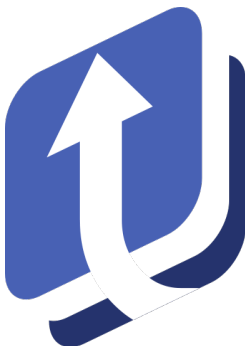


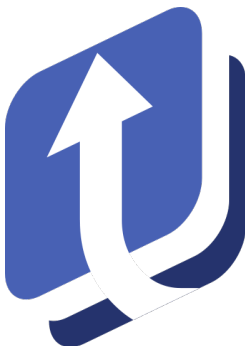
Scala tips and tricks

David Pollak
San Francisco Java Users Group
January 20th, 2009



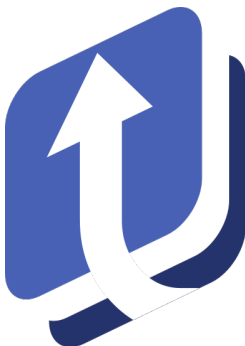
David Pollak

- Not strict, but pretty lazy
- Lead developer for *Lift* web framework
- Scala since November 2006, Ruby/Rails, Java/J2EE
- Spreadsheet junky (writing more than using)



Scala in the Wild

- Twitter loves Scala
- Scala powers community apps at SAP
- Scala and Lift run Buy a Feature @ Enthiosys
- Scala powers travel apps
- Scala runs time billing apps
- Scala courses at Stanford, San Jose State, EPFL, etc.



At the command line

```
File Edit View Terminal Tabs Help
dpp@horse:~$ scala
Welcome to Scala version 2.7.1.final (Java HotSpot(TM) 64-Bit Server VM, Java 1.6.0_07).
Type in expressions to have them evaluated.
Type :help for more information.

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

scala> val k = j.map(_ * 4)
k: List[Int] = List(4, 8, 12)

scala> val m = 99 :: j
m: List[Int] = List(99, 1, 2, 3)

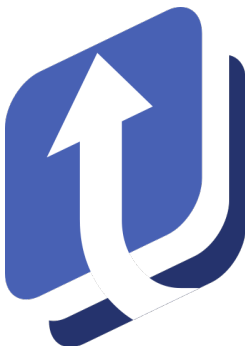
scala> j
res0: List[Int] = List(1, 2, 3)

scala> □
```



Collections

- List[T]
 - Immutable
 - Linked List of type T (String, Int, FooClass)
 - Very efficient for small collections
- Array[T]
 - Mutable
 - Thin veneer on JVM Array
 - Lots of methods (map, filter, toList, etc.)



Collections

File Edit View Terminal Tabs Help

```
scala> val a = Array(1,2,3)
a: Array[Int] = Array(1, 2, 3)
```

```
scala> a(2) = 99
```

```
scala> a
res2: Array[Int] = Array(1, 2, 99)
```

```
scala> val b = List(1,2,3)
b: List[Int] = List(1, 2, 3)
```

```
scala> b(2) = 99
<console>:6: error: value update is not a member of List[Int]
    b(2) = 99
      ^
```

```
scala> a.map(_ + 99)
res4: Array[Int] = Array(100, 101, 198)
```

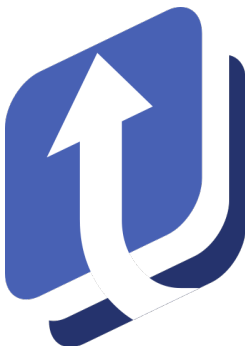
```
scala> a
res5: Array[Int] = Array(1, 2, 99)
```

```
scala> █
```



map & filter

- map
 - Apply a function to each element in original
 - Return a new collection containing the result
 - `List(1,2,3).map(_ + 1) // List(2,3,4)`
- filter
 - Test each element against a function that returns Boolean
 - Return a new collection containing items that passed



map & filter

File Edit View Terminal Tabs Help

```
scala> List(1,2,3).map(_ + 1)
res6: List[Int] = List(2, 3, 4)
```

```
scala> List(1,2,3).map(_.toString + " Cats")
res7: List[java.lang.String] = List(1 Cats, 2 Cats, 3 Cats)
```

```
scala> def even(in: Int) = (in % 2) == 0
even: (Int)Boolean
```

```
scala> List(1,2,3,4,5).filter(even)
res8: List[Int] = List(2, 4)
```

```
scala> List(1,2,3,4,5).filter(v => even(v))
res9: List[Int] = List(2, 4)
```

```
scala> List(1,2,3,4,5).filter(even).map(_.toString)
res10: List[java.lang.String] = List(2, 4)
```

```
scala> List(1,2,3,4,5).filter(even).map(_.toString + " is even")
res11: List[java.lang.String] = List(2 is even, 4 is even)
```

```
scala> □
```



flatMap

- flatMap
 - Flattens the collection the function returns
 - Useful for nesting operations on collections
 - "99 red balloons".toList
List[Char] = List(9, 9, , r, e,...)
 - List("99 red balloons", "88 lines", "44 women").
flatMap(s => s.filter(Character.isDigit))
List[Char] = List(9, 9, 8, 8, 4, 4)



