

Listas

Taller de Álgebra I

2do Cuatrimestre 2017

Un nuevo tipo: Listas

Tipo Lista

Las listas pueden contener elementos de cualquier tipo (incluso listas).

- ▶ `[1, 2] :: [Integer]a.`
- ▶ `[div 1 1, div 2 1] :: [Integer].`
- ▶ `[1.0, 2] :: [Float].`
- ▶ `[[1], [2,3], [], [1,1000,2,0]] :: [[Integer]]`
- ▶ `[1, True]` NO ES UNA LISTA VÁLIDA, ¿por qué?
- ▶ `[1.0, div 1 1]` NO ES UNA LISTA VÁLIDA, ¿por qué?

^aEn realidad el tipo es `:t [1,2] :: Num t => [t]`

Un nuevo tipo: Listas

Tipo Lista

Las listas pueden contener elementos de cualquier tipo (incluso listas).

- ▶ `[1, 2] :: [Integer]a.`
- ▶ `[div 1 1, div 2 1] :: [Integer].`
- ▶ `[1.0, 2] :: [Float].`
- ▶ `[[1], [2,3], [], [1,1000,2,0]] :: [[Integer]]`

▶ `[1, True]`

NO ES UNA LISTA VÁLIDA, ¿por qué?

▶ `[1.0, div 1 1]`

NO ES UNA LISTA VÁLIDA, ¿por qué?

^aEn realidad el tipo es `:t [1,2] :: Num t => [t]`

Ejercicio entre todos

Tipar las siguientes expresiones

- ▶ `[(1,2), (3,4), (5,2)]`
- ▶ `[maximo 2 3, fst (2+2, 3+4), 3+4 - 3/4]`
- ▶ `[[], [], [], [], []]`
- ▶ `[]`

Algunas operaciones

- ▶ `head :: [a] -> a`
- ▶ `tail :: [a] -> [a]`
- ▶ `(:) :: a -> [a] -> [a]`
- ▶ `(++) :: [a] -> [a] -> [a]`
- ▶ `length :: [a] -> Integer`
- ▶ `reverse :: [a] -> [a]`

Algunas operaciones

- ▶ `head :: [a] -> a`
- ▶ `tail :: [a] -> [a]`
- ▶ `(:) :: a -> [a] -> [a]`
- ▶ `(++) :: [a] -> [a] -> [a]`
- ▶ `length :: [a] -> Integer`
- ▶ `reverse :: [a] -> [a]`

Tipar y evaluar las siguientes expresiones

- ▶ `head [(1,2), (3,4), (5,2)]`
- ▶ `tail [1,2,3,4,4,3,2,1]`
- ▶ `head []`
- ▶ `head [1,2,3] : [2,3]`
- ▶ `[True, True] ++ [False, False]`
- ▶ `[1,2] : []`

Formas rápidas para crear listas

Prueben las siguientes expresiones en GHCi

- ▶ `[1..100]`
- ▶ `[1,3..100]`
- ▶ `[100..1]`

Formas rápidas para crear listas

Prueben las siguientes expresiones en GHCi

- ▶ `[1..100]`
- ▶ `[1,3..100]`
- ▶ `[100..1]`

Ejercicios

- ▶ Definir la función `listar :: a -> a -> a -> [a]` que toma 3 elementos y los convierte en una lista.
- ▶ Escribir una expresión que denote la lista estrictamente decreciente de enteros que comienza con el número 1 y termina con el número -100.

Pattern matching en listas

Ya vimos cómo hacer *pattern matching* sobre distintos tipos (Bool, Integer, tuplas).
¿Se puede hacer *pattern matching* en listas?

Pattern matching en listas

Ya vimos cómo hacer *pattern matching* sobre distintos tipos (Bool, Integer, tuplas).

¿Se puede hacer *pattern matching* en listas?

¿Cuál es la verdadera forma de las listas?

Las listas tienen dos “pintas”:

- ▶ [] (lista vacía)
- ▶ algo : lista (lista no vacía)

¿Cómo escribir la función `sumatoria :: [Integer] -> Integer` usando *pattern matching*?

Pattern matching en listas

Ya vimos cómo hacer *pattern matching* sobre distintos tipos (Bool, Integer, tuplas).

¿Se puede hacer *pattern matching* en listas?

¿Cuál es la verdadera forma de las listas?

Las listas tienen dos “pintas”:

- ▶ [] (lista vacía)
- ▶ algo : lista (lista no vacía)

¿Cómo escribir la función `sumatoria :: [Integer] -> Integer` usando *pattern matching*?

```
sumatoria [] = 0
sumatoria (x:xs) = sumatoria xs + x
```

Pattern matching en listas

Ya vimos cómo hacer *pattern matching* sobre distintos tipos (`Bool`, `Integer`, tuplas).
¿Se puede hacer *pattern matching* en listas?

¿Cuál es la verdadera forma de las listas?

Las listas tienen dos “pintas”:

- ▶ `[]` (lista vacía)
- ▶ `algo : lista` (lista no vacía)

¿Cómo escribir la función `sumatoria :: [Integer] -> Integer` usando *pattern matching*?

```
sumatoria [] = 0
sumatoria (x:xs) = sumatoria xs + x
```

Las listas también admiten el patrón `_`, que se corresponde con cualquier valor, pero no liga ninguna variable. Por ejemplo:

```
tieneDosElementos :: [a] -> Bool
tieneDosElementos (_:_:[]) = True
tieneDosElementos _ = False
```

Ejercicio: máximo

Repensar la función `maximo` utilizando *pattern matching*.

