Funkcionalno programiranje

Vežbe

061 Kolekcije

Zadatak 1

- Ispisati elemente sekvencijalne strukture zajedno sa njihovim rednim brojevima.

```
val fruits = Array("apple", "banana", "orange")
//> fruits  : Array[String] = Array(apple, banana, orange)

for (i <- 0 until fruits.size) println(s"element $i is ${fruits(i)}")
                                                  //> element 0 is apple
                                                  //| element 1 is banana
                                                  //| element 2 is orange

for ((elem, count) <- fruits.zipWithIndex)
 println(s"element $count is $elem")              //> element 0 is apple
                                                  //| element 1 is banana
                                                  //| element 2 is orange
// zipWithIndex:
// Array( (apple, 0), (banana,1), (orange,2) )
```

Zadatak 1

```
for ((elem, count) <- fruits.view.zipWithIndex)
  println(s"element $count is $elem")
                                          //> element 0 is apple
                                          //| element 1 is banana
                                          //| element 2 is orange
```

```
for ((elem,count) <- fruits.zip(Stream from 1))
println(s"element $count is $elem")
                                          //> element 1 is apple
                                          //| element 2 is banana
                                          //| element 3 is orange
```

Zadatak 2

- Realizovati for/yield nad kolekcijom primenom metode map

Metoda map primenjuje zadatu funkciju nad svim elementima kolekcije i pravi novu kolekciju.

```
val gradovi = List("Beograd", "Cacak", "Pozega", "Novi Sad", "Nis")
//> gradovi  :
//  List[String] = List(Beograd, Cacak, Pozega, Novi Sad, Nis)

for(g <- gradovi if g.length() < 6) yield g.toUpperCase()
//> res0: List[String] = List(CACAK, NIS)


gradovi.map(x => if(x.length()<6) x.toUpperCase() )
//> res1: List[Any] = List((), CACAK, (), (), NIS)
```

Zadatak 2

- U opštem slučaju, samo `map` nije dovoljno
  - `map` mora da se primeni nad svakim elementom i da rezultujuće preslikavanje
  - ako postoji uslov koji nije ispunjen, prazna else grana tumači se kao `Unit`.

```
gradovi.map(x => if(x.length()<6) x.toUpperCase() )
//> res1: List[Any] = List((), CACAK, (), (), NIS)

gradovi.map(x => if(x.length()<6) x.toUpperCase() ).filter( _ != () )
//> res2: List[Any] = List(CACAK, NIS)

gradovi.filter( _.length()<6 ).map(_.toUpperCase())
//> res3: List[String] = List(CACAK, NIS)


gradovi.view.filter(_.length()<6).map(_.toUpperCase()).force
//> res4: Seq[String] = List(CACAK, NIS)
```

Zadatak 3

- Data je mapa koja preslikava nazive regija neke države u nazive vrsta stabala koje u njima rastu.
  Naći jedinstvene nazive vrsta stabala koje rastu u obuhvaćenim regijama.

```
/*
var r1 = List("Omorika", "Breza", "Javor", "Hrast")

var r2 = List("Jasen", "Bukva", "Kesten", "Bor")

var r3 = List("Lipa", "Omorika", "Topola", "Jablan", "Bukva")
*/

var vrste = Map("R1" -> r1, "R2" -> r2, "R3" -> r3)
```

Zadatak 3

- Zadata je kolekcija lista
- Potrebno je:
  - napraviti jedinstvenu listu (kolekciju)
  - izbaciti duplikate

```
vrste.flatten
```

No implicit view available from (String, List[String])
⇒ scala.collection.GenTraversableOnce[B].

```
vrste.flatten( (x) => x._2.iterator )
```

```
//> res8: scala.collection.immutable.Iterable[String] =
//  List(Omorika, Breza, Javor, Hrast, Jasen, Bukva, Kesten, Bor,
//       Lipa, Omorika, Topola, Jablan, Bukva)
```

