Prácticas Haskell

Contents

Chapter 1

Labo1

```
module Labo1 (
    time, yearsSeg, daysSeg, hoursSeg, minsSeg, years, days, hours,
    mins, segs, sol1a, sol1b, sol2A, sol2B, sol3, calculoDigitos,
    sumaDigitos, reduccion, bools
  ) where
```

1.1 EJERCICIO 1

1.1.1 A)

`time :: Integer`

tiempo base del ejercicio 1

`yearsSeg :: Integer`

Segundos totales en un año

`daysSeg :: Integer`

Segundos totales en un día

`hoursSeg :: Integer`

Segundos totales en una hora

`minsSeg :: Integer`

Segundos totales en un minuto

`years :: Integer`

Caclulo de los años equivalentes a `time`

`days :: Integer`

Calculo de las horas equivalentes a `time` quitando `years`

`hours :: Integer`

Calculo de las horas equivalentes a `time` quitando `days` y `years`

`mins :: Integer`

Calculo de las horas equivalentes a `time` quitando `days`, `years` y `hours`

`segs :: Integer`

Calculo de las horas equivalentes a `time` quitando `days`, `years`, `hours` y `mins`

`sol1a :: (Integer, Integer, Integer, Integer, Integer)`

Tupla con el tiempo total de `time` en años, dias, horas, minutos y segundos

1.1.2 B)

```
sol1b :: Integer -> (Integer, Integer, Integer, Integer, Integer)
```

Descripción

Coje el tiempo dado y lo convierte a (Años, Días, Horas, Minutos, Segundos)

Ejemplos

```
>>> sol1b 1000000
(0,11,13,46,40)
```

1.2 EJERCICIO 2

1.2.1 A)

```
sol2A ::  Integer -> Bool
```

Descripción

Coje el año dado y devuelve True si es bisiesto, False en caso contrario

Ejemplos

```
>>> sol2A 2008
True
```

```
>>> sol2A 2022
False
```

1.2.2 B)

```
sol2B ::  Integer -> Bool
```

Descripción

Coje el año dado y devuelve True si es bisiesto, False en caso contrario

Ejemplos

```
>>> sol2A 2008
True


>>> sol2A 2022
False
```

1.3 EJERCICIO 3

```
sol3 ::  (Fractional a1, Integral a2, Foldable t) => t a2 -> a1
```

Descripción

Error de tipado ya que la division / no trabaja con Integer

1.4 EJERCICIO 4

```
calculoDigitos ::  Integer -> Integer
```

Descripción

Cálculo del número de dijitos de un numero dado

Ejemplos

```
>>> calculoDigitos 0
1


>>> calculoDigitos 1984
4
```

```
sumaDigitos ::  Integral t => t -> t
```

Descripción

Suma todos los digitos de un numero dado

Ejemplos

```
>>> sumaDigitos 0
0


>>> sumaDigitos 1984
22
```

```
reduccion ::  Integral t => t -> t
```

Descripción

Suma todos los digitos de un numero dado hasta que el total es menor que 10

Ejemplos

```
>>> reduccion 0
0


>>> reduccion 1984
4
```

1.5 EJERCICIO 5

```
bools ::  Bool -> Bool -> Bool
```

Disyunción booleana definida por ajuste de patrones

Chapter 2

Labo2

```
module Labo2 (
    cuadrados, cuadradoInverso, rsumcos, sumMenores,
    siguientePrimo, factores, isPrime, iguales, menor, mayorA, ex,
    filter2, filters, mapx
  ) where
```

2.1 EJERCICIO 1

2.1.1 A)

```
cuadrados ::  (Num b, Enum b) => b -> [b]
```

Descripción

Coje un números y devuelve una lista con los cuadrados desde 0 hasta el número

Ejemplos

```
>>> cuadrados 3
[0,1,4,9]
```

2.1.2 B)

```
cuadradoInverso ::  (Num a, Enum a) => a -> [(a, a)]
```

Descripción

Coje un números y devuelve una lista con los cuadrados desde 0 hasta el número en orden inverso y emparejado con su número inicial

