

CURSO DE PROGRAMACION SCALA

Sesión 1

Sergio Couto Catoira
ingscc00@gmail.com

Información del curso

› Repositorio

- <https://github.com/SCouto/cursoScala>

› Slack => amaris-scala.slack.com

› Herramientas

- IntelliJ
- Consola (cmdr) con editor de texto (vim, gedit, notepad++ etc..)
- SBT

SBT - Instalación

› Herramienta de construcción de proyectos

- Se configura con un fichero build.sbt

› Instalación en Windows

- Descargar de <http://www.scala-sbt.org/0.13/docs/Installing-sbt-on-Windows.html>

› Instalación en Linux

- `echo "deb https://dl.bintray.com/sbt/debian /" | sudo tee -a /etc/apt/sources.list.d/sbt.list`
- `sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 2EE0EA64E40A89B84B2DF73499E82A75642AC823`
- `sudo apt-get update`
- `sudo apt-get install sbt`

Índice

- › SBT
- › REPL
- › Un poco sobre sintaxis
- › Variables
- › Colecciones
- › Pattern matching
- › Recursividad
- › Orden superior
- › Testing unitario
- › Ejercicios

SBT - Opciones

› Opciones

- compile
- run
- test
- console: abrir REPL (Read-Evaluate-Print-Loop) de scala
- Reload

› Identifica cambios de código. Si lanzamos run tras haber tocado algo, ejecutará compile antes.

› Interactivo (vírgula antes de la opción)

- ~compile
- ~test

REPL

- › Consola interactiva de scala
 - Autocompletado
 - Similar a la worksheet de IntelliJ
 - Permite hacer imports y cualquier sentencia de scala
- › Se arranca con
 - scala
 - sbt console
- › Útil para “trastear”

```
scouto@:cursoScala(master)$ sbt console
[info] Loading project definition from /home/scouto/
[info] Set current project to cursoScala (in build f
[info] Starting scala interpreter...
[info]
Welcome to Scala 2.12.1 (OpenJDK 64-Bit Server VM, J
Type in expressions for evaluation. Or try :help.

scala> val x =1
x: Int = 1

scala> x
res0: Int = 1
```

Scala

- › Multiparadigma
 - Programación funcional
 - Orientación a objetos
- › Sintaxis más sencilla
 - Inferencia de tipos
 - Evita caracteres innecesarios

Sintaxis : Ejemplo 1

› Main java

```
public class MyApp {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

› Main scala

```
object MyApp extends App {  
    println("Hello World!")  
}
```

› Función java

```
public Boolean hasUpperCase(String word) {  
    Boolean hasUpperCase = false;  
    for (int i = 0; i < word.length(); i++) {  
        if (Character.isUpperCase(word.charAt(i))) {  
            return true;  
        }  
        i += 1;  
    }  
    return false;  
}
```

› Función scala java-like

```
def hasUpperCase(word:String): Boolean = {  
  
    var wordHasUpperCase = false  
    var i = 0  
    while (i < word.length && ! wordHasUpperCase) {  
        if (Character.isUpperCase (word.charAt(i))) {  
            wordHasUpperCase = true;  
        }  
        i += 1;  
    }  
}
```


Sintaxis : Ejemplo 2 y 3

