Computación y estructuras discretas II

Unidad Programación Funcional: Scala básico, tipos, funciones, estructuras de decisión y control y comparación con Java

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

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

Tabla de contenido

Contenido

¿Qué es Scala?

Scala

Scala es un lenguaje:

- Estáticamente tipado
- Compilado
- Orientado a objetos
- Funcional
- Que corre sobre la JVM

¿Qué es Scala?

Scala

Scala es un lenguaje:

- Estáticamente tipado
- Compilado
- Orientado a objetos
- Funcional
- Que corre sobre la JVM

- Combina programación funcional y orientada a objetos.
- Es interoperable con Java.
- Tiene inferencia de tipos.

Sistema de tipos en Scala

Sistema de tipos estático

El tipo de una variable es conocido en tiempo de compilación.

Sistema de tipos en Scala

Sistema de tipos estático

El tipo de una variable es conocido en tiempo de compilación.

Ejemplo sin inferencia:

```
val x: Int = 5
```

Sistema de tipos en Scala

Sistema de tipos estático

El tipo de una variable es conocido en tiempo de compilación.

Ejemplo sin inferencia:

```
val x: Int = 5
```

Ejemplo con inferencia de tipos:

```
val x = 5
```

Variables en Scala

Declaración

val → inmutable var → mutable

Variables en Scala

Declaración

val → inmutable var → mutable

Ejemplos:

```
val a = 10      // no se puede modificar
var b = 20      // sí se puede modificar

b = 30
```

Variables en Scala

Declaración

val → inmutable var → mutable

Ejemplos:

```
val a = 10       // no se puede modificar
var b = 20       // sí se puede modificar

b = 30
```

Recomendación: usar `val` siempre que sea posible.

Tipos básicos en Scala

- Int
- Double
- Float
- Boolean
- Char
- String

Tipos básicos en Scala

- Int
- Double
- Float
- Boolean
- Char
- String

Ejemplo:

```
val edad: Int = 25
val precio: Double = 19.99
val activo: Boolean = true
```

Contenido

Definición de funciones

Sintaxis general

def nombre(parametros): TipoRetorno = cuerpo

Definición de funciones

Sintaxis general

def nombre(parametros): TipoRetorno = cuerpo

Ejemplo:

```
def sumar(a: Int, b: Int): Int = {
  a + b
}
```

Definición de funciones

Sintaxis general

def nombre(parametros): TipoRetorno = cuerpo

Ejemplo:

```
def sumar(a: Int, b: Int): Int = {
  a + b
}
```

En Scala, el último valor es el valor de retorno.

Funciones concisas

```
def multiplicar(a: Int, b: Int): Int = a * b
```

Funciones concisas

```
def multiplicar(a: Int, b: Int): Int = a * b

def esPar(n: Int): Boolean = n % 2 == 0
```

Funciones concisas

```
def multiplicar(a: Int, b: Int): Int = a * b

def esPar(n: Int): Boolean = n % 2 == 0
```

No es necesario escribir `return`.

Contenido

Estructura if

Sintaxis

if (condicion) bloque

Estructura if

Sintaxis

if (condicion) bloque

Ejemplo:

```
val x = 10

if (x > 5) {
  println("Mayor que 5")
}
```

if como expresión

```
val a = 10
val b = 20

val mayor = if (a > b) a else b
```

if como expresión

```
val a = 10
val b = 20

val mayor = if (a > b) a else b
```

En Scala, **if devuelve un valor**.

Operadores en Scala

Relacionales

- >, <, >=, <=
- ==, !=

Operadores en Scala

Relacionales

- >, <, >=, <=
- ==, !=

Lógicos

- (AND)
- || (OR)
- ! (NOT)

Colecciones: Array

```
val numeros = Array(1,2,3,4)

println(numeros(0))  // acceso por índice
```

Colecciones: Array

```
val numeros = Array(1,2,3,4)

println(numeros(0))  // acceso por índice

val nombres = Array("Ana", "Luis", "Carlos")
```

Listas en Scala

```
// Lista literal
val lista = List(1, 2, 3, 4)

// ::  (cons) agrega al inicio
val nuevaLista = 0 :: lista   // List(0,1,2,3,4)

// Construcción explícita
val otra = 1 :: 2 :: 3 :: Nil

// ::: concatena listas
val l1 = List(1,2)
val l2 = List(3,4)

val concatenada = l1 ::: l2   // List(1,2,3,4)
```

Recorrer listas con foreach

```
val lista = List(1, 2, 3, 4)

// Imprimir cada elemento
lista.foreach(x => println(x))

// Forma más corta
lista.foreach(println)
```

Recorrer listas con foreach

```
val lista = List(1, 2, 3, 4)

// Imprimir cada elemento
lista.foreach(x => println(x))

// Forma más corta
lista.foreach(println)
```

foreach: - Aplica una función a cada elemento - No retorna nueva lista (retorna Unit)

Bucle while

```
var i = 0

while (i < 5) {
  println(i)
  i = i + 1
}
```

Bucle while

```
var i = 0

while (i < 5) {
  println(i)
  i = i + 1
}
```

Scala favorece estilo funcional sobre bucles imperativos.

Contenido

Java vs Scala — Declaración de variables

Java

```
int x = 10;
x = 20;
```

- Variables mutables por defecto
- Tipo obligatorio
- No hay inferencia fuerte

Scala

```
val x = 10
var y = 20
```

- **val = inmutable** (por defecto recomendado)
- var = mutable
- **Inferencia automática de tipos**

Java vs Scala — Funciones

Java

```
public static int suma(int a, int
b) {
  return a + b;
}
```

- Verboso
- Requiere clase
- return obligatorio

Scala

```
def suma(a: Int, b: Int): Int = a
+ b
```

- Sintaxis compacta
- No necesita return
- Todo es expresión

Java vs Scala — Ciclos

Java

```
for(int i = 0; i < 5; i++) {
  System.out.println(i);
}
```

- Estilo imperativo
- Control manual del índice

Scala

```
for(i <- 0 until 5) {
  println(i)
}
```

- Más declarativo
- No se maneja el incremento
- Basado en rangos

Java vs Scala — Operaciones sobre coleccior

Java

```
List<Integer> nums =
Arrays.asList(1,2,3,4);
for(Integer n : nums) {
  System.out.println(n*2);
}
```

- Iteración explícita
- Enfoque imperativo

Scala

```
val nums = List(1,2,3,4)
nums.map(n => n*2)
  .foreach(println)
```

- **Funciones como valores**
- map, foreach
- Estilo funcional

Java vs Scala — Funciones como valores

Java

```
Function<Integer,Integer>
f = x -> x * 2;
```

- Requiere interfaces
- Sintaxis más pesada

Scala

```
val f = (x: Int) => x * 2
```

- Funciones son valores reales
- Sintaxis directa
- Natural en el lenguaje

Contenido

Recursividad: Factorial

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

Recursividad: Factorial

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

Caso base: n == 0 Caso recursivo: n * factorial(n - 1)

Recursividad: Fibonacci

```
def fibonacci(n: Int): Int = {
  if (n <= 1) n
  else fibonacci(n - 1) + fibonacci(n - 2)
}
```

Recursividad: Fibonacci

```
def fibonacci(n: Int): Int = {
  if (n <= 1) n
  else fibonacci(n - 1) + fibonacci(n - 2)
}
```

Nota: esta versión no es eficiente.