Módulo Profesional: Big Data Aplicado

Scala

MAP – FlatMAP – FILTER - FOR

```scala
val list = List(1,2,3)
println(list)
println(list.head)
println(list.tail)
```

1. val list = List (1,2,3): Lista inmutable en Scala con los elementos 1, 2 y 3.
2. list.head: devuelve el primer elemento de la lista.
3. list.tail: devuelve una nueva lista que contiene todos los elementos excepto el primero.

MAP – FlatMAP – FILTER - FOR

```scala
val list = List(1,2,3)
//map
println(list.map(_ +1))
println(list.map(_+ " is un número"))
```

1. La función map aplica una operación a cada elemento de la lista y devuelve una nueva lista con los resultados.

2. (_ + 1) es una función anónima que toma cada elemento de la lista y le suma 1. Resultado:

1 suma 1 = 2
2 suma 1 = 3 ⇨ **List(2, 3, 4)**
3 suma 1 = 4

MAP – FlatMAP – FILTER - FOR

```
val list = List(1,2,3)
//map
println(list.map(_ +1))
println(list.map(_+ " is un número"))
```

3. list.map (_ + “ is un número”): concatena el texto “es un número” a cada elemento de la lista.

1 se convierte “1 es un número”
2 se convierte “2 es un número”
3 se convierte “3 es un número”

MAP – FlatMAP – FILTER - FOR

```
val list = List(1,2,3)

//filter
println(list.filter(_ % 2 == 0))
```

filter se utiliza para filtrar elementos de una colección que cumplen una condición.

_ % 2 == 0: función anónima (lambda) que verifica si un número es divisible entre 2.

_ representa cada elemento de la lista

% el operador residuo. En este caso dividir el número entre 2 es 0.

El resultado de filter es una nueva lista que contiene los números pares en este caso 2.

MAP – FlatMAP – FILTER - FOR

```
 val list = List(1,2,3)
//flatMap
val toPair = (x: Int) => List(x, x+1)
println(list.flatMap(toPair))
```

ToPair: función anónima que toma un entero y devuelve una lista que contiene x y x +1. Ejemplo toPair(1) devuelve List(1,2)
List.flatMap(toPair):

flatMap: método que combina dos operaciones map y aplanado flatten. Generando una sola lista.

toPair: se aplica a cada elemento de la lista.

Resultado: (1, 2, 2, 3, 3, 4)

MAP – FlatMAP – FILTER - FOR

```
//imprime todas las combinaciones entre dos listas
val numero = List (1,2,3,4)
val caracteres = List ('a','b','c','d')
val colores = List ("claro", "oscuro")
//List ("a1","a2"...."d4")

//iteraciones
val combinaciones = numero.flatMap(n=> caracteres.map(c=> "" + c + n))
println(combinaciones)
```

MAP – FlatMAP – FILTER - FOR

```
//imprime todas las combinaciones entre dos listas
val numero = List (1,2,3,4)
val caracteres = List ('a','b','c','d')
val colores = List ("claro", "oscuro")
//List ("a1","a2"...."d4")

//iteraciones
val combinaciones = numero.flatMap(n=> caracteres.map(c=> "" + c + n))
println(combinaciones)
```

Resultado:

```
List(a1, b1, c1, d1, a2, b2, c2, d2, a3, b3, c3, d3, a4, b4, c4, d4)
```

MAP – FlatMAP – FILTER - FOR

```
//imprime todas las combinaciones entre dos listas
val numero = List (1,2,3,4)
val caracteres = List ('a','b','c','d')
val colores = List ("claro", "oscuro")
//List ("a1","a2"...."d4")

//iteraciones
val combinaciones = numero.map(n=> caracteres.map(c=> "" + c + n))
println(combinaciones)
```

Resultado: ¿cómo lo devuelve?

