Updated: 2021-10-18

3.10
Lazy Lists

Northeastern University

Non-strict collections

- There are three special types of collection which are *non-strict* (lazy to you and me):
  - *Iterator*: evaluates elements as needed but they cannot be revisited;
  - *LazyList*: evaluates element as needed and they can be revisited [*LazyList* was formerly *Stream* in 2.12 and earlier);
  - *SeqView* (actually, there are other types of view, too): essentially just “decorates” a collection with a transformation function
- Each of these has different behavior but what is generally common is that an element in the sequence will not be evaluated if you never actually need it.

LazyLists are lazy lists

- We've briefly mentioned *LazyList* before:
  - Like a *List*, a *LazyList* has a *head* and a *tail* but…
  - …Unlike in a *List*, the *tail (and head)* of a *LazyList* are call-by-name parameters.

```
trait LazyList[A] {
  def head: A
  def tail: LazyList[A]
}
case class Cons[A](head:=>A, tail:=>LazyList[A]) extends LazyList[A]
case object empty extends LazyList[Nothing] {
  def head: throw NoSuchElementException("head of empty lazy list")
  def tail: throw UnsupportedOperationException("tail…lazy list")
}
```

  - A *LazyList* is ideal for memoizing something.

Working with LazyLists

- Ways to create a *LazyList*:
    - `import LazyList._`
    - `1 #:: 2 #:: empty`
    - `cons(1, cons(2, empty))`
    - `from(1)`
    - `continually(9)`
    - `range(1, 20, 3)`
- A *LazyList* has no definite length.
    - In order to turn an (infinite) *LazyList* into a (finite) *List,* you need to do two things: force a definite size, and convert it to a *List:*
    - `From 1 take 10 to List`

What do you think this function does?

Note: recursive even though ***f*** is a val.

A bit like *foldLeft* but retains shape

```
val f: LazyList[Long] = 0L #:: f.scanLeft(1L)(_ + _)
```

```
val g: LazyList[Long] = 0L #:: 1L #:: g.zip(g tail).map (n =>
n._1 + n._2)
```

Should be a bit easier to understand.

Fibonacci

```
scala> val f: LazyList[BigInt] = BigInt(0) #:: f.scanLeft(BigInt(1))(_ + _)
f: LazyList[BigInt] = LazyList(0, ?)

scala> LazyList.from(0) zip f take 100 foreach println
(0,0)
(1,1)
(2,1)
(3,2)
(4,3)
(5,5)
(6,8)
(7,13)
(8,21)
(9,34)
etc. etc.
(99,218922995834555169026)
```

Changes in 2.13

- *Stream* has been deprecated in favor of *LazyList*.
  - In *LazyList,* both *tail* <u>and</u> *head* are lazily evaluated.
- There's no *Traversable* any more:
  - They decided that *Iterable* and *Traversable* were so similar that it wasn't worth maintaining a distinction.
- *StringOps* has methods *toIntOption*, etc.
- Converting collections is now done as follows:
  - `xs.to(List)`, or
  - `xs to List`