Exploring Scala

https://github.com/CodeSolid/ScalaTalkSacjug

Who:	John Lockwood
Web:	http://codesolid.com
	http://scalausergroup.org
Twitter:	@JohnLockwood
Mail:	john@codesolid.com
LinkedIn:	http://www.linkedin.com/in/codesolid

Thanks and How Did We Get Here?

- Thank you for having me.

Starting another side-project, are we?
Yeah... Well no actually, I've nearly finished the other one so I just wanted to test out this other thing
CommitStrip.com

Broad Topics

- What is Scala? Getting Our Bearings Part I.
- Getting Started – Using The Tools
- Scala's Relation to Java. Getting Our Bearings Part II.
- A Brief Introduction to Functional Programming.
- Concurrency with Scala – A quick look at akka

What Is Scala?

Two points of view:

- Scala is an awesome, elegant, fun language running on the JVM that supports both object oriented and functional programming styles. Learning it will lead you to cool opportunities or at the very least to be a better programmer.
- Scala is an obtuse, unclear, non-mainstream, pedantic, slow-to-compile beast with only one job to be had in the market for every 33 jobs in Java. It's what you get when bad operator overloading happens to good people.

Scala Tools

- “scala” (REPL or script tool)
- “scalac” (Compiler, more like javac)
- “sbt” (“Simple Build Tool”
- Scala in the IDE
  - Idea (demo)
  - Eclipse (mention) “Scala IDE for Eclipse”

Installing Scala

I wasn't going to mention this because I know you can all figure it out, but in the interests of being thorough / verbose:

- Have a JDK already on your path (javac / java).

  (I haven't tested doing without it).
- Download the right scala.zip /.tgz file for your environment. Or use package mgr / msi etc.
- Extract it to a directory
- Add bin directory to your path

Hello World

- Simplest possible version
  hello/small
- Coming from javac – scalac
  samples/hello/java
- A common idiom – unclear magic is a virtue:
  samples/hello/common
- Running as a bash script, because we can
  hello/

The Scala REPL – An awesome exploration tool.

- Type “scala” with no arguments.
- Can try out snippets of code interactively, and can load / save scripts.
- Some helpful commands:
  - :help
  - :load
  - :save

The Scala REPL Continued

- Command completion.

  -
    ```
    :reset
    :sh ls .
    res0.<tab>
    res0.lines.foreach(println)
    :type res0.lines
    ```

Scala as a Scripting Language

- Full disclosure: a beginner idea that has not aged well.
- More possible than Java, less plausible than bash, Perl.

scripting/script_enable.scala
scripting/curlit
scripting/ListFiles

SBT

- Scala's “**S**imple” **B**uild **T**ool
- It really is Simple if you try to use it like Maven. If you want it to act like Ant or Rake, put scare quotes around “Simple” and bring a lunch – you're in for a learning curve.

Scala's Relation to Java

- Natural Order:
  - Similarities / Differences
  - What's Better / What's Worse
- Presentation Order:
  - Similarities
  - What's Worse
  - What's Better
  - Differences

Similarities:
How Are Java & Scala **The Same**?

- Both run on the JVM and share common JVM features internally.
- Because of this Scala can trivially use existing Java libraries / tools.
- Both are strongly typed.
- Great support for object oriented programming.
- Runtime performance is comparable for comparable language features.

What Scala Does **WORSE** than Java

- Compile time is worse (But... Play!).
- Learning curve / complexity “for some features”.
- Obscurity / pedantry:
  - operator overloading =~!*# operator overdoing
- Some tools particularly difficult / obtuse (Parts of SBT for example).
- Still a Little Bit of Newness Penalty:
  - Third party tooling catching up (less confident now).
  - Java has longer history of documentation & support.

What Scala Does **BETTER** than Java

- Natural support for functional programming
  - Compare my history with C → C++
- A different, and much simplified, way of doing concurrent / parallel programming.
- Better support for scripting.
- I can haz REPL!
  - Tee shirt idea: *“Programmers do it interactively.”*
- Generally much more concise. (Same Java program in ½ or less lines of code).

How Java and Scala are **DIFFERENT**

- Lots of syntactical differences
  - Good ones
  - Surprising ones
- Supporting functional programming / stuff borrowed from other FP languages (Erlang , etc.)
  - Good stuff
  - Ugly stuff

How Java and Scala are **DIFFERENT**

- Objects Everywhere in Scala

  Scala will compile many expressions to primitive types for performance reasons, but in terms of the programming interface, everything is an object.

Any ← AnyRef

← AnyVal

How Java and Scala are **DIFFERENT** (Continued)

```
Scala has type inference in many
cases:

val greeting = “Hello World”
val answer = 42

var states = Array(“RI”, “IN”, “CA”)
var counter = 0


// But can also be explicit...
val theAnswer : Int = getAnswer()
```

Scala has no static methods. Instead uses “object” to define singletons.

```
// HelloWorld.scala

object HelloWorld {
  def main(args: Array[String]):Unit = {
    println("Hello World!")
  }
}

HelloWorld.main(Array[String]())
```

How Java and Scala are **DIFFERENT** (Continued)

- Many constructs return a value in Scala that don't in Java (i.e., more “Expression Based”):

```
// if_expression.scala

val answer = 42
val size = if (answer < 100) "big"
    else "little"
```

For Comprehensions with Yield Made Easy

- For also returns a value in an expression called a “for comprehension”. Python and Ruby also do these.

