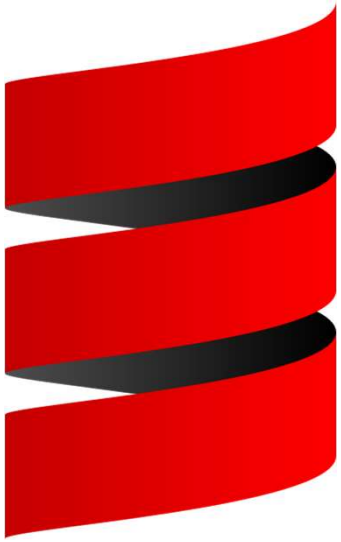
Refreshing Your Knowledge: Scala Fundamentals for This Course

Xavier Morera

HELPING DEVELOPERS UNDERSTAND SEARCH & BIG DATA

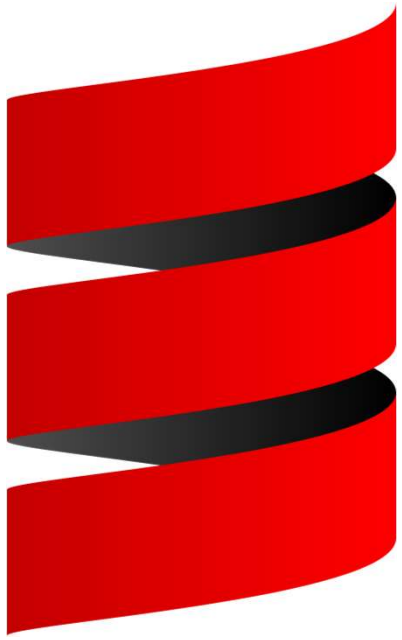
@xmorera www.xaviermorera.com

Refreshing Your Knowledge: Scala Fundamentals for This Course



History of Scala

```
Using Scala version 2.11.8
```



Design started in 2001

- École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
- Martin Odersky

Internal release in 2003, publicly in 2004

- Version 2 in 2006

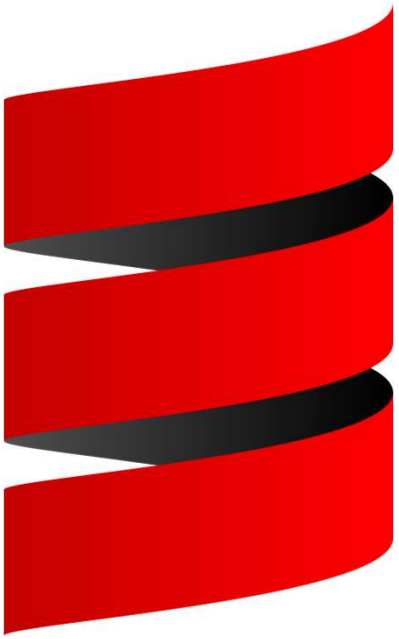
Research grant in 2011 by ERC

- Launched Typesafe (now Lightbend)

Scalable language

What does it mean?

Scala Overview



General-purpose programming language

- Object oriented
- Functional programming

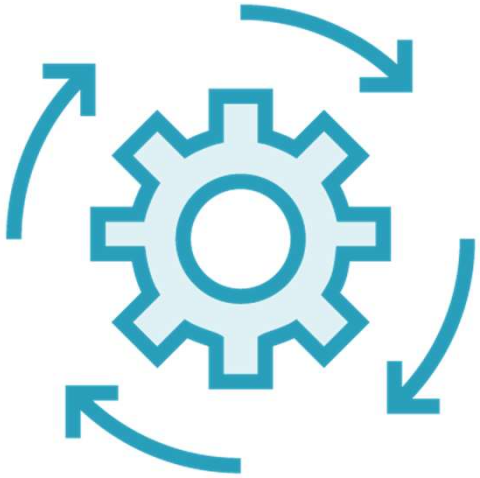
Statically typed

Compiled language

- Java bytecode
- Runs on the JVM
  - Interoperability with Java

```
Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_144)
```

Building & Running Scala Applications



Scala is a compiled language

- Compiled into bytecode
- Executed in the JVM

Can also "feel" interpreted

Building & Running Scala Applications

scalac + scala

Compile and run simple programs

sbt

Simple Build Tool
For more complex applications

REPL

Interactive expressions

Building & Running Scala Applications

scalac + scala

Compile and run simple programs

sbt

Simple Build Tool
For more complex applications

REPL

Interactive expressions

scalac + scala

Compile and run simple programs

Compile & run simple programs

- `scalac` for compiling
- `scala` for running

Process is very similar to Java

- Generates bytecode
- Multiple class files
- Specify `classpath`

```
scalac Count_Committers.scala

scala Count_Committers
```

