Programación Funcional en Haskell

Paradigmas de Lenguajes de Programación

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

29 de Enero de 2018

Repaso: Expresiones y tipos básicos

Tipos elementales

```
1                -- Int
'a'              -- Char
1.2              -- Float
True             -- Bool
```

Repaso: Expresiones y tipos básicos

Tipos elementales

```
1                -- Int
'a'              -- Char
1.2              -- Float
True             -- Bool
[1,2,3]          -- [Int]
```

Repaso: Expresiones y tipos básicos

Tipos elementales

```
1                -- Int
'a'              -- Char
1.2              -- Float
True             -- Bool
[1,2,3]          -- [Int]
(1, True)        -- (Int, Bool)
```

Repaso: Expresiones y tipos básicos

Tipos elementales

```
1                -- Int
'a'              -- Char
1.2              -- Float
True             -- Bool
[1,2,3]          -- [Int]
(1, True)        -- (Int, Bool)
length           -- [a] -> Int
```

Repaso: Expresiones y tipos básicos

Tipos elementales

```
1                -- Int
'a'              -- Char
1.2              -- Float
True             -- Bool
[1,2,3]          -- [Int]
(1, True)        -- (Int, Bool)
length           -- [a] -> Int
length [1,2,3]   -- Int
```

Repaso: Expresiones y tipos básicos

Tipos elementales

```
1                   -- Int
'a'                 -- Char
1.2                 -- Float
True                -- Bool
[1,2,3]             -- [Int]
(1, True)           -- (Int, Bool)
length              -- [a] -> Int
length [1,2,3]      -- Int
\x -> x             -- a -> a
```

Repaso: Expresiones y tipos básicos

Tipos elementales

```
1                -- Int
'a'              -- Char
1.2              -- Float
True             -- Bool
[1,2,3]          -- [Int]
(1, True)        -- (Int, Bool)
length           -- [a] -> Int
length [1,2,3]   -- Int
\x -> x          -- a -> a
```

Guardas

```
signo n | n >= 0    = True
        | otherwise = False
```

Repaso: Expresiones y tipos básicos

Tipos elementales

```
1                -- Int
'a'              -- Char
1.2              -- Float
True             -- Bool
[1,2,3]          -- [Int]
(1, True)        -- (Int, Bool)
length           -- [a] -> Int
length [1,2,3]   -- Int
\x -> x          -- a -> a
```

Guardas

```
signo n | n >= 0    = True
        | otherwise = False
```

Pattern matching

```
longitud [] = 0
longitud (x:xs) = 1 + (longitud xs)
```

Repaso: Polimorfismo paramétrico

`todosIguales` es una función que determina si todos los elementos de una lista son iguales entre sí.

Ejercicio

```
todosIguales :: ??
todosIguales = ...
```

Repaso: Polimorfismo paramétrico

`todosIguales` es una función que determina si todos los elementos de una lista son iguales entre sí.

Ejercicio

```
todosIguales :: ??
todosIguales = ...
```

- El sistema de tipos de Haskell permite definir funciones para ser usadas con más de un tipo
- Su tipo se expresa con *variables de tipo*

Repaso: Polimorfismo paramétrico

`todosIguales` es una función que determina si todos los elementos de una lista son iguales entre sí.

Ejercicio

```
todosIguales :: ??
todosIguales = ...
```

- El sistema de tipos de Haskell permite definir funciones para ser usadas con más de un tipo
- Su tipo se expresa con *variables de tipo*

Ejemplo

```
sort :: Ord a => [a] -> [a]
```

Clases de tipos: conjuntos de tipos con ciertas operaciones

- `Eq`
- `Ord`
- `Num`
- `Show`

Repaso: Polimorfismo paramétrico

`todosIguales` es una función que determina si todos los elementos de una lista son iguales entre sí.

Ejercicio

```
todosIguales :: ??
todosIguales = ...
```

- El sistema de tipos de Haskell permite definir funciones para ser usadas con más de un tipo
- Su tipo se expresa con *variables de tipo*

Ejemplo

```
sort :: Ord a => [a] -> [a]
```

Clases de tipos: conjuntos de tipos con ciertas operaciones

- `Eq`
- `Ord`
- `Num`
- `Show`

Listas infinitas

Definición de listas

- Listas por extensión
  `[0, 3, 0, 3, 4, 5, 6]`
- Secuencias aritméticas
  `[1..4]  [5, 7..13]`
- Listas por comprensión
  `[expresion | selectores, condiciones]`
  `[(x, y) | x <-[0..3], y <-[0..3]]`

¿Las listas pueden ser infinitas?

Listas infinitas

Definición de listas

- Listas por extensión
  `[0, 3, 0, 3, 4, 5, 6]`
- Secuencias aritméticas
  `[1..4]  [5, 7..13]`
- Listas por comprensión
  `[expresion | selectores, condiciones]`
  `[(x, y) | x <-[0..3], y <-[0..3]]`

¿Las listas pueden ser infinitas?

