The Scala programming ecosystem

Leveraging functional, OO, libraries and frameworks

Markus Dale, 2016

Scala - The Good

The Scala Programming Language

- Martin Odersky, EPFL, Switzerland
  - Worked on javac (1.3)
  - Java Generics
- Lightbend (formerly Typesafe)
- Multi-paradigm language
  - Functional and Object-Oriented
- Statically typed
- Scalable language - script to large program
- Stretch your mind - functions and immutability

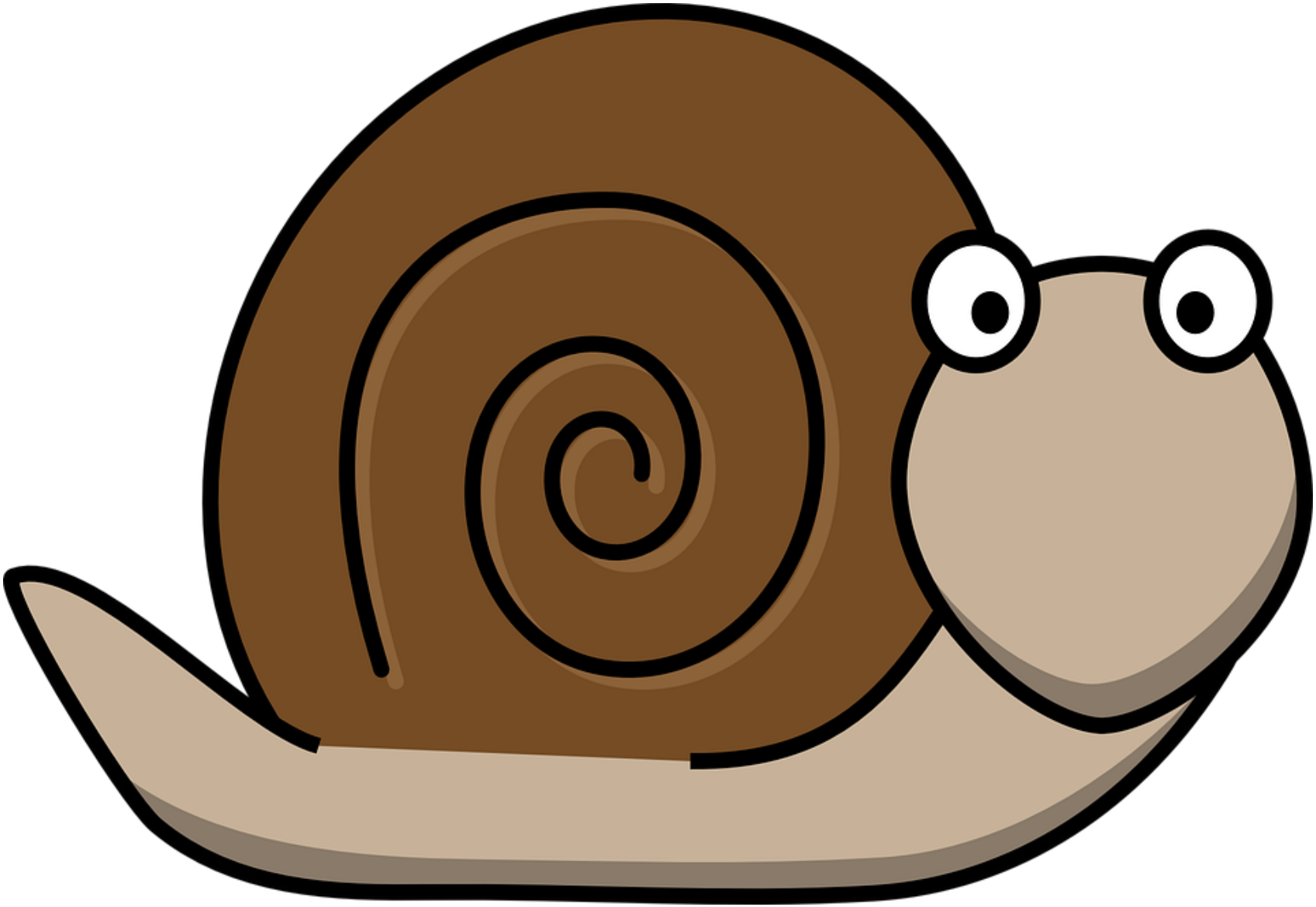
Sca(lable) la(nguage)

- Apache Kafka (LinkedIn)
- Apache Spark (Databricks)
- Finagle (Twitter)
- Akka (Lightbend)
- Lucid Software - scala.js presentation
- Play Web Framework
  - Lichess Online Chess
- Lightbend customers: Walmart, Verizon, Twitter, LinkedIn, Coursera, The Guardian, Airbnb...

