First day of Scala

COLLABORATORS			
	TITLE : First day of Scala		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Bart Schuller	2011-09-21	

Contents

1 Today's menu

- Introduction to the Scala language
- Scala tools and resources
- Setting up your environment
- Excercises

Actually, we're going the agile route and iterate these subjects.

This workshop on github (please switch branch):

https://github.com/bartschuller/scala-workshop

2 About

- what
- when
- who

2.1 What is Scala?

A programming language which is

- Statically typed
- Object Oriented
- Functional
- and more
- focus on concurrency
- Open Source
- runs on JVM
- is compiled
- has a REPL

2.2 History

Written by Martin Odersky, who also added generics to Java and wrote the current java compiler.

- Design started in 2001
- first release in 2003
- 2.0 in 2006
- Current version is 2.9.1

Odersky is professor at EPFL in Switzerland, where Scala releases come from.

2.3 Commercial backing

The company Typesafe was founded in 2011 by Odersky and others to promote and support Scala and the Akka middleware framework.

Advisors

- James Gosling
- Doug Lea

Investors

- Greylock Partners
- founders of VMWare

3 Language introduction

- general syntax
- val, var, def
- types

3.1 Values

Scala has expressions that look the same as in most other languages. You can store the value of an expression like this:

```
val subtotal = 42.0
val tax = 1.19
val fees = 10
val total = subtotal*tax + fees

println("Please pay us €"+ total +" promptly.")
```

3.2 Values are forever

A `val` never changes value once initialized, and initialization has to happen at the declaration site.

```
val total = 42.0
total = total * 1.19 // ❶
```

❶ Compilation error

3.3 Variables, if you must

A `var` is like the variables from most other programming languages (but not math!)

Scala programmers try to keep their use to a minimum.

```
var total = 42.0
total = total * 1.19
```

3.4 methods

A method or a function uses the `def` keyword:

```
def sayit = println("it!")

def taxed(untaxed: Double) = untaxed * 1.19

val total = taxed(subtotal) + fees

def isEven(n: Int) = {
    if (n % 2 == 0)
        true
    else
        false
}
```

Note the equals sign.

What else do you notice?

3.5 Types

Values, variables and return types have optional type annotations.

```
val i = 10
val j: Int = 20
var k: String = _

def m: Unit = println("no return value")

def n { // ❶
    println("also Unit (’void’ in java)")
}
```

❶ No equals sign means it’s Unit

3.5.1 Built-in types and literals

Byte
`1, -127`

Short
`32767, 0xff, 0777`

Int
`10, -46565`

Long
`2744L, 5845776520L`

Boolean
`true, false`

Float
`0.01F, -1e8F`

Double
`3.002, 34D`

Char

```
'a', '€'
```

String

```
"Note the \"escapes\"", """multi-line, embedded "quotes""""
```

3.5.2 Supertypes

Not that important.

- Any
- AnyVal
- AnyRef

3.5.3 Compound types

- Tuples
- Arrays
- Collections

Tuples

```
def divide(a: Int, b: Int): (Int, Int) =
    (a / b, a % b)

val (result, remainder) = divide(72, 30)

val asl = (30, true, "Rotterdam")

println("Welcome to %s".format(asl._3))
```

Note

Even though you can make very flexible compound types, the following will give a compile error because each element still has a static type:

```
val halfsex = asl._2 / 2
```

error: value / is not a member of Boolean

Semicolons

Scala *infers* semicolons at the end of a line where that line could validly end.

Problems can arise.

```
a = 1 + 1 + 1      // ❶ don't do this
    + 1            // ❷
b = 1 + 1 + 1 +    // ❸ do this
    1
```

❶ Statement looks finished at the end of the line, so compiler infers a semicolon. **a == 3**

❷ New statement: throw away positive one

❸ End with an operator, the compiler will expect more and continues looking at the next line. **b == 4**

Arrays

```
val blob: Array[Byte] = fetchBlob
val first = blob(0)

def fetchBlob = Array[Byte](0, 1, 2, 3)
def newArray = new Array[String](10)
```

Collections

We'll get to collections once we've covered what they're made of: classes and objects.

But first, it's time t get our hands dirty.

4 Installing the Tools

- scala
- git
- sbt

4.1 Installing Scala

Recommendation: skip the standalone scala compiler, go straight to the build tool.

`brew install scala [--with-docs]`

Install sbt instead.

4.1.1 SBT

- Simple Build Tool
- Downloads deps (a.k.a. the internet), builds, tests
- *Using* it is simple
- https://github.com/harrah/xsbt/wiki/
- Watch out, 0.10/0.11 is latest, not compatible with 0.7 or earlier

4.1.2 Installing sbt

We use sbt 0.10.1 for this workshop.

Mac with HomeBrew
`brew install sbt`

Everything else
Use `sbt` in the root of the workshop project from github and peruse https://github.com/harrah/xsbt/wiki/Setup at your leasure.

