Computación y estructuras discretas II

Unidad Programación Funcional: Introducción a Scala

Juan Marcos Caicedo Mejía – jmcaicedo@icesi.edu.co

Departamento de Computación y Sistemas Inteligentes
Facultad Barberi de Ingeniería, Diseño y Ciencias Aplicadas
Universidad ICESI

Material adaptado del material original de DataCamp - Introduction to Scala

Tabla de contenido

Contenido

Introducción a Scala

Aprender haciendo

- ¿Qué es Scala?
- ¿Por qué usar Scala?
- ¿Quién usa Scala?

¿Qué es Scala?

Scala es un lenguaje de programación de propósito general que proporciona soporte para programación funcional y un sistema de tipos estático fuerte.
Fue diseñado para ser conciso, y muchas de sus decisiones de diseño buscan responder a críticas hechas a Java.

¿Qué es Scala?

El código fuente en Scala está diseñado para compilarse a *bytecode* de Java, de modo que el ejecutable resultante se ejecute en la Máquina Virtual de Java (JVM).

¿Por qué usar Scala?

SCAlable LAnguage (Lenguaje Escalable)

Flexible y conveniente

Flexible
Scala permite agregar nuevos tipos, colecciones y estructuras de control que se sienten como si fueran parte del lenguaje.

Conveniente
La biblioteca estándar de Scala ofrece tipos, colecciones y estructuras de control predefinidas muy útiles.

¿Quién usa Scala?

Roles

- Ingeniero de Software
- Ingeniero de Datos
- Científico de Datos
- Ingeniero de Machine Learning

Industrias

- Finanzas
- Tecnología
- Salud
- Y muchas más...

Contenido

¿Qué es Scala?

Scala es un lenguaje de programación de propósito general que proporciona soporte para programación funcional y un sistema de tipos estático fuerte.
Fue diseñado para ser conciso, y muchas de sus decisiones de diseño buscan responder a críticas realizadas a Java.

El código fuente en Scala está pensado para compilarse a *bytecode* de Java, de modo que el código ejecutable resultante se ejecute sobre la **Máquina Virtual de Java (JVM)**.

¿Qué es Scala?

Scala combina programación orientada a objetos y programación funcional en un único lenguaje conciso y de alto nivel.

Los tipos estáticos de Scala ayudan a evitar errores en aplicaciones complejas.

Sus entornos de ejecución sobre la JVM y JavaScript permiten construir sistemas de alto rendimiento con acceso sencillo a grandes ecosistemas de librerías.

`https://www.scala-lang.org/`

Scala combina OOP y FP

Scala combina programación orientada a objetos y programación funcional en un lenguaje conciso y de alto nivel.
Los tipos estáticos ayudan a evitar errores en aplicaciones complejas.
Se ejecuta sobre la JVM y también puede compilar a JavaScript.

Scala es orientado a objetos

- Cada valor es un objeto
- Cada operación es una llamada a método

```
val sumA = 2 + 4
sumA: Int = 6
```

Operaciones como métodos

```
val sumA = 2.+(4)
sumA: Int = 6
```

Scala es funcional

1. Las funciones son valores de primera clase.
2. Las operaciones deberían transformar entradas en salidas, en lugar de modificar datos directamente.

Ejemplo en el intérprete

```
scala> 2 + 3
res0: Int = 5
res0 * 2
res1: Int = 10
```

Impresión en consola

```
scala> println("Aprendiendo Scala!")
Aprendiendo Scala!
```

Contenido

Dos tipos de variables

val (inmutable)

- No puede reasignarse

```
scala> val fourHearts: Int = 4
fourHearts: Int = 4
```

Reasignar un val produce error

```
scala> val four: Int = 4
scala> four = 5
error: reassignment to val
```

Tipos de valor en Scala

- Double
- Float
- Long
- Int
- Short
- Byte
- Char
- Boolean
- Unit

Double

Número de punto flotante de 64 bits (IEEE-754).
Rango: 4.94e-324 a 1.79e+308 (positivo o negativo)

Ejemplo Double

```
scala> val piDouble: Double = 3.14
piDouble: Double = 3.14
```

Int

Entero con signo de 32 bits.
Rango: -2,147,483,648 a 2,147,483,647

Boolean

Puede ser:

- true
- false

```
scala> val esViernes: Boolean = true
esViernes: Boolean = true
```

Char y String

Char

- Entero Unicode sin signo de 16 bits
- Rango: 0 a 65,535

String

- Secuencia de caracteres (Char)

Contenido

var (mutable)

```
scala> var uno: Int = 1
uno: Int = 1

scala> uno = 11
uno: Int = 11
```

Ventajas y desventajas de la inmutabilidad

Ventajas

- Los datos no cambian accidentalmente
- Código más fácil de razonar
- Menos pruebas necesarias

Desventajas

- Mayor uso de memoria por copias

Inferencia de tipos

```
scala> val cuatro = 4
cuatro: Int = 4

scala> var uno = 1
uno: Int = 1
```

Contenido

Scripts en Scala

- Secuencia de instrucciones en un archivo
- Se ejecutan de manera secuencial
- Útiles para proyectos pequeños
- El comando `scala` envuelve, compila y ejecuta el script automáticamente

Ejemplo de Script

Archivo: `saludo.scala`

```
// Programa simple
println("¡Bienvenido al curso de Scala!")
```

Ejecución:

```
$ scala saludo.scala
¡Bienvenido al curso de Scala!
```

Lenguaje interpretado vs compilado

Intérprete

- Ejecuta instrucciones directamente
- No requiere compilación previa

Compilador

- Traduce código fuente a código de bajo nivel
- Genera un ejecutable

Aplicaciones Scala

- Se compilan explícitamente
- Pueden contener múltiples archivos
- Útiles para programas grandes
- Mayor rendimiento al estar precompiladas

Ejemplo de Aplicación

Archivo: `Saludo.scala`

```
object Saludo extends App {
  println("Aplicación Scala ejecutándose")
}
```

Compilar:

```
$ scalac Saludo.scala
```

Ejecutar:

```
$ scala Saludo
```

Ventajas y desventajas de lenguajes compila

Ventajas

- Mayor rendimiento

Desventajas

- Tiempo de compilación

Formas de trabajar en Scala

- Línea de comandos
- IDE (Entorno de Desarrollo Integrado)

IDE

- Muy útil para proyectos grandes
- IntelliJ IDEA es el IDE más usado

Simple Build Tool

- Compila
- Ejecuta
- Testea aplicaciones Scala

Contenido

Funciones

En este módulo aprenderemos:

- Qué es una función
- Cómo invocar una función

¿Qué es una función?

- Recibe argumentos
- Produce un resultado
- Tiene:
  1. Lista de parámetros
  2. Cuerpo
  3. Tipo de retorno

Una pregunta específica

```
scala> 85 >= 60
true
```

¿El estudiante aprobó?

Generalizando la pregunta

```
scala> nota >= 60
```

Ahora usamos una variable en vez de un número fijo.

Función: aprobar

```
// Determina si un estudiante aprueba
def aprueba(nota: Int): Boolean = {
  nota >= 60
}
```

El cuerpo está dentro de llaves .

Usando la función

```
println(aprueba(75))
println(aprueba(40))

true
false
```

Llamar función con variables

```
val notaParcial = 55
val notaFinal = 20
val notaTotal = notaParcial + notaFinal

println(aprueba(notaTotal))
```

Contenido

Colecciones

Mutables

- Pueden modificarse

Inmutables

- No cambian

Array

```
val estudiantes = Array("Ana", "Luis", "Carla")
```

Secuencia mutable de elementos del mismo tipo.

Inicialización con tamaño

```
val estudiantes = new Array[String](3)

estudiantes: Array[String] = Array(null, null, null)
```

Tipo: String
Valor: tamaño 3

Modificando un Array

```
estudiantes(0) = "Pedro"
```

Los arrays son mutables.

Error de tipo

```
estudiantes(0) = 100
```

Error: el tipo debe ser String.

Recomendación

Usar:

- val con Array

Permite modificar elementos, pero no reasignar el array completo.

