



# Computación y estructuras discretas II

Unidad Programación Funcional: Principios de Programación Funcional

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 los profesores Juan Francisco Díaz, Angela Villota, Jenifer Viafara*

## 1. Introducción

## 2. Principios

- 2.1 Paradigmas de programación
- 2.2 Principios de la Programación Funcional

## 3. Elementos básicos

- 3.1 Scala: El lenguaje de programación del curso
- 3.2 El Read-Eval-Print Loop

## 4. Definición de funciones

## 5. Condicionales y definición de valores

- 5.1 Expresiones condicionales

## 6. Fundamentos

## 1. Introducción

## 2. Principios

## 3. Elementos básicos

## 4. Definición de funciones

## 5. Condicionales y definición de valores

## 6. Fundamentos

# Programación funcional

**Objetivos:** al final del curso el estudiante estará en la capacidad de:

1. Diseño e implementación de programas funcionales puros con estructuras de datos inmutables utilizando recursión, reconocimiento de patrones, mecanismos de encapsulación, funciones de alto orden e iteradores para resolver problemas de programación.
2. Aplicar las reglas de inferencia de tipos genéricos en los programas funcionales para determinar errores de programación.
3. Aplicar conceptos fundamentales de la programación funcional, utilizando un lenguaje de programación adecuado como Scala, para analizar un problema, modelar, diseñar y desarrollar su solución.
4. Razonar sobre la estructura de programas funcionales utilizando la inducción como mecanismo de argumentación para demostrar propiedades de los programas que construye.

# Programación funcional

**Contenido:** para cumplir estos objetivos estudiaremos:

- Principios, fundamentos y lenguajes de programación funcional.
- Elementos de programación funcional: elementos básicos, estrategias de evaluación y terminación, condicionales y definición de valores.
- Funciones y los procesos que ellas generan: procesos recursivos y procesos iterativos.
- Listas: la descomposición en el diseño de datos y el reconocimiento de patrones, pares y tuplas.
- Funciones de alto orden: funciones como parámetro, funciones anónimas y funciones como respuesta.
- Funciones y datos: construyendo abstracciones de datos de la nada e incrementalmente, organización de clases en Scala.
- Funciones de alto orden sobre listas.
- Colecciones: Secuencias, Conjuntos, expresiones `for`, Maps o asociaciones, Streams o flujos.

