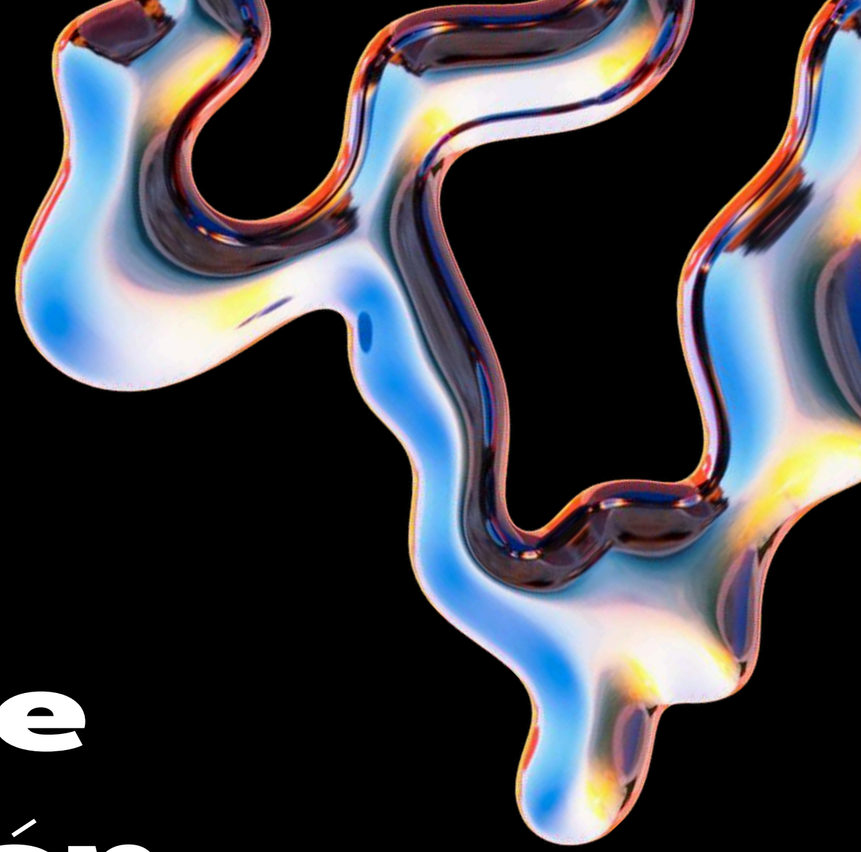
2024-1

Scala 3



Teoría de Lenguajes de Programación

Sara Acevedo Maya
saacevedom@unal.edu.co
Universidad Nacional de Colombia

UNIVERSIDAD NACIONAL DE COLOMBIA

Contenido

UNIVERSIDAD NACIONAL DE COLOMBIA

01 - Listas

UNIVERSIDAD
NACIONAL
DE COLOMBIA

Listas

Una lista en Scala es una colección **inmutable** y **ordenada** de elementos del mismo tipo.

Características :

- **Ordenada:** Los elementos se mantienen en el orden en que se agregaron.
- **Tipo de elementos:** Todos los elementos deben ser del mismo tipo.
- **Inmutabilidad:** Una vez que una lista es creada, no se puede cambiar; cualquier operación que parece modificar la lista, en realidad crea una nueva lista.

```
val listaNumeros = List(1, 2, 3, 4, 5)
```

UNIVERSIDAD NACIONAL DE COLOMBIA

Operaciones Básicas con Listas :

- Crear una lista:

SAM <3

```
val listaVacia = List()
val listaCadenas = List("Scala", "Java", "Python")
```

- Acceso a elementos:

```
val listaNumeros = List(1, 2, 3, 4, 5)
```

SAM <3

```
val primerElemento = listaNumeros.head  // 1
val restoElementos = listaNumeros.tail  // List(2, 3, 4, 5)
val tercerElemento = listaNumeros(2)    // 3
```

UNIVERSIDAD NACIONAL DE COLOMBIA

- Modificar listas:
  - Añadir elementos:

```
SAM <3

val nuevaLista = 0 :: listaNumeros  // List(0, 1, 2, 3, 4, 5)
```