Ejemplos

```
>>> cuadradoInverso 3
[(3,9),(2,4),(1,1),(0,0)]
```

2.1.3 C)

```
rsumcos ::  (Floating a, Enum a) => a -> a
```

Descripción

Sumatorio del coseno de 1 hasta el número dado

Ejemplos

```
>>> rsumcos 90
2592.852064773843
```

2.1.4 D)

```
sumMenores ::  Integral a => a -> a
```

Descripción

Sumatorio de los números menores que el dado que sean multiplos de 5 o 3

Ejemplos

```
>>> sumMenores 10
33
```

2.1.5 E)

```
siguientePrimo ::  Integral a => a -> a
```

Descripción

calcula el siguiente número primo a uno dado.

Usa `isPrime`

Ejemplos

```
>>> siguientePrimo 10
11
```

```
factores ::  Integral a => a -> [a]
```

Descripción

Calcula todos los divisores de un número dado

Ejemplos

```
>>> factores 10
[1,2,5,10]
```

```
isPrime ::  Integral a => a -> Bool
```

Descripción

Comprueba si un número es primo

Usa `factores`

Ejemplos

```
>>> isPrime 10
False


>>> isPrime 5
True
```

2.2 EJERCICIO 2

2.2.1 A)

```
iguales ::  Eq b => (a -> b) -> (a -> b) -> a -> a -> Bool
```

Descripción

Comprueba si dos funciones son iguales para un ranfo dado de valores

Ejemplos

```
>>> iguales siguientePrimo sumMenores  0 10
False


>>> iguales sumMenores sumMenores  0 10
True
```

2.2.2 B)

```
menor ::  (Enum a, Num a) => a -> (a -> Bool) -> a
```

Descripción

Devuelve el menor numero mayor que n que compla una función dada

Ejemplos

```
>>> menor 10 isPrime
11
```

2.2.3 C)

```
mayorA ::  Enum a => a -> a -> (a -> Bool) -> a
```

Descripción

Mayor número en un intervalo dado que cumple una función dada

Ejemplos

```
>>> mayorA 10 20 isPrime
19
```

2.2.4 D)

```
ex ::  Enum a => a -> a -> (a -> Bool) -> Bool
```

Descripción

Comprueba si existe algún número en el intervalo dado que verifique una función

Ejemplos

```
>>> ex 14 16 isPrime
False


>>> ex 10 15 isPrime
True
```

2.3 EJERCICIO 3

2.3.1 A)

```
filter2 ::  [a] -> (a -> Bool) -> (a -> Bool) -> ([a], [a])
```

Descripción

Devuelve dos listas con los elementos de una lista dada que cumplen dos funciones

Ejemplos

```
>>> filter2 [1..10] odd even
([1,3,5,7,9],[2,4,6,8,10])
```

2.3.2 B)

```
filters ::  [a] -> [a -> Bool] -> [[a]]
```

Descripción

Devuelve una lista con las listas de numeros que cumplen unas funciones dadas en formato de lista

Ejemplos

```
>>> filters [1..10] [odd, even, isPrime]
[[1,3,5,7,9],[2,4,6,8,10],[2,3,5,7]]
```

2.3.3 C)

```
mapx ::  t -> [t -> b] -> [b]
```

Descripción

Devuelve una lista de valores resultado de aplicar a un valor un listado de funciones. Todos los tipos de las funciones de la lista deben ser iguales

Ejemplos

```
>>> mapx 10 [even, odd]
[True,False]
```

```
>>> mapx 10 [siguientePrimo, sumMenores]
[11,33]
```

Chapter 3

Labo3

```
module Labo3 (
    last', reverse', all', min', map', filter', takeWhile',
    listaNegPos, listaParejas', listaParejas, sufijos, sublistas,
    permuta, intercalado, sumandos, sumandos'
  ) where
```

