

What The Hell Is This „Scala“ Anyway?

Sebastian Jackel

What's to come?

- What is Scala?
- Basic Language Features
- Collections
- Functional Code
- Concurrency and Actors

What is Scala?

- Scala is a hybrid OO/functional language
- It runs on the JVM
- It is statically typed
- It was designed by Martin Odersky

Odersky? Who's that?

- Not the guy who invented peanut butter!
- But he co-wrote GJ which became javac of Java 1.3+.
- He also co-designed Java Generics
- He's currently a professor at EPFL in Lausanne, Switzerland

Why do we even need another language?

- Concurrency is more prevalent than ever
- In bigger environments, you want more powerful, higher level abstractions
- You want functional programming (oh yes, you do!)

```
10  BASICS
```

```
20  GOTO 10
```

The obligatory (and only) comparison to Java

```
public class Employee {  
    private final String name;  
    private double salary;  
  
    public Employee(String name, double salary) {  
        this.name = name;  
        this.setSalary(salary);  
    }  
  
    public String getName() { return name; }  
    public double getSalary() { return salary; }  
    public void setSalary(double salary) {  
        this.salary = salary;  
    }  
}
```

The obligatory (and only) comparison to Java

That code becomes

```
class Employee(val name: String, var salary: Double)
```


So the basic syntax is...?

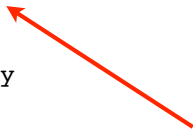
Class declaration



Fields & Constructor args



```
class Employee(val name: String, var salary: Double) {  
  def modifySalary(modify: Double => Double) {  
    salary = modify(salary)  
  }  
  def total = salary  
}
```



Method declarations

What else is there to know?

- There are no static methods. You place those in singleton objects instead.
- In Scala everything REALLY is an object:

```
val x = 1 + 2  
val y = 2.+(3)
```



**Many things that seem to be keywords
are just methods!**

Inheritance & Traits

- Like Java, Scala does not do straight multiple inheritance
- Scala's *traits* are richer than interfaces though.
 - They may implement methods and fields.
 - They can be mixed in at runtime

Ye olde Logger example

```
trait Logging {  
  val level: Level  
  
  def log(msg: String) {  
    preferredLogLib.log(level, msg)  
  }  
}  
  
val app = new ImportantApp(...) with  
Logging { val level = DEBUG }  
  
app.log("Shit -> Fan!!!")
```

["Coll", "ect", "ions"]

Let's have a look at collections

Lists are fun(ctional):

```
scala> val nums = 1 :: 2 :: 3 :: 4 :: 5 :: Nil  
nums: List[Int] = List(1, 2, 3, 4, 5)
```

```
scala> nums head  
res0: Int = 1
```

```
scala> nums tail  
res1: List[Int] = List(2, 3, 4, 5)
```

Let's have a look at collections

The compiler does some awesome inference!

```
scala> nums map {_ * 2}
res2: List[Int] = List(2, 4, 6, 8, 10)
```

```
scala> nums reduceLeft {_ + _}
res3: Int = 15
```

```
scala> nums filter {_ % 2 == 0}
res4: List[Int] = List(2, 4)
```

Function Literals instead of clunky anon inner classes

Don't go without a map

```
val salaries = Map(  
  "Ralf" -> 2000.0,  
  "Erik" -> 1500.0,  
  "Don" -> 1500.0  
)
```

```
val employees = for((name, salary) <- salaries)  
  yield new Employee(name, salary)
```


The Option type

```
scala> salaries.get("Ralf")  
res0: Option[Double] = Some(2000.0)
```

```
scala> salaries.get("Kurt")  
res1: Option[Double] = None
```

- It is either None or Some(value)
- It is iterable via for-comprehension
- It helps prevent NPEs

We have more Options

```
for(kurt <- salaries.get("Kurt")) println(kurt)
```

won't print anything, but won't throw an NPE either,
because kurt has value „None“

- Java collections return null all the time
- Those can be wrapped, because

```
scala> Option(null)  
res0: Option[Null] = None
```

**Note: If you're using null and you're not
working with Java libraries, you're doing it
wrong!**

While we're looking at for...

for can also accumulate results:

```
for(i <- 1 to 5) yield {i*2}
```

returns

```
Vector(2, 4, 6, 8, 10)
```

```
for(i <- 1 to 5 if (i % 2 == 0)) yield i
```

returns

```
Vector(2, 4)
```

More

Functional
Code

Why would you want functional programming?

