=>x)(2,5)    // Double = 120.0
```

b)

```
def fact(n: Int) = product(x=>x)(1,n)

fact(5)               // Double = 120.0
```

Zadatak 1.5 – generalizacija pristupa

c)

```
def genop(f: Int => Double, id: Int,
          g: (Double, Double)=> Double)(a: Int, b: Int) = {
  def tr(a: Int, r: Double): Double =
    if ( a>b )      r
    else            tr(a+1, g(r,f(a)))

  tr(a, id)
}

def fact2(n: Int)=genop(x=>x, 1, (x,y)=>x*y)(1,n) //(n: Int)Double
fact2(5)
```

Zadatak 1.6 – Rekurzija i kompozicija

- a) Napisati definiciju funkcije koja vraća funkciju koja vraca vrednost jedinog parametra tipa `Int`

- b) Napisati funkciju koja za date dve funkcije `f` i `g`, koje preslikavaju `Int` u `Int`, vraća funkciju koja vrši kompoziciju funkcija `f` i `g`: `y = g(f(x))`

- c) Napisati funkciju koja vraća funkciju koja predstavlja `n` puta komponovanu funkciju `f` nad parametrom `x`:
  `f(f(f ... f(x) ...))`

- d) Napisati funkciju iz tačke (c) upotrebom funkcija iz tačaka (a) i (b)

Zadatak 1.6 – Rekurzija i kompozicija

a)

```
def id = (x: Int) => x
```

b)

```
def compose(f: Int => Int, g: Int => Int): Int => Int =
   x => g(f(x))
```

c)

```
def repeated(f: Int => Int, n: Int): Int => Int =
   x => if (n == 0) x else repeated(f, n-1)(f(x))
```

```
repeated(f, 3)(10)
→  10           => repeated(f, 2)(f(10))
→  f(10)        => repeated(f, 1)(f(f(10)))
→  f(f(10))     => repeated(f, 0)(f(f(f(10))))
→  f(f(f(10))) => f(f(f(10)))
```

Zadatak 1.6 – Rekurzija i kompozicija

a)

```
def id = (x: Int) => x
```

b)

```
def compose(f: Int => Int, g: Int => Int): Int => Int =
   x => g(f(x))
```

c)

```
def repeated(f: Int => Int, n: Int): Int => Int =
   x => if (n == 0) x else repeated(f, n-1)(f(x))
```

d)

```
def repeated(f: Int => Int, n: Int): Int => Int =
   if(n == 0) id
   else compose(f, repeated(f, n-1))
```

Zadatak 1.7 – Nalaženje fiksne tačke funkcije

- Vraćanje na problem pronalaženja kvadratnog korena uz upotrebu funkcija višeg reda
- Broj x zove se fiksna tačka funkcije `f` ako važi `f(x) = x`
- Za određene funkcije, fiksna tačka se može odrediti uzastopnom primenom funkcije f, počevši od inicijalne procene: `x, f(x), f(f(x)), f(f(f(x))), ...`
- Napisati funkciju koja određuje fiksnu tačku zadate funkcije
- Zatim upotrebiti tu funkciju za definisanje funkcije sqrt(x)

Zadatak 1.7 – Nalaženje fiksne tačke funkcije

```
val tolerance = 0.0001
def isCloseEnough(x : Double, y : Double) =
   abs((x - y) / x) < tolerance
def fixedPoint(f: Double => Double)(firstGuess: Double) = {
   def iterate(guess : Double): Double = {
       val next = f(guess)
       if(isCloseEnough(guess, next)) next
       else iterate(next)
   }
   iterate(firstGuess)
}
```

Zadatak 1.7 – Nalaženje fiksne tačke funkcije

sqrt(x) = broj **y** takav da **y * y = x**
= broj **y** takav da **y = x / y**

Dakle, `sqrt(x)` se može napisati kao `(y => x / y)`

```
def sqrt(x: Double) = fixedPoint( y => x / y )(1.0)
```

Na žalost, funkcija ne konvergira (alternira 1.0, 2.0, 1.0, ...)

Rešenje, koje odgovara onom iz zadatka 1.3, je da se
procena računa kao srednja vrednost
dve uzastopne procene

```
def sqrt(x: Double) = fixedPoint( y => (y + x/y)/2 )(1.0)
```

Zadatak 1.7 – Nalaženje fiksne tačke funkcije

Pošto je tehnika "stabilizacije" fiksne tačke dovoljno generalna, zaslužuje da se apstrahuje posebnom funkcijom

```
def averageDamp(f: Double => Double)(x: Double) =
  (x + f(x)) / 2
```

Sada se može ponovo formulisati funkcija `sqrt(x)`

```
def sqrt(x: Double) =
  fixedPoint( averageDamp( y => x / y ) ) (1.0)
```

Pitanja:

- kako izračunati 3. koren?
- kako izračunati n-ti koren?