  - Concatenar listas:

```
SAM <3

val listaCombinada = listaNumeros ++ List(6, 7, 8)
```

  - Filtrar elementos:

```
SAM <3

val listaPares = listaNumeros.filter(_ % 2 == 0)  // List(2, 4)
```

02 - Tuplas

UNIVERSIDAD
NACIONAL
DE COLOMBIA

Tuplas

Una tupla es una colección que puede contener elementos de diferentes tipos

- **Uso común:** Utilizadas cuando se desea agrupar múltiples valores sin crear una clase específica.
- **Inmutabilidad:** Similar a las listas, las tuplas son inmutables.

SAM <3

```
val tuplaPersona = ("Juan", 30, true)  // (String, Int, Boolean)
```

UNIVERSIDAD NACIONAL DE COLOMBIA

Sintaxis para Definir y Trabajar con Tuplas

Definir una tupla:

SAM <3

```
val tuplaEjemplo = ("Scala", 2024, 3.14)
```

Acceder a elementos de la tupla:

SAM <3

```
val primerElemento = tuplaEjemplo._1  // "Scala"
val segundoElemento = tuplaEjemplo._2  // 2024
```

UNIVERSIDAD NACIONAL DE COLOMBIA

Sintaxis para Definir y Trabajar con Tuplas

Desempaquetar tupla en variables:

SAM <3

```scala
val tuplaEjemplo = ("Scala", 2024, 3.14)
```

SAM <3

```scala
val (lenguaje, anio, valorPi) = tuplaEjemplo
```

SAM <3

```scala
println(lenguaje)  // Imprime: Scala
println(anio)      // Imprime: 2024
println(valorPi)   // Imprime: 3.14
```

***desestructuración** es una característica que permite extraer los elementos de una tupla y asignarlos directamente a variables individuales.

UNIVERSIDAD NACIONAL DE COLOMBIA

Ejemplos de Uso de Tuplas en Contextos Prácticos

- **Almacenar datos relacionados:** Almacenar información de una persona

SAM <3

```
val persona = ("Marta", 28, "Ingeniera")
val (nombre, edad, profesion) = persona
```

- **Retorno múltiple:** Devolver múltiples valores de una función

SAM <3

```
def calcular(a: Int, b: Int): (Int, Int) = (a + b, a * b)
val (suma, producto) = calcular(3, 4)
```

- La función calcular toma dos parámetros enteros, a y b.
- Devuelve una tupla con dos elementos: la suma de a y b, y el producto de a y b.
- En el caso de calcular(3, 4):
  Suma: 3 + 4 = 7
  Producto: 3 * 4 = 12
- La función retorna la tupla (7, 12).
- Luego, la tupla se desestructura en dos variables:
  suma recibe el valor 7.
  producto recibe el valor 12.

UNIVERSIDAD NACIONAL DE COLOMBIA

SAM <3

```
def calcular(a: Int, b: Int): (Int, Int) = (a + b, a * b)
val (suma, producto) = calcular(3, 4)
```

- La función calcular toma dos parámetros enteros, a y b.
- Devuelve una tupla con dos elementos: la suma de a y b, y el producto de a y b.
- En el caso de calcular(3, 4):
  Suma: 3 + 4 = 7
  Producto: 3 * 4 = 12
- La función retorna la tupla (7, 12).
- Luego, la tupla se desestructura en dos variables:
  suma recibe el valor 7.
  producto recibe el valor 12.

