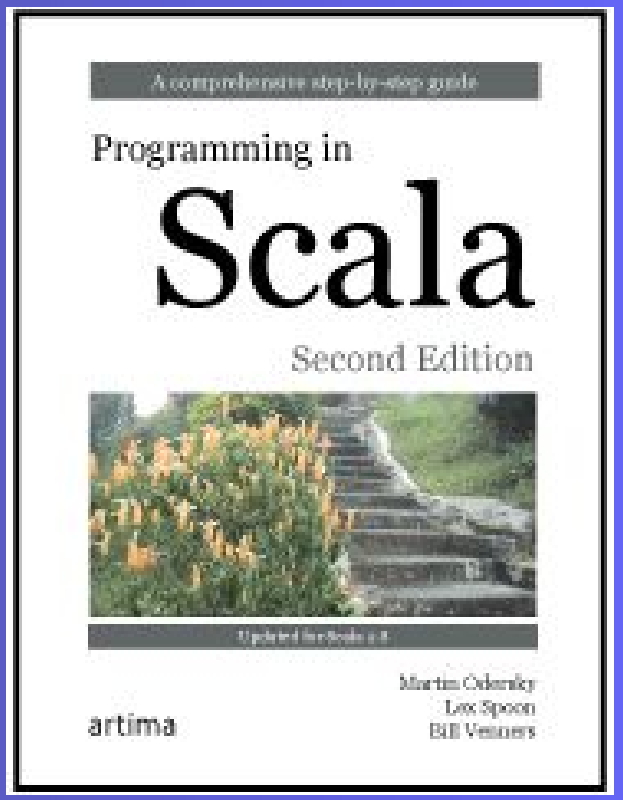
Stairway to Scala - Flight 14

Working with Lists

Bill Venners
Dick Wall



escalatesoft.com

Copyright (c) 2010-2014 Escalate Software, LLC.
All Rights Reserved.

Flight 14 goal

Get to know Scala's main functional data structure: the list.*

* Except perhaps for Vector.

List literals

```
val fruit = List("apples", "oranges", "pears")
val nums = List(1, 2, 3, 4)
val diag3 =
  List(
    List(1, 0, 0),
    List(0, 1, 0),
    List(0, 0, 1)
  )
val empty = List()
```

Lists are homogeneous and covariant

```
val fruit: List[String] = List("apples", "oranges", "pears")
val nums: List[Int] = List(1, 2, 3, 4)
val diag3: List[List[Int]] =
  List(
    List(1, 0, 0),
    List(0, 1, 0),
    List(0, 0, 1)
  )
val empty: List[Nothing] = List()

// List() is also of type List[String]!
val xs: List[String] = List()
```

Constructing lists

```
x :: xs // element x, rest of list xs

val fruit = "apples" :: ("oranges" :: ("pears" :: Nil))
val nums  = 1 :: (2 :: (3 :: (4 :: Nil)))
val diag3 = (1 :: (0 :: (0 :: Nil))) ::
            (0 :: (1 :: (0 :: Nil))) ::
            (0 :: (0 :: (1 :: Nil))) :: Nil
val empty = Nil

val moreNums = 1 :: 2 :: 3 :: 4 :: Nil
```

Basic operations on lists

head - returns the first element
tail - returns a list of all but the first element
isEmpty - returns true if the list is empty

Example: insertion sort

To sort a non-empty list x :: xs, sort the remainder xs and insert the first element x at the correct position in the result. Sorting an empty list yields the empty list.

Insertion sort with head, tail, isEmpty

```
def isort(xs: List[Int]): List[Int] =
  if (xs.isEmpty) Nil
  else insert(xs.head, isort(xs.tail))

def insert(x: Int, xs: List[Int]): List[Int] =
  if (xs.isEmpty || x <= xs.head) x :: xs
  else xs.head :: insert(x, xs.tail)
```

List patterns

```
scala> val List(a, b, c) = fruit
a: String = apples
b: String = oranges
c: String = pears

scala> val a :: b :: rest = fruit
a: String = apples
b: String = oranges
rest: List[String] = List(pears)
```

Insertion sort with pattern matching

```
def isort(xs: List[Int]): List[Int] = xs match {
  case Nil                 => Nil
  case head :: tail    => insert(head, isort(tail))
}

def insert(x: Int, xs: List[Int]): List[Int] = xs match {
  case Nil                          => x :: Nil
  case hd :: _ if x <= hd     => x :: xs
  case head :: tail              => head :: insert(x, tail)
}
```

