Funkcionalno programiranje

Vežbe

03 Kontrolne strukture

Zadatak 3.1

- (a) Napisati for ciklus koji ispiše sve elemente niza stringova

```
object loops {

 val marke = Array("BMW", "Mercedes", "Audi", "Porsche")
 //> marke  : Array[String] = Array(BMW, Mercedes, Audi, Porsche)

 for( marka <- marke )
       println(marka)
                                                    //> BMW
                                                     //| Mercedes
                                                     //| Audi
                                                     //| Porsche
}
```

```
def apply[T: ClassTag](xs: T*): Array[T] = {
   val array = new Array[T](xs.length)
   var i = 0
   for (x <- xs.iterator) { array(i) = x; i += 1 }
   array
}
```

Zadatak 3.1

- (b) Napisati for ciklus koji ispiše sve elemente niza stringova nakon pretvaranja svih slova u velika

```
object loops {

 val marke = Array("BMW", "Mercedes", "Audi", "Porsche")
 //> marke  : Array[String] = Array(BMW, Mercedes, Audi, Porsche)

 for( marka <- marke )  {
    val m = marka.toUpperCase     /* println( marka.toUpperCase ) */
    println(m)
 }
                                                    //> BMW
                                                     //| MERCEDES
                                                     //| AUDI
                                                     //| PORSCHE

}
```

Zadatak 3.2

- Napisati for ciklus koji formira nov niz elemenata tipa string u kojem su svi znaci velika slova

```
object loops {

 val marke = Array("BMW", "Mercedes", "Audi", "Porsche")
 //> marke  : Array[String] = Array(BMW, Mercedes, Audi, Porsche)

 val marke2 = for( marka <- marke )  yield marka.toUpperCase
//> marke2  : Array[String] = Array(BMW, MERCEDES, AUDI, PORSCHE)


 // yield vraća tip po kojem se iterira
 val marke3 = for(marka <- marke.toList) yield marka.toUpperCase
 //> marke3  : List[String] = List(BMW, MERCEDES, AUDI, PORSCHE)
}
```

Zadatak 3.3: string interpolation

```
val valuesAndNames = List( (1, "One"), (2, "Two") )
//> valuesAndNames  : List[(Int, String)] = List((1,One), (2,Two))

for(valueAndName <- valuesAndNames)
  println(s"${valueAndName._1} ... ${valueAndName._2}")
                                                  //> 1 ... One
                                                  //| 2 ... Two

val namesAndWeights = List ( ("John", 76.57), ("Andy", 88.863) )
//> namesAndWeights  : List[(String, Double)] = List((John,76.57),
//| (Andy,88.863))
var i = 0
for(nameAndWeight <- namesAndWeights) {
  i = i+1
  println(f"$i. ${nameAndWeight._1} weights ${nameAndWeight._2}%.1f")
}                                                 //> 1. John weights 76.6
                                                  //| 2. Andy weights 88.9
```

Zadatak 3.4 – custom string interpolation

http://docs.scala-lang.org/overviews/core/string-interpolation.html

- Napisati funkciju za namensku obradu znakovnih literala

```
implicit class Upper(val sc: StringContext) {
  def up(args: Any*): String = {
    val strings = sc.parts.iterator
    val expressions = args.iterator
    var buf = new StringBuffer(strings.next.toUpperCase)
    while(strings.hasNext) {
      buf append expressions.next
      buf append strings.next.toUpperCase
    }
    buf.toString
  }
}

val weight1 = 76.57
  val weight2 = 88.863
  println(up"average weight: ${ (weight1 + weight2)/2 }")
  //> AVERAGE WEIGHT: 82.7165
```

Zadatak 3.5

- Napisati funkciju za nalaženje zbira elemenata kvadratne matrice koji se nalaze u gornjem trouglu

```
type Matrix[T] = Array[ Array[T] ]

def sumTri[T <% Double](m : Matrix[T]) = {
  val length = m.length
  var value = 0 : Double
  for(row <- 0 until length; col <- row until length)
    value = value + m(row)(col)
  value
}

val matrix = Array.tabulate(3,3)( (x,y) => x+y )
//> matrix  : Array[Array[Int]] = Array(  Array(0, 1, 2),
//|                                      Array(1, 2, 3),
//|                                      Array(2, 3, 4))

sumTri(matrix)                  //> res0: Double = 12.0
```

Naznaka kako treba tumačiti literal

Zadatak 3.6

- Napisati funkciju koja vrši zadatu numeričku obradu nad zadatim elementima zadate matrice brojeva.

```
def processMatrix[@specialized(Int, Double, Long, Float) T ]
  (m : Matrix[T])
  (ident : Double)
  (ind : Iterable[(Int, Int)])
  (op : (Double, Double) => Double) : T =
  {
    var res = ident;
    for(i <- ind)
      res = op(res, matrix(i._1)(i._2));

    res.asInstanceOf[T]
  }
```

Generiše specijalizovanu metodu i metodu sa brisanjem tipa

Zadatak 3.6

```
val indices = List( (1, 2), (0, 2) )
//> indices  : List[(Int, Int)] = List((1,2), (0,2))

def maxEl(x:Double, y:Double) = if(x > y) x else y
//> maxEl: (x: Double, y: Double)Double

processMatrix(matrix)(0)(indices)(maxEl)
//> res1: Int = 3
```