## 1. Introducción

## 2. Principios

- 2.1 Paradigmas de programación
- 2.2 Principios de la Programación Funcional

## 3. Elementos básicos

## 4. Definición de funciones

## 5. Condicionales y definición de valores

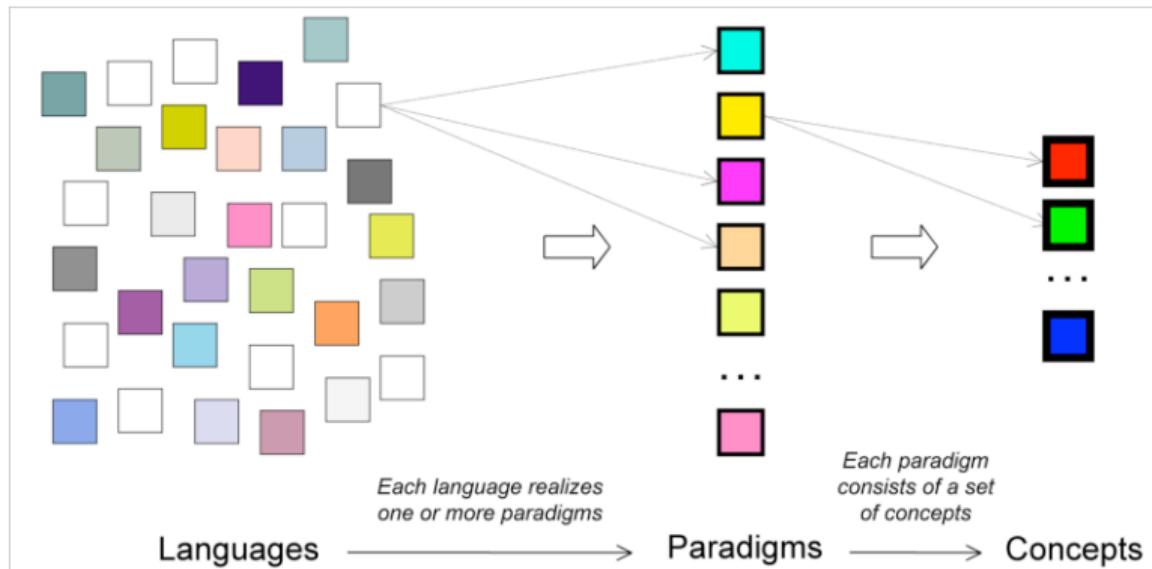
## 6. Fundamentos

# Paradigmas de programación

- Un **paradigma** describe distintos conceptos y patrones de pensamiento en una disciplina científica.

- Un **paradigma** describe distintos conceptos y patrones de pensamiento en una disciplina científica.
- El **paradigma de programación** define las principales características de un lenguaje de programación, esto es, la forma en que se diseñan y construyen los programas. Por ejemplo, en el paradigma orientado a objetos los programas se construyen como un conjunto de objetos que interactúan entre ellos por medio de llamados a métodos, también conocidos como paso de mensajes.

# Paradigmas de programación



Lenguajes, paradigmas y conceptos

1

<sup>1</sup>Imagen tomada de: Programming Paradigms for Dummies: What Every Programmer Should Know, Peter Van Roy.

# Principios de la PF

---

- En un sentido operativo, la PF significa programar sin variables mutables, ni asignación, ni ciclos, ni otras estructuras de control imperativas.

- En un sentido operativo, la PF significa programar sin variables mutables, ni asignación, ni ciclos, ni otras estructuras de control imperativas.
- En un sentido más amplio, la PF significa que un programa es un conjunto de funciones e invocaciones a funciones.

- En un sentido operativo, la PF significa programar sin variables mutables, ni asignación, ni ciclos, ni otras estructuras de control imperativas.
- En un sentido más amplio, la PF significa que un programa es un conjunto de funciones e invocaciones a funciones.
- En particular, las funciones son valores que pueden ser producidos, consumidos y compuestos.

# Principios de la PF

- En un sentido operativo, la PF significa programar sin variables mutables, ni asignación, ni ciclos, ni otras estructuras de control imperativas.
- En un sentido más amplio, la PF significa que un programa es un conjunto de funciones e invocaciones a funciones.
- En particular, las funciones son valores que pueden ser producidos, consumidos y compuestos.
- Los lenguajes de programación funcionales hacen que esto sea sencillo.

# Principios de los lenguajes de PF

- En un sentido operativo, un lenguaje de PF no ofrece variables mutables, ni asignación, ni ciclos, ni otras estructuras de control imperativas.

- En un sentido operativo, un lenguaje de PF no ofrece variables mutables, ni asignación, ni ciclos, ni otras estructuras de control imperativas.
- En un sentido más amplio, un lenguaje de PF permite la construcción de programas elegantes enfocados en funciones.

- En un sentido operativo, un lenguaje de PF no ofrece variables mutables, ni asignación, ni ciclos, ni otras estructuras de control imperativas.
- En un sentido más amplio, un lenguaje de PF permite la construcción de programas elegantes enfocados en funciones.
- En particular, **las funciones son valores** que pueden ser producidos, consumidos y compuestos, es decir:
  - Pueden ser definidas en cualquier parte, incluso dentro de otras funciones.
  - Al igual que cualquier otro valor, las funciones pueden ser pasadas como parámetros a otras funciones y devueltas como resultado.
  - Como para cualquier valor, existen operadores sobre las funciones (para componerlas y para invocarlas).

## 1. Introducción

## 2. Principios

## 3. Elementos básicos

- 3.1 Scala: El lenguaje de programación del curso
- 3.2 El Read-Eval-Print Loop

## 4. Definición de funciones

## 5. Condicionales y definición de valores

## 6. Fundamentos

# Scala como lenguaje del curso

Usaremos **Scala** como el lenguaje de programación del curso.

¿Por qué Scala?

- Porque corre sobre la JVM y su sintaxis es parecida a la de Java (menos esfuerzo inicial).



# Scala como lenguaje del curso

Usaremos **Scala** como el lenguaje de programación del curso.

¿Por qué Scala?