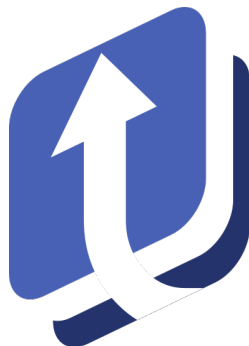
foldLeft & reduceLeft

- Perform an operation on adjacent members of a collection
- Good for: sum, product, etc.
- `List(1,2,3,4).reduceLeft(_ + _)`
`Int = 10`
- ```
def sq(in: Int) = in * in
def sum(in: List[Int]) = in.foldLeft(0)(_ + _)
def sumSq(in: List[Int]) = sum(in.map(sq))
def sqSum(in: List[Int]) = sq(sum(in))
```



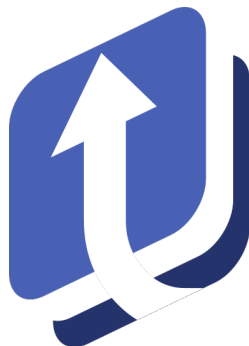
# Parser Calculator

- object Calc extends JavaTokenParsers {  
  
 def expr = term ~ rep("+ ~ term | "-" ~ term)  
  
 def term = factor ~ rep("\* ~ factor | "/" ~ factor)  
  
 def factor = floatingPointNumber | "(" ~ expr ~ ")"  
}



# Parser Calculator

- import scala.util.parsing.combinator.\_  
object Calc extends JavaTokenParsers {  
  
 def expr = term ~ rep("+ ~ term | "-" ~ term) ^^ {  
 case v ~ f => f.foldLeft(v) {  
 case (x, "+" ~ term) => x + term  
 case (x, "-" ~ term) => x - term }}  
  
 def term = factor ~ rep("\* ~ factor | "/" ~ factor) ^^ {  
 case v ~ f => f.foldLeft(v) {  
 case (x, "\*" ~ term) => x \* term  
 case (x, "/" ~ term) => x / term }}  
  
 def factor: Parser[Double] =  
 (floatingPointNumber ^^ (\_.toDouble)) |  
 "(" ~> expr <~ ")"  
}



# Calculator in Action

```
File Edit View Terminal Tabs Help

scala> parseAll(expr, "1 + 1")
res7: Arith.ParseResult[Double] = [1.6] parsed: 2.0

scala> parseAll(expr, "2 * 8")
res8: Arith.ParseResult[Double] = [1.6] parsed: 16.0

scala> parseAll(expr, "2 * 8 + 9")
res9: Arith.ParseResult[Double] = [1.10] parsed: 25.0

scala> parseAll(expr, "2 * 8 + 9 / 3")
res10: Arith.ParseResult[Double] = [1.14] parsed: 19.0

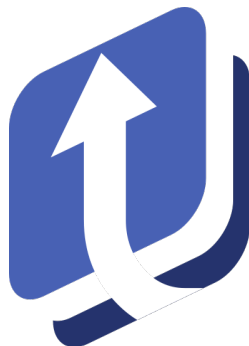
scala> parseAll(expr, "2 * (8 + 9) / 3")
res11: Arith.ParseResult[Double] = [1.16] parsed: 11.333333333333334

scala> □
```



# Domain Specific Languages

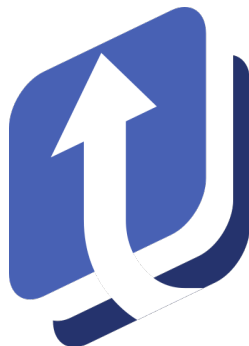
- `User.find(By(email, "dpp@athena.com"),  
          OrderBy(signupDate, Descending))`  
**Type Safe**
- `SetValById(inputName, "") &  
SetValById(hiddenName, "") &  
SetElemById(clrLinkName, "none", "style.display")`
- `<button onclick={  
  jsonCall("submit", JsArray(ValById(inputName),  
  ValById(hiddenName)))}>Yep</button>`



# DSL definitions

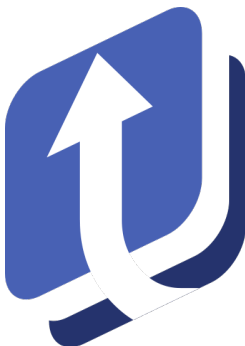
- This separates the library consumers from the architects

- ```
def find(by: QueryParam[A]*)  
abstract class QueryParam[A<:Mapper[A]]  
case class Cmp[A<:Mapper[A], T](...)  
    extends QueryParam[A]  
object By {  
    import OprEnum._  
    def apply[O <: Mapper[O], T, U <% T](...) =  
        Cmp[O,T](field, Eql, Full(value), Empty)
```



DSL definitions

- `case class SetValById(id: String, right: JsExp) extends JsCmd {
 def toJsCmd = "document.getElementById(\"+id.encJs+\").value = "+
 right.toJsCmd+";"}`
- `trait JsCmd extends HtmlFixer with ToJsCmd {
 def &(other: JsCmd): JsCmd =
 JsCmds.CmdPair(this, other)
 def toJsCmd: String }`



Questions?

- Thanks for participating!