3.1 EJERCICIO 1

3.1.1 A)

```
last' ::  [a] -> a
```

Descripción

Coge el último elemento de una lista dada

Ejemplos

```
>>> last' []
Lista vacia
```

```
>>> last [1,2,3,4]
4
```

3.1.2 B)

```
reverse' ::  [a] -> [a]
```

Descripción

Hace la lista inversa de una dada

Ejemplos

```
>>> reverse' [1,2,3,4]
[4,3,2,1]
```

3.1.3 C)

```
all' ::  (a -> Bool) -> [a] -> Bool
```

Descripción

Devuelve true si todos los elementos de la lista cumplen una condición

Ejemplos

```
>>> all' (<10) [1,2,3,4]
True
```

```
>>> all' (>10) [1,20,30,40]
False
```

3.1.4 D)

```
min' ::  Ord a => [a] -> a
```

Descripción

Devuelve el menor elemento de una lista dada

Ejemplos

```
>>> min' []
Lista vacia


>>> min' [2,5,8,2,0]
0
```

3.1.5 E)

```
map' ::  (a -> b) -> [a] -> [b]
```

Descripción

Genera una lista de elementos resultantes de aplicar un filtro a otra lista de elementos

Ejemplo

```
>>> map' (*3) [1,2,3,4]
[3,6,9,12]
```

3.1.6 F)

```
filter' ::  (a -> Bool) -> [a] -> [a]
```

Descripción

Genera una lista de elementos con los elementos de otra lista que hayan pasado un filtro

Ejemplo

```
>>> filter' (>10) [1,20,30,40]
[20,30,40]
```

3.1.7 G)

```
takeWhile' ::  (a -> Bool) -> [a] -> [a]
```

Descripción

Coge elementos de la lista dada hasta que la condición falla

Ejemplos

```
>>> takeWhile' (<3) [1,2,3,4,5]
[1,2]
```

```
>>> takeWhile' (>3) [1,2,3,4,5]
[]
```

3.2 EJERCICIO 2

```
listaNegPos ::  [Integer]
```

Descripción

Genera una lista alternada de elementos del 1 al 100 de forma [1, -1]

3.3 EJERCICIO 3

```
listaParejas' ::  (Num b, Enum b, Eq b) => b -> [(b, b)]
```

Descripción

Versión acotada de `listaParejas`

Ejemplo

```
>>> listaParejas' 4
[(0,0),(0,1),(1,0),(0,2),(1,1),(2,0),(0,3),(1,2),(2,1),(3,0),(0,4),(1,3),(2,2),(3,1),(4,0)]
```

```
listaParejas ::  Integral a => [(a, a)]
```

Descripción

Genera parejas ordenadas que cumplen x + y = z, en orden de z

Ver `listaParejas'` para ejemplos

3.4 EJERCICIO 4

3.4.1 A)

```
sufijos ::  [a] -> [[a]]
```

Descripción

Genera todos los sufijos de una lista

Ejemplo

```
>>> sufijos [1,2,3,4]
[[1,2,3,4],[2,3,4],[3,4],[4],[]]
```

3.4.2 B)

```
sublistas ::  [a] -> [[a]]
```

Descripción

Genera todas las sublistas de una lista dada

Ejemplo

```
>>> sublistas [1,2,3]
[[1],[2],[3],[1,2],[2,3],[1,2,3]]
```

3.4.3 C)

```
permuta ::  [a] -> [[a]]
```

Descripción

Genera todas las permutaciones posibles de una lista dada

Ejemplo

```
>>> permuta [1,2,3]
[[1,2,3],[2,1,3],[2,3,1],[1,3,2],[3,1,2],[3,2,1]]
```

```
intercalado ::  a -> [a] -> [[a]]
```

Descripción

Devuelve una lista de listas con el elemento dado en cada posición posible de la lista dada

Ejemplo

```
>>> intercalado 3 [1,2]
[[3,1,2],[1,3,2],[1,2,3]]
```

3.4.4 D)

```
sumandos ::  Integral a => a -> [[a]]
```

Descripción

Genera un listado de de listas que al sumar todos sus elementos dan como resultado el numero dado

