PLRG

CS320 - Scala Seminar

Working with Scala Collections

Jaemin Hong

hjm0901@gmail.com

September 12, 2018

PLRG

Why Functional?

- Good to reason and to verify programs.
- Easy to parallelize.
- Make high-level abstraction available.
- Can model and treat complex structures.

Immutability

Higher-order functions

ADT & Pattern Matching

Review

```
trait List

case object Nil extends List

case class Cons(head: Int, tail: List) extends List
```

PLRG

Review

```
def inc1(l: List): List =

  list_map(l, _ + 1)


def square(l: List): List =

  list_map(l, h => h * h)
```

PLRG

Review

```
def odd(l: List): List =

  list_filter(l, _ % 2 != 0)


def positive(l: List): List =

  list_filter(l, _ > 0)
```

Review

```
def sum(l: List): Int =

  list_foldRight(l, 0, _ + _)


def product(l: List): Int =

  list_foldRight(l, 1, _ * _)
```

PLRG

Review

```
def sum(l: List): Int =

  list_foldLeft(l, 0, _ + _)
```

```
def product(l: List): Int =

  list_foldLeft(l, 1, _ * _)
```

List in Scala Standard Library

PLRG

```
Nil
```

```
Cons(0, Nil)
```

```
Cons(0, Cons(1, Nil))
```

```
List(0, 1, 2)
```

```
Nil
```

```
0 :: Nil
```

```
0 :: 1 :: Nil
```

```
List(0, 1, 2)
```

List in Scala Standard Library

PLRG

```
List(0, 1, 2) match {
  case Nil => "foo"
  case Cons(h, t) => “bar"
}
```

```
List(0, 1, 2) match {
  case Nil => "foo"
  case h :: t => "bar"
}
```

List in Scala Standard Library

PLRG

```
List(): List

List(0): List

List(0, 1): List

List(0, 1, 2): List
```

```
List(): List[Nothing]

List(true): List[Boolean]

List("a", "b"): List[String]

List(0.0, 0.1, 0.2): List[Double]
```

List in Scala Standard Library

PLRG

```
def inc1(l: List): List =
  l match {
    case Nil => Nil
    case Cons(h, t) =>
      Cons(h + 1, t)
  }
```

```
def inc1(l: List[Int]): List[Int] =
  l match {
    case Nil => Nil
    case h :: t =>
      (h + 1) :: t
  }
```

List in Scala Standard Library

PLRG

```
list_map(
  List(1, 2, 3),
  _ + 1
)
```

```
List(1, 2, 3).map(_ + 1)
```

```
List(1, 2, 3).map(_ == 1)
```

List in Scala Standard Library

PLRG

```
list_map(
  List(1, 2, 3),
  _
)
```

```
List(1, 2, 3).map(println(_))
```

```
List(1, 2, 3).foreach(
  println(_)
)
```

List in Scala Standard Library

PLRG

```
list_map(
  List(1, 2, 3),
  _
)
```

```
List(1, 2, 3).flatMap(
  List(_)
)
```

```
List(1, 2, 3).flatMap(
  1 to _
)
```

List in Scala Standard Library

PLRG

```
list_filter(
  List(1, 2, 3),
  _ % 2 == 1
)
```

```
List(1, 2, 3)
  .filter(_ % 2 != 0)
```

```
List("", "a", "ab")
  .filter(_.length == 1)
```

List in Scala Standard Library

PLRG

```
list_foldRight(
  List(1, 2, 3),
  0,
  _ + _
)
```

```
List(1, 2, 3)
  .foldRight(0)(_ + _)
```

```
(List(1, 2, 3) :\ 0)(_ + _)
```

List in Scala Standard Library

PLRG

```
list_foldRight_list(
  List(1, 2, 3),
  List(4),
  Cons(_, _)
)
```

```
List(1, 2, 3)
  .foldRight(List(4))(_ :: _)
```

```
(List(1, 2, 3) :\ List(4))
  (_ :: _)
```