First-order methods on List

Concatenate lists with :::

```
scala> List(1, 2) ::: List(3, 4, 5)
res0: List[Int] = List(1, 2, 3, 4, 5)

scala> List() ::: List(1, 2, 3)
res1: List[Int] = List(1, 2, 3)

scala> List(1, 2, 3) ::: List(4)
res2: List[Int] = List(1, 2, 3, 4)
```

How is xs ::: ys ::: zs interpreted?

length, init, last

length is expensive

```
scala> List(1, 2, 3).length // linear time
res3: Int = 3
```

So is init and last

```
scala> val abcde = List('a', 'b', 'c', 'd', 'e')
abcde: List[Char] = List(a, b, c, d, e)

scala> abcde.last // linear time
res4: Char = e

scala> abcde.init // linear time
res5: List[Char] = List(a, b, c, d)
```

reverse, take, drop

```
scala> abcde.reverse
res6: List[Char] = List(e, d, c, b, a)

scala> abcde
res7: List[Char] = List(a, b, c, d, e)

scala> abcde take 2
res8: List[Char] = List(a, b)

scala> abcde drop 2
res9: List[Char] = List(c, d, e)
```

apply, indices

```
 scala> abcde apply 2 // linear time
res11: Char = c

scala> abcde(2)      // linear time
res12: Char = c

scala> abcde.indices
res13: scala.collection.immutable.Range =
    Range(0, 1, 2, 3, 4)
```

zip, zipWithIndex, unzip

```
scala> abcde.indices zip abcde
res17: scala.collection.immutable.IndexedSeq[(Int, Char)] =
    IndexedSeq((0,a), (1,b), (2,c), (3,d), (4,e))

scala> val zipped = abcde zip List(1, 2, 3)
zipped: List[(Char, Int)] = List((a,1), (b,2), (c,3))

scala> abcde.zipWithIndex
res18: List[(Char, Int)] = List((a,0), (b,1), (c,2), (d,3),
    (e,4))

scala> zipped.unzip
res19: (List[Char], List[Int]) =
    (List(a, b, c),List(1, 2, 3))
```

toString, mkString

```
scala> abcde.toString
res20: String = List(a, b, c, d, e)
scala> abcde mkString ("[", ",", "]")
res21: String = [a,b,c,d,e]

scala> abcde mkString ""
res22: String = abcde

scala> abcde.mkString
res23: String = abcde

scala> abcde mkString ("List(", ", ", ")")
res24: String = List(a, b, c, d, e)
```

Higher-order functions: map

```
scala> List(1, 2, 3) map (_ + 1)
res32: List[Int] = List(2, 3, 4)

scala> val words = List("the", "quick", "brown", "fox")
words: List[java.lang.String] = List(the, quick, brown, fox)

scala> words map (_.length)
res33: List[Int] = List(3, 5, 5, 3)

scala> words map (_.toList.reverse.mkString)
res34: List[String] = List(eht, kciuq, nworb, xof)
```

flatMap

```
scala> words map (_.toList)
res35: List[List[Char]] = List(List(t, h, e), List(q, u, i,
    c, k), List(b, r, o, w, n), List(f, o, x))

scala> words flatMap (_.toList)
res36: List[Char] = List(t, h, e, q, u, i, c, k, b, r, o, w,
    n, f, o, x)
```

foreach

```
scala> var sum = 0
sum: Int = 0

scala> List(1, 2, 3, 4, 5) foreach (sum += _)

scala> sum
res39: Int = 15
```

filter, partition, exists

```
scala> List(1, 2, 3, 4, 5) filter (_ % 2 == 0)
res40: List[Int] = List(2, 4)

scala> words filter (_.length == 3)
res41: List[java.lang.String] = List(the, fox)

scala> List(1, 2, 3, 4, 5) partition (_ % 2 == 0)
res42: (List[Int], List[Int]) = (List(2, 4),List(1, 3, 5))

scala> List(1, 2, 3, 4, 5) exists (_ == 3)
res43: Boolean = true
```

