

# Exercitium (curso 2018–19)

## 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, 23 de noviembre 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.

# Índice general

1 Listas equidigitales	7
2 Distancia de Hamming	9
3 Último dígito no nulo del factorial	13
4 Diferencia simétrica	17
5 Números libres de cuadrados	19
6 Capicúas productos de dos números de dos dígitos	23



# Introducción

Este libro es una recopilación de las soluciones de los ejercicios propuestos en el blog [Exercitium](https://www.glc.us.es/~jalonso/exercitium)<sup>1</sup> durante el curso 2018-19.

El principal objetivo de Exercitium es servir de complemento a la asignatura de [Informática](https://www.cs.us.es/~jalonso/cursos/ilm-18)<sup>2</sup> de 1º del Grado en Matemáticas de la Universidad de Sevilla.

Con los problemas de Exercitium, a diferencia de los de las [relaciones](https://www.cs.us.es/~jalonso/cursos/ilm-18/ejercicios/ejercicios-I1M-2018.pdf)<sup>3</sup>, 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](https://www.glc.us.es/~jalonso/vestigium/category/curso/ilm/ilm2018)<sup>4</sup>.

En el libro se irán añadiendo semanalmente las soluciones de los ejercicios del curso.

El código del libro se encuentra en [GitHub](https://github.com/jalonso/Exercitium2018)<sup>5</sup>

---

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

<sup>2</sup><https://www.cs.us.es/~jalonso/cursos/ilm-18>

<sup>3</sup><https://www.cs.us.es/~jalonso/cursos/ilm-18/ejercicios/ejercicios-I1M-2018.pdf>

<sup>4</sup><https://www.glc.us.es/~jalonso/vestigium/category/curso/ilm/ilm2018>

<sup>5</sup><https://github.com/jalonso/Exercitium2018>



# Ejercicio 1

## Listas equidigitales

### Ejercicio propuesto el 9-11-18

Una lista de números naturales es equidigital si todos sus elementos tienen el mismo número de dígitos.

Definir la función

---

```
equidigital :: [Int] -> Bool
```

---

tal que (equidigital xs) se verifica si xs es una lista equidigital. Por ejemplo,

---

```
equidigital [343,225,777,943] == True
equidigital [343,225,777,94,3] == False
```

---

## Soluciones

```
-- 1ª definición
-- =====
```

```
equidigital :: [Int] -> Bool
equidigital xs = todosIguales (numerosDeDigitos xs)
```

```
-- (numerosDeDigitos xs) es la lista de los números de dígitos de
```

```

-- los elementos de xs. Por ejemplo,
--   numerosDeDigitos [343,225,777,943] == [3,3,3,3]
--   numerosDeDigitos [343,225,777,94,3] == [3,3,3,2,1]
numerosDeDigitos :: [Int] -> [Int]
numerosDeDigitos xs = [numeroDeDigitos x | x <- xs]

-- (numeroDeDigitos x) es el número de dígitos de x. Por ejemplo,
--   numeroDeDigitos 475 == 3
numeroDeDigitos :: Int -> Int
numeroDeDigitos x = length (show x)

-- (todosIguales xs) se verifica si todos los elementos de xs son
-- iguales. Por ejemplo,
--   todosIguales [3,3,3,3] == True
--   todosIguales [3,3,3,2,1] == False
todosIguales :: Eq a => [a] -> Bool
todosIguales (x:y:zs) = x == y && todosIguales (y:zs)
todosIguales _        = True

-- 2ª definición
-- =====

equidigital2 :: [Int] -> Bool
equidigital2 []      = True
equidigital2 (x:xs) = and [numeroDeDigitos y == n | y <- xs]
                    where n = numeroDeDigitos x

-- 3ª definición
-- =====

equidigital3 :: [Int] -> Bool
equidigital3 (x:y:zs) = numeroDeDigitos x == numeroDeDigitos y &&
                        equidigital3 (y:zs)
equidigital3 _        = True

```



## Ejercicio 2

# Distancia de Hamming

### Ejercicio propuesto el 10-11-18

La distancia de Hamming entre dos listas es el número de posiciones en que los correspondientes elementos son distintos. Por ejemplo, la distancia de Hamming entre `romaz` y `loba` es 2 (porque hay 2 posiciones en las que los elementos correspondientes son distintos: la 1ª y la 3ª).

