Funciones sobre listas

Funciones sobre listas

Las funciones sobre listas son todas aquellas que reciben como argumento una o varias listas y/o devuelven como resultado una lista.

Construcción de listas

`[] :: [a]`
`[]` es la lista vacía.

```
λ> []
[]
```

Lista con varios elementos (la coma separa los elementos): `[x,y,z]`

```
λ> [1,2,3,4]
[1,2,3,4]
```

Lista con la secuencia de elementos desde `x` hasta `y`: `[x..y]`

```
λ> [1..5]
[1,2,3,4,5]
λ> [5..1]
[]
```

Lista con la secuencia de elementos de la progresión aritmética que comienza en `x1` y cuyo segundo elemento es `x2`, hasta el que sea menor o igual a `y`: `[x1,x2..y]`

```
λ> [2,4..9]
[2,4,6,8]
λ> [5,4..1]
[5,4,3,2,1]
```

`(:) :: a -> [a]`
`(x:xs)` es la lista que se obtiene añadiendo el elemento `x` al principio de la lista `xs`.

```
λ> 0:[1,2,3]
[0,1,2,3]
λ> 0:1:[2..3]
[0,1,2,3]
λ> 0:1:2:[3]
[0,1,2,3]
λ> 0:[]
[0]
```

Funciones para obtener los elementos de una lista

`head :: [a] -> a`
`(head xs)` es el primer elemento de la lista `xs`.

```
λ> head [1..5]
1
```

`tail :: [a] -> [a]`
`(tail xs)` es el resto de los elementos de la lista `xs`, quitando el primer elemento.

```
λ> tail [1..5]
[2,3,4,5]
```

`last :: [a] -> a`
`(last xs)` es el último elemento de la lista `xs`.

```
λ> last [1..5]
5
```

`init :: [a] -> [a]`
`(init xs)` es el trozo inicial de la lista `xs`, quitando el último elemento.

```
λ> init [1..5]
[1,2,3,4]
```

`(!!) :: [a] -> Int -> a`

`(xs !! i)` es el elemento de la lista `xs`, que ocupa la posición `i`, contando las posiciones desde 0.

```
λ> [1..5] !! 0
1
λ> [1..5] !! 3
4
```

`take :: Int -> [a] -> [a]`

`(take n xs)` es la lista formada por los `n` primeros elementos de la lista `xs`.

```
λ> take 3 [1..5]
[1,2,3]
λ> take 7 [1..5]
[1,2,3,4,5]
λ> take 0 [1..5]
[]
λ> take (-1) [1..5]
[]
```

`drop :: Int -> [a] -> [a]`

`(drop n xs)` es la lista obtenida quitando los `n` primeros elementos de la lista `xs`.

```
λ> drop 3 [1..5]
[4,5]
λ> drop 7 [1..5]
[]
λ> drop 0 [1..5]
[1,2,3,4,5]
λ> drop (-1) [1..5]
[1,2,3,4,5]
```

Otras características

`length :: [a] -> Int`

`(length xs)` es la longitud de la lista `xs`.

```
λ> length [1..5]
5
```

`null :: [a] -> Bool`

`(null xs)` comprueba si la lista `xs` es la lista vacía.

```
λ> null []
True
λ> null [1..5]
False
```

`elem :: a -> [a] -> Bool`

`(elem x xs)` comprueba si el elemento `x` está en la lista `xs`.

```
λ> elem 2 [1..5]
True
λ> elem 7 [1..5]
False
```

`notElem :: a -> [a] -> Bool`

`(notElem x xs)` comprueba si el elemento `x` **no** está en la lista `xs`.

```
λ> notElem 2 [1..5]
False
λ> notElem 7 [1..5]
True
```

Combinación de listas

`(++) :: [a] -> [a] -> [a]`

`(xs ++ ys)` es la concatenación de las listas `xs` e `ys`.

```
λ> [1..3] ++ [4..6]
[1,2,3,4,5,6]
```

`reverse :: [a] -> [a]`

`(reverse xs)` es la inversa de la lista `xs`.

```
λ> reverse [1..5]
[5,4,3,2,1]
```

`zip :: [a] -> [b] -> [(a,b)]`

`(zip xs ys)` empareja los elementos de las listas `xs` e `ys`, hasta la longitud de la más corta.

```
λ> zip [1..5] [2,3,5,7]
[(1,2),(2,3),(3,5),(4,7)]
λ> zip [1..3] [2,3,5,7]
[(1,2),(2,3),(3,5)]
λ> zip [1..4] [2,3,5,7]
[(1,2),(2,3),(3,5),(4,7)]
```

Operaciones sobre los elementos de una lista

`sum :: (Num a) => [a] -> a`
`(sum xs)` es la suma de los elementos de la lista de números `xs`. Se trata de la generalización a listas de `+`.

```
λ> sum [1..5]
15
```

`product :: (Num a) => [a] -> a`
`(product xs)` es el producto de los elementos de la lista de números `xs`. Se trata de la generalización a listas de `*`.

```
λ> product [1..5]
120
```

`maximum :: (Ord a) => [a] -> a`
`(maximum xs)` es el mayor de los elementos de la lista `xs`. Se trata de la generalización a listas de `max`.

```
λ> maximum [1..5]
5
```

`minimum :: (Ord a) => [a] -> a`
`(minimum xs)` es el menor de los elementos de la lista `xs`. Se trata de la generalización a listas de `min`.

```
λ> minimum [1..5]
1
```

`and :: [Bool] -> Bool`
`(and xs)` es la conjunción de los elementos de la lista `xs`. Se trata de la generalización a listas de `&&`.

```
λ> and [True,True,True]
True
λ> and [True,False,True]
False
```

`or :: [Bool] -> Bool`
`(or xs)` es la disyunción de los elementos de la lista `xs`. Se trata de la generalización a listas de `||`.

```
λ> or [False,True,False]
True
λ> or [False,False,False]
False
```

`concat :: [[a]] -> [a]`
`(concat xs)` es la concatenación de los elementos de la lista `xs`. Se trata de la generalización a listas de `++`.

```
λ> concat [[1,2],[3],[],[4,5]]
[1,2,3,4,5]
```