Optionally create `~/.sbtconfig`, mine contains

```
SBT_OPTS="-Dsbt.boot.directory=$HOME/.sbt/boot/
-XX:+CMSClassUnloadingEnabled -server -Xss2m -Xms128m
-Xmx1024m -XX:MaxPermSize=512M -Dfile.encoding=UTF-8"
```

4.1.3 Using sbt

Existing project (directory contains `build.sbt` and/or `project/*.scala`):

- Open a terminal
- cd to the project directory
- type `sbt` (or `./sbt` for the workshop)

New project:

- Make empty project directory, cd to it
- `mkdir -p src/main/scala src/test/scala`
- Optionally copy and change this workshop's `build.sbt`

Other options include

- giter8
- sbteclipse *create-src* option
- np sbt plugin

4.1.4 SBT commands

- `compile`
- `test`
- `run`
- `~test` — keep testing while you make changes
- `console` — finally a Scala REPL

4.1.5 Exploration time

Start the console and type some expressions. Try the TAB completion. Define some functions.

Notice that every expression gets assigned to a new variable name `res0` etc., so creating a `val` is optional.

If you want to paste larger snippets then start by typing `:paste`, paste your code, then type *Ctrl*-D.

5 Language Intro part 2

- code structures
- collections and functions
- exceptions and pattern matching
- for-comprehensions

5.1 Organizing code

- Classes
- Objects
- Traits
- Namespaces
- Case Classes

5.1.1 Classes

The bread and butter of every program.

```
class Person(val name: String, var address: Address) {
  var moved = false
  override def toString = "%s from %s".format(name,
                  address.municipality)
  def move(newAddress: Address) {
    address = newAddress
    moved = true
  }
}

class Address(val municipality: String, val country: String) {
    override def toString = "%s, %s".format(municipality,
                  country)
}
```

Using classes looks pretty familiar.

```
val bart = new Person("Bart", new Address("Heemraadssingel 70",
                                    "Rotterdam", "Holland"))
println(bart)
bart.move(new Address("Oudemansstraat",
                      "Den Haag", "The Netherlands"))
println(bart)
```

5.1.2 Objects

Mr. Singleton

```
object Person {
  private var peopleCount = 0
  def total = peopleCount
  def apply(name: String, address: Address) = {
    peopleCount += 1
    new Person(name, address)
  }

  def swapHomes(a: Person, b: Person) {
    val aHome = a.address
    a.address = b.address
    b.address = aHome
  }
}
```

Out with the new

```
val bart = Person("Bart", Address("Oudemansstraat",
                          "Den Haag", "The Netherlands"))
val paco = Person("Francisco", Address("Zuid",
                               "Rotterdam", "Holland"))
Person.total should_== 2
Person.swapHomes(bart, paco)
bart.address.municipality should_== "Rotterdam"
```

I almost forgot

```
object MainProgram {
  def main(args: Array[String]) {
    println("Hello, world!")
  }
}
```

Or shorter

```
object HelloWorld extends App {
  println("Hello, world!")
}
```

5.1.3 Traits

```
trait Named {
  def name: String
}

trait Ordered[A] {
  def compare(that: A): Int
}

class Person(val name: String) extends
                Named with Ordered[Person] {
  def compare(that: Person) = name.compare(that.name)
}
```

```
var n: Named = new Person("Bart")

n = new Named { def name = "name " + math.random }
n = new Named { val name = "Bart" }
```

- Traits can include concrete methods
- Create mixin types on the spot

```
trait Damned extends Named {
  def damned = name.reverse
}

val bart = new Person with Damned
bart.damned
```

5.1.4 Packages and visibility

- packages
- imports
- privacy
- import whatever
- wherever

```
package com.lunatech.helloworld

import com.lunatech.handy._

object Hello extends App {
  Handy.foo()

  import Handy._
  foo()
}
```

- default is public
- ultra-privacy is available

```
package com.lunatech.foo

class Foo(private var i: Int) {
  private[this] val orig = i
  protected def printOrig = println(orig)
  def otherI(o: Foo) = o.i

  // error: value orig is not a member of Foo
  def otherOrig(o: Foo) = o.orig
}
```

```
val foo = new Foo(7) { def gimme = printOrig }
foo.gimme
```

5.1.5 Case Classes

```
case class Person(name: String, address: Address)
case class Address(municipality: String, country: String)

val bart1 = Person("Bart", Address(
                     "Den Haag", "The Netherlands"))
val bart2 = bart.copy(address = Address(
                               "Rotterdam", "Holland"))
```

6 Installing more Tools

- IntelliJ
- or Eclipse
- Scala plugin
- sbt plugin for generating intellij/eclipse files

6.1 IntelliJ IDEA

- Community Edition from http://www.jetbrains.com/idea/
- Scala Plugin: *Preferences...* → *Plugins*

sbt plugin: https://github.com/mpeltonen/sbt-idea/
or rather:

```
mkdir -p ~/.sbt/plugins
edit ~/.sbt/plugins/build.sbt
```