Supertipo Any

```
val mixto = new Array[Any](3)

mixto(0) = "Hola"
mixto(1) = 100
mixto(2) = true
```

Contenido

List

- Colección inmutable
- Elementos del mismo tipo

Lista de tareas

```
val tareas = List("Estudiar", "Practicar", "Repasar")
```

Agregar elemento (::)

```
val nuevasTareas = "Investigar" :: tareas
```

Agrega al inicio y retorna nueva lista. Conocidio típicamente como `cons`.

Nil

```
val listaVacia = Nil
```

Lista vacía.

Construcción con :: y Nil

```
val tareas =
  "Estudiar" ::
  "Practicar" ::
  "Repasar" ::
  Nil
```

Concatenación (:::)

```
val grupoA = List("Ana", "Luis")
val grupoB = List("Carla", "Pedro")

val todos = grupoA ::: grupoB
```

No muta listas originales.

Inmutabilidad

Ventajas

- Evita cambios accidentales
- Código más claro
- Menos pruebas necesarias

Desventajas

- Mayor uso de memoria

Contenido

Más razones para usar Scala

Scala combina programación orientada a objetos y funcional en un lenguaje conciso y de alto nivel.
Los tipos estáticos ayudan a evitar errores en aplicaciones complejas.
La JVM y JavaScript permiten construir sistemas de alto rendimiento con acceso a grandes ecosistemas de librerías.

Definición: Tipo

Tipo:
Restringe los posibles valores a los que una variable puede referirse o que una expresión puede producir en tiempo de ejecución.

Tipos Scala y sus equivalentes en Java

Scala

- scala.Double
- scala.Float
- scala.Long
- scala.Int
- scala.Short
- scala.Byte
- scala.Char
- scala.Boolean
- scala.Unit

Java

- java.lang.Double
- java.lang.Float
- java.lang.Long
- java.lang.Integer

Tiempo de compilación vs ejecución

Compile time: Cuando el código fuente se traduce a código máquina.
Run time: Cuando el programa está ejecutándose.

Sistemas de tipos

Estático

- Tipo conocido en compilación
- Verificado antes de ejecutar
- Ejemplos: C, Java, Scala

Dinámico

- Tipo verificado en ejecución
- Ejemplos: Python, JavaScript, Ruby

Pros y Contras del tipado estático

Ventajas

- Mayor rendimiento
- Prevención de errores de tipo
- Refactorización segura
- Documentación mediante anotaciones

Desventajas

- Tiempo de compilación
- Código más verboso
- Menos flexibilidad

Reduciendo verbosidad (variables)

Sin inferencia:

```
val cuatro: Int = 4
```

Con inferencia:

```
val cuatro = 4
```

Reduciendo verbosidad (colecciones)

Sin inferencia:

```
val estudiantes: Array[String] =
  Array("Ana", "Luis", "Carla")
```

Con inferencia:

```
val estudiantes = Array("Ana", "Luis", "Carla")
```

Promoviendo flexibilidad

- Pattern matching
- Nuevas formas de componer tipos

Lenguajes compilados y estáticos

Lenguajes compilados → Mayor rendimiento
Lenguajes estáticos → Mayor verificación en compilación

Contenido

Estructuras de control

Una estructura de control analiza variables y dirige el flujo del programa.
Ejemplo:

- if / else

Un if simple

```
val nota = 75

if (nota >= 60) {
  println("El estudiante aprueba")
}
```

Un if que no se ejecuta

```
val nota = 45

if (nota >= 60) {
  println("El estudiante aprueba")
}
```

Control if-else

```
def mayorNota(notaA: Int, notaB: Int): Int = {
  if (notaA > notaB) notaA
  else notaB
}
```

if-else en acción

```
val notaA = 55
val notaB = 80

if (notaA > notaB) println(notaA)
else println(notaB)
```

if-else en acción

```
val notaA = 55
val notaB = 80

if (notaA > notaB) println(notaA)
else println(notaB)
```

if – else if – else

```
val notaA = 70
val notaB = 50

if (aprueba(notaA) && aprueba(notaB)) println("Ambos aprueban")
else if (aprueba(notaA)) println("Aprueba A")
else if (aprueba(notaB)) println("Aprueba B")
else println("Ninguno aprueba")
```

if como expresión

```
val notaA = 70
val notaB = 85

val mejorNota =
  if (notaA > notaB) notaA
  else notaB
```