SAM <3

```
println(suma)     // Imprime: 7
println(producto) // Imprime: 12
```

UNIVERSIDAD NACIONAL DE COLOMBIA

03-Listas vs Tuplas

UNIVERSIDAD
NACIONAL
DE COLOMBIA

Listas vs Tuplas

Característica	Listas (List)	Tuplas (Tuple)
Tipos de elementos	Todos los elementos deben ser del mismo tipo.	Pueden contener elementos de diferentes tipos.
Tamaño	Puede contener un número arbitrario de elementos.	El tamaño de la tupla es fijo y determinado en el momento de la creación.
Acceso a elementos	Acceso a través de índices (comenzando en 0), como list(0).	Acceso a través de _1, _2, etc., dependiendo de la posición del elemento. Ejemplo: tupla._1.
Operaciones comunes	Métodos como map, filter, reduce, fold, etc.	No se aplican operaciones como map o filter directamente sobre tuplas; se utilizan principalmente para acceder a elementos.
Ejemplo	List(1, 2, 3)	(1, "Scala", 3.14)
Mutabilidad	Las listas estándar en Scala (List) son inmutables, pero se puede usar ListBuffer para una lista mutable.	Las tuplas son inmutables; sus elementos no se pueden cambiar después de la creación.

UNIVERSIDAD NACIONAL DE COLOMBIA

Listas vs Tuplas

- **Concatenación de Listas:**

SAM <3

```
val lista1 = List(1, 2, 3)
val lista2 = List(4, 5)
val listaConcatenada = lista1 ++ lista2  // Resultado: List(1, 2, 3, 4, 5)
```

- **Concatenación de Duplas:**

SAM <3

```
val tupla1 = (1, "Scala")
val tupla2 = (3.14, "Rocks")

// No puedes concatenar tuplas directamente, pero puedes crear una nueva tupla combinando elementos manualmente:
val nuevaTupla = (tupla1._1, tupla1._2, tupla2._1, tupla2._2)  // Resultado: (1, "Scala", 3.14, "Rocks")
```

UNIVERSIDAD NACIONAL DE COLOMBIA

04-Recursión

UNIVERSIDAD
NACIONAL
DE COLOMBIA

Recursión

*match es una estructura similar a switch en otros lenguajes de programación, pero mucho más potente y expresiva. Te permite realizar verificaciones sobre una variable y manejar diferentes casos basándote en su estructura o valor.

SAM <3

```
val x = 3

x match {
  case 1 => println("Uno")
  case 2 => println("Dos")
  case 3 => println("Tres") //en este caso se cumpliría esta
  case _ => println("Otro número")
}
```

***Nil** en Scala representa una lista vacía. Es un objeto que actúa como una constante para denotar que una lista no contiene elementos.

SAM <3

```
val listaVacia: List[Int] = Nil  // Lista vacía
```

UNIVERSIDAD NACIONAL DE COLOMBIA

Recursión

- **Caso básico:** Suma de los elementos de una lista

SAM <3

```
def suma(lista: List[Int]): Int = lista match {
  case Nil => 0  // Caso base: la suma de una lista vacía es 0
  case head :: tail => head + suma(tail)  // Caso recursivo: suma el primer elemento y llama a la función con el resto de la lista
}
```

UNIVERSIDAD NACIONAL DE COLOMBIA

Recursión

- Encontrar la longitud de una lista

SAM <3

```
def longitud(lista: List[Int]): Int = lista match {
  case Nil => 0  // Caso base: la lista está vacía, su longitud es 0
  case _ :: tail => 1 + longitud(tail)  // Caso recursivo: contar el primer elemento y seguir
con el resto de la lista
}
```

_ -> comodín, no nos importa que valor va ahí

UNIVERSIDAD NACIONAL DE COLOMBIA

Recursión

- Sumatoria con tail recursion

SAM <3

```scala
@annotation.tailrec
def sumaTailRec(lista: List[Int], acumulador: Int = 0): Int = lista match {
  case Nil => acumulador  // Caso base: retornar el acumulador
  case head :: tail => sumaTailRec(tail, acumulador + head)  // Caso recursivo: acumular la suma y continuar
}
```

@annotation.tailrec es una anotación en Scala que se utiliza para indicar que una función es recursiva en cola (tail-recursive). Esta anotación le dice al compilador que debe verificar si la recursión de la función es realmente en cola y, si lo es, optimizarla para evitar desbordamientos de pila.

UNIVERSIDAD NACIONAL DE COLOMBIA

Tarea

Grupo 1,2,3:
Plazo hasta el Domingo 25 de Agosto a las 10am:

https://forms.gle/m2SGixdcNhZWWgHm9

UNIVERSIDAD NACIONAL DE COLOMBIA