

# The Scala Programming Language

## Description

The Scala Programming Language was invented in 2004 in an attempt to bring functional programming to the jvm, and the java development community. It has risen to prominence in recent years as the de-facto language to compliment and replace java in existing teams, and as an advanced and capable language on greenfield projects.

Scala combines advanced techniques long established in the functional programming community with the best practices of object-orientation, allowing developers to pick and chose a style and an approach that suits their team and their problems.

By the end of this course delegates will have a clear understanding of object orientation and functional programming, and in particular, all the major features of scala which enable these styles of programming to be used and combined.

### **At the end of this course you will be able to**

- Understand the difference between mutable and immutable data
- Understand the basic syntax used by Scala
- Be able to create objects and classes using Scala
- Understand variable scoping and Implicit function calls
- Understand traits and inheritance
- Have an understanding of the functional programming paradigm and how it differs from the object oriented approach
- Understand the concept of currying and lambda functions
- Be able to use collections and generics with functional programming
- Understand pattern matching

# Outline

## 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

## Prerequisites

Delegates must have prior programming experience in java or a related object-oriented language.