

Exercitium (curso 2013–14)

Ejercicios de programación funcional con Haskell

José A. Alonso Jiménez

Grupo de Lógica Computacional
Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, 8 de diciembre de 2018

Esta obra está bajo una licencia Reconocimiento–NoComercial–CompartirIgual 2.5 Spain de Creative Commons.

Se permite:

- copiar, distribuir y comunicar públicamente la obra
- hacer obras derivadas

Bajo las condiciones siguientes:

Reconocimiento. Debe reconocer los créditos de la obra de la manera especificada por el autor.



No comercial. No puede utilizar esta obra para fines comerciales.



Compartir bajo la misma licencia. Si altera o transforma esta obra, o genera una obra derivada, sólo puede distribuir la obra generada bajo una licencia idéntica a ésta.

- Al reutilizar o distribuir la obra, tiene que dejar bien claro los términos de la licencia de esta obra.
- alguna de estas condiciones puede no aplicarse si se obtiene el permiso del titular de los derechos de autor.

Esto es un resumen del texto legal (la licencia completa). Para ver una copia de esta licencia, visite <http://creativecommons.org/licenses/by-nc-sa/2.5/es/> o envíe una carta a Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

Para Guiomar

Índice general

| | | |
|---|--------------------------|----|
| 1 | Iguals al siguiente | 9 |
| 2 | Ordenación por el máximo | 11 |

Introducción

"The chief goal of my work as an educator and author is to help people learn to write beautiful programs."

(Donald Knuth en [Computer programming as an art](#))

Este libro es una recopilación de las soluciones de los ejercicios propuestos en el blog [Exercitium](#)¹ durante el curso 2013-14.

El principal objetivo de Exercitium es servir de complemento a la asignatura de [Informática](#)² de 1º del Grado en Matemáticas de la Universidad de Sevilla.

Con los problemas de Exercitium, a diferencia de los de las [relaciones](#)³, se pretende practicar con los conocimientos adquiridos durante todo el curso, mientras que con las relaciones están orientadas a los nuevos conocimientos.

Habitualmente de cada ejercicio se muestra distintas soluciones y se compara sus eficiencias.

La dinámica del blog es la siguiente: cada día, de lunes a viernes, se propone un ejercicio para que los alumnos escriban distintas soluciones en los comentarios. Pasado 7 días de la propuesta de cada ejercicio, se cierra los comentarios y se publica una selección de sus soluciones.

Para conocer la cronología de los temas explicados se puede consultar el [diario de clase](#)⁴.

El código del libro se encuentra en [GitHub](#)⁵

¹<https://www.glc.us.es/~jalonso/exercitium>

²<https://www.cs.us.es/~jalonso/cursos/ilm-13>

³<https://www.cs.us.es/~jalonso/cursos/ilm-13/ejercicios/ejercicios-I1M-2013.pdf>

⁴<https://www.glc.us.es/~jalonso/vestigium/category/curso/ilm/ilm2013>

⁵<https://github.com/jaalonso/Exercitium2013>

Ejercicio 1

Iguales al siguiente

Ejercicio propuesto el 21-4-14

Definir la función

```
igualesAlSiguiente :: Eq a => [a] -> [a]
```

tal que (igualesAlSiguiente xs) es la lista de los elementos de xs que son iguales a su siguiente. Por ejemplo,

```
igualesAlSiguiente [1,2,2,2,3,3,4] == [2,2,3]
igualesAlSiguiente [1..10]         == []
```

Soluciones

```
import Data.List (group)
import Test.QuickCheck

-- 1ª definición (por comprensión):
igualesAlSiguiente :: Eq a => [a] -> [a]
igualesAlSiguiente xs =
  [x | (x,y) <- zip xs (tail xs), x == y]

-- 2ª definición (por recursión):
igualesAlSiguiente2 :: Eq a => [a] -> [a]
```

```

igualesAlSiguiente2 (x:y:zs)
  | x == y    = x : igualesAlSiguiente2 (y:zs)
  | otherwise = igualesAlSiguiente2 (y:zs)
igualesAlSiguiente2 _ = []

-- 3ª definición (con concat y comprensión):
igualesAlSiguiente3 :: Eq a => [a] -> [a]
igualesAlSiguiente3 xs = concat [ys | (_,ys) <- group xs]

-- 4ª definición (con concat y map):
igualesAlSiguiente4 :: Eq a => [a] -> [a]
igualesAlSiguiente4 xs = concat (map tail (group xs))

-- 5ª definición (con concatMap):
igualesAlSiguiente5 :: Eq a => [a] -> [a]
igualesAlSiguiente5 xs = concatMap tail (group xs)

-- 6ª definición (con concatMap y sin argumentos):
igualesAlSiguiente6 :: Eq a => [a] -> [a]
igualesAlSiguiente6 = concatMap tail . group

-- Equivalencia
-- =====

-- La propiedad es
prop_igualesAlSiguiente_equiv :: [Int] -> Bool
prop_igualesAlSiguiente_equiv xs =
  igualesAlSiguiente xs == igualesAlSiguiente2 xs &&
  igualesAlSiguiente xs == igualesAlSiguiente3 xs &&
  igualesAlSiguiente xs == igualesAlSiguiente4 xs &&
  igualesAlSiguiente xs == igualesAlSiguiente5 xs &&
  igualesAlSiguiente xs == igualesAlSiguiente6 xs

-- La comprobación es
--    +++ OK, passed 100 tests.
--    (0.07 secs, 9,911,528 bytes)

```

Ejercicio 2

Ordenación por el máximo

Ejercicio propuesto el 22-4-18

Definir la función

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

tal que (ordenadosPorMaximo xss) es la lista de los elementos de xss ordenada por sus máximos. Por ejemplo,

```
ghci> ordenadosPorMaximo [[3,2],[6,7,5],[1,4]]
[[3,2],[1,4],[6,7,5]]
ghci> ordenadosPorMaximo ["este","es","el","primero"]
["el","primero","es","este"]
```

Soluciones

```
import Data.List (sort)
import GHC.Exts  (sortWith)
import Test.QuickCheck

-- 1ª definición
ordenadosPorMaximo :: Ord a => [[a]] -> [[a]]
ordenadosPorMaximo xss =
  map snd (sort [(maximum xs,xs) | xs <- xss])
```

```

-- 2ª definición
ordenadosPorMaximo2 :: Ord a => [[a]] -> [[a]]
ordenadosPorMaximo2 xss =
  [xs | (_,xs) <- sort [(maximum xs,xs) | xs <- xss]]

-- 3ª definición:
ordenadosPorMaximo3 :: Ord a => [[a]] -> [[a]]
ordenadosPorMaximo3 = sortWith maximum

-- Equivalencia
-- =====

verificaOrdenadosPorMaximo :: IO ()
verificaOrdenadosPorMaximo =
  quickCheck prop_ordenadosPorMaximo

prop_ordenadosPorMaximo :: [[Int]] -> Bool
prop_ordenadosPorMaximo xs =
  ordenadosPorMaximo ys == ordenadosPorMaximo2 ys
  where ys = filter (not . null) xs

-- Comprobación
--    λ> verificaOrdenadosPorMaximo
--    +++ OK, passed 100 tests.

```