Definir la función

---

```
distancia :: Eq a => [a] -> [a] -> Int
```

---

tal que `(distancia xs ys)` es la distancia de Hamming entre `xs` e `ys`. Por ejemplo,

---

```
distancia "romano" "comino" == 2
distancia "romano" "camino" == 3
distancia "roma"   "comino" == 2
distancia "roma"   "camino" == 3
distancia "romano" "ron"    == 1
distancia "romano" "cama"   == 2
distancia "romano" "rama"   == 1
```

---

Comprobar con QuickCheck si la distancia de Hamming tiene la siguiente propiedad: `distancia(xs,ys) = 0` si, y sólo si, `xs = ys` y, en el caso de que no se verifique, modificar ligeramente la propiedad para obtener una condición necesaria y suficiente de `distancia(xs,ys) = 0`.

## Soluciones

```
import Test.QuickCheck
```

```
-- 1ª definición:
```

```
distancia :: Eq a => [a] -> [a] -> Int
distancia xs ys = length [(x,y) | (x,y) <- zip xs ys, x /= y]
```

```
-- 2ª definición:
```

```
distancia2 :: Eq a => [a] -> [a] -> Int
distancia2 [] _ = 0
distancia2 _ [] = 0
distancia2 (x:xs) (y:ys) | x /= y = 1 + distancia2 xs ys
                        | otherwise = distancia2 xs ys
```

```
-- La propiedad es
```

```
prop_distancial :: [Int] -> [Int] -> Bool
prop_distancial xs ys =
  (distancia xs ys == 0) == (xs == ys)
```

```
-- La comprobación es
```

```
-- ghci> quickCheck prop_distancial
-- *** Failed! Falsifiable (after 2 tests and 1 shrink):
-- []
-- [1]
```

```
-- En efecto,
```

```
-- ghci> distancia [] [1] == 0
-- True
-- ghci> [] == [1]
-- False
```

```
-- La primera modificación es restringir la propiedad a lista de igual
-- longitud:
```

```
prop_distancia2 :: [Int] -> [Int] -> Property
prop_distancia2 xs ys =
  length xs == length ys ==>
  (distancia xs ys == 0) == (xs == ys)
```

```
-- La comprobación es
```

```
-- ghci> quickCheck prop_distancia2
-- *** Gave up! Passed only 33 tests.

-- Nota. La propiedad se verifica, pero al ser la condición demasiado
-- restringida sólo 33 de los casos la cumple.

-- La segunda restricción es limitar las listas a la longitud de la más
-- corta:
prop_distancia3 :: [Int] -> [Int] -> Bool
prop_distancia3 xs ys =
  (distancia xs ys == 0) == (take n xs == take n ys)
  where n = min (length xs) (length ys)

-- La comprobación es
-- ghci> quickCheck prop_distancia3
-- +++ OK, passed 100 tests.
```



## Ejercicio 3

# Último dígito no nulo del factorial

### Ejercicio propuesto el 13-11-18

El factorial de 7 es  $7! = 1 * 2 * 3 * 4 * 5 * 6 * 7 = 5040$ . Por tanto, el último dígito no nulo del factorial de 7 es 4.

Definir la función

---

```
ultimoNoNuloFactorial :: Integer -> Integer
```

---

tal que (ultimoNoNuloFactorial n) es el último dígito no nulo del factorial de n. Por ejemplo,

---

```
ultimoNoNuloFactorial 7  == 4
ultimoNoNuloFactorial 10 == 8
ultimoNoNuloFactorial 12 == 6
ultimoNoNuloFactorial 97 == 2
ultimoNoNuloFactorial 0  == 1
```

---

Comprobar con QuickCheck que si n es mayor que 4, entonces el último dígito no nulo del factorial de n es par.

