Introducción a la Programación Funcional

- Pliegues

Pliegues

- Es muy común que trabajemos con listas y también es muy común que tengamos que recorrerlas para obtener un valor. Por ejemplo si queremos el promedio de una lista de números, debemos sumar todos los números para luego dividirlos por la cantidad.
- Este es un patrón muy común y por lo tanto en Haskell o scala nos brinda unas cuantas funciones muy útiles para encapsular este comportamiento. Estas funciones son llamadas pliegues (o folds en ingles).

Pliegues

•Un pliegue toma una función binaria, un valor inicial (llamémoslo acumulador) y una lista que plegar. La función binaria toma dos parámetros por si misma. La función binaria es llamada con el acumulador y el primer (o último) elemento y produce un nuevo acumulador. Luego, la función binaria se vuelve a llamar junto al nuevo acumulador y al nuevo primer (o último) elemento de la lista, y así sucesivamente. Cuando se ha recorrido la lista completa, solo permanece un acumulador, que es el valor al que se ha reducido la lista.

Pliegues

• La función foldl, también llamada pliegue por la izquierda. Esta pliega la lista empezando desde la izquierda. La función binaria es aplicada junto a el valor inicial y la cabeza de la lista. Esto produce un nuevo acumulador y la función binaria es vuelta a llamar con ese nuevo valor y el siguiente elemento, etc.

Pliegues

Haskell

```
sumaLista :: (Num a) => [a] -> a
sumaLista xs = foldl (\acc x -> acc + x) 0 xs

ghci> sumaLista [1,2,3,4]
10
```

Pliegues

- Ahora los pliegues por la derecha funcionan igual que los pliegues por la izquierda, solo que el acumulador consume elemento por la derecha. La función binaria de los pliegues por la izquierda como primer parámetro el acumulador y el valor actual como segundo parámetro, la función binaria de los pliegues por la derecha tiene el valor actual como primer parámetro y el acumulador después. Tiene sentido ya que el pliegue por la derecha tiene el acumulador a la derecha.

Pliegues

Ruby

```
> (1..4).inject(&:+)
=> 10
>(1..4).reduce(&:+)
=> 10
```

Groovy

```
[1, 2, 3, 4].inject(0, { sum, value -> sum + value })
```

Pliegues

Javascript

```
[1, 2, 3,4].reduce(function(x,y){return x+y})
```

10

```
[1, 2, 3,4].reduceRight(function(x,y){return x+y})
```

10

Clojure

```
(reduce + [1 2 3 4])
```

10

Scala

```
List(1,2,3,4).foldLeft(0)((x, sum) => sum + x)  //> res0: Int = 10
```

Pliegues

Erlang

```
lists:foldl(fun(X, Sum) -> X + Sum end, 0, [1, 2, 3, 4]).
10
```