Módulo Profesional: Big Data Aplicado

Scala

TUPLAS. LISTAS FINITAS Y ORDENADAS

```scala
//lists. Listas finitas y ordenadas
val aTuple = new Tuple2 (2, "Hola, Scala") // Tuple2 [Int, String] = (Int, String)
val aTuple2 = (2, "Hola, Scala2") //sin lpalabra new

println(aTuple._1) //2
println(aTuple.copy(_2 ="Adiós Java"))
println(aTuple.swap) //intercambia los elementos
```

- **Declaración**: con la palabra reservada “new” sin new. Tenemos hasta 22 elementos de una tupla.
- **nombre_tupla._x:** donde nombre_tupla es el nombre declarado y x es la posición del elemento que queremos obtener.
- **copy**: función que nos permite copiar todos los elementos de una tupla o asignar otro valor a un elemento _x =”…” donde x es la posición del elemento que queremos cambiar.
- **swap**: intercambia la posición de los elementos. Es decir, tupla (2, “Hola,Scala”) pasa a ser (“Hola, Scala”,2)

MAPS

```
//Maps: son colecciones que se usan para asociar unas con otras cosas
//Maps : keys --> values
val aMap: Map[String, Int] = Map()
val dirTelefono = Map (("Jim", 999), "Daniel" -> 777)
//a-> b === (a, b)
println (dirTelefono)
```

```
//operaciones básica Maps
println(dirTelefono.contains("Jim")) //devuelve verdadero o falso si contiene la clave
println(dirTelefono("Jim")) //devuelve el valor asociado a la clave
println(dirTelefono("maria")) //¿qué sucede?
```

MAPS

```
val dirTelefono = Map (("Jim", 999), "Daniel" -> 777)
println(dirTelefono("maria")) //¿qué sucede?
```

Si intentamos imprimir un elemento de Map que no existe el resultado que devuelve es un error

```
Exception in thread "main" java.lang.ExceptionInInitializerError
        at TupesAndMaps.main(16_TupesAndMaps.scala)
Caused by: java.util.NoSuchElementException: key not found: maria
        at scala.collection.immutable.Map$Map2.apply(Map.scala:316)
        at TupesAndMaps$.<clinit>(16_TupesAndMaps.scala:21)
        ... 1 more
```

Para evitar que muestre error añadimos a continuación .withDefaultValue(-1)

```
val dirTelefono = Map (("Jim", 999), "Daniel" -> 777).withDefaultValue(-1)  //para evitar errores
println(dirTelefono("maria")) //¿qué sucede?
```

MAPS

Para añadir un nuevo emparejamiento a Map:

```
//añadir un emparejamiento
val nuevoEmpar = "Maria" -> 789
val dirTelefono2 = dirTelefono + nuevoEmpar
println(dirTelefono2)
```

MAPS

Funciones Map:

```
//funciones en mapas
//map, flatMap, filter
println(dirTelefono.map(pair => pair._1.toLowerCase -> pair._2)) //jim y daniel se imprimen en minúsculas
//filterkeys
println(dirTelefono.filterKeys(x => x.startsWith("J"))) //sintaxis lambda

//maps Values
println(dirTelefono.mapValues(number => number * 10)) //pasará por todos los valores y les agregar un 0

println(dirTelefono.mapValues (number => "789-" + number)) //¿qué pasará?
```

MAPS

Conversiones a otras colecciones:

```
//conversiones a otras colecciones
val dirTelefono = Map (("Jim", 999), "Daniel" -> 777).withDefaultValue(-1)  //para evitar errores
println(dirTelefono.toList) //pasar a lista
```

```
val nombres = List("Carlos", "Miguel", "Manuel","Roberto","David", "Guillermo", "Daniel")
println(nombres.groupBy(name => name.charAt(0) )) //agrupa los nombres por lista donde la letra inicial sea la misma
```

EJERCICIOS

```
/*Ejercicio 1
    En este ejercicio 1 programa una lista de números enteros y devuelva una nueva lista
    en la que cada número sea el cuadrado del número original. Utiliza la función map para transformar la lista.
*/
```

```
/* Ejercicio 2
   En este ejercicio 2 programa una lista de palabras y devuelva una nueva lista
    en la que cada palabra esté convertida a mayúsculas. Utiliza la función map para transformar la lista.
*/
```

```
/* Ejercicio 3
En este ejercicio 3 programa una lista de nombres completos (formados por nombre y apellido, separados por un espacio) y
devuelva una nueva lista donde cada nombre esté en formato "Apellido, Nombre".
Utiliza la función map para realizar esta transformación.

*/
```