- Porque corre sobre la JVM y su sintaxis es parecida a la de Java (menos esfuerzo inicial).
- Porque implementa el paradigma funcional de forma elegante.



# Scala como lenguaje del curso

Usaremos **Scala** como el lenguaje de programación del curso.

¿Por qué Scala?

- Porque corre sobre la JVM y su sintaxis es parecida a la de Java (menos esfuerzo inicial).
- Porque implementa el paradigma funcional de forma elegante.
- Porque ha sido diseñado para el procesamiento de grandes cantidades de datos, y eso puede ser muy útil para los estudiantes.



# El Read-Eval-Print Loop

- El *Read-Eval-Print Loop* o **REPL** es un ambiente de programación simple e interactivo que permite interactuar con el intérprete del lenguaje como si fuera una calculadora.

# El Read-Eval-Print Loop

- El *Read-Eval-Print Loop* o **REPL** es un ambiente de programación simple e interactivo que permite interactuar con el intérprete del lenguaje como si fuera una calculadora.
- **REPL** es una capa interactiva que permite escribir expresiones y responde con su valor.

# El Read-Eval-Print Loop

- El *Read-Eval-Print Loop* o **REPL** es un ambiente de programación simple e interactivo que permite interactuar con el intérprete del lenguaje como si fuera una calculadora.
- **REPL** es una capa interactiva que permite escribir expresiones y responde con su valor.
- El REPL de **Scala** se puede iniciar de una de las siguientes formas:

Interacciones sencillas con REPL:

- Como calculadora

```
1 scala> 87+145
2 val res3: Int = 232
```

- Es más que las calculadoras pues permite definir valores y calcular con ellos

```
1 scala> def size = 2
2 def size: Int
3
4 scala> 5*size
5 val res4: Int = 10
```

- Expresiones
  - Valores
  - Operandos
- Definición de nombres

**Nota sobre tipos:** En Scala todo es un objeto, así que los tipos de datos son como en Java, pero con mayúscula. → [ir a la doc de Scala](#)

# Parámetros

Las definiciones pueden tener parámetros:

```
1 scala> def square(x: Double) = x*x
2 def square(x: Double): Double
3
4 scala> square(2)
5 val res5: Double = 4.0
6
7 scala> square(5+4)
8 val res6: Double = 81.0
9
10 scala> square(square(4))
11 val res7: Double = 256.0
12
13 scala> def sumOfSquares(x: Double, y: Double) = square(x) + square(y)
14 def sumOfSquares(x: Double, y: Double): Double
```

Los tipos de los parámetros de las funciones se escriben después de :.

El tipo devuelto por la función se escribe después de :, posterior a la lista de parámetros.

```
1 scala> def sumOfSquares(x: Double, y: Double): Double = square(x) + square(y)
2 def sumOfSquares(x: Double, y: Double): Double
```

## 1. Introducción

## 2. Principios

## 3. Elementos básicos

## 4. Definición de funciones

## 5. Condicionales y definición de valores

## 6. Fundamentos

# Definición de funciones

The diagram shows a Scala function definition with various parts labeled by arrows:

- “def” starts a function definition
- function name
- parameter list in parentheses
- function’s result type
- equals sign
- function body, indented

```
def max(x: Int, y: Int): Int =  
  if x > y then  
    x  
  else  
    y
```

# Funciones simples

Por ejemplo:

```
1 scala> def cubo(n: Int) = n * n * n  
2 scala> cubo(3)    // llamado
```

1. No necesito definir tipo de retorno, se puede inferir.
2. No hay `return`.
3. No hay carácter que señale el fin de una instrucción.
4. No hay caracteres para marcar el inicio-fin de un bloque.

## 1. Introducción

## 2. Principios

## 3. Elementos básicos

## 4. Definición de funciones

## 5. Condicionales y definición de valores

### 5.1 Expresiones condicionales

## 6. Fundamentos

# Expresiones condicionales

Para expresar escogencia entre dos alternativas, Scala tiene una expresión condicional:

```
1 if-else
```

Como es una expresión, **siempre devuelve algo**.

Por ejemplo:

```
1 scala> def abs(x: Int) = if (x >= 0) x else -x
2 def abs(x: Int): Int
```

$(x \geq 0)$  es un **predicado** de tipo Boolean.