Zadatak 3

```
vrste.flatten( (x) => x._2.iterator ).toList.distinct
//> res8: List[String] =
//  List(Omorika, Breza, Javor, Hrast, Jasen, Bukva, Kesten
//       Bor, Lipa, Topola, Jablan)

vrste.values.flatten.toList.distinct
//> res9: List[String] =
// List(Omorika, Breza, Javor, Hrast, Jasen, Bukva, Kesten
//      Bor, Lipa, Topola, Jablan)

vrste.values.flatten.toStream.distinct.force
//> res10: scala.collection.immutable.Stream[String] =
//  Stream(Omorika, Breza, Javor, Hrast, Jasen, Bukva, Kesten, Bor,
//         Lipa, Topola, Jablan)


vrste.values.toStream.flatten.distinct.force
//> res11: scala.collection.immutable.Stream[String] =
//  Stream(Omorika, Breza, Javor, Hrast, Jasen, Bukva, Kesten, Bor,
//         Lipa, Topola, Jablan)
```

Zadatak 3

```
vrste.values.flatten.toStream.distinct.force
//> res10: scala.collection.immutable.Stream[String] =
//  Stream(Omorika, Breza, Javor, Hrast, Jasen, Bukva, Kesten, Bor,
//         Lipa, Topola, Jablan)


vrste.values.toStream.flatten.distinct.force
//> res11: scala.collection.immutable.Stream[String] =
//  Stream(Omorika, Breza, Javor, Hrast, Jasen, Bukva, Kesten, Bor,
//         Lipa, Topola, Jablan)


vrste.values.view.flatten.toList.distinct
//> res12: List[String] =
//  List(Omorika, Breza, Javor, Hrast, Jasen, Bukva, Kesten,
//       Bor, Lipa, Topola, Jablan)
```

Zadatak 4

- Data je lista reči koja sadrži cele brojeve zapisane kao tekst i u decimalnoj cifarskoj reprezentaciji. Naći sumu brojeva zapisanih u decimalnoj cifarskoj reprezentaciji.

```
val brojevi = List("1", "2", "tri", "4", "sto sedamdeset pet")
//> brojevi  : List[String] = List(1, 2, tri, 4, sto sedamdeset pet)

def strToInt(s: String) : Option[Int] = {
  try { Some(Integer.parseInt(s.trim)) }
  catch { case e: NumberFormatException => None }
}
//> strToInt: (s: String)Option[Int]
```

Zadatak 4

```
brojevi.map(strToInt)
//> res0: List[Option[Int]] =
//        List(Some(1), Some(2), None, Some(4), None)

for(e <- brojevi.map(strToInt) if e != None ) yield e.get
//> res1: List[Int] = List(1, 2, 4)

brojevi.map(strToInt).flatten.sum          //> res2: Int = 7

brojevi.flatMap(strToInt).sum              //> res3: Int = 7
```

Zadatak 5

- Implementirati algoritam sortiranja metodom brojanja upotrebom funkcija višeg reda nad kolekcijama.

```
def countingSort(l : List[Int]) : List[Int] = {
  var maxVal = l.reduceLeft( _ max _ )
  var C = Array.fill(maxVal){0}

  for(e <- l) C(e-1) += 1
  C = C.scanLeft(0)(_ + _)

  var B = Array.ofDim[Int](l.length)

  for( e <- l.reverseIterator) {
    B(C(e)-1) = e
    C(e) -= 1
  }

  B.toList
}              //> countingSort: (l: List[Int])List[Int]
```

Zadatak 5

```
val niz = List(5, 12, 2, 8, 4, 16, 32, 12, 44, 5, 8, 6, 7, 5)
//> niz  : List[Int] =
//         List(5, 12, 2, 8, 4, 16, 32, 12, 44, 5, 8, 6, 7, 5)

countingSort(niz)
//| res0: List[Int] =
//        List(2, 4, 5, 5, 5, 6, 7, 8, 8, 12, 12, 16, 32, 44)
```