Práctica 8: Programación Funcional en Scala

Ejercicio 1

a) Define en Scala la función recursiva pura (sin *tail recursion*) `aplica3D` que reciba una lista de coordenadas 3D en forma de tupla, una función unaria y una coordenada (“x”, “y” ó “z”). Deberá aplicar la función a los elementos correspondientes a esa coordenada y devolver las nuevas coordenadas. Ejemplo:

```
val lista = List((1,2,3), (4,5,6), (7,8,9), (10,11,12))
def suma2(x: Int) = x + 2
aplica3D(lista, suma2 _, “y”)
⇒ ((1,4,3), (4,7,6), (7,10,9), (10,13,12))
```

b) Define la función anterior utilizando funciones de orden superior.

Ejercicio 2

a) Escribe en Scala la función `intercambia(lista)` que reciba una lista de tuplas de dos elementos de tipo entero y devuelva una lista con las mismas tuplas pero con los elementos de cada tupla intercambiados.

Ejemplo:

```
val lista = List((1,2), (3,4), (5,6))
intercambia(lista)
⇒ List((2,1), (4,3), (6,5))
```

b) Escribe en Scala la función `asocia(func, lista)` que reciba una función y una lista de números enteros, y devuelva una lista de tuplas. La función asocia debe recorrer la lista de números y devolver aquellos números contiguos (en forma de tupla) que cumplan que el número de la derecha es el resultado de aplicar la función al número de su izquierda. La función asocia tiene que ser recursiva y debes definir por completo su prototipo.

Ejemplo:

```
def cuadrado(x:Int) = x*x
val lista = List(2, 4, 16, 5, 10, 100, 105)
asocia(cuadrado _, lista)
⇒ List((2,4),(4,16),(10,100))
```

Ejercicio 3

a)
Escribe en Scala el procedimiento `numTests(List[(Int)=>Boolean], Int): Int` que toma una lista de tests y un número n y que devuelva el número de tests de la lista que pasa el número n. Por ejemplo, supongamos los tests: `mayorQue8(x)`, `par(x)`, `impar(x)`

```
val listaTests = List(mayorQue8 _,  par _,  impar _)
numTests(listaTests, 12)
⇒ 2
numTests(listaTests, 3)
⇒ 1
```

b)
Implementa en Scala el procedimiento `generaTest(List[String]): List[(Int)=>Boolean]` que devuelva una lista de tests a partir de una lista de expresiones. La lista de expresiones tendrá la forma de ("op1" "n1" … "opn" "nn"), donde los operadores pueden ser las cadenas ">", "<" o "=". Puedes definir funciones auxiliares.

Por ejemplo:

```
generaTests(List(">","3","<","5",">","8","=","10"))
```

devolverá una lista con 4 tests: la comprobación de si un número es mayor que 3, menor que 5, mayor que 8 y igual que 10. Esta lista se podría utilizar en el ejercicio anterior:

```
val listTest = generaTests(List(">","3","<","5",">","8","=","10"))
numTests(listTest, 10)
⇒ 3
```

Nota: Puedes utilizar el método de cadenas `toInt` para transformar una cadena a un entero.

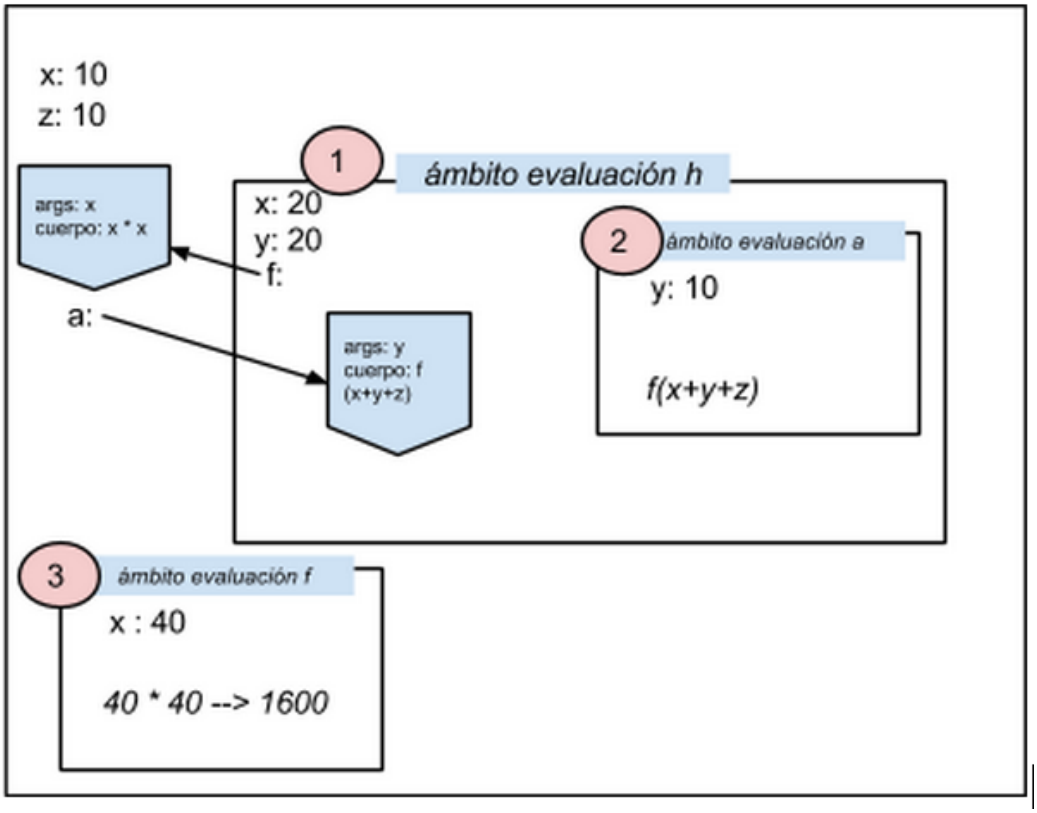
Ejercicio 4

a) Dibuja y explica los ámbitos generados por las siguientes instrucciones en Scala
b) ¿Se ha generado alguna *closure*? Explícalo

```
val x = 1
val y = 2
def f(x:Int) : (Int) => Int =
  (y:Int) => x + y
def g(h : (Int) => Int) : Int = {
    val x = 20
    h(100)
}
val a = f(10)
g(a)
```

Ejercicio 5

a) Explica y escribe en Scala las instrucciones que han generado los siguientes ámbitos.
b) ¿Se ha creado alguna *closure*? Explícalo



image