Ejemplos

```
>>> sumandos 3
[[1,1,1],[1,2],[3]]
```

```
sumandos' ::  Integral a => a -> a -> a -> [[a]]
```

Descripción

Genera listados de tamaños desde x hasta n que al ser sumados dan como resultado x, siendo i el numero de listados generados

Ejemplo

```
>>> sumandos' 4 1 1
[[1,1,1],[1,2],[3]]
```

```
>>> sumandos' 4 2 1
[[1,1],[2]]
```

```
>>> sumandos' 4 2 2
[[2]]
```

Chapter 4

Labo4

```
module Labo4 (
    Punto(P), pointSum,
    Direccion(DERECHA, IZQUIERDA, ABAJO, ARRIBA), mueve, destino,
    trayectoria, Nat(Suc, Cero), (~+), (~*), natToInt, Complejo(C),
    Medible(medida)
  ) where
```

4.1 EJERCICIO 1

`data Punto`

Representación de un punto de forma (x,y)

Constructors

=	`P`	Constructor del punto
	`Int`	Coordenada X
	`Int`	Coordenada Y

`instance Show Punto`

Muestra el punto como "(a,b)""

```
pointSum ::  (Ord a, Num a) => a -> a -> a
```

Descripción

Suma un numero a una parte de una coordenada dada. Tiene como rango [0 .. 100]

Ejemplos

```
>>> pointSum 100 10
9


>>> pointSum 0 (-1)
100


>>> pointSum 100 1
0
```

`data Direccion`

Representación de una dirección

Constructors

```
=  ARRIBA
|  ABAJO
|  IZQUIERDA
|  DERECHA
```

`instance Eq Direccion`

Derivación estandar de Eq

`instance Ord Direccion`

Derivación estandar de Ord

`instance Show Direccion`

Derivación estandar de Show

4.1.1 A)

```
mueve ::  Punto -> Direccion -> Punto
```

Descripción

Mueve un punto en la dirección dada

Ejemplos

```
>>> mueve (P 0 1) IZQUIERDA
(100,1)

>>> mueve (P 0 1) DERECHA
(1,1)

>>> mueve (P 0 1) ARRIBA
(0,2)

>>> mueve (P 0 1) ABAJO
(0,0)
```

4.1.2 B)

```
destino ::  Punto -> [Direccion] -> Punto
```

Descripción

Mueve un punto en las direcciones dadas

Ejemplos

```
>>> destino (P 0 0) [ARRIBA, ARRIBA, ABAJO, ABAJO, IZQUIERDA, DERECHA, IZQUIERDA, DERECHA]
(0,0)

>>> destino (P 0 0) [ARRIBA, ARRIBA, ABAJO, IZQUIERDA, IZQUIERDA, DERECHA]
(100,1)
```

4.1.3 C)

```
trayectoria ::  Punto -> [Direccion] -> [Punto]
```

Descripción

Mueve un punto en las direcciones dadas mostrando los puntos por los que se han pasado

Ejemplos

```
>>> trayectoria (P 0 0) [ARRIBA, ARRIBA, ABAJO, ABAJO, IZQUIERDA, DERECHA, IZQUIERDA,
[(0,0),(0,1),(0,2),(0,1),(0,0),(100,0),(0,0),(100,0),(0,0)]
```

```
>>> trayectoria (P 0 0) [ARRIBA, ARRIBA, ABAJO, IZQUIERDA, IZQUIERDA, DERECHA]
[(0,0),(0,1),(0,2),(0,1),(100,1),(99,1),(100,1)]
```

4.2 EJERCICIO 2

`data Nat`

Representación de numeros naturales según la aritmética de Peano

Constructors

=	`Cero`	Representación del 0
|	`Suc Nat`	Representación de cualquier sucesor de 0