Recursión sobre listas

¿Se puede pensar recursivamente en listas? ¿Cómo?

Pensar las siguientes funciones

- ▶ `sumatoria :: [Integer] -> Integer`
que indica la suma de los elementos de una lista.
- ▶ `pertenece :: Integer -> [Integer] -> Bool`
que indica si un elemento aparece en la lista. Por ejemplo:
`pertenece 9 [] ~> False`
`pertenece 9 [1,2,3] ~> False`
`pertenece 9 [1,2,9,9,-1,0] ~> True`

Recursión sobre listas

¿Se puede pensar recursivamente en listas? ¿Cómo?

Pensar las siguientes funciones

- ▶ `sumatoria :: [Integer] -> Integer`
que indica la suma de los elementos de una lista.
- ▶ `pertenece :: Integer -> [Integer] -> Bool`
que indica si un elemento aparece en la lista. Por ejemplo:
`pertenece 9 [] ~> False`
`pertenece 9 [1,2,3] ~> False`
`pertenece 9 [1,2,9,9,-1,0] ~> True`

¿Me sirve de algo para definir la función, el resultado sobre la cola de la lista?

- ▶ Me sirve para sumar la lista `[a, b, c]` saber cuánto es la suma de la lista `[b, c]` (la cola de la lista) y saber que el primer elemento es `a`?
- ▶ Me sirve para saber si pertenece `x [a, b, c, d]` saber si pertenece `x [b, c, d]` y saber que el primer elemento es `a`?

Recursión sobre listas

¿Se puede pensar recursivamente en listas? ¿Cómo?

Pensar las siguientes funciones

- ▶ `sumatoria :: [Integer] -> Integer`
que indica la suma de los elementos de una lista.
- ▶ `pertenece :: Integer -> [Integer] -> Bool`
que indica si un elemento aparece en la lista. Por ejemplo:
`pertenece 9 [] ~> False`
`pertenece 9 [1,2,3] ~> False`
`pertenece 9 [1,2,9,9,-1,0] ~> True`

¿Me sirve de algo para definir la función, el resultado sobre la cola de la lista?

- ▶ Me sirve para sumar la lista `[a, b, c]` saber cuánto es la suma de la lista `[b, c]` (la cola de la lista) y saber que el primer elemento es `a`?
- ▶ Me sirve para saber si pertenece `x [a, b, c, d]` saber si pertenece `x [b, c, d]` y saber que el primer elemento es `a`?

Idea: Pensar como combinar el resultado de la función sobre la cola de la lista con el primer elemento. Recordar:

- ▶ `head [1, 2, 3] ~> 1`
- ▶ `tail [1, 2, 3] ~> [2, 3]`

Recursión sobre listas

¿Se puede pensar recursivamente en listas? ¿Cómo?

Pensar las siguientes funciones

- ▶ `sumatoria :: [Integer] -> Integer`
que indica la suma de los elementos de una lista.
- ▶ `pertenece :: Integer -> [Integer] -> Bool`
que indica si un elemento aparece en la lista. Por ejemplo:
`pertenece 9 [] ~> False`
`pertenece 9 [1,2,3] ~> False`
`pertenece 9 [1,2,9,9,-1,0] ~> True`

¿Me sirve de algo para definir la función, el resultado sobre la cola de la lista?

- ▶ Me sirve para sumar la lista `[a, b, c]` saber cuánto es la suma de la lista `[b, c]` (la cola de la lista) y saber que el primer elemento es `a`?
- ▶ Me sirve para saber si pertenece `x [a, b, c, d]` saber si pertenece `x [b, c, d]` y saber que el primer elemento es `a`?

Idea: Pensar como combinar el resultado de la función sobre la cola de la lista con el primer elemento. Recordar:

- ▶ `head [1, 2, 3] ~> 1`
- ▶ `tail [1, 2, 3] ~> [2, 3]`

Ejercicios

- ▶ `productoria :: [Integer] -> Integer` que devuelve la productoria de los elementos.
- ▶ `sumarN :: Integer -> [Integer] -> [Integer]` que dado un número N y una lista xs , suma N a cada elemento de xs .
- ▶ `sumarElUltimo :: [Integer] -> [Integer]` que dada una lista no vacía xs , suma el último elemento a cada elemento de xs . Ejemplo `sumarElUltimo [1,2,3] ~> [4,5,6]`
- ▶ `sumarElPrimero :: [Integer] -> [Integer]` que dada una lista no vacía xs , suma el primer elemento a cada elemento de xs . Ejemplo `sumarElPrimero [1,2,3] ~> [2,3,4]`
- ▶ `pares :: [Integer] -> [Integer]` que devuelve una lista con los elementos pares de la lista original. Ejemplo `pares [1,2,3,8] ~> [2,8]`
- ▶ `multiplosDeN :: Integer -> [Integer] -> [Integer]` que dado un número N y una lista xs , devuelve una lista con los elementos multiplos N de xs .
- ▶ `quitar :: Integer -> [Integer] -> [Integer]` que elimina la primera aparición del elemento en la lista (de haberla).
- ▶ `hayRepetidos :: [Integer] -> Bool` que indica si una lista tiene elementos repetidos.
- ▶ `eliminarRepetidos :: [Integer] -> [Integer]` que deja en la lista una única aparición de cada elemento, eliminando las repeticiones adicionales.
- ▶ `maximo :: [Integer] -> Integer` que calcula el máximo elemento de una lista no vacía.
- ▶ `ordenar :: [Integer] -> [Integer]` que ordena los elementos de forma creciente.