# Scala Programming

Duration: 3 days

Audience: This course is aimed at those who have some Java experience, and anyone who hasn’t is probably going to find it hard going in places. If you need to deliver to a non-Java audience, consider doing an extra Day 0 on the Java world (JVM, class files, Eclipse, JARs and stuff like that). It is also worth finding out whether delegates have any existing functional programming experience.

# Setup

For closed courses, the setup is driven by whatever tools the delegates want to use. For public courses, I recommend

* A recent version of Scala from scala-lang.org. I’ve been using 2.10 up to now.
* Matching versions of sbt and ScalaTest
* An IDE. I tend to use ScalaIDE, but you could also use IntelliJ

The labs are designed so that they can be done from the command line or in using an IDE.

# Content

My overall approach is to emphasize the “why” rather than the “how”. I assume that everyone on the course is a competent programmer (which is *normally* the case) and so should be able to pick up new syntax and tools. What needs to be explained is the “why”, and in particular the concepts behind functional programming. This means that I do tend to skip over some of the syntax, letting people pick it up themselves.

## Introduction to Scala Programming

A general intro to Scala, the environment and tools. How long you take on this depends a lot on how much experience delegates have with both Scala and Java. I mention sbt here, but if no-one is interested I don’t bother with a demo. If there is interest, then I will go into more detail on how to set up a project.

## Scala Syntax Fundamentals

A quick intro to basic Scala syntax, majoring on the differences between Scala and Java. Emphasize the use of val rather than var when declaring values, and possibly use this to start introducing functional ideas.

## Object-Oriented Programming with Scala

An introduction to OO programming in Scala, once again majoring on the differences between Scala and Java. It is a good idea to keep emphasizing the functional nature of Scala and the difference this makes to programming style. For example, discuss why fields are public by default (because values are supposed to be immutable) and how to create/design and work with immutable objects.

There is often a good opportunity for a discussion of the difference between val and def, emphasizing that it definitely isn’t a case of val for declaring values and def for functions!

The chapter finishes with a brief discussion of Java-style exception handling. You can mention that there are more functional ways to do this (ie. Try) that will be introduced later on.

## Inheritance and Traits

Inheritance is the normal JVM model, although the syntax is a bit different. The main thing to cover here is traits, because they are fundamental to an understanding of Scala. I normally spend quite a lot of time on demos and examples from ScalaDoc so people understand how to use them.

## Test-Driven Development with Scala

We would hope that Java developers coming on this course would be thoroughly familiar with (and using) unit testing and TDD. If they are, then you can skip the first few slides and just dive into how to use ScalaTest. If not, you may want to go into more detail about the how and why of TDD.

## Functional Programming with Scala

For most delegates, this is the first difficult chapter, and you may need to take it slowly with lots of demos. I normally do up to currying before lunch, with lots of demos. This is definitely a chapter where you need to emphasize the “why”, otherwise it can seem just a collection of confusing techniques, and do emphasize the advantages of the techniques being discussed.

## Collections and Generics

For many, this will provide a bit of light relief after the functional chapter, and there isn’t too much that will cause trouble here. However, if you have experienced Java delegates, be prepared for a discussion on the efficiency of using immutable collections, and the trade-off of efficiency against immutability.

The section on type bounds and variance at the end is important, but can be left out if you are pressed for time, or if you think it is too much detail for the class.

## Functional programming and collections

This chapter pulls together what delegates have learned about OO, functional programming and collections, and shows many of the common idioms used in Scala programming.

The chapter ends with for comprehensions, which were introduced in the basic syntax chapter, but here I emphasize that they provide a more Java-friendly alternative to map, fold and filter, although they come to the same thing in the end.

## Pattern Matching

The course wraps up with pattern matching, emphasizing how much this is good, idiomatic Scala style. If you are running short of time you can omit the extractors.

# Timings

I normally do the first 4 chapters on the first day, finishing off with the inheritance lab. Day 2 starts with a review and then the TDD chapter: how much you do on this depends on how interested delegates are and how much experience they have. The rest of the morning covers the functional chapter up to currying, and I then normally do a short lab (the one on isEven and isOdd) after lunch before finishing it. The afternoon finishes with the generics and collections chapter, up to Option.

Day 3 starts with a review, then does Option and possibly the variance slides (although you can leave this out if it seems too heavy). Then there is the functional collections chapter, with Pattern Matching about 3pm or thereabouts. I usually do an informal review and wrapup after the patterns lab, so that everyone doesn’t just push off ;-)

# Threading and Concurrency in Scala

Duration: 1 day

This follows on from the introductory course, and assumes that delegates know at least what was covered there. I have found that some people want this as an overview, in which case it may be mainly talk and demos, while others are more interested in actually using it, in which case there will be more labs. And if delegates are only interested in Akka, you can omit the first two chapters altogether.

# Setup

The setup required for the intro course, plus a version of Akka from akka.io to match the Scala version.

# Content

I present this as a progression from old fashioned manual thread management, through futures, to the highly decoupled actor model. The morning is spent on basic threads and futures, possibly introducing actors just before lunch.

## Creating and Using Threads

Since Scala supports the Java threading APIs, this chapter shows how to do basic thread manipulation in Scala. I normally do this very quickly, only really covering it for completeness, and emphasizing that it really isn’t the way you’d tend to do things.

## Futures and Promises

Scala also gives access to the futures and executors provided by Java, but also provides ways to use them in a functional manner. The way in which futures can be used with for comprehensions is particularly worth spending time on.

## Introduction to Actors and Akka

The main things to get over in this chapter are the idea behind actor systems, and the advantages and disadvantages of such systems.

## Fault Tolerance

This chapter shows how actors are built into a hierarchy, so that parents can supervise children, and how this can be used to build robust, self-repairing systems.

## Routing and Dispatching

The final topic shows how message delivery can be customized. It is more important to cover routing in detail, and depending on how time goes, you may want to omit the lab that goes with this topic.