`instance Eq Nat`

Derivación estandar de Eq

`instance Ord Nat`

Derivación estandar de Ord

`instance Show Nat`

Muestra el punto como un número normal

4.2.1 A)

```
(~+) ::  Nat -> Nat -> Nat
```

Descripción

Suma de dos números siguiendo la aritmética de Peano

Ejemplo

```
>>> (Suc (Cero)) ~+ (Suc (Suc (Cero)))
3
```

```
(~*) ::  Nat -> Nat -> Nat
```

Descripción

Multiplicación de dos números siguiendo la aritmética de Peano

Ejemplo

```
>>> (Suc (Cero)) ~* (Suc (Suc (Cero)))
2
```

4.2.2 B)

```
natToInt ::  Nat -> Int
```

Descripción

Convierte un número representado en aritmética de Peano en un `Int` normal

Ejemplo

```
>>> natToInt (Suc (Suc (Cero)))
2
```

4.2.3 C)

4.3 EJERCICIO 3

```
data Complejo
```

Representación de un numero complejo

Constructors

`= C Float Float`

`instance Eq Complejo`

Derivación estandar de Eq

`instance Fractional Complejo`

Implementación especial para `(/)`

`instance Num Complejo`

Implementación especial para `(+)` `(-)` y `(*)`

`instance Show Complejo`

Representa el número como "a (+-) bi"

4.4 EJERCICIO 4

`class Medible a where`

Clase para medir otros tipos de datos

Methods

`medida`

`:: a`
`-> Int` Coge el dato dado y saca una medida de él

`instance Medible Bool`

Mediremos True como 0 y false como 1

`instance Medible a => Medible [a]`

Suma todas las medidas de la lista

`instance (Medible a, Medible b) => Medible (a, b)`

Suma la medida de los dos elementos dados

Chapter 5

Labo5

```
module Labo5 (
    getInt, adivina, Matriz, getFloat, getMatriz, getFilas,
    getDatosMatriz, dibujaMatriz, mostrarMatrizNueva, formatea,
    formatear, justify, addSpaces
  ) where
```

5.1 EJERCICIO 1

```
getInt ::  IO Int
```

Descripción

Coge un número desde la entrada.

```
adivina ::  Int -> IO ()
```

Descripión

Pide un número por teclado a adivinar. Esto es bastante estúpido de por si solo porque tienes que introducir el número como argumento

5.2 EJERCICIO 2

```
type Matriz = [[Float]]
```

Representación de una matriz como lista de listas

```
getFloat ::  IO Float
```

Descripción

Coge un número desde la entrada.

5.2.1 A)

```
getMatriz ::  IO Matriz
```

Descripción

Coge una matriz desde la entrada.

Ver `getDatosMatriz` para ver el funcionamiento total

```
getFilas ::  Int -> IO [Float]
```

Descripción

Coge una fila de una matriz según el número de columnas dadas.

```
getDatosMatriz ::  Int -> Int -> IO Matriz
```

Descripción

Coge todas las filas de una matriz según el número de filas y columnas dadas.

Ver `getFilas` para el funcionamiento completo

5.2.2 B)

```
dibujaMatriz ::  Matriz -> IO ()
```

Descripción

Dibuja una matriz dada como argumento.

```
mostrarMatrizNueva ::  IO ()
```

Descripción

Pide una matriz por teclado y la muestra una vez construida.

Ver `getMatriz` y `dibujaMatriz`.

5.3 EJERCICIO 3

```
formatea ::  String -> String -> Int -> IO ()
```

Descripción

Formatea a n columnas cada linea de un fichero dado en otro.

Ver `formatear` para el funcionamiento.

```
formatear ::  Int -> String -> String
```

Descripción

Formatea cada linea a un total de n caracteres por fila.
Ver `justify` para el funcionamiento completo.

```
justify ::  Int -> String -> String
```

Descripción

Añade espacios a una línea si es necesario.
Ver `addSpaces` para el funcionamiento completo.

```
addSpaces ::  Int -> String -> String
```

Descripción

Añade n espacios a una lina dada.