Funciones de Haskell y sus tipados

//OJO// Algunas de estas funciones no se encuentran definidas en el preludio, sino en los módulos List o Maybe. Para usarlas, incluir la(s) línea(s) import Data.List y/o import Data.Maybe respectivamente en el archivo fuente.

Funciones de plegado (folds)

- `foldr :: (a -> b -> b) -> b -> [a] -> b`
  Pliega una lista de derecha a izquierda, aplicando una función binaria a cada elemento y el resultado acumulado.
- `foldl :: (a -> b -> b) -> b -> [a] -> b`
  Pliega una lista de izquierda a derecha, aplicando una función binaria a cada elemento y el resultado acumulado.
- `foldr1 :: (a -> a -> a) -> [a] -> a`
  Similar a `foldr`, pero requiere una lista no vacía.
- `foldl1 :: (a -> a -> a) -> [a] -> a`

  Similar a `foldl`, pero requiere una lista no vacía.

Funciones de mapeo (map)

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

  Aplica una función a cada elemento de una lista y devuelve una nueva lista con los resultados.

Funciones de combinación (zipWith)

- `zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]`

  Combina dos listas elemento a elemento, aplicando una función binaria a cada par de elementos y devolviendo una nueva lista con los resultados.

Funciones de predicados (all, any)

- `all :: (a -> Bool) -> [a] -> Bool`
  Verifica si todos los elementos de una lista satisfacen un predicado.
- `any :: (a -> Bool) -> [a] -> Bool`

  Verifica si al menos un elemento de una lista satisface un predicado.

Funciones sobre listas (null, nub, sort, sortBy, ...)

- `null :: [a] -> Bool`
  Verifica si una lista está vacía.
- `nub :: Eq a => [a] -> [a]`
  Elimina elementos duplicados de una lista.
- `sort :: Ord a => [a] -> [a]`
  Ordena una lista de elementos comparables.

- `sortBy :: (a -> a -> Ordering) -> [a] -> [a]`
  Ordena una lista de elementos utilizando una función de comparación personalizada.

- `mod :: Integral a => a -> a -> a`
  Calcula el resto de la división entre dos números.
- `odd :: Integral a => a -> Bool`
  Verifica si un número es impar.
- `even :: Integral a => a -> Bool`
  Verifica si un número es par.
- `(++) :: [a] -> [a] -> [a]`
  Concatena dos listas.
- `head :: [a] -> a`
  Devuelve el primer elemento de una lista.
- `tail :: [a] -> [a]`
  Devuelve la lista sin el primer elemento.
- `init :: [a] -> [a]`
  Devuelve la lista sin el último elemento.
- `last :: [a] -> a`
  Devuelve el último elemento de una lista.
- `length :: [a] -> Int`
  Devuelve la longitud de una lista.
- `replicate :: Int -> a -> [a]`
  Crea una lista con un elemento repetido varias veces.
- `repeat :: a -> [a]`
  Crea una lista infinita con un elemento repetido.
- `iterate :: (a -> a) -> a -> [a]`
  Aplica una función a un valor inicial repetidamente y devuelve una lista infinita de los resultados.
- `filter :: (a -> Bool) -> [a] -> [a]`
  Filtra una lista basada en un predicado.
- `take :: Int -> [a] -> [a]`
  Toma los primeros n elementos de una lista.
- `drop :: Int -> [a] -> [a]`
  Descarta los primeros n elementos de una lista.
- `elem :: Eq a => a -> [a] -> Bool`
  Verifica si un elemento está en una lista.
- `find :: (a -> Bool) -> [a] -> Maybe a`
  Encuentra el primer elemento de una lista que satisfaga un predicado.
- `isNothing :: Maybe a -> Bool`
  Verifica si un valor `Maybe` es `Nothing`.
- `fromJust :: Maybe a -> a`
  Extrae el valor de un `Just` (puede fallar si el valor es `Nothing`).

- `maybe :: b -> (a -> b) -> Maybe a -> b`
  Aplica una función a un valor `Maybe`, devolviendo un valor predeterminado si es `Nothing`.
- `lookup :: Eq a => a -> [(a, b)] -> Maybe b`
  Busca un elemento en una lista de pares clave-valor.
- `reverse :: [a] -> [a]`
  Invierte el orden de los elementos de una lista.
- `concat :: [[a]] -> [a]`
  Concatena una lista de listas.
- `union :: Eq a => [a] -> [a] -> [a]`
  Calcula la unión de dos conjuntos.
- `intersect :: Eq a => [a] -> [a] -> [a]`
  Calcula la intersección de dos conjuntos.
- `(>>=) :: Monad m => m a -> (a -> m b) -> m b`
  Operador de unión de monadas (por ejemplo, para encadenar operaciones sobre valores `Maybe` o `Either`).
- `span :: (a -> Bool) -> [a] -> ([a], [a])`
  Divide una lista en dos listas: la primera mientras una condición se cumple y la segunda con los elementos restantes.
- `takeWhile :: (a -> Bool) -> [a] -> [a]`
  Toma elementos de una lista mientras una condición se cumple.
- `dropWhile :: (a -> Bool) -> [a] -> [a]`
  Descarta elementos de una lista mientras una condición se cumple.
- `concatMap :: (a -> [b]) -> [a] -> [b]`

  Aplica una función a cada elemento de una lista y concatena los resultados.

Funciones aritméticas y lógicas

- `and :: [Bool] -> Bool`
  Operador lógico AND.
- `or :: [Bool] -> Bool`
  Operador lógico OR.
- `sum :: Num a => [a] -> a`
  Calcula la suma de los elementos de una lista numérica.
- `max :: Ord a => a -> a -> a`
  Devuelve el máximo de dos valores comparables.
- `maximum :: Ord a => [a] -> a`
  Devuelve el máximo elemento de una lista.
- `maximumBy :: (a -> a -> Ordering) -> [a] -> a`
  Devuelve el máximo elemento de una lista según una función de comparación personalizada.
- `min :: Ord a => a -> a -> a`
  Devuelve el mínimo de dos valores comparables.

- `minimum :: Ord a => [a] -> a`
  Devuelve el mínimo elemento de una lista.
- `minimumBy :: (a -> a -> Ordering) -> [a] -> a`

  Devuelve el mínimo elemento de una lista según una función de comparación personalizada.

Operadores de comparación

- `(==) :: Eq a => a -> a -> Bool`
  Operador de igualdad.
- `(/=) :: Eq a => a -> a -> Bool`
  Operador de desigualdad.
- `compare :: Ord a => a -> a -> Ordering`
  Compara dos elementos y devuelve `LT`, `EQ` o `GT`.
- `comparing :: Ord a => (b -> a) -> b -> b -> Ordering`

  Crea una función de comparación basada en un campo de un tipo de datos.

Funciones sobre caracteres

- `not :: Bool -> Bool`
  Negación lógica.
- `ord :: Char -> Int`
  Devuelve el código ASCII de un carácter.
- `chr :: Int -> Char`

  Devuelve el carácter correspondiente a un código ASCII.