## Solución

```

import Test.QuickCheck

-- 1ª definición
-- =====

ultimoNoNuloFactorial :: Integer -> Integer
ultimoNoNuloFactorial n = ultimoNoNulo (factorial n)

-- (ultimoNoNulo n) es el último dígito no nulo de n. Por ejemplo,
--   ultimoNoNulo 5040 == 4
ultimoNoNulo :: Integer -> Integer
ultimoNoNulo n
  | m /= 0    = m
  | otherwise = ultimoNoNulo (n `div` 10)
  where m = n `rem` 10

-- (factorial n) es el factorial de n. Por ejemplo,
--   factorial 7 == 5040
factorial :: Integer -> Integer
factorial n = product [1..n]

-- 2ª definición
-- =====

ultimoNoNuloFactorial2 :: Integer -> Integer
ultimoNoNuloFactorial2 n = ultimoNoNulo2 (factorial n)

-- (ultimoNoNulo2 n) es el último dígito no nulo de n. Por ejemplo,
--   ultimoNoNulo2 5040 == 4
ultimoNoNulo2 :: Integer -> Integer
ultimoNoNulo2 n = read [head (dropWhile (=='0') (reverse (show n)))]

-- Comprobación
-- =====

-- La propiedad es
prop_ultimoNoNuloFactorial :: Integer -> Property
prop_ultimoNoNuloFactorial n =

```

---

```
n > 4 ==> even (ultimoNoNuloFactorial n)

-- La comprobación es
--   ghci> quickCheck prop_ultimoNoNuloFactorial
--   +++ OK, passed 100 tests.
```





## Ejercicio 4

### Diferencia simétrica

#### Ejercicio propuesto el 14-11-18

La **diferencia simétrica** de dos conjuntos es el conjunto cuyos elementos son aquellos que pertenecen a alguno de los conjuntos iniciales, sin pertenecer a ambos a la vez. Por ejemplo, la diferencia simétrica de 2,5,3 y 4,2,3,7 es 5,4,7.

Definir la función

---

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

---

tal que (diferenciaSimetrica xs ys) es la diferencia simétrica de xs e ys. Por ejemplo,

---

```
diferenciaSimetrica [2,5,3] [4,2,3,7] == [4,5,7]
diferenciaSimetrica [2,5,3] [5,2,3]   == []
diferenciaSimetrica [2,5,2] [4,2,3,7] == [3,4,5,7]
diferenciaSimetrica [2,5,2] [4,2,4,7] == [4,5,7]
diferenciaSimetrica [2,5,2,4] [4,2,4,7] == [5,7]
```

---

## Soluciones

```
import Data.List
```

```
-- 1ª definición
```

```
diferenciaSimetrica :: Ord a => [a] -> [a] -> [a]
diferenciaSimetrica xs ys =
  sort (nub ([x | x <- xs, x 'notElem' ys] ++ [y | y <- ys, y 'notElem' xs]))

-- 2ª definición
diferenciaSimetrica2 :: Ord a => [a] -> [a] -> [a]
diferenciaSimetrica2 xs ys =
  sort (nub (union xs ys \\ intersect xs ys))

-- 3ª definición
diferenciaSimetrica3 :: Ord a => [a] -> [a] -> [a]
diferenciaSimetrica3 xs ys =
  [x | x <- sort (nub (xs ++ ys))
    , x 'notElem' xs || x 'notElem' ys]
```

## Ejercicio 5

# Números libres de cuadrados

### Ejercicio propuesto el 15-11-18

Un número entero positivo es libre de cuadrados si no es divisible por el cuadrado de ningún entero mayor que 1. Por ejemplo, 70 es libre de cuadrado porque sólo es divisible por 1, 2, 5, 7 y 70; en cambio, 40 no es libre de cuadrados porque es divisible por  $2^2$ .