Scala to Java bytecode

- Leverage Java Virtual Machine (JVM)
  - Over 20 years of optimizations
  - Java Interpreter and Just-in-time (JIT) compilers
  - Portability and Security
  - Ever-evolving garbage collectors
- Full interoperability with Java and Java libraries

Exploration - Scala Shell and Worksheet



Scala Tour

- Conciseness
- Mixed Paradigms
  - Object Oriented
  - Functional
- Options, Collections
- Functional Pattern Matching
- Implicits
- Spark

Vals and vars but no semicolons

```scala
val helloWorld = "Hello, Scala World!"

//vals are immutable
//helloWorld2 = "this is a different string"

val names = List("Markus", "Joe", "Jane")

//vars are mutable
var allHellos = ""

names.foreach(name =>
    allHellos += s"Hello, ${name}! ")

println(allHellos)
> Hello, Markus! Hello, Joe! Hello, Jane!
```

Defining a function, higher-order functions

```
def hasAtLeastThreeLetters(input: String): Boolean = {
    if ((input != null) && (!input.isEmpty)) {
        val letters = input.filter(c => c.isLetter)
        letters.size >= 3
    } else {
      false
    }
}
```

Calling a function - syntactic sugar

```
val testInputs = List(null, "", "lower", "Upper")

testInputs.map((input: String) =>
    hasAtLeastThreeLetters(input))

testInputs.map((input) =>
        hasAtLeastThreeLetters(input))

testInputs.map(input => hasAtLeastThreeLetters(input))

testInputs.map(hasAtLeastThreeLetters(_))

testInputs.map(hasAtLeastThreeLetters)
> res0: List[Boolean] = List(false, false, true, true)
```

Assigning functions/function literals to variables

```
val vowels = List('a','e','i','o','u')

val threeLs: String => Boolean = hasAtLeastThreeLetters

threeLs("abcd")
> res1: Boolean = true

val removeVowels: (String) => String = { (str) =>
    str.filter(c => !vowels.contains(c))
}

val removeNonLetters: String => String = { str =>
    str.filter(c => c.isLetter)
}

removeVowels("wabbit")
> res2: String = wbbt
```

Everything's an object, more syntactic sugar, == equality

```
3 * 10

3.*(10)

1 to 10
1.to(10)
> res2: scala.collection.immutable.Range.Inclusive =
 Range(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

val foo = "foo"
val bar = new String("foo")

foo == bar
```

Built-in tuples

```
val tuple = ("hello", 42)

val tuple2: (String, Int) = ("hello", 42)

val tuple3: Tuple2[String, Int] = ("hello", 42)

val triple = ("123-22-2111", "Joe", "443.998.8899")

tuple._1
tuple._2

val (word, count) = tuple
> word: String = hello
> count: Int = 5
```

Options - no more NullPointerExceptions!

```
val portOpt: Option[Int] = Some(5123)
val port2Opt: Option[Int] = None
portOpt.get
> res0: Int = 5123

port2Opt.get -
> java.util.NoSuchElementException: None.get

port2Opt.getOrElse(3306)
> res1: Int = 3306

portOpt.foreach(port => println(s"opening port ${port}"))
> res2: Unit = ()

Option(null)
> res3: Option[Null] = None
```

Collections - Arrays (with syntactic sugar)

```
val a : Array[Int] = Array(1,3,7,9)
//val b = Array.apply(1,3,7,9)

a(0)
//b.apply(0)

a(0) = 5
//b.update(0, 5)

a.mkString(",")
> res1: String = 5,3,7,9
```

Collections - Lists

```
val ws = List("When", "shall", "we", "three")

val longWords = ws.filter(s => s.length > 4)

val lowers = ws.map(_.toLowerCase)

lowers.flatMap(_.permutations)
> res3: List[String] = List(when, whne, wehn...

//how many letters in our list?
val lengths = ws.map(_.length)
lengths.reduce(_ + _)
lengths.sum
```