- You want to take away the pain from coding concurrent or parallel algorithms.
- Means: You want to minimize shared mutable state.
- You want to keep your code concise.

An immutable Employee

```
case class Employee(name: String, salary: Double) {  
  
  def modifySalary(modify: Double => Double) = {  
    Employee(name, modify(salary))  
  }  
  ...  
}
```

**Since we're immutable now, we
return a new instance here**

- Just adding the keyword „case“ to a class gives us:

- A factory method for object construction
- All fields are automatically considered as vals
- Implementations of toString, hashCode and equals
- Enables pattern matching

**Allows for testing
structural equality**

Pattern Matching

Remember Option?

```
Option(dbResultSet.getString("name")) match {  
  case Some(s: String) => s  
  case None => ""  
}
```

Looks like a mightier switch doesn't it?


Pattern Matching

Let's take a simple model of expressions:

```
sealed trait Expr
case class Num(value: Int) extends Expr
case class Plus(left: Expr, right: Expr) extends Expr
case class Minus(left: Expr, right: Expr) extends Expr
case class Mult(left: Expr, right: Expr) extends Expr
case class Div(left: Expr, right: Expr) extends Expr
```

To evaluate instances of those, we can just do this:

```
def eval(e: Expr): Int = e match {
  case Num(n) => n
  case Plus(l, r) => eval(l) + eval(r)
  case Minus(l, r) => eval(l) - eval(r)
  case Mult(l, r) => eval(l) * eval(r)
  case Div(l, r) if (eval(r) != 0) => eval(l) / eval(r)
  case _: Div => throw new Exception("Division by zero")
}
```



We can inspect the structure of case class instances

Function literals and Currying

You can declare anonymous functions („lambdas“) like this:

```
scala> val fun = (x: Int) => x * 2  
fun: (Int) => Int = <function1>
```

(You've already seen it used with collections:)

```
scala> nums map {_ * 2}  
res2: List[Int] = List(2, 4, 6, 8, 10)
```

Function literals and Currying

Currying means partially applying a function

```
scala> val fun = (x: Int) => (y: Int) => x * y  
fun: (Int) => (Int) => Int = <function1>
```

```
scala> val timesTwo = fun(2)  
bytwo: (Int) => Int = <function1>
```

```
scala> nums map timesTwo  
res0: List[Int] = List(2, 4, 6, 8, 10)
```

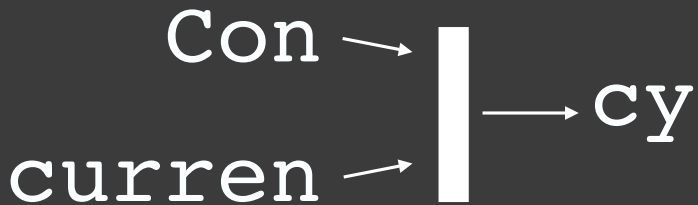
Functions as objects

- Any object can be used like a function by implementing an `apply()` method.
- In fact, functions are just objects:

```
trait Function1[-T1, +R] {  
  def apply(v1: T1): R  
}
```



**We're of course eliding some
convenience methods here**



Hanging by a thread...

Just use Java threads if you want to:

```
scala> new Thread {println("Serious Business!")}  
Serious Business!  
res0: java.lang.Thread = Thread[Thread-4,5,main]
```

But if you do, you'll be facing the lock and synchronization issues that come with the paradigm.