```
// for_comprehension.scala

val shouting = "HEY SHOUTING IS
  IMPOLITE".split(" ")
val whisper = for (s <- shouting)
  yield s.toLowerCase()
```

Match Expressions (“pattern matching”)

- Scala has no switch statement, but match:
  - Can take any input
  - Can return a value or Unit (Unit is like void in Java, C, etc).

An untyped example matchUntyped.scala

```
// matchUntyped.scala

def roseColored(beloved: Any): Any = {
  beloved match {
    case "green eggs and ham" => -1
    case _ => beloved.toString() +
                 " is beautiful!"
  }
}
```

“Typed match” example matchTyped.scala

```
// matchTyped.scala

def matchIntToString(someNumber: Int):
   String =    someNumber match {
     case 1 => "The loneliest number"
     case 2 => "It's company."
     case 3 => "It's a crowd."
     case _ => someNumber.toString()
   }
```

Match Expressions

- Are used heavily in akka, Scala's concurrency library, which we'll see later on.
- Often use “case classes”, which we'll get into when we look at akka and which will make sense after we look at our next awesome slide, and oh by the way...

  … topic segue ...

Functional Programming

- Is Scala a functional programming language?
- What Is It?
- What are the benefits?
- What are the difficulties?
- Scala as an impure functional language language

Functional Programming Defined

Functional programming is a style of programming that emphasizes programming using **pure functions** – that is, functions that do not have side effects and always return the same result given the same input.

"It treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data." – Wikipedia.

Scala – A Functional Language?

Unlike languages such as Haskell which are (or at least, which purport to be) purely functional, Scala is not a purely functional language BUT...

It has excellent support for functional and imperative programming.

Wikipedia: *“An interesting case is that of Scala – it is frequently written in a functional style, but the presence of side effects and mutable state place it in a grey area between imperative and functional languages.”*

Functional Programming Defined (Continued)

Functional programming languages feature **first class functions**, that is, functions that can be assigned to variables, passed to other functions, and returned from functions.

This in turn supports the creation of **high order functions**, that is, functions that call other functions passed in as arguments.

Functional Programming Defined (Continued)

Functional programming languages support a style of programming that is **declarative** (tell the machine what you want done) rather than **imperative** (tell the machine how to do every step, and as a result, tell it what you want done).

Scala – A Functional Language?

Should you mix imperative and functional code?

- Yes if you haven't gotten through the “Functional Programming” book yet. Pure functional programming is non-trivial.
- Maybe. A reasonable use for Scala is a concise, script-able language on the JVM.
- Sometimes. Some Java libraries may force your hand. What to do with rs.next()?
- No as much as possible if doing concurrency or otherwise trying to get the most benefit out of Scala.

Imperative Programming
Copying an Array of Ints

```
// Imperative.scala

def copyIntArray(src: Array[Int]): Array[Int] = {

   var dest: Array[Int] = new Array[Int](src.length)
      var i = 0
      while (i < src.length) {
          dest(i) = src(i)
          i = i + 1
      }

   dest
}
```

Declarative Programming
Function does the same thing

```
// Declarative.scala

def copyIntArray(src: Array[Int]): Array[Int] = {
   src.map(x => x)
}
```

Functional Programming Topics We've Glossed Over

- Examples of higher-order functions
- Examples of partial functions
- A LOT of other stuff. Currying and monads and monoids, oh my!
- See Paul Chiusano & Rúnar Bjarnason: Functional Programming in Scala

Concurrency With Scala Actors

- Currently implemented in “akka” library.
- “Simple high-level abstractions for concurrency and parallelism.”
- Event driven – relies on asynchronous message passing between threads in the same process.
- Can use the same mechanism across a cluster of machines – “location transparency”.
- Messages passed are generally immutable.

Actors “ask” and “tell”

- Tell is very common, recommended, high performance, fire and forget:

  myActorRef ! “Hello” // tell myActorRef “hello”

- Ask is low performance way of returning a future, should use only when you must:

  val future = myActorRef ? “Hello”

  // … Use future

“Typed match” example (Revisited)

```
// matchTyped.scala

def matchIntToString(someNumber: Int):
  String =    someNumber match {
    case 1 => "The loneliest number"
    case 2 => "It's company."
    case 3 => "It's a crowd."
    case _ => someNumber.toString()
  }
```

Match expressions are basis of Actor “receive” method

```
class SimpleActor extends Actor {
    def receive  = {
       case "Hey" => sender ! "Hay is for horses"
       case _ => sender ! "Unknown message type"
    }
```

Other examples

- Idea project at akka_messages
- Case classes: samples/CaseClassPatternMatching.scala

Scala Popularity: Dice “Jobs” (Nationwide)

- Mainstream languages:

17,448	Java
17,163	C++
12,349	Javascript
8,478	C#

Etc.

- Niche languages:

543	Scala
470	Groovy
99	Clojure
74	Erlang
43	Haskel
34	Lisp
26	F#

Etc., Etc., Etc.