# Seven Languages in Seven Weeks

Correl Roush

July 8, 2015

| | |
|---|---|
| Created | 2003 |
| Author | Martin Odersky |

A multi-paradigm language on the JVM, bringing support for functional programming and a strong static type system.

## GETTING SCALA

```
http://scala-lang.org/
```

## DAY 1: THE CASTLE ON THE HILL

- Types and Expressions
- Loops
- Ranges and Tuples
- Classes

Everything is an object.

```
1 + 1
// 2

(1).+(1)
// 2

"abc".size
// 3
```

## EXPRESSIONS AND CONDITIONS

```
5 < 6
// true

5 <= 2
// false

val a = 1
val b = 2

if (a > b) {
  println("True")
} else {
  println("False")
}
// False
```

## LOOPS

```
def whileLoop {
    var i = 1
    while(i <= 3) {
        println(i)
        i += 1
    }
}
def forLoop {
    println( "for loop using Java-style iteration" )
    for(i <- 0 until args.length) {
        println(args(i))
    }
}
def rubyStyleForLoop {
    println( "for loop using Ruby-style iteration" )
    args.foreach { arg =>
        println(arg)
    }
}
```

## RANGES AND TUPLES

```
val range = 0 until 10
// Range(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
range.start
// 0
range.end
// 10
range.step
// 1

(0 to 10) by 5
// Range(0, 5, 10)

val person = ("Elvis", "Presley")
person._1 // "Elvis"
person._2 // "Presley"

val (x, y) = (1, 2)
// x = 1, y = 2
```

```scala
class Compass {
  val directions = List("north", "east", "south", "west")
  var bearing = 0
  print("Initial bearing: ")
  println(direction)

  def direction() = directions(bearing)

  def inform(turnDirection: String) {
    println("Turning " + turnDirection + ". Now bearing " + direction)
  }
  def turnRight() {
    bearing = (bearing + 1) % directions.size
    inform("right")
  }
  def turnLeft() {
    bearing = (bearing + (directions.size - 1)) % directions.size
    inform("left")
  }
}
```

## EXTENDING CLASSES

- Companion Objects and Class Methods
- Inheritance
- Traits

## EXERCISES

## DAY 2: CLIPPING BUSHES AND OTHER NEW TRICKS

- Simple Functions
- Mutable and Immutable Variables
- Collections

## A SIMPLE FUNCTION

```
def double(x:Int):Int = {
  x * 2
}
```

## VAR VS. VAL

|  |  |
|---|---|
| var | mutable |
| val | immutable |

*Mutable state limits concurrency.*

## COLLECTIONS

- Lists
- Sets
- Maps

## ANY AND NOTHING

Any is the root class in the Scala class hierarchy.
Nothing is a subtype of every type.

*Everything inherits from Any, and Nothing inherits from everything.*

- foreach

```
hobbits.foreach(hobbit => println(hobbit._1))
// frodo
// samwise
// pippin
```

- List methods (isEmpty, length, size)

```
list.isEmpty // Boolean = false
Nil.isEmpty // Boolean = true
list.head // java.lang.String = frodo
list.tail // List[java.lang.string] = List(samwise, pippin)
```

- count, map, filter, and others

```
words.count(word => word.size > 2)
words.filter(word => word.size > 2)
words.map(word => word.size)
words.exists(word => word.size > 4)
```

```scala
val list = List(1, 2, 3)
val sum = (0 /: list) {(sum, i) => sum + i}
// sum: Int = 6
```

- We invoke the operator with a value and a code block. The code block takes two arguments, `sum` and `i` .
- Initially, /: takes the initial value, 0, and the first element of `list`, 1, and passes them into the code block. `sum` is 0, `i` is 1, and the result of `0 + 1` is `1` .
- Next, /: takes 1, the result returned from the code block, and folds it back into the calculation as `sum` . So, `sum` is `1` ; `i` is the next element of `list`, or 2; and the result of the code block is 3.
- Finally, /: takes 3, the result returned from the code block, and folds it back into the calculation as `sum`. So, `sum` is 3; `i` is the next element of `list`, or 3; and `sum + i` is 6.

## EXERCISES

## DAY 3: CUTTING THROUGH THE FLUFF

- XML DSL
- Pattern Matching
- Guards
- Regular Expressions
- Concurrency

```
val movies =
  <movies>
      <movie genre="action">Pirates of the Caribbean</movie>
      <movie genre="fairytale">Edward Scissorhands</movie>
  </movies>

movies.text
// String =
//
//       Pirates of the Caribbean
//       Edward Scissorhands
//
```

## PATTERN MATCHING

```
def doChore(chore: String): String = chore match {
  case "clean dishes" => "scrub, dry"
  case "cook dinner" => "chop, sizzle"
  case _ => "whine, complain"
}

println(doChore("clean dishes"))
// scrub, dry

println(doChore("mow lawn"))
// whine, complain
```

## GUARDS

```
def factorial(n: Int): Int = n match {
    case 0 => 1
    case x if x > 0 => factorial(n - 1) * n
}
```

## REGULAR EXPRESSIONS

```scala
val reg = """^(F|f)\w*""".r

println(reg.findFirstIn("Fantastic"))
// Some(Fantastic)

println(reg.findFirstIn("not Fantastic"))
// None
```

## XML WITH MATCHING

```scala
val movies = <movies>
    <movie>The Incredibles</movie>
    <movie>WALL E</movie>
    <short>Jack Jack Attack</short>
    <short>Geri's Game</short>
</movies>

(movies \ "_").foreach { movie =>
    movie match {
        case <movie>{movieName}</movie> => println(movieName)
        case <short>{shortName}</short> => println(shortName + " (short)")
    }
}
// The Incredibles
// WALL E
// Jack Jack Attack (short)
// Geri's Game (short)
```

## CONCURRENCY

```
import scala.actors._
import scala.actors.Actor._

case object Poke
case object Feed

class Kid() extends Actor {
  def act() {
    loop {
      react {
        case Poke => {
          println("Ow...")
          println("Quit it...")
        }
        case Feed => {
          println("Gurgle...")
          println("Burp...")
        }
      }
    }
  }
```

## EXERCISES

## WRAPPING UP SCALA: STRENGTHS

- Concurrency
- Evolution of Legacy Java
- Domain-Specific Languages
- XML
- Bridging

## WRAPPING UP SCALA: WEAKNESSES

- Static Typing (with mixed paradigms)
- Syntax
- Mutability

Scala represents a bridge between the large Java community and functional, concurrent programming.