```
word.exists(x => x.isUpper)
```

```
word.exists(_.isUpper)
```

Variables

› Val vs var

- Value => immutable (final)
- Variable => mutable

› Sintaxis

- No es necesario indicar tipo de parámetro => Inferencia de tipos

› getClass

› Interpolación de Strings

- `s"Mi edad es ${x}, por lo que naci en ${currentYear-x}"`

Tipos

Data Type	Precision
Byte	8 bit signed value. Range from -128 to 127
Short	16 bit signed value. Range from -32768 to 32767
Int	32 bit signed value. Range from -2147483648 to 2147483647
Long	64 bit signed value. Range from -9223372036854775808 to 9223372036854775807
Float	32 bit IEEE 754 single-precision float
Double	64 bit IEEE 754 double-precision float
Char	16 bit unsigned Unicode character
String	A sequence of Chars
Boolean	Either the literal true or the literal false

Funciones

› Ejemplo:

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

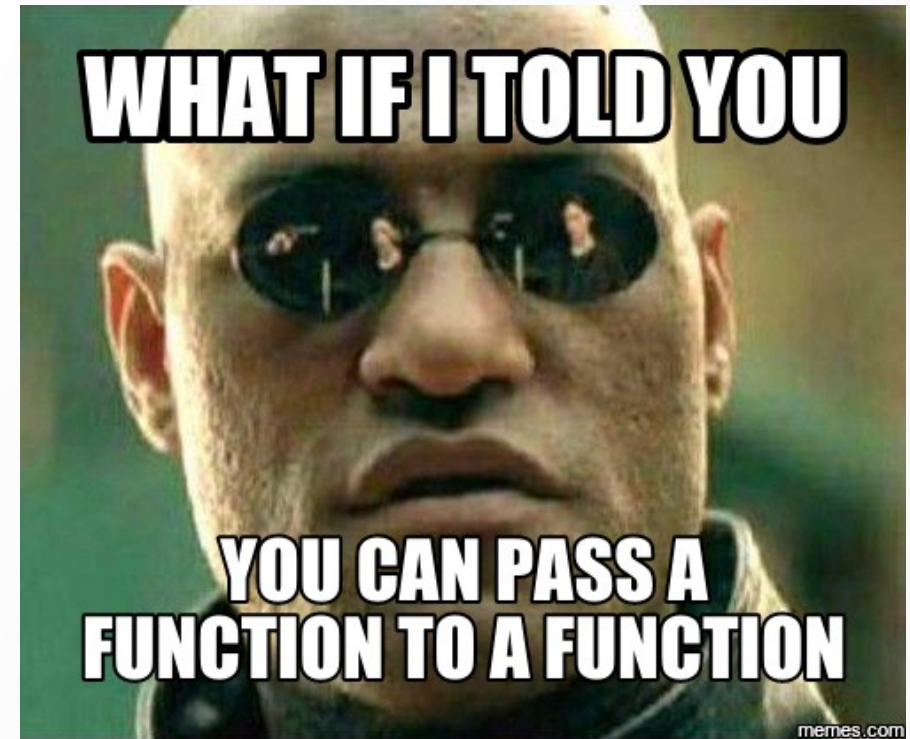
```
def sum2(x:Int, y:Int): Int = x+y
```

› Ejercicio

- Apoyándote en la función sum definida anteriormente, define las funciones multiplicacion, resta y división

Conceptos sobre funciones

- › Recursivas
- › Puras \Rightarrow sin efectos laterales
- › Anónimas: $(x: \text{Int}) \Rightarrow x + 1$
- › Orden superior
- › Parciales
- › Ejercicio:
 - Las funciones definidas en el ejercicio anterior son prácticamente idénticas. ¿Podrías generalizarla en una única función?



Colecciones

› Array

- `Val a = Array("OneString", "Another")`
- Lectura: `a(0)`
- Asignación: `a(0) = "thirdString"`

› List

- `List(1, 2, 3)`
- `Nil` => lista vacía
- `tail`, `reverse`, `head`, `dropRight`, `take`, `mkString`, `foreach`, `contains` y más
- Concatenar => `:::`
- Prepend => `::`

Pattern matching

- › “Similar” al switch de Java

```
x match {  
  case 1 => "one"  
  case 2 => "two"  
  case _ => "many" //default  
}
```

- > Permite descomponer elementos

```
list match {  
  case x::t => x  
  case Nil => Nil  
case _ => list  
}
```


Funciones de orden superior

- › Funciones que reciben como parámetro otra función
 - `def operate(x: Int, y: Int, f:(Int, Int) => Int) = f(x,y)`
- › Algunas conocidas
 - Map vs foreach
 - filter
 - flatMap
 - Exists
 - takeWhile, dropWhile

Funciones recursivas

```
def factorial(n: Int): Int = {  
    If (n<=0) n  
    else n* factorial(n-1)  
}
```

- › Ojo con tail calls
 - `@annotation.tailrec`
 - nested functions

Testing unitario

› ScalaTest

- FlatSpec => Permite mezclar el test con texto que define el comportamiento esperado
- Matchers => Establece condiciones de éxito

```
"sum" should "work with natural numbers" in {  
  sum(2,3) should be (5)  
}
```

Ejercicios

- › La función creada en esta sesión, que generaliza las operaciones sólo es válida para enteros, podrías generalizarla:
 - `def operate[A]...`
 - `def operate[A, B]`
- › Define una función recursiva que devuelva el máximo de una lista de enteros
 - `def max(list: List[Int]): Int = ???`
- › Función que devuelve el segundo elemento de una lista
 - `def second(list: List[Int]) : Int = ???`
- › Función que devuelva el nth elemento de una lista
 - `def nth(list: List[Int], n: Int): Int = ???`