List in Scala Standard Library

PLRG

```
list_foldLeft(
  List(1, 2, 3),
  1,
  _ * _
)
```

```
List(1, 2, 3)
  .foldLeft(1)(_ * _)
```

```
(1 /: List(1, 2, 3))(_ * _)
```

List in Scala Standard Library

PLRG

```
list_foldLeft_list(
  List(1, 2, 3),
  List(),
  (t, h) => Cons(h, t)
)
```

```
List(1, 2, 3)
  .foldLeft
    (List[Int]())
    ((t, h) => h :: t)
```

```
(List[Int]() /: List(1, 2, 3))
  ((t, h) => h :: t)
```

List in Scala Standard Library

PLRG

```
addBack(List(1, 2, 3), 4)
```

```
List(1, 2, 3) :+ 4
```

```
append(
  List(1, 2, 3),
  List(4, 5, 6)
)
```

```
List(1, 2, 3) ++ List(4, 5, 6)
```

List in Scala Standard Library

PLRG

```
length(List(1, 2, 3))
```

```
List(1, 2, 3).length
```

```
reverse(List(1, 2, 3))
```

```
List(1, 2, 3).reverse
```

List in Scala Standard Library

PLRG

www.scala-lang.org/api/current/scala/collection/immutable/List.html

PLRG

Review

```
trait Option

case object None extends Option

case class Some(n: Int) extends Option
```

PLRG

Review

```
def list_getOrElse(l: List, i: Int, default: Int): Int =
  option_getOrElse(
     list_getOption(l, i),
     default
  )
```

PLRG

Review

```
def getSquare(l: List, i: Int): Option =

  option_map(

    list_getOption(l, i),

    n => n * n

  )
```

PLRG

Review

```
def getAndDiv100(l: List, i: Int): Option =
  option_flatMap(
    list_getOption(l, i),
    div100Safe
  )
```

Option in Scala Standard Library

PLRG

```
None
```

```
Some(0)
```

```
Some(1)
```

```
None
```

```
Some(0)
```

```
Some(1)
```

Option in Scala Standard Library

```
Some(0) match {
  case None => "foo"
  case Some(n) => "bar"
}
```

```
Some(0) match {
  case None => "foo"
  case Some(n) => "bar"
}
```

Option in Scala Standard Library

```
None: Option

Some(0): Option

Some(1): Option

Some(2): Option
```

```
None: Option[Nothing]

Some(true): Option[Boolean]

Some("a"): Option[String]

Some(1.0): Option[Double]
```

Option in Scala Standard Library

```
option_getOrElse(
  Some(0),
  1
)
```

```
Some(0).getOrElse(1)
```

```
Some("foo").getOrElse("bar")
```

Option in Scala Standard Library

```
option_map(
  Some(0),
  n => n * n
)
```

```
Some(0).map(n => n * n)
```

```
Some(0).map(_.toString)
```

Option in Scala Standard Library

```
option_flatMap(
  Some(0),
  div100Safe
)
```

```
Some(0).flatMap(div100Safe)
```

List in Scala Standard Library

PLRG

```
list_get(List(1, 2, 3), 0)
```

```
List(1, 2, 3)(0)
```

```
list_getOption(
  List(1, 2, 3), 0)
```

```
List(1, 2, 3).lift(0)
```

```
lift(
  list_getOption(
    List(1, 2, 3), _))(0)
```

List in Scala Standard Library

PLRG

```
list_filter(
  list_map(
    List(0, 1, 2),
    n => option_getOrElse(
      div100Safe(n), 99)
  ), _ != 99)
```

```
List(0, 1, 2).flatMap(
  div100Safe
)
```

Option in Scala Standard Library

PLRG

www.scala-lang.org/api/current/scala/collection/immutable/Option.html

For Loop in Scala

```
for (n <- List(1, 2, 3))
  println(n)


List(1, 2, 3).foreach(n =>
  println(n))
```

