Scala goodies for CompProg

Daniel M. German

January 11, 2017

Contents

1 Conversions

Iterator	toArray	
	toList	
	toSet	
	mkString	Can include a separator
string	toInt	
	BigInt(st)	
	toFloat	

2 Functions

abs	Math.abs

3 Input

3.1 Read one line

```
val tests = scala.io.StdIn.readLine().toInt
```

3.2 Read all lines until certain condition

It is an iterator. Optionally convert to Array or List depending on processing

- takeWhile can specify any condition

```
(* until end of file *)
val lines = Iterator.continually(scala.io.StdIn.readLine()).takeWhile(x => x != null).toLis
```

3.3 Read n lines

```
val lines = Iterator.continually(scala.io.StdIn.readLine()).take(r).toArray
```

4 Output

4.1 format output

Scanf type

```
println("%.1f".format(x._3*ratio))
```

5 Algorithms

5.1 Sort

descending order

```
sortWith(_ < _)
```

5.2 Foreach in a range

Equivalent to the usually for loop

```
1 to x foreach { i =>
}
```

5.3 permutations

List(1,2,3).permutations.

5.4 combinations

val words = List(1,2,2).toSet.subsets.map(_.toList).toList

6 Mutable

```
import scala.collection.mutable.{HashMap, Map}
```

7 Maps

7.1 getOrElse

```
// states is a map
states.getOrElse("FOO", "No such state")
```

7.2 Declare them

```
var save:Map[(Int,List[Int]),Set[Int]] = Map()
```

8 priority queue

```
import scala.collection.mutable.PriorityQueue

case class Donut(name: String, price: Double)

def donutOrder(d: Donut) = d.price

val priorityQueue1: PriorityQueue[Donut] = PriorityQueue(
 Donut("Plain Donut", 1.50),
 Donut("Strawberry Donut", 2.0),
 Donut("Chocolate Donut", 2.50))(Ordering.by(donutOrder))
println(s"Elements of priorityQueue1 = $priorityQueue1")

priorityQueue1.enqueue(Donut("Vanilla Donut", 1.0))
```

9 Memoize

- Should it be a var? I **think** so

```
var save:Map[(Int,List[Int]),Set[Int]] = Map()

def my_partitions(size: Int, part: List[Int]): Set[Int] = {
   if (save.contains((size,part))) {
      save((size,part))
   } else {
    (* whatever *)
      val result = ...
      save += ((size, part)-> result)
      result
   }
}
```

9.1 check if element include

```
if (penalty.contains(problem))
  penalty(problem) += 1
else
  penalty += (problem -> 1)
}
```

9.2 Mutable

```
var penalty : scala.collection.mutable.Map[String,Int] = scala.collection.mutable.Map()
```

9.3 with default value

non mutable

```
val m = Map[Int, Int]().withDefaultValue(0)
```

9.4 processing the "values"

```
val totalTime = good.map(_._2).sum
```

10 Sets

10.1 Ops

x1 union x2	
x1 intersect x2	
x1 diff x2	
x1.empty	returns an empty set of x1 type
xs contains x	
xs subsetOf yx	is xs a subset of yx

10.2 Mutable

xs += elem	
xs -= elem	
xs ++= otherSet	
xs retain p	retain elements that satisfy p
xs.clear	
xs clone	create a new set
xs.update(x,b)	if b true, add to x, otherwise remove x

11 Strings

11.1 Prefix of a string

st toLowerCase
st contains st2
st startsWith st2
st equalsIgnoreCase st2

12 Dates

12.1 get parts of the date

```
val today = Calendar.getInstance().getTime()
val now = Calendar.getInstance()
now.get(Calendar.MINUTE)
now.get(Calendar.HOUR)
now.get(Calendar.MONTH)
now.get(Calendar.DAY_OF_WEEK)
now.get(Calendar.DAY_OF_MONTH)
now.get(Calendar.DAY_OF_WEEK_IN_MONTH)
now.get(Calendar.DAY_OF_YEAR)
now.get(Calendar.WEEK_OF_YEAR)
```

12.2 parse a date

```
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

DateFormat df = new SimpleDateFormat("dd/MM/yyyy");
try {
  Date today = df.parse("19/08/2011");
  System.out.println("Today = " + df.format(today));
}
catch (ParseException e)
{
e.printStackTrace();
}
```

12.3 convert string to time

```
// times contains collection of times 12:00 p.m. or 11:30 a.m.
      val times = lines.map( l => {
        val pm = 60 *12 * 60;
        val x = l.split("[: ]")
        val h = x(0).toInt

        (l,
          (if (h == 12) 0 else h) * 60 +
          x(1).toInt +
          pm * (if (x(2) == "a.m.") 0 else 1))

      })
```

13 generics

13.1 Breaking a loop

```
var last = 0
(x to 1 by -1).iterator.takeWhile(i => !condition).foreach(i => {
   assert(!flag)
   last = i
})
if (last != 1) {
   // we "broke" the loop
}
```

14 Tuples

14.1 process tuples

need to use case expression in lambda

```
val totalPenalty = penalty.map({case (problem,count) =>
})
```

15 Regular expressions

15.1 Simple substitutioj

```
val regex = "[0-9][0-9]?".r
val z = regex.replaceAllIn(x, x => rename(x.toString.toInt))
```

15.2 Complex substitution

```
val cons = "([^aeiou]+)([^ ]+)".r
val z = cons.replaceAllIn(line, _ match { case cons(pre, rest) => "[" + rest + pre + "]"})
```

15.3 matching

```
val cons = "(^[^aeiou]+)(.*)".r
cons.findFirstIn(w) match {
  case Some( cons(pre,rest)) => rest + pre + "ay"
  case None => {
    w + "yay"
  }
}
```

16 Chars

toUpper
toLower