Definir la función

---

```
libreDeCuadrados :: Integer -> Bool
```

---

tal que (libreDeCuadrados x) se verifica si x es libre de cuadrados. Por ejemplo,

---

libreDeCuadrados 70	==	True
libreDeCuadrados 40	==	False
libreDeCuadrados 510510	==	True
libreDeCuadrados (((10 <sup>10</sup> ) <sup>10</sup> ) <sup>10</sup> )	==	False

---

## Soluciones

```
import Data.List (nub)
```

```
-- 1ª definición
```

```

-- =====

libreDeCuadrados :: Integer -> Bool
libreDeCuadrados x = x == product (divisoresPrimos x)

-- (divisoresPrimos x) es la lista de los divisores primos de x. Por
-- ejemplo,
--   divisoresPrimos 40 == [2,5]
--   divisoresPrimos 70 == [2,5,7]
divisoresPrimos :: Integer -> [Integer]
divisoresPrimos x = [n | n <- divisores x, primo n]

-- (divisores n) es la lista de los divisores del número n. Por ejemplo,
--   divisores 30 == [1,2,3,5,6,10,15,30]
divisores :: Integer -> [Integer]
divisores n = [x | x <- [1..n], n `mod` x == 0]

-- (primo n) se verifica si n es primo. Por ejemplo,
--   primo 30 == False
--   primo 31 == True
primo :: Integer -> Bool
primo n = divisores n == [1, n]

-- 2ª definición
-- =====

libreDeCuadrados2 :: Integer -> Bool
libreDeCuadrados2 n =
  null [x | x <- [2..n], rem n (x^2) == 0]

-- 3ª definición
-- =====

libreDeCuadrados3 :: Integer -> Bool
libreDeCuadrados3 n =
  null [x | x <- [2..floor (sqrt (fromIntegral n))],
    rem n (x^2) == 0]

-- 4ª definición
-- =====

```

```
libreDeCuadrados4 :: Integer -> Bool
libreDeCuadrados4 x =
    factorizacion x == nub (factorizacion x)

-- (factorizacion n) es la lista de factores primos de n. Por ejemplo,
--   factorizacion 180 == [2,2,3,3,5]
factorizacion :: Integer -> [Integer]
factorizacion n | n == 1    = []
                | otherwise = x : factorizacion (div n x)
    where x = menorFactor n

-- (menorFactor n) es el menor divisor de n. Por ejemplo,
--   menorFactor 15 == 3
menorFactor :: Integer -> Integer
menorFactor n = head [x | x <- [2..], rem n x == 0]

-- Comparación de eficiencia
-- =====

--   λ> libreDeCuadrados 510510
--   True
--   (0.76 secs, 89,522,360 bytes)
--   λ> libreDeCuadrados2 510510
--   True
--   (1.78 secs, 371,826,320 bytes)
--   λ> libreDeCuadrados3 510510
--   True
--   (0.01 secs, 0 bytes)
--   λ> libreDeCuadrados4 510510
--   True
--   (0.00 secs, 153,216 bytes)
```



## Ejercicio 6

# Capicúas productos de dos números de dos dígitos

### Ejercicio propuesto el 16-11-18

El número 9009 es capicúa y es producto de dos números de dos dígitos, pues  $9009 = 91 \times 99$ .

Definir la lista

---

```
capicuasP2N2D :: [Int]
```

---

cuyos elementos son los números capicúas que son producto de 2 números de dos dígitos. Por ejemplo,

---

```
take 5  capicuasP2N2D  ==  [121,242,252,272,323]
length  capicuasP2N2D  ==  74
drop 70  capicuasP2N2D  ==  [8008,8118,8448,9009]
```

---

## Soluciones

```
import Data.List (nub, sort)
```

```
capicuasP2N2D :: [Int]
```

```
capicuasP2N2D = [x | x <- productos, esCapicua x]
```

```
-- productos es la lista de números que son productos de 2 números de
-- dos dígitos.
productos :: [Int]
productos = sort (nub [x*y | x <- [10..99], y <- [x..99]])

-- (esCapicua x) se verifica si x es capicúa.
esCapicua :: Int -> Bool
esCapicua x = xs == reverse xs
  where xs = show x
```