Put this in (including the empty line)

```
resolvers += "sbt-idea-repo" at "http://mpeltonen.github.com/maven/"

libraryDependencies += "com.github.mpeltonen" %% "sbt-idea" % "0.10.0"
```

6.2 Collections

- List
- Vector
- Option
- Map

6.2.1 List

Constructing lists

```
val l1 = List(1, 2, 3)
val l2 = 2 :: 3 :: Nil
val l3 = 1 :: l2
l1 should_== l3
val a1 = Array(1, 2, 3)
val l4 = a1.toList
l1 should_== l4
```

6.2.2 Matching on List

Deconstructing lists

```
def listLen[T](l: List[T]): Int = {
  l match {
    case x :: xs => 1 + listLen(xs)
    case _ => 0
  }
}

listLen(List(1, 2, 3)) should_== 3
```

Also note `listLen` is a generic function: it works not just for `List[Int]` but for any `List[T]`.

6.3 For comprehensions

Scala doesn't have `for` loops, but it does have the `for` keyword. Let's explore what it does.

```
for (i <- 1 to 10) { println(i) }
```

Spoiler alert: the next slides will show you my solution to problem number 9 of the Euler project.

6.3.1 Euler problem 9

A Pythagorean triplet is a set of three natural numbers,
a < b < c, for which,

$a^2 + b^2 = c^2$

For example, $3^2 + 4^2 = 9 + 16 = 25 = 5^2$.

There exists exactly one Pythagorean triplet for which
a + b + c = 1000. Find the product *abc*.

6.3.2 Analysis

All are Natural numbers, so > 0
a < b < c
$a^2 + b^2 = c^2$
a + b + c = 1000

- a, b and c are smaller than 1000
- c = 1000 - a - b
- let's just try all a and b below 1000

6.3.3 for

```
def euler9 = {
  val ans =
    for (b <- 2 to 1000;  // ❶
         a <- 1 to b;     // ❷
         c = 1000 - a - b
             if c*c == a*a + b*b)
      yield a*b*c
  ans.head
}
```

❶ *b* is a fresh variable, taking on the succesive values 2 to 1000 inclusive

❷ This is a loop within a loop, *a* loops from 1 to the current value of b, so we generate all possible combinations of a and b.

```
def euler9 = {
  val ans =
    for (b <- 2 to 1000;
         a <- 1 to b;
         c = 1000 - a - b // ❶
             if c*c == a*a + b*b) // ❷
      yield a*b*c
  ans.head
}
```

1. Assignment just gives a name to an expression, we still loop just over b, then a.
2. An `if` statement can appear anywhere to add a constraint to the combination of values. If not met, then inner loops and the body are skipped.

6.4 Exceptions

```
val x = List(1, 2)
try {
  x.tail.tail.head
  failure("Should have thrown")
} catch {
  case _: NoSuchElementException => success
  case e => failure("Unexpectedly got "+e.toString)
}
```

7 Where to go from here

Martin Odersky classifies the journey to Scala mastery as follows:

- Level A1: Beginning application programmer
  - Java-like statements and expressions: standard operators, method calls, conditionals, loops, try/catch
  - class, object, def, val, var, import, package
  - Infix notation for method calls
  - Simple closures
  - Collections with map, filter, etc
  - for-expressions
- Level A2: Intermediate application programmer
  - Pattern matching
  - Trait composition
  - Recursion, in particular tail recursion
  - XML literals
- Level A3: Expert application programmer
  - Folds, i.e. methods such as foldLeft, foldRight
  - Streams and other lazy data structures
  - Actors
  - Combinator parsers
- Level L1: Junior library designer
  - Type parameters
  - Traits
  - Lazy vals
  - Control abstraction, currying
  - By-name parameters

- Level L2: Senior library designer
  - Variance annotations
  - Existential types (e.g., to interface with Java wildcards)
  - Self type annotations and the cake pattern for dependency injection
  - Structural types (aka static duck typing)
  - Defining map/flatmap/withFilter for new kinds of for-expressions
  - Extractors
- Level L3: Expert library designer
  - Early initializers
  - Abstract types
  - Implicit definitions
  - Higher-kinded types

8 The Scala community

- Twitter: https://twitter.com/#!/BartSchuller/scala
- scala-user list: https://groups.google.com/forum/#!forum/scala-user
- Scala Types podcast: http://itunes.apple.com/us/podcast/the-scala-types/id443785200

News feeds

- Scala News: http://www.scala-lang.org/rss.xml
- Reddit Scala http://reddit.com/r/scala/.rss
- http://implicit.ly/ (release announcements for libraries)
- Scala Scoop: http://scalascoop.tumblr.com/rss (mostly dupes though)

8.1 Interesting Scala projects

Scalaz
: Hardcore Haskell-style functional programming concepts.

Lift
: The first well-known Scala web-framework. Best for stateful, interactive sites.

9 The End

Write code, have fun, be awesome