Actors

- Small entities that communicate by sending each other messages
- side-effecting behaviour can be contained within one actor.
- Only the actors' inboxes need to be synchronized.
- In Scala, actors run on a threadpool that is expanded as needed.
- not necessarily 1 actor == 1 thread

Actors

A Scala actor implements the Actor trait:

```
class UselessActor extends Actor {  
  def act() = {  
    println("Look ma, I'm acting!")  
  }  
}
```

```
scala> val wahlberg = new UselessActor()  
wahlberg: UselessActor = UselessActor@7bc8b313
```

```
scala> wahlberg.start()  
res0: scala.actors.Actor = UselessActor@7bc8b313  
Look ma, I'm acting!
```

Actors

Using `react()` or `receive()` and `loop()`, an actor can communicate via messages.

```
class SimpleActor extends Actor {  
  def act() = loop {  
    react {  
      case s: String => println("Received: " + s)  
    }  
  }  
}
```


Actor communication

```
case class Ping(sender: Actor)
case class Pong(count: Int, sender: Actor)
case object Stop
```

```
class PingActor(val max: Int) extends Actor {
  var count = 0

  def act() = loop {
    react {
      case Ping(sender) => if (count == max) {
        sender ! Stop
        self ! Stop
      } else {
        count += 1
        println("Got Ping no. " + count)
        sender ! Pong(count, self)
      }
      case Stop => {
        println("Exiting!")
        exit()
      }
    }
  }
}
```

```
class PongActor extends Actor {

  def replyTo(target: Actor) {
    val sender = self
    actor {
      Thread.sleep(1000)
      target ! Ping(sender)
    }
  }

  def act() = loop {
    react {
      case Pong(count, sender) => {
        println("Got Pong no. " + count)
        replyTo(sender)
      }
      case Stop => {
        println("Exiting!")
        exit()
      }
    }
  }
}
```

Also:

```
val billMurray = new GreatActor()
```

Oh the songs we didn't sing...

- implicit conversions and parameters
- XML literals
- combinator parsing
- the power of the type system
 - advanced inheritance
 - type bounds
 - structural typing
 - type classes and higher kinded types
 - fluffy kittens

Scala Sources

Sites

www.scala-lang.org - main language site

www.implicit.ly - news about Scala software releases

www.scala-ide.org - home of the Scala Eclipse plugin

www.akka.io - Akka concurrency framework

www.liftweb.net - Lift web framework

Free online books

programming-scala.labs.oreilly.com - „Programming Scala“
by Dean Wampler and Alex Payne

www.artima.com/pinsltd - „Programming in Scala“ 1st ed. by
Lex Spoon, Bill Venners and Martin Odersky

simply.liftweb.com - Simply Lift by David Pollak

www.scala-lang.org/docu/files/ScalaByExample.pdf - „Scala
By Example“ (Draft), a PDF tutorial on Scala by Martin
Odersky

That's all folks!
Questions?

Thanks for advice and encouragement to:

Dean Wampler (@deanwampler)

Heiko Seeberger (@hseeberger)

Mario Gleichmann (@mariogleichmann)

Ralf Lämmel (@reallynotabba)

scalaz

scalate

Akka

squeryl

Bonus Round: The Scala Environment

specs

Scala-IDE

Play

sbt

ScalaTest

Lift

How to manage Scala projects

- Ye olde ant
- maven-scala-plugin
 - From the official scala-tools Maven Repo:
<http://scala-tools.org/repo-releases>
- sbt - Simple Build Tool

Simple Build Tool

- Basically replaces Maven
- Uses Apache Ivy for dependency management
- Comfortably XML-free
- Lots of plugins

IDE Integration

- IntelliJ IDEA plugin
 - Great code completion, analysis and refactoring support
 - Sometimes a bit unstable
- Scala-IDE for Eclipse:
 - Has for a long time been so-so
 - On it's way for a complete overhaul for Scala 2.9 (looks promising)
- Bundles for vim, emacs, TextMate, Insert your favorite editor here...

Akka

- Platform for scalable, event-driven and fault tolerant applications
- Provides very lightweight actors (~600 bytes per Instance)
- Software Transactional Memory
- „Let-it-Crash“ philosophy
- Java and Scala API

Noteworthy Libs & Frameworks

- Lift - a View-First web application framework
- Play - If you prefer MVC instead
- scalaz - Pure functional data structures for Scala (inspired by Haskell)
- Scalate - Scala Template Engine
- Scalatest, Scalacheck and Specs2 - three TDD/BDD libraries