Using scalac & scala

Very simple application

Compile with `scalac` and review created files

Run with `scala`

sbt

Simple Build Tool
For more complex applications

Easy and convenient to use scalac & scala

Except when your project starts to grow

Use sbt, or Simple Build Tool

- Familiar if you have used Maven

Used widely in open source

sbt

Simple Build Tool
For more complex applications

Describe your project with a build definition

- build.sbt
- Subprojects, with settings and imports

Specify dependencies

Predefined folder structure

- Source, resources and tests

Packed with features

Features of sbt

- Little or no configuration required for simple projects
- Scala-based build definition that can use the full flexibility of Scala code
- Accurate incremental recompilation using information extracted from the compiler
- Continuous compilation and testing with triggered execution
- Packages and publishes jars
- Generates documentation with scaladoc
- Supports mixed Scala/Java projects
- Supports testing with ScalaCheck, specs, and ScalaTest. JUnit is supported by a plugin.
- Starts the Scala REPL with project classes and dependencies on the classpath
- Modularization supported with sub-projects
- External project support (list a git repository as a dependency!)
- Parallel task execution, including parallel test execution
- Library management support: inline declarations, external Ivy or Maven configuration files, or manual management

```
mkdir -p src/{main,test}/{java,resources,scala}
mkdir lib project target
find .
```

The sbt Directory Structure

Predefined directory structure

- Same one used by Maven

Code (Java | Scala), resources and test

```
 sbt package
scala simple-project_2.11-1.0.jar
```

Using sbt

Use sbt with a simple exercise using Scala

Create build definition, build.sbt

Package and ready to run

The Scala Shell: REPL

REPL

Interactive expressions

scalac

Compile and run simple programs

sbt

Simple Build Tool
For more complex applications

REPL

Interactive expressions

- Read Evaluate Print Loop
- Interactive shell
- Immediate feedback, including errors
- Typeahead (tab completion)
- Extremely useful
- Suitable for exploration and testing
- Not the best for production

```
scala
```

Starting the REPL

Simply start with scala

Check the version

Ready to explore Scala

In Scala
everything is an object

```
40

40.getClass

40 + 2

// resxx can be reused

40.+(2)
```

Everything Is an Object

Numbers are objects

Get the class

And perform an operation

```
x = 41

var x = 41

x = x + 1

x = "forty one"

var x: Int = 41
```

Variables

Need to define variables, using var

Some are mutable, others immutable. But always statically typed

Types not required when compiler can infer type

```
var platform: String = "Spark";
var platform: String = "Spark"
platform
val fixedPlatform = "Spark"
fixedPlatform = "Apache Spark"
platform = "Apache Spark"
println(fixedPlatform)
```

Variables

Semicolon is optional

For fixed variables, use val

Important with Spark

```
true
false
platform = "Apache Spark"
platform == "Spark"
10 * 100
10 * (100 * 1000) == (10 * 100) * 1000
10 * 100 * 1000 == 1000 * 100 * 10
10 - 100 - 1000 == 1000 - 100 - 10
:history
```

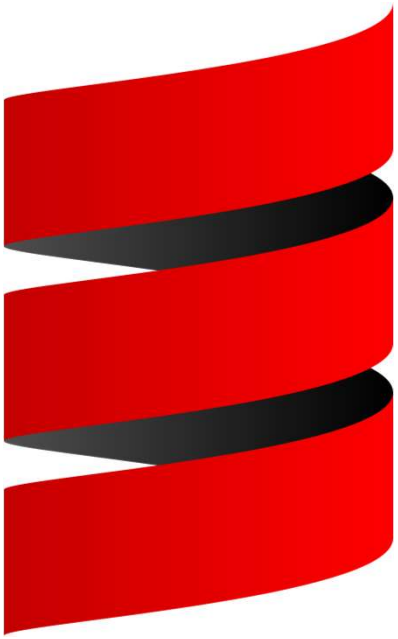
Variables and Operations

Strings, integers, booleans...

