Funciones de orden superior

Funciones de orden superior

Las funciones de orden superior son todas aquellas que reciben como argumento una función y/o devuelven como resultado una función.

Funciones que actúan sobre listas

`map :: (a -> b) -> [a] -> [b]`
`(map f xs)` es la lista con los resultados de evaluar la función `f` sobre los elementos de la lista `xs`.

```
λ> map (^2) [1..5]
[1,4,9,16,25]
```

`filter :: (a -> Bool) -> [a] -> [a]`
`(filter p xs)` es la lista con los elementos de la lista `xs` que cumplen la propiedad `p` .

```
λ> filter (even) [1..5]
[2,4]
```

`all :: (a -> Bool) -> [a] -> Bool`
`(all p xs)` comprueba que todos los elementos de lista `xs` cumplen la propiedad `p` .

```
λ> all (even) [1..5]
False
λ> all (>0) [1..5]
True
```

`any :: (a -> Bool) -> [a] -> Bool`
`(any p xs)` comprueba que algún elemento de lista `xs` cumple la propiedad `p` .

```
λ> any (even) [1..5]
True
λ> any (<0) [1..5]
False
```

`takeWhile :: (a -> Bool) -> [a] -> [a]`
`(takeWhile p xs)` e s la lista resultado de coger en orden desde el principio elementos de la lista `xs` mientras cumplan la propiedad `p` .

```
λ> takeWhile (<4) [1..5]
[1,2,3]
λ> takeWhile (odd) [1..5]
[1]
λ> takeWhile (even) [1..5]
[]
```

`dropWhile :: (a -> Bool) -> [a] -> [a]`
`(dropWhile p xs)` es la lista resultado de quitar en orden desde el principio elementos de la lista `xs` mientras cumplan la propiedad `p` .

```
λ> dropWhile (<4) [1..5]
[4,5]
λ> dropWhile (odd) [1..5]
[2,3,4,5]
λ> dropWhile (even) [1..5]
[1,2,3,4,5]
```

`zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]`
`(zipWith f xs ys)` es la lista con los resultados de evaluar la función f sobre las parejas de elementos de las listas `xs` e `ys` hasta la longitud de la más corta.

```
λ> zipWith (+) [1,3,5] [2,4,6]
[3,7,11]
λ> zipWith (+) [1,3] [2,4,6]
[3,7]
λ> zipWith (+) [1,3,5] [2,4]
[3,7]
```

Operaciones con funciones

`(.) :: (b -> c) -> (a -> b) -> a -> c`
`(g . f)` es la función composición de la función `f` con la función `g`.

```
λ> ((^2) . (+1)) 1
4
```

Patrones de definición

`foldr :: (a -> b -> b) -> b -> [a] -> b`
`(foldr f w xs)` es el resultado de evaluar recursivamente la función `f` por la derecha sobre la lista `xs` con valor inicial `w`.
`foldr f w [x1, x2, ..., xn] == f x1 (f x2 ... (f xn w) ... )`

- `foldr (+) 0 == sum`
- `foldr (*) 1 == product`
- `foldr (&&) True == and`
- `foldr (||) False == or`
- `foldr (++) [] == concat`
- `foldr (\ x r -> r++[x]) [] == reverse`
- `foldr (\ x r -> 1+r) 0 == length`
- `foldr (\ x r -> (f x):r) [] == map`
- `foldr (\ x r -> if p x then x:r else r) [] == filter`

```
λ> foldr (+) 0 [1..5]
15
λ> foldr (\ x r -> r++[x]) [] [1..5]
[5,4,3,2,1]
```

`foldl :: (b -> a -> b) -> b -> [a] -> b`
`(foldl f w xs)` es el resultado de evaluar recursivamente la función `f` por la izquierda sobre la lista `xs` con valor inicial `w`.
`foldl f w [x1, x2, ..., xn] == f ( ... (f (f w x1) x2) ... ) xn`

- `foldl (+) 0 == sum`
- `foldl (*) 1 == product`
- `foldl (&&) True == and`
- `foldl (||) False == or`
- `foldl (++) [] == concat`
- `foldl (\ a x -> x:a) [] == reverse`
- `foldl (\ a x -> 1+a) 0 == length`
- `foldl (\ a x -> a++[f x]) [] == map`
- `foldl (\ a x -> if p x then a++[x] else a) [] == filter`

```
λ> foldl (+) 0 [1..5]
15
λ> foldl (\ a x -> x:a) [] [1..5]
[5,4,3,2,1]
```