Módulo Profesional: Big Data Aplicado

Scala

Recursividad

Recursividad no final:
Después de calcular factorial (n - 1) todavía tiene que multiplicar ese resultado por n antes de devolverlo.

Esto significa que cada llamada recursiva debe esperar el resultado de las llamadas siguientes para completar su cálculo.

- Consumo de memoria
- Menor optimización

Recursividad

```
object Recursividad1 extends App{
    def factorial (n: Int): Int =
        else {
            println ("Ejecutando Factorial de " + n + "factorial de n - 1 " + (n - 1))
            val resultado = n * factorial (n - 1)
            println("El Factorial de n" + n)

            resultado
        }
    println (factorial (10))
    println(factorial(5000))
```

Recursividad

Recursividad final:

La llamada recursiva es la última operación que se realiza en cada invocación de factCalcula.

La función se llama así misma con los valores actualizados y no realiza ningún cálculo después de la llamada recursiva

- Eficiencia
- Optimización atuomática. El compilador puede optimizar recursiones finales para transformarlas en un simple bucle.

Recursividad

```scala
import scala.annotation.tailrec
run | debug
object Recursividad2 extends App{

    //nuevo concepto --BigInt

    def otroFactorial(n: Int): Int = {
      def factCalcula (x: Int, acumulador: Int) : Int =
          if ( x <= 1) acumulador
          else factCalcula (x - 1, x * acumulador)

      factCalcula(n ,1)

    }
```

Recursividad

```
/* otroFactorial (10) = factCalcula (10,1)
= factCalcula (9, 10 * 1)
= factCalcula (8, 9 * 10 * 1)
= factCalcula (7, 9 * 9 * 10 * 1)
=......
= factCalcula (2, 3 * 4 * ...* 10 * 1)
= factCalcula (1, 2 * 3 * 4 *...* 10 * 1)
= 1 * 2 * 3 * 4 * ... * 10
*/
```

Recursividad

Ejercicios

1. Función que concatena una cadena n veces