find, takeWhile, dropWhile

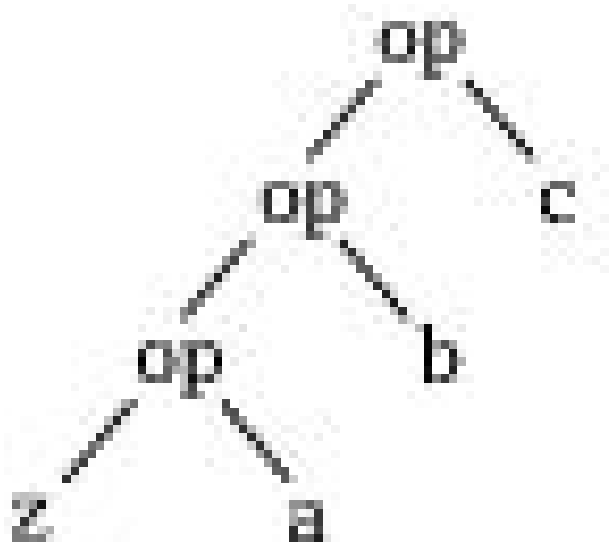
```
scala>  List(1, 2, 3, 4, 5) find (_ % 2 == 0)
res43: Option[Int] = Some(2)

scala>  List(1, 2, 3, 4, 5) find (_ <= 0)
res44: Option[Int] = None

scala> List(1, 2, 3, -4, 5) takeWhile (_ > 0)
res45: List[Int] = List(1, 2, 3)

scala> words dropWhile (_ startsWith "t")
res46: List[java.lang.String] = List(quick, brown, fox)
```

Fold left concept

sum(List(a, b, c)) *equals* 0 + a + b + c

product(List(a, b, c)) *equals* 1 * a * b * c

op(op(op(z, a), b), c)

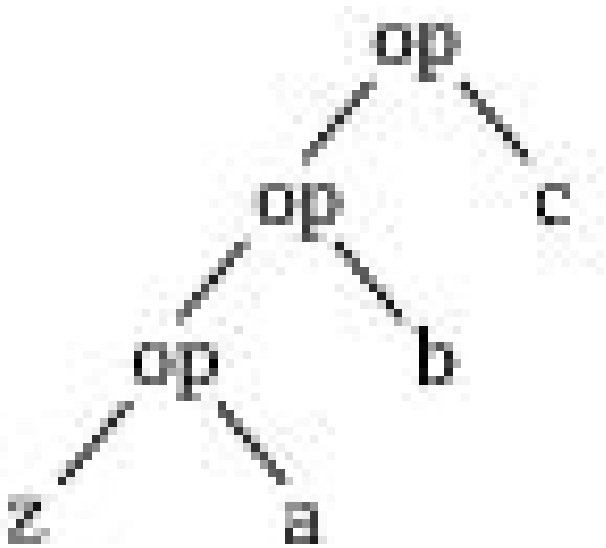


Fold left code

op(op(op(z, a), b), c)

```
scala> def sum(xs: List[Int]): Int = (xs foldLeft(0))(_ + _)
sum: (xs: List[Int])Int
```

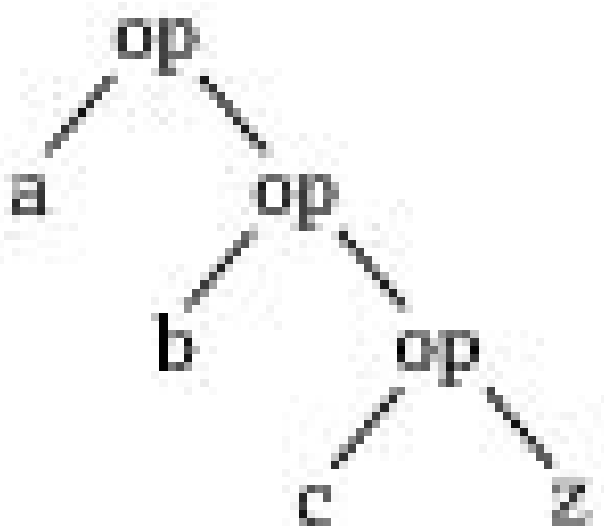
```
scala> def sum(xs: List[Int]): Int = (0 /: xs) (_ + _)
sum: (xs: List[Int])Int
```



def /:[B](z: B)(op: (B, A) => B): B

Fold right

(List(a, b, c) :\ z)(op) *equals* op(a, op(b op(c, z)))



sortWith

```
scala> List(1, -3, 4, 2, 6) sortWith (_ < _)
res4: List[Int] = List(-3, 1, 2, 4, 6)

scala> case class Person(first: String, last: String)
defined class Person

scala> val ps = List(Person("Harry", "Potter"), Person("Hermione", "Granger"), Person
("Ronald", "Weasley"))
ps: List[Person] = List(Person(Harry,Potter), Person(Hermione,Granger), Person
(Ronald,Weasley))

scala> ps.sortBy(_.first)
res0: List[Person] = List(Person(Harry,Potter), Person(Hermione,Granger), Person
(Ronald,Weasley))

scala> ps.sortBy(_.last)
res1: List[Person] = List(Person(Hermione,Granger), Person(Harry,Potter), Person
(Ronald,Weasley))
```

Exercises for Flight 14