# Expresiones Booleanas

Las expresiones booleanas son:

```
1 true  false    // Constantes
2 !b              // Neg
3 b && b        // Conj
4 b || b         // Disy
```

Y las operaciones típicas de comparación:

```
1 e <= e, e >= e, e < e, e > e, e == e, e != e
```

Reglas de evaluación de las expresiones booleanas:

```
1 !true      --> false
2 !false     --> true
3 true && e   --> e
4 false && e  --> false
5 true || e   --> true
6 false || e  --> e
```

Nótese que el operando a la derecha no necesita ser evaluado siempre.

Reglas de evaluación de la expresión condicional:

Se evalúa primero el predicado, y luego:

```
1 if true  e1 else e2  --> e1
2 if false e1 else e2  --> e2
```

## 1. Introducción

## 2. Principios

## 3. Elementos básicos

## 4. Definición de funciones

## 5. Condicionales y definición de valores

## 6. Fundamentos

# Evaluación de expresiones

- Toda **expresión no primitiva** (con un operador y unos operandos) se evalúa de la siguiente manera:

# Evaluación de expresiones

- Toda **expresión no primitiva** (con un operador y unos operandos) se evalúa de la siguiente manera:
  1. Se identifica el operador principal de la expresión.

# Evaluación de expresiones

- Toda **expresión no primitiva** (con un operador y unos operandos) se evalúa de la siguiente manera:
  1. Se identifica el operador principal de la expresión.
  2. Se evalúan sus operandos, de izquierda a derecha.

# Evaluación de expresiones

- Toda **expresión no primitiva** (con un operador y unos operandos) se evalúa de la siguiente manera:
  1. Se identifica el operador principal de la expresión.
  2. Se evalúan sus operandos, de izquierda a derecha.
  3. Se aplica el operador a los operandos.

- Toda **expresión no primitiva** (con un operador y unos operandos) se evalúa de la siguiente manera:
  1. Se identifica el operador principal de la expresión.
  2. Se evalúan sus operandos, de izquierda a derecha.
  3. Se aplica el operador a los operandos.
- Un **nombre** se evalúa sustituyéndolo por lo que hay en el lado derecho de su definición.

- Toda **expresión no primitiva** (con un operador y unos operandos) se evalúa de la siguiente manera:
  1. Se identifica el operador principal de la expresión.
  2. Se evalúan sus operandos, de izquierda a derecha.
  3. Se aplica el operador a los operandos.
- Un **nombre** se evalúa sustituyéndolo por lo que hay en el lado derecho de su definición.
- El proceso de evaluación **termina** una vez se tiene un valor (un valor es, por ahora, un número).

¿Cómo se evaluaría la expresión:  $(2 * \text{Size}) * \text{Size}$ ?

**Nota:** operador principal, de izq a der.

1. Identificar el operador principal:  $(2 * \text{Size}) * \text{Size}$

# Ejemplo de evaluación de una expresión sencilla



¿Cómo se evaluaría la expresión:  $(2 * \text{Size}) * \text{Size}$ ?

**Nota:** operador principal, de izq a der.

1. Identificar el operador principal:  $(2 * \text{Size}) * \text{Size}$
2. Evaluar el primer operando  $(2 * \text{Size})$

# Ejemplo de evaluación de una expresión sencilla



¿Cómo se evaluaría la expresión:  $(2 * \text{Size}) * \text{Size}$ ?

**Nota:** operador principal, de izq a der.

1. Identificar el operador principal:  $(2 * \text{Size}) * \text{Size}$
2. Evaluar el primer operando  $(2 * \text{Size})$ 
  - 2.1 Identificar el operador principal:  $2 * \text{Size}$

¿Cómo se evaluaría la expresión:  $(2 * \text{Size}) * \text{Size}$ ?

**Nota:** operador principal, de izq a der.

1. Identificar el operador principal:  $(2 * \text{Size}) * \text{Size}$
2. Evaluar el primer operando  $(2 * \text{Size})$ 
  - 2.1 Identificar el operador principal:  $2 * \text{Size}$ 
    - 1.1 Evaluar el primer operando 2

¿Cómo se evaluaría la expresión:  $(2 * \text{Size}) * \text{Size}$ ?