Ejemplo

- `infinitosUnos = 1 : infinitosUnos`
- `naturales =`

Listas infinitas

Definición de listas

- Listas por extensión
  `[0, 3, 0, 3, 4, 5, 6]`
- Secuencias aritméticas
  `[1..4]  [5, 7..13]`
- Listas por comprensión
  `[expresion | selectores, condiciones]`
  `[(x, y) | x <-[0..3], y <-[0..3]]`

¿Las listas pueden ser infinitas?

Ejemplo

- `infinitosUnos = 1 : infinitosUnos`
- `naturales =  [0..]`
- `multiplosDe3 =`

Listas infinitas

Definición de listas

- Listas por extensión
  `[0, 3, 0, 3, 4, 5, 6]`
- Secuencias aritméticas
  `[1..4]  [5, 7..13]`
- Listas por comprensión
  `[expresion | selectores, condiciones]`
  `[(x, y) | x <-[0..3], y <-[0..3]]`

¿Las listas pueden ser infinitas?

Ejemplo

- `infinitosUnos = 1 : infinitosUnos`
- `naturales = [0..]`
- `multiplosDe3 = [0,3..]`
- `repeat "hola"`

Listas infinitas

Definición de listas

- Listas por extensión
  `[0, 3, 0, 3, 4, 5, 6]`
- Secuencias aritméticas
  `[1..4]  [5, 7..13]`
- Listas por comprensión
  `[expresion | selectores, condiciones]`
  `[(x, y) | x <-[0..3], y <-[0..3]]`

¿Las listas pueden ser infinitas?

Ejemplo

- `infinitosUnos = 1 : infinitosUnos`
- `naturales = [0..]`
- `multiplosDe3 = [0,3..]`
- `repeat "hola"`
- `primos =`

Listas infinitas

Definición de listas

- Listas por extensión
  `[0, 3, 0, 3, 4, 5, 6]`
- Secuencias aritméticas
  `[1..4]  [5, 7..13]`
- Listas por comprensión
  `[expresion | selectores, condiciones]`
  `[(x, y) | x <-[0..3], y <-[0..3]]`

¿Las listas pueden ser infinitas?

Ejemplo

- `infinitosUnos = 1 : infinitosUnos`
- `naturales = [0..]`
- `multiplosDe3 = [0,3..]`
- `repeat "hola"`
- `primos = [n | n <- [2..], esPrimo n]`

Listas infinitas

Definición de listas

- Listas por extensión
  `[0, 3, 0, 3, 4, 5, 6]`
- Secuencias aritméticas
  `[1..4]  [5, 7..13]`
- Listas por comprensión
  `[expresion | selectores, condiciones]`
  `[(x, y) | x <-[0..3], y <-[0..3]]`

¿Las listas pueden ser infinitas?

Ejemplo

- `infinitosUnos = 1 : infinitosUnos`
- `naturales = [0..]`
- `multiplosDe3 = [0,3..]`
- `repeat "hola"`
- `primos = [n | n <- [2..], esPrimo n]`

Módelo de Cómputo

Ejercicio

Mostrar los pasos necesarios para reducir `nUnos 2`

```
take :: Int -> [a] -> [a]
take 0 l      = []
take n []     = []
take n (x:xs) = x : (take (n-1) xs)

infinitosUnos :: [Int]
infinitosUnos = 1 : infinitosUnos

nUnos :: Int -> [Int]
nUnos n = take n infinitosUnos
```

Funciones de alto orden

Definamos las siguientes funciones
Precondición: las listas tienen algún elemento.

- `maximo :: Ord a => [a] -> a`
- `minimo :: Ord a => [a] -> a`
- `listaMasCorta :: [[a]] -> [a]`

Funciones de alto orden

Definamos las siguientes funciones
Precondición: las listas tienen algún elemento.

- `maximo :: Ord a => [a] -> a`
- `minimo :: Ord a => [a] -> a`
- `listaMasCorta :: [[a]] -> [a]`

Siempre hago lo mismo... ¿Se podrá generalizar? ¿Cómo?

Ejercicio

- `mejorSegun ::`

Funciones de alto orden

Definamos las siguientes funciones
Precondición: las listas tienen algún elemento.

- `maximo :: Ord a => [a] -> a`
- `minimo :: Ord a => [a] -> a`
- `listaMasCorta :: [[a]] -> [a]`

Siempre hago lo mismo... ¿Se podrá generalizar? ¿Cómo?

Ejercicio

- `mejorSegun :: (a -> a -> Bool) -> [a] -> a`

Funciones de alto orden

Definamos las siguientes funciones
Precondición: las listas tienen algún elemento.

- `maximo :: Ord a => [a] -> a`
- `minimo :: Ord a => [a] -> a`
- `listaMasCorta :: [[a]] -> [a]`

Siempre hago lo mismo... ¿Se podrá generalizar? ¿Cómo?

Ejercicio

- `mejorSegun :: (a -> a -> Bool) -> [a] -> a`
- Reescribir `maximo` y `listaMasCorta` en base a `mejorSegun`