Scala cruise

Snorkelling in some of the Scala features

Alessandro Candolini

October 1, 2018

Itinerary

1. Let's meet Scala
2. Tools
3. The Play framework
4. Learn more

Let’s meet Scala

Figure 1: Martin Odersky, the creator of Scala

Scala programming language is:

- general purpose
- strongly statically typed (with type inference)
- compiling primarily to JVM 8+ bytecode[1]
- interoperable with Java 8+

[1] `scala.js` compiles to JS; `scala native` targets LLVM; jdk 9+ compatibility might require support; there is a Scala REPL.

Scala supports

- OOP
- FP

```
val value : Int    = 2;
val text : String = "hello";
val value : Int    = 2
val text : String = "hello"
val value = 2
val text = "hello"
```

```scala
val value = 2
val text = "hello"

print(text)
print(s"$value-$text") // string interpolation
print(s"${value*2}")
```

```
var value = 2
val text = "hello"

value = 4
text = "hello" // !!! forbidden
```

```
def square(x : Int) : Int = {
  return x*x
}
```

```scala
def square(x : Int) : Int = x*x
// evaluation strategy
```

```
def square(x : Int) = x*x
def abs(x : Int) : Int =
  if (x >= 0)
    x
  else
    -x

// if statement vs if control flow
```

```
def id(x : Int) : Int = {
  if (x >= 0) {
    print(s"$x")
  }
  x
}


id(-2)
id(2)
```

```
def square(x : Int) = x*x

square(2+3+4) // <-- how is this evaluated
```

```
def log(message : String) = ???
def log(message : String) : Unit = ???

// unit type
```

```
class Logger {
  def log(message : String) = ???
}
```

```scala
trait Logger {
  def log(message : String)
}

class PrintLogger extends Logger {
  override def log(message: String) = {
    print(message)
  }
}
class ProductionLogger() extends Logger {
  override def log(message: String) = {
  }
}
```

```
val logger : Logger = ???
// ...
logger.log("hello")
```

```
val logger : Logger = ???
// ...
logger.log("hello" + sqrt(2-3+10))
```

```
trait Logger {
  def log(message : => String)
 // call by name vs call by value
}
```

```
def squareCallByName(x : => Int) = x*x
```

```
val function : Int => Int = square
```

```
def f(x : Int, y : Int) : Int = x + y
def g(y : Int) : Int = f(1,y)
def h(y : Int) : Int => Int = f(_,y)

f(2,3)
g(3)
h(2)(3)
```

```scala
def div(x : Int) : Int = 1/x

div(1)
div(2)
div(0) // java.lang.ArithmeticException: / by zero
```

```
def div(x : Int) : Integer = {
  if ( x != 0 ) 1/x else null // !!!
}
```

```
def div(x : Int) : Option[Int] = {
  if ( x != 0 ) Some(1/x) else None
}
```

```
def div(x : Int) : Option[Int] = {
  if ( x != 0 ) Some(1/x) else None
}

div(0).map(square)
```

```
def div(x : Int) : Option[Int] = {
  if ( x != 0 ) Some(1/x) else None
}

def anotherFunction(x : Int) : Option[Int] = ???


div(0).map(square).flatMap(anotherFunction)
```

Tools

- IDE: intellij with Scala plugin
- Build tool: sbt
- Tests: scalatest vs specs2

The Play framework

What is play?

// not really FP // based on Options / Futures, must be already introduced at this point

Structure of a Play project

g8 template

Sbt configuration

Route file

A taste of contract driven development:

- apiary / blueprint
- dredd testing (local / staging)

Controller

Dependency injection (imperative): Guice

Controllers must be stupid, keep them simple (this is general architectural rule, no scala specific, no play specific)

Clients with play : write data classes and write the client

Json parsing

From futures to HTTP responses

Hydration

Learn more

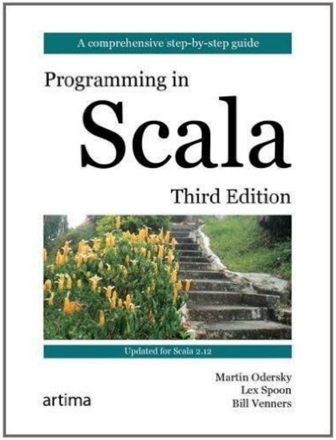


Figure 2: The reference guide

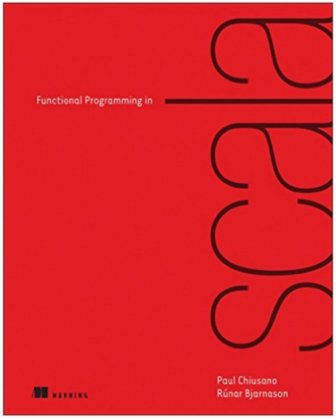
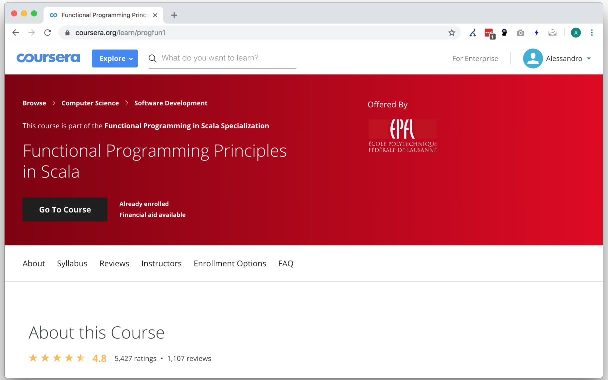


Figure 3: Gym to master FP in Scala



This course is part of the **Functional Programming in Scala Specialization**

Functional Programming Principles in Scala

About this Course

4.8 5,427 ratings • 1,107 reviews

Figure 4: Functional programming principles in Scala MOOC by Martin Odersky

Online exercises:

- `https://www.scala-exercises.org/`
- `http://www.scalakoans.org/`

Questions?