Lenguajes de programación

Evaluación Perezosa y otras monerías en Haskell

"Guardias"

- Se usan en definiciones por casos
- Ejemplo:

```
fact1 :: Int -> Int
fact1 n   -- usa "guardias"
  | n == 0 = 1
  | n > 0 = n * fact1 (n-1)
```

Ejemplo

- Extraer los números pares de una lista

```
getEven [1..10] => [2,4,6,8,10]

getEven :: [Int] -> [Int]
getEven [] = []
getEven (x:xs)
    | even x = x : getEven xs
    | otherwise = getEven xs
```

Comprensión de listas

- `[ f x | x <- xs ]` significa la lista de todas las `f x` tales que `x` pertenece a `xs`
  - `x <- xs` es denominado un **generador**
- Ejemplo: producto cartesiano

```
prodcart [1,2] [3,4]
=> [(1,3),(1,4),(2,3),(2,4)]

prodcart :: [a] -> [b] -> [(a, b)]
prodcart xs ys =
  [ (x,y) | x <- xs, y <- ys ]
```

Ejemplo

- Filtrar una lista de números seleccionando sólo los positivos

```
positivos [-3..3] => [1,2,3]

positivos :: [Int] -> [Int]
positivos l = [x | x <- l, x > 0]
```

Ejemplo

- Contar el número de ocurrencias de un elemento entero en una lista de enteros

```
ocurre 3 [1,2,3,2,3,2,1] => 2

ocurre :: Int -> [Int] -> Int
ocurre e l =
   length [x | x <- l, x == e]
```

Definiciones Locales con constructor “where”

- Ordenar ascendentemente una lista de números utilizando el algoritmo quicksort.

```
qsort [1,2,6,3,5,4] => [1,2,3,4,5,6]

qsort :: [Int] -> [Int]
qsort []     = []
qsort (x:xs) =
      qsort lt ++ [x] ++ qsort ge
  where
      lt = [y | y <- xs, y < x]
      ge = [y | y <- xs, y >= x]
```

Estructuras de datos “infinitas”

- `[1..]` es `[1,2,3,4,5..]`
- `[1,3..]` es `[1,3,5,7,…]`

- Uso:

```
take 6 [1..]
=> [1,2,3,4,5,6]
takeWhile (< 30) [5,11..]
=> [5,11,17,23,29]
```

Evaluación perezosa

- Las expresiones son evaluadas hasta que se necesitan:

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

- ¿Cómo se evalúa la expresión take 2 [5..]?

```
take 2 [5..] =
  take 2 5:[6..] =
  5: (take 1 [6..])=
  5: (take 1 6:[7..])=
  5: 6: (take 0 [7..])=
  5: 6: [] => [5,6]
```

Ejemplos

```
-- lista infinita de unos
unos :: [Int]
unos = 1:unos

-- lista infinita de números a
-- partir de n
numsDesde :: Int -> [Int]
numsDesde n = n:numsDesde (n+1)

-- lista infinita de números al
-- cuadrado
cuadrados :: [Int]
cuadrados = map (^2) [1..]
```

Ejemplos

```
-- factorial con listas infinitas
factorial:: Int -> Int
factorial n = product (take n [1..])

-- suma primeras n raíces cuadradas
sumfirstsqrt :: Int -> Float
sumfirstsqrt n =
   sum (map sqrt (take n [1.0..]))
```

Ejemplo fibonacci

- Generar la serie de Fibonacci
- Uso de Función `zip :: [a] -> [b] -> [(a,b)]`

```
zip [1,2,3,4,5] [6..]
=> [(1,6),(2,7),(3,8),(4,9),(5,10)]

take 8 fibo => [1,1,2,3,5,8,13,21]

fibo = 1:1:[a+b | (a,b) <-
            zip fibo (tail fibo)]
```