MAP – FlatMAP – FILTER - FOR

```
//imprime todas las combinaciones entre dos listas
val numero = List (1,2,3,4)
val caracteres = List ('a','b','c','d')
val colores = List ("claro", "oscuro")
//List ("a1","a2"...."d4")

//iteraciones
val combinaciones = numero.map(n=> caracteres.map(c=> "" + c + n))
println(combinaciones)
```

Resultado:

```
List(List(a1, b1, c1, d1), List(a2, b2, c2, d2), List(a3, b3, c3, d3), List(a4, b4, c4, d4))
```

MAP – FlatMAP – FILTER - FOR

```
//imprime todas las combinaciones entre dos listas
val numero = List (1,2,3,4)
val caracteres = List ('a','b','c','d')
val colores = List ("claro", "oscuro")
val combinaciones1=  numero.flatMap(n=> caracteres.map(c=> colores.map(color => ""+ c + n + color)))
```

Resultado: ¿?

MAP – FlatMAP – FILTER - FOR

```
//imprime todas las combinaciones entre dos listas
val numero = List (1,2,3,4)
val caracteres = List ('a','b','c','d')
val colores = List ("claro", "oscuro")
val combinaciones2=  numero.filter(_ % 2 ==0).flatMap(n=> caracteres.map(c=>
                                    colores.map(color => ""+ c + n + color)))

println(combinaciones2)
```

Resultado: ¿?

MAP – FlatMAP – FILTER - FOR

```
//foreach
list.foreach(println)
```

Resultado de Imprimir:

```
1
2
3
```

MAP – FlatMAP – FILTER - FOR

```
//for-comprehensions
val numero = List (1,2,3,4)
val caracteres = List ('a','b','c','d')
val colores = List ("claro", "oscuro")
    //for-comprehensions
    val forCombinaciones = for {
        n <- numero
        c <- caracteres
        color <- colores

    } yield "" + c + n + "-" + color
    println (forCombinaciones)
```

MAP – FlatMAP – FILTER - FOR

```
//for-comprehensions
val forCombinaciones = for {
    n <- numero
    c <- caracteres
    color <- colores

} yield "" + c + n + "-" + color
println (forCombinaciones)
```

For {...} yield...: se utiliza para generar una nueva colección aplicando combinaciones de elementos de las listas de entrada.
N < - numero: itera sobre cada elemento de la lista número.
Yield: especifica qué valor se generará para cada combinación.

MAP – FlatMAP – FILTER - FOR

Resultado de Imprimir:

```
List(a1-claro, a1-oscuro, b1-claro, b1-oscuro, c1-claro, c1-oscuro, d1-claro, d1-oscuro, a2-claro, a2-osc
uro, b2-claro, b2-oscuro, c2-claro, c2-oscuro, d2-claro, d2-oscuro, a3-claro, a3-oscuro, b3-claro, b3-osc
uro, c3-claro, c3-oscuro, d3-claro, d3-oscuro, a4-claro, a4-oscuro, b4-claro, b4-oscuro, c4-claro, c4-osc
uro, d4-claro, d4-oscuro)
```

Resultado genera una cadena con el formato <c><n>-<color> y las acumula en una nueva lista.

EJERCICIOS

```
/* Ejercicio 1:
    Dada una lista de alumnos y alumnas con sus nombres y una lista de asignaturas disponibles,
    genera todas las combinaciones posibles de ambas creando una cadena con el siguiente formato:
        "Nombre del estudiante" está inscrito en "Asignaturas"
*/
```

```
/*Ejercicio 2:
    En una tienda de frutas, cada fruta tiene un precio por unidad. Se necesita realizar las siguientes
    operaciones sobre una lista de compras de un cliente:
        1. Filtrar las frutas cuyo precio por unidad sea mayor a 3
        2. Calcular el precio total para cada fruta filtrada considerando la cantidad comprada
        3. Generar una lista de mensajes indicando cuánto se gastará por cada fruta en el formato:
            "Fruta: <nombre>, Total: <total>"
*/
```