Assignment and comparison

Operations: associative and commutative

Syntax

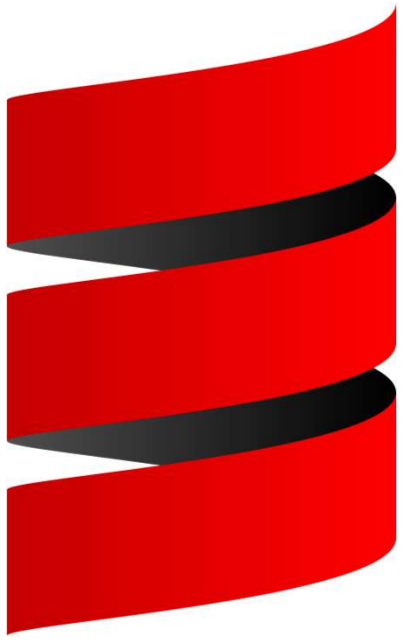


Case sensitive

CamelCase

- ClassName
- methodNames
- Program name match object name

Identifiers



Starts with a letter or underscore

Followed by letters, digits or underscores

- Don't use $
- + means .+()
- i.e. :- as $colon$minus

Don't use reserved words

decomposes into the three identifiers `big_bob`, `++=`, and **`def`**. The rules for pattern matching further distinguish between *variable identifiers*, which start with a lower case letter, and *constant identifiers*, which do not.

The '`$`' character is reserved for compiler-synthesized identifiers. User programs should not define identifiers which contain '`$`' characters.

The following names are reserved words instead of being members of the syntactic class `id` of lexical identifiers.

abstract	**case**	**catch**	**class**	**def**
do	**else**	**extends**	**false**	**final**
finally	**for**	**forSome**	**if**	**implicit**
import	**lazy**	**match**	**new**	**null**
object	**override**	**package**	**private**	**protected**
return	**sealed**	**super**	**this**	**throw**
trait	**try**	**true**	**type**	**val**
var	**while**	**with**	**yield**	
`_`	`:`	`=`	`=>`	`<-`	`<:`	`<%`	`>:`	`#`	`@`

The Unicode operators \u21D2 '⇒' and \u2190 '←', which have the ASCII equiva-

```
'c'
"string"
"string".length
"string".take(1)
"string".take(4)
val name = "Xavier"
println(s"Hello, $name")
```

More on Types, Functions, and Operations

Chars and strings

Remember... objects

Format strings using string interpolation

```
"""This is a
"multiline" string"""
<xml>this is an xml sample</xml>
```

More on Types, Functions and Operations

Multiline strings with """

XML support

```
val myList = List[String] ("Xavier", "Irene")

val myMap = Map[String, String] ("Xavier" -> "Author",
"Irene" -> "QA")

myList.getClass

Seq(1, 2, 3, 4)
```

More on Types, Functions and Operations

Generics used

- Useful in collections

Mutable and Immutable collections

```
def sumOfTwoValues(x: Int, y: Int): Int = {
  x + y
}
```

Expressions, Functions and Methods

Define a function using def

- Parameters
- Return value
- But return keyword is optional

Difference Between Function & Method

Method	Function
Part of a class	An object
Name	Traits to represent these objects
Signature	Parameters
def	val

```
val f1 = (x:Int) => x+1

f1.getClass

f1(2)
```

Functions

Quite frequently used

Named and anonymous functions

```
class PostC(Id: Integer, PostTypeId: String, Score:
Integer, ViewCount: Integer, AnswerCount: Integer,
OwnerUserId: String)
```

Classes

Blueprint for creating objects

Represent objects

```
case class Post (Id: Integer, PostTypeId: String, Score:
Integer, ViewCount: Integer, AnswerCount: Integer,
OwnerUserId: String)

val post: Post = Post(1, "1", 29, 34, 12, "32")

Post.Id
```

Case Classes

Immutable classes

What we will use in Spark with Datasets

```
val x = 9
if (x < 10) "Less" else "More"
val x = 11
if (x < 10) "Less" else "More"
for ( i <- 0 to 10 ) println(i)
var i = 0
while (i < 15) {
   println(i)
   i = i + 1
}
```

Flow Control

Basics of programming is if then

To iterate use for

And loop using while

```
Seq(1, 2, 3).map(x => x+1)
```

```
Seq(1, 2, 3).filter(x => x %2 == 1)
```

Functional Programming

Computation as the evaluation of mathematical functions

- Avoid changing state of mutable data

Map applies to each element

Filter removes elements where function evaluates to false

spark2-shell: Spark in the Scala Shell

```
scala>
```

- Scala shell
- Use the Spark Scala API
- Get all of Scala
- In the Spark world

```
:q
# spark2-shell
```

Spark in the Scala Shell

Exit the Scala shell

Now take a look at spark2-shell

Takeaway

- Scalable Language
- Started in 2001, publicly in 2004
- Statically typed
- Compiled language
- Runs on the JVM

Takeaway

Build and run

- scalac and scala
- sbt
- REPL

Takeaway

Everything is an object

Char, String, Integer, List, Map, Tuple...

val vs. var

Methods and functions

Flow control

- If, for, while

Takeaway

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.2.0.cloudera1
      /_/
```