For Loop in Scala

```
for (n <- List(1, 2, 3);
     m <- 1 to n)
  println(m)
```

```
List(1, 2, 3).foreach(n =>
  (1 to n).foreach(m =>
    println(m)))
```

For Loop in Scala

```
for (n <- List(1, 2, 3) if n % 2 != 0)
  println(n)


List(1, 2, 3).filter(n => n % 2 != 0).foreach(n =>
    println(n))
```

For Loop in Scala

```
for (n <- List(1, 2, 3) if n % 2 != 0;
     m <- 1 to n)
  println(m)
```

```
List(1, 2, 3).filter(n => n % 2 != 0).foreach(n =>
  (1 to n).foreach(m =>
    println(m)))
```

For Loop in Scala

```
for (n <- List(1, 2, 3))
  yield n * n


List(1, 2, 3).map(n =>
  n * n)
```

For Loop in Scala

```
for (n <- List(1, 2, 3);
     m <- 1 to n)
  yield m * m
```

```
List(1, 2, 3).flatMap(n =>
  (1 to n).map(m =>
    m * m))
```

For Loop in Scala

```
for (n <- List(1, 2, 3) if n % 2 != 0;
      m <- 1 to n)
  yield m * m
```

```
List(1, 2, 3).filter(n => n % 2 != 0).flatMap(n =>
  (1 to n).map(m =>
    m * m))
```

For Loop in Scala

```
def f(x: Int): Option[Int] = ...

def g(x0: Int, x1: Int, x2: Int, x3: Int): Int = ...
```

Apply `x` to four integers. Then, if it succeeds, apply `g`.

For Loop in Scala

```
(f(n0), f(n1), f(n2), f(n3)) match {
  case (Some(m0), Some(m1), Some(m2), Some(m3)) =>
    Some(g(m0, m1, m2, m3))
  case _ =>
    None
}
```

For Loop in Scala

```
f(n0) match {
  case Some(m0) => f(n1) match {
    case Some(m1) => f(n2) match {
      case Some(m2) => f(n3) match {
        case Some(m3) => Some(g(m0, m1, m2, m3))
        case None => None } case None => None }
    case None => None } case _ => None }
```

For Loop in Scala

```
f(n0).flatMap(m0 =>

  f(n1).flatMap(m1 =>

    f(n2).flatMap(m2 =>

      f(n3).map(m3 => g(m0, m1, m2, m3)))))
```

For Loop in Scala

```
for (
  m0 <- f(n0);
  m1 <- f(n1);
  m2 <- f(n2);
  m3 <- f(n3)
) yield m0 * m1 * m2 * m3
```

Example: KAIST Bus Chatbot

```
$ scala Chatbot.scala "when OLEV come to E11 and when downtown bus
come to time world??"

  No more OLEV at E11 today.

  Weolpyeong Shuttle arrives at Galleria at 17:25.

$ scala Chatbot.scala "Electric car arrive at cafeteria at what
time?"

  OLEV arrives at Cafeteria at 10:15, 10:25, 10:45...

$ scala Chatbot.scala "When campus shuttle go to gungdong?"

  Please say stops of OLEV properly: Cafeteria, Sports Complex,
E11, Medical Science Center, KAIST Clinic, South Gate, Duck Pond,
Main Hall
```

Example: KAIST Bus Chatbot

PLRG

```
def query(msg: String, day: Int, hour: Int, min: Int):
String
```

```
def queryBus(msg: String, day: Int, hour: Int, min: Int,
bus: Bus): Option[String]
```

```
def queryStop(msg: String, day: Int, hour: Int, min: Int,
bus: Bus, stop: Stop): Option[String]
```

PLRG

Summary

- Scala offers useful collection library, which includes List and Option with many helper methods.
- `for` statement of Scala is transformed into code using `foreach`, `filter`, `map`, `flatMap` at the compile-time.
    - `for` is functional - completely immutable - and can yield value
- We can implement real-world applications in functional style.
- See you again after the midterm!