Módulo Profesional: Big Data Aplicado

Scala

LISTS

```
//Lists
val aList = List (1,2,3)
val prepended = 42 :: aList //añade a la lista como primer elemento el 42
println(prepended)
```

```
//Lists
val aList = List (1,2,3)
val prepended = 42 +: aList :+ 89 //añade a la lista como primer elemento el 42 y como último elemento 89
println(prepended)
```

```
val manzanas5 = List.fill(5) ("manzana")
println(manzanas5) //imprime una lista de 5 elementos con el valor "manzana"
```

```
val aList2 = List(1,2,3)
println(aList2.mkString("-|-")) //imprime los elementos de la lista separados -| === 1-|-2-|-3
```

mkString convierte los elementos de una colección en una sola cadena, separando los elementos con un delimitador definido entre paréntesis.

ARRAY

```
//arrays
val numeros = Array (1,2,3,4)
val tresElementos = Array.ofDim[Int](3) // Array.ofDim[String](3)
tresElementos.foreach(println) //¿Qué imprime por pantalla? 0    0     0 // null    null  null

//mutation
numeros(2) = 0 //estamos actualizando el valor de la posición 2 a 0
println(numeros.mkString(" ")) //imprime por pantalla 1 2 0 4

//arrays y secuencias
val numerosSeq: Seq[Int] = numeros //conversión implícita
println(numerosSeq)
```

¿Cuándo utilizar “Array” o “List”?

Mutabilidad. Si necesitamos modificar los valores de los elementos con tamaño fijo entonces utilizamos ***Array*** porque ***List*** es inmutable (se usa más en programación funcional) no permitiendo cambiar los valores una vez creados .

```
//array
val arr = Array (1, 2, 3)
arr(0) = 10 //modificamos el valor del primer elemento
println(arr.mkString(", "))
```

Estructura. Si necesitamos una colección fija de elementos de tamaño predefinido con acceso rápido a los elementos entonces utilizamos ***Array***. En ***List*** cada elemento apunta al siguiente y el último apunta a una lista vacía y acceder a un elemento específico por índice es más lento.

EJERCICIOS: ¿Array o List?

```
/* Ejercicios: Resuelve los siugientes ejercicios utilizando ¿Array o List? y el por qué.

1.- Implementa una aplicación que almacene las lecturas de un sensor de temperatura. Cada vez que llega una nueva lectura, se debe agregar en la colección
sin eliminar las lecturas anteriores.
2.- Necesitamos almacenar las puntuaciones de 6 jugadores en una partida, donde las puntuaciones deben ser accesibles para calcular su promedio al final.
3.- En una lista de compras, los productos comprados deben almacenarse en el orden en el que se van comprando.
4.- Tenemos un inventario donde los elementos pueden cambiar de cantidad rápidamente.
*/
```

VECTOR

- ***Inmutabilidad***: no puedes cambiar sus elementos directamente. Para modificar los valores de un Vector como *update* devuelve un nuevo vector con los cambios aplicados, sin modificar el vector original.
- ***Estructura basada en 32 ramas***, dividiendo los datos en bloques pequeños permitiendo el acceso por índice (O(log32(n))) y actualizaciones eficientes. Representación podría ser:

```
Nivel 1: [ Nodo_1, Nodo_2, Nodo_3, ... ]   // Raíz del árbol (32 referencias)
Nivel 2: [ Elementos 1-32, Elementos 33-64, ..., Elementos 993-1000 ]
```

- ***Acceso rápido*** pero no tan rápido como Array.

```
//Vector (Inmutable)
val vector = Vector(1, 2, 3)
```

- ***Es dinámico*** porque te permite agregar o modificar obteniendo una nueva colección con los cambios.

Comparar Vector vs Array

```
//Vector (Inmutable)
val vector = Vector(1, 2, 3)
val updatedVector = vector.updated(1, 42) // Modificar un elemento devuelve un nuevo Vector
println(vector)        // Vector(1, 2, 3) - Sin cambios
println(updatedVector) // Vector(1, 42, 3) - Nuevo Vector

//Array (mutable)
val array = Array(1, 2, 3)
array(1) = 42 //Modificar un elemento cambia el Array original
println(array.mkString(", ")) // 1, 42, 3
```

RESUMEN:

- Array: Ideal para trabajar con datos mutables y de tamaño fijo.

- List: Ideal para operaciones funcionales y cuando las inserciones al inicio son frecuentes.

- Vector: Opción predeterminada para colecciones inmutables con buen rendimiento general.

Ejercicio

En Aules, disponéis de un ejemplo de código en Scala donde el objetivo del ejercicio es analizar y realizar una breve explicación del resultado al ejecutar el código de Scala.
“15_Ejercicio_Vector.scala”