

Pre-Práctica de Programación Funcional

Para resolver esta práctica, recomendamos usar el intérprete “GHCi”, de distribución gratuita, que puede bajarse de <https://www.haskell.org/ghc/download>.

Ejercicio 1

Dar el tipo y describir el comportamiento de las siguientes funciones del módulo `Prelude` de Haskell:

`null head tail init last take drop (++) concat reverse elem`

Ejercicio 2

Definir las siguientes funciones:

- `valorAbsoluto :: Float → Float`, que dado un número devuelve su valor absoluto.
- `bisiesto :: Int → Bool`, que dado un número que representa un año, indica si el mismo es bisiesto.
- `factorial :: Int → Int`, definida únicamente para enteros positivos, que computa el factorial.
- `cantDivisoresPrimos :: Int → Int`, que dado un entero positivo devuelve la cantidad de divisores primos.

Ejercicio 3

Contamos con los tipos `Maybe` y `Either` definidos como sigue:

```
data Maybe a = Nothing | Just a
data Either a b = Left a | Right b
```

- Definir la función `inverso :: Float → Maybe Float` que dado un número devuelve su inverso multiplicativo si está definido, o `Nothing` en caso contrario.
- Definir la función `aEntero :: Either Int Bool → Int` que convierte a entero una expresión que puede ser booleana o entera. En el caso de los booleanos, el entero que corresponde es 0 para `False` y 1 para `True`.

Ejercicio 4

Definir las siguientes funciones sobre listas:

- `limpiar :: String → String → String`, que elimina todas las apariciones de cualquier carácter de la primera cadena en la segunda. Por ejemplo, `limpiar "susto" "puerta"` evalúa a `"pera"`. Nota: `String` es un renombre de `[Char]`. La notación `"hola"` es equivalente a `['h','o','l','a']` y a `'h':'o':'l':'a':[]`.
- `difPromedio :: [Float] → [Float]` que dada una lista de números devuelve la diferencia de cada uno con el promedio general. Por ejemplo, `difPromedio [2, 3, 4]` evalúa a `[-1, 0, 1]`.
- `todosIguales :: [Int] → Bool` que indica si una lista de enteros tiene todos sus elementos iguales.

Ejercicio 5

Dado el siguiente modelo para árboles binarios:

```
data AB a = Nil | Bin (AB a) a (AB a)
```

definir las siguientes funciones:

- `vacioAB :: AB a → Bool` que indica si un árbol es vacío (i.e. no tiene nodos).
- `negacionAB :: AB Bool → AB Bool` que dado un árbol de booleanos construye otro formado por la negación de cada uno de los nodos.
- `productoAB :: AB Int → Int` que calcula el producto de todos los nodos del árbol.