Scala in 15 minutes

Guillaume Belrose / @gbelrose

Jozi Jug 30/01/2017

A **Scala**ble language

- Made in Switzerland
- Hybrid **OO** / **FP**
- Statically **typed** language
- Compiles to bytecode / JVM interop
- See also scala.js / scala native

Tooling

- IDEs
  - Eclipse / IntelliJ / Emacs
- Build tools
  - Maven / SBT / Gradle
- Scala is just a **.jar**
- **REPL**

Types

- **Inference**

```
val x : Int = 5;
```

```
val flag = false
```

- Generics

```
val numbers = List(42,26,39,666)
```

- Types, types and more types

“make illegal states unrepresentable”

–Yaron Minsky

Functional programming

- Computation as a chain of functions
- Functions as 1st class citizens
- Composition / Higher-order functions
- Handling state / side effects

Functions

```
val double = { x: Int => x * 2 }

val quadruple = double andThen double

quadruple(2) // 8

val doSomething = { (x: Int, fn: Int => Int) => fn(x)

doSomething(42, quadruple)
```

Immutability

- **values** not variables
- Persistent data structures
  - List, Set, Vectors, etc..
- Functional data structures

```
List(1,3,4,7,8,9)
  .filter( x => x > 5)
  .map( quadruple )
```

Data types

```
case class Person(firstname: String, surname: String)

val me = Person("Guillaume", "Belrose")

val adwin = me.copy(firstname="Adwin")
```

Pattern matching

```
def greet(p : Person) : String = p match{
  case Person("Jacob", "Zuma") =>
    "Hey JZ, how is your fire pool?"

  case Person("François", surname ) =>
    "Hey, are you french?"

  case Person(firstname, _) =>
    "Hello " + firstname
}
```

Grow a language

```
class PersonSpec extends WordSpec with Matchers{

  "Donald's son" should {
    "have his father's surname" in {
      val potus = Person("Donald","Trump")
      val son = potus.copy(firstname = "Eric")
      son.surname should be (potus.surname)
    }
  }
}
```

Scala flavours

- Java-flavoured with syntax sugar
- Mix OO / FP
- Haskell-like FP

Getting started

- Plenty of books
- Coursera MOOCs
- Write tests