Zadatak 3.6

```
val processMatrixSpec = processMatrix(matrix)(0)(_)
//> processMatrixSpec  :
//| Iterable[(Int, Int)] => (((Double, Double) => Double) => Int)
//| = <function1>

def elems[T](matrix : Matrix[T]) = {
   for(i <- 0 until matrix.length;
       j <- 0 until matrix.length if((i+j)%2==0) )
      yield (i, j)
}
//> elems: [T](matrix: w04.ranges.Matrix[T])(implicit evidence$2:
//| T => Double)scala.collection.immutable.IndexedSeq[(Int, Int)]

val myElems = elems(matrix)
//> myElems  : scala.collection.immutable.IndexedSeq[(Int, Int)]
//| = Vector((0,0), (0,2), (1,1), (2,0), (2,2))

val findAll = processMatrixSpec(myElems)(maxEl)
//> findAll  : Int = 4
```

Zadatak 3.7

- Za spisak knjiga, dat u formi liste naslova i autora, napisati iskaze u vidu *for-comprehension* za nalaženje
  - svih knjiga čiji je jedan od autora zadatog imena
  - svih knjiga čiji naslovi sadrže zadatu znakovni niz
  - svih autora koji su napisali najmanje dve knjige iz spiska

```
case class Book(title: String, authors: List[String])
val books: List[Book] = List(
    Book("Structure and Interpretation of Computer Programs",
       List("Abelson, Harold", "Sussman, Gerald J.")),
    Book("Principles of Compiler Design",
       List("Aho, Alfred", "Ullman, Jeffrey")),
    Book("Programming in Modula-2",
       List("Wirth, Niklaus")),
    Book("Introduction to Functional Programming",
       List("Bird, Richard")),
    Book("The Java Language Specification",
       List("Gosling, James", "Joy, Bill", "Steele, Guy",
            "Bracha, Gilad")))
```

Zadatak 3.7

- Za spisak knjiga, dat u formi liste naslova i autora, napisati iskaze u vidu *for-comprehension* za nalaženje
  - svih knjiga čiji je jedan od autora zadatog imena

```
for (b <- books; a <- b.authors if a startsWith "Ullman")
       yield b.title
//> res0: List[String] = List(Principles of Compiler Design)
```

Zadatak 3.7

- Za spisak knjiga, dat u formi liste naslova i autora, napisati iskaze u vidu *for-comprehension* za nalaženje
  - svih knjiga čiji naslovi sadrže zadati znakovni niz

```
for (b <- books if (b.title indexOf "Program") >= 0)
       yield b.title
//> res1: List[String] = List(Structure and Interpretation of
//| Computer Programs, Programming in Modula-2, Introduction to
//| Functional Programming)
```

Zadatak 3.7

- Za spisak knjiga, dat u formi liste naslova i autora, napisati iskaze u vidu *for-comprehension* za nalaženje
  - svih autora koji su napisali najmanje dve knjige sa spiska

```
for (   b1 <- books;
                b2 <- books if b1 != b2;
                a1 <- b1.authors;
                a2 <- b2.authors if a1 == a2)
        yield a1
//> res2: List[String] = List()
```

```
for {   b1 <- books
                b2 <- books if b1 != b2
                a1 <- b1.authors
                a2 <- b2.authors if a1 == a2
        }
        yield a1
//> res3: List[String] = List()
```

Mana rešenja:
dupliraće imena autora.

Za vežbu: popraviti.

Zadatak 3.8

- Prekidanje i nastavljanje ciklusa bez naredbi `break` i `continue`

```
import scala.util.control.Breaks._

object breakandcontinue {
  breakable {
    for (i <- 1 to 10) {
       println(i)
       if (i > 4) break  // break out of the for loop
    }
  }                                                    //> 1
                                                       //| 2
                                                       //| 3
                                                       //| 4
                                                       //| 5
  ...
}
```

Zadatak 3.8

Kako izgleda klasa Breaks?

```
package scala
package util.control

class Breaks {

  private val breakException = new BreakControl

  def breakable(op: => Unit) {
    try {
      op
    } catch {
      case ex: BreakControl =>
        if (ex ne breakException) throw ex
    }
  }

  def break(): Nothing = { throw breakException }
}
```

Zadatak 3.8

```
object breakandcontinue {
 ...
 var sumOdd = 0
 for (i <- 0 until 10) {
   breakable {
     if (i % 2 == 0)
     {
       break
     }
     else
     {
       sumOdd += i
     }
   }
  }
  println("Sum of odd numbers in range 0 to 10: " + sumOdd)
  //> Sum of odd numbers in range 0 to 10: 25
}
```

Zadatak 3.8

```
import util.control._

  val Inner = new Breaks
  val Outer = new Breaks
  Outer.breakable {
   for (i <- 1 to 5) {
     Inner.breakable {
       for (j <- 'a' to 'e')
       {
          if (i == 1 && j == 'c') Inner.break
          else println(s"i: $i, j: $j")

          if (i == 2 && j == 'b') Outer.break
       }
     }
  }
}
```

```
//> i: 1, j: a
//| i: 1, j: b
//| i: 2, j: a
//| i: 2, j: b
```

Zadatak 3.9

- Napisati funkciju koja sintaksno i semantički odgovara `while` ciklusu

```
sveDok(uslov) { telo }
```

U jeziku Scala ovo se može rešiti kerifikovanom funkcijom

- kada f-ja ima 1 parametar, on može biti u ( ) ili { }

```
@tailrec
def sveDok(testCondition: => Boolean)(codeBlock: => Unit) {
  if (testCondition) {
      codeBlock
      sveDok(testCondition)(codeBlock)
  }
}
```