Collections - Maps 1

```
var transMap = Map("when" -> "wann",
    "shall" -> "sollen", "we" -> "wir")

val entryTuple1 = ("three" -> "drei")
val entryTuple2 = ("meet", "treffen")

transMap = transMap + entryTuple1
transMap = transMap + entryTuple2

transMap("when")
//transMap("who") //java.util.NoSuchElementException

transMap.get("when")
> res10: Option[String] = Some(wann)

transMap.get("who")
res11: Option[String] = None
```

Collections - Maps 1

```
val whenGerman = if (transMap.contains("when")) {
    transMap("when")
} else {
    "unbekannt"
}

val whenGerman2 = transMap.getOrElse("when", "unbekannt")

val transMap2 = transMap.withDefaultValue("unbekannt")

transMap2("when")
> res12: String = wann

transMap2("who")
> res13: String = unbekannt
```

Collections - higher-order functions

```
val wordLengthTuples = ws.map(s => (s, s.length))

val lengthMap =
   wordLengthTuples.groupBy { case (word, length) =>
      length }
> lengthMap: immutable.Map[Int,List[(String, Int)]]

lengthMap(5)
>res14: List[(String, Int)] = List((shall,5), (three,5))
```

For Comprehensions, yield, guards

```
val input = "afed-123-ghi-45-67"

//if we did not have RichChar.isDigit...
def isDigit(c : Char): Boolean = {
    ('0' to '9').contains(c)
}

var digits = ""
for (c <- input) {
    if (isDigit(c)) digits += c
}
digits
> res1: String = 1234567

val digits2 = for(c <- input if isDigit(c)) yield c
> digits2: String = 1234567
```

Scala Docs

Scala Standard ×

www.scala-lang.org/api/2.11.8/#scala.collection.immutable.Vector

Apps Asymmetrik Wildfire GeneralDev Scala

Vector

#ABCDEFGHIJKLMNOPQRSTUVWXYZ – deprecated

display packages only

scala.collection.immutable hide focus

- Vector
- VectorBuilder
- VectorIterator

scala.collection.parallel.immutable hide focus

- ParVector

scala.collection.immutable

Vector

```
final class Vector[+A] extends AbstractSeq[A] with IndexedSeq[A] with Gen
VectorPointer[A] with Serializable with CustomParallelizable[A, ParVector
```

Vector is a general-purpose, immutable data structure. It provides random access and updates in effectively consta
random functional updates, they are currently the default implementation of immutable indexed sequences. It is ba
very large sequences.

A	the element type

Self Type	Vector[A]
Source	Vector.scala
See also	"Scala's Collection Library overview" section on `Vectors` for more information.

▶ Linear Supertypes

Ordering Alphabetic By Inheritance

Inherited Vector CustomParallelizable Serializable Serializable VectorPointer IndexedSeq
GenSeq GenSeqLike PartialFunction Function1 AbstractIterable Iterable Iterab
GenericTraversableTemplate TraversableLike GenTraversableLike Parallelizable Trav

scalatour/07-MultilineStrings

- Triple quotes
- substitution (f for printf formatting)

```
val d = 100
val s = f"${d}%05d"
> s: String = 00100
```

scalatour/08-FunctionalPatternMatching

- match construct
- match by type, structure
- default case or MatchError

scalatour/09-ParsingConfig

- Match on regular expressions
- Go Options

scalatour/10-ClassesTraitsMixins

- class - constructor/body
- constructor args - val, var, no modifier
- traits

scalatour/11-CaseClasses

- provide val accessors
- apply/unapply, hashCode, toString
- pattern matching

scalatour/12-Scripting

- In the small
- sys.process
- sys.env
- sys.props

scalatour/13-JavaInterop

- to/from Java/Scala collections
- BeanProperty for getters/setters

scalatour/14-Implicits

- Use sparingly!
- Powerful way to extend closed classes

scalatour/Spark15

- Implemented in Scala
- Powerful functional primitives for scalable cluster processing

scalatour/exercises

- See scalatour_exercises and scalatour_solutions

Resources

- Coursera/EPFL Functional Programming in Scala Specialization
- Horstmann, Scala for the Impatient Video
- Odersky et al., Programming in Scala, 3rd Edition
- Payne, Wampler, Programming Scala, 2nd Edition
- Alexander, Scala Cookbook
- Chiusano, Bjarnason, Functional Programming in Scala
- Twitter Scala School