**Nota:** operador principal, de izq a der.

1. Identificar el operador principal:  $(2 * \text{Size}) * \text{Size}$
2. Evaluar el primer operando  $(2 * \text{Size})$ 
  - 2.1 Identificar el operador principal:  $2 * \text{Size}$ 
    - 1.1 Evaluar el primer operando 2
    - 2.1 Evaluar el segundo operando Size. Como es un nombre, devuelve el lado derecho: 2

# Ejemplo de evaluación de una expresión sencilla



¿Cómo se evaluaría la expresión:  $(2 * \text{Size}) * \text{Size}$ ?

**Nota:** operador principal, de izq a der.

1. Identificar el operador principal:  $(2 * \text{Size}) * \text{Size}$
2. Evaluar el primer operando  $(2 * \text{Size})$ 
  - 2.1 Identificar el operador principal:  $2 * \text{Size}$ 
    - 1.1 Evaluar el primer operando 2
    - 2.1 Evaluar el segundo operando  $\text{Size}$ . Como es un nombre, devuelve el lado derecho: 2
    - 3.1 Aplicar el operador  $*$  a los operandos 2 y 2 devolviendo 4

# Ejemplo de evaluación de una expresión sencilla



¿Cómo se evaluaría la expresión:  $(2 * \text{Size}) * \text{Size}$ ?

Nota: operador principal, de izq a der.

1. Identificar el operador principal:  $(2 * \text{Size}) * \text{Size}$
2. Evaluar el primer operando  $(2 * \text{Size})$ 
  - 2.1 Identificar el operador principal:  $2 * \text{Size}$ 
    - 1.1 Evaluar el primer operando 2
    - 2.1 Evaluar el segundo operando Size. Como es un nombre, devuelve el lado derecho: 2
    - 3.1 Aplicar el operador  $*$  a los operandos 2 y 2 devolviendo 4
  4. Evaluar el segundo operando Size. Como es un nombre, devuelve el lado derecho: 2

¿Cómo se evaluaría la expresión:  $(2 * \text{Size}) * \text{Size}$ ?

Nota: operador principal, de izq a der.

1. Identificar el operador principal:  $(2 * \text{Size}) * \text{Size}$
2. Evaluar el primer operando  $(2 * \text{Size})$ 
  - 2.1 Identificar el operador principal:  $2 * \text{Size}$ 
    - 1.1 Evaluar el primer operando 2
    - 2.1 Evaluar el segundo operando Size. Como es un nombre, devuelve el lado derecho: 2
    - 3.1 Aplicar el operador  $*$  a los operandos 2 y 2 devolviendo 4
  4. Evaluar el segundo operando Size. Como es un nombre, devuelve el lado derecho: 2
  5. Aplicar el operador  $*$  a los operandos 4 y 2 devolviendo 8

# Evaluación de aplicación de funciones definidas por el programador

---

La evaluación de funciones con parámetros se hace de manera similar a las expresiones no primitivas:

- Evaluar los argumentos de izquierda a derecha

# Evaluación de aplicación de funciones definidas por el programador

---

La evaluación de funciones con parámetros se hace de manera similar a las expresiones no primitivas:

- Evaluar los argumentos de izquierda a derecha
- Reemplazar el llamado de la función por su cuerpo (después del =)

# Evaluación de aplicación de funciones definidas por el programador

---

La evaluación de funciones con parámetros se hace de manera similar a las expresiones no primitivas:

- Evaluar los argumentos de izquierda a derecha
- Reemplazar el llamado de la función por su cuerpo (después del =)
- Reemplazar en el cuerpo de la función, los parámetros formales por los argumentos actuales **evaluados**

# Evaluación de aplicación de funciones definidas por el programador

---

La evaluación de funciones con parámetros se hace de manera similar a las expresiones no primitivas:

- Evaluar los argumentos de izquierda a derecha
- Reemplazar el llamado de la función por su cuerpo (después del =)
- Reemplazar en el cuerpo de la función, los parámetros formales por los argumentos actuales **evaluados**
- Evalúe esta nueva expresión

# Evaluación de aplicación de funciones definidas por el programador (2)

---

Por ejemplo:

```
1 sumOfSquares(3, 2+2)
2 sumOfSquares(3, 4)
3 square(3) + square(4)
4 3*3 + square(4)
5 9 + square(4)
6 9 + 4*4
7 9 + 16
8 25
```