Scala with Functional Programming (5DAY)

* **Mon**
  + 00.Overview
  + 01.Introduction
  + 02.Fundamentals
  + 03.Flow
* **Tue**
  + 04.Methods
  + 05.Functions
  + 06.Collections
* **Wed**
  + 07.Transformation
  + 08.OO
  + 09.Inheritance
* **Thu**
  + 10.Design
  + 11.SBTest
  + 12.Implicits
* **Fri**
  + 13.HigherKinds
  + 14.Monads
  + 15.Appendix
* **Day**
  + Morning Module: 9.30am – 10.45am
  + Morning Break: 10.45am – 11.00am
  + Noon Module: 11.00am – 12.30pm
  + Noon Break: 1.30pm – 2.45pm
  + Afternoon Module: 3.00pm – 4.30pm
* **Lesson**
  + Demonstrations with Questions & Answers: 1 hour
  + Exercise: ½ hour

# Introduction

* Why scala?
  + Libraries
  + Static typing
  + Compositional syntax
  + OO
  + Functional
  + Powerful
* Paradigms
  + OO
  + Functional
* Running scala
  + Anatomy of scala programs
  + Try the repl
  + Scalac
  + Scala interpreter
  + Sbt

# Fundamentals

* Language
  + Objects
  + Calling methods
  + Operators as methods
  + Values and variables
  + Types
* Basic types
  + Boolean
  + Numeric types
  + Unit
  + Strings
  + String methods
* Introduction to collections
  + Type arguments
  + Tuples, Lists, Maps

# Flow

* Branching, matching & selecting
  + Conditionals
  + Intro to pattern matching
  + Destructuring
* For comprehensions
  + Yield vs Unit
  + Comprehensions over lists
  + Comprehensions over maps
  + Comprehensions over ranges
  + Option
  + Multiple extraction
  + Guards
  + Ranges
  + Let expressions
* While loops

# Methods

* Code blocks
* Methods
* Method bodies
* Returning unit
* Passing arguments
* Variadics
* Def vs val
* Lazy vals
* Lazy arguments
* Recursion

# Functions

* What Is Functional Programming?
* Pure Functions
* Functions
* The Function Type
* ‘Function’ vs Method
* Higher Order Functions
* Functions as Data
* Currying
* Aside: Type Aliases

# Collections

* Review: types
* Collections heirachy
* Creating collections
* Range
* Array & array buffer
* List & list buffer
* Vector
* Maps
* Sets
* Empty
* Idioms
  + Pattern matching
  + For comprehnsions review
  + Zipping
  + Traits

# Transformation

* Streams
* Combinators
  + .map & .flatMap
  + Folding & Reducing
  + .exists and .forall
  + Option
* For comprehensions
  + Foreach & Flatmap
* Pattern matching
  + Review
  + Casting
  + Sequences
  + For-yield-match
  + Regex
  + Guards
  + Option

# OO

* Classes
  + Reading class definitions
  + Defining classes
  + Constructors
  + Properties
  + Method overloading
  + Java inspection of scala classes
  + Getters and setters
  + Operator methods
* Objects
  + The object keyword
  + Companions
  + The mechanism of pattern matching
  + Case classes
  + Pattern matching with case classes
  + Aside: case objects

# Inheritance

* Inheritance: a type relation
* Inheritance: class relationship
* Overriding methods
* Overriding tostring
* Calling the parent method
* Abstracts
* Traits
  + Trait inheritance
  + Composing objects with traits
  + The meaning of super
  + Traits as mixins
  + Self types
  + Cake pattern
  + Aside: type members

# Design

* Imports
  + Package objects
* Algebra
  + Algebraic data types: defining
  + Algebraic data types
  + Example: document store
  + Tuples types
* Generics
  + Type arguments
  + Variance
* Special types
  + Option
  + Example: user data
  + Bad: exceptions
  + Try
  + Either

# SBTest

* Simple build tool
  + Build files
  + Creating a project
  + Dependencies
  + Project structure
  + Running tests
  + Sbt task runner
* Testing
  + Type systems as tests
  + Using type systems
  + Assertions
  + Scala test
  + Flatspec
  + Featurespec
  + Matchers

# Implicits

* Implicit conversions
  + Implicit conversions: def
  + Implicit conversions: implicit classes
* Implicit parameters
  + Contexts
  + Scoping
* Type classes
  + Typeclasses with implicit classes
  + Without typeclasses
  + Typeclasses: generalizing
  + Typeclasses: implicit objects
  + Typeclasses: multiple instances

# HigherKinds

* Review: Polymorphism
  + Parametric
  + Subtyping
  + Ad-hoc
* Review: Typeclasseses: Monoid
* Recipe For A Typeclass
* Scalaz
* Sbt for Scalaz
* Simple Typeclass: Equal
* Simple Typeclass: Order
* Simple Typeclass: Implementing Equal
* Kinds
* Higher-kinded Types: Functor
* Generalizing Functor
* Higher-kinded Types: Applicative

# Monads

* Prelude
  + Lifting
  + Type lambdas
* Applicative functors
  + APPLICATIVE (vs. Monadic) style
  + Collections are applicatives
  + Functions are applicatives
* Applicatives to monads
  + Generalizing applicatives
  + For-comprehensions
* Monads
  + The writer monad
  + The function monad ("reader")
* Monad transformers
  + Monad transformers: why
  + Monad transformers: using

# Appendix

* Concurrency
* Libraries