Ejercicio 2

Definir las siguiente funciones:

* **hd** :: [a] -> a retorna el primer elemento de una lista
* **tl** :: [a] -> [a] retorna toda la lista menos el primer elemento
* **last** :: [a] -> a retorna el último elemento de la lista
* **init** :: [a]-> [a] retorna toda la lista menos el último elemento

Respuesta

```haskell
hd :: [a] -> a
hd (x:xs) = x

tl :: [a] -> [a]
tl (x:xs) = xs

lst :: [a] -> a
lst [x] = x
lst (x:xs) = lst xs

ini :: [a] -> [a]
ini [x] = []    --(caso base)
ini (x:xs) = x : ini xs     --(caso recursivo)
```

Ejercicio 3

Defina una función máximo de tres, tal que maxTres x y z es el máximo valor entre x, y, z. Por ejemplo: maxTres 6 7 4 = 7.

Respuesta

```haskell
maxTree :: Int -> Int -> Int -> Int
maxTree a b c = (max a (max b c))
```

Ejercicio 4

Defina las siguientes operaciones sobre listas (vistas en el teórico): concatenar, tomar, tirar y agregar final.

Respuesta

```haskell
concateniar :: [a] -> [a] -> [a]
concateniar [] ys = ys
concateniar xs [] = xs
concateniar (x:xs) ys = x : concateniar xs ys

tomar :: Int ->[a] -> [a]
tomar n [] = []
tomar 0 (x:xs) = []
tomar n (x:xs) = x: tomar(n-1) XS
```

```haskell
tirar :: Int -> [a] -> [a]
tirar n [] = []
tirar 0 (x:xs) = x:xs
tirar n (x:xs) = tirar(n-1) XS

agregarFinal :: a -> [a] -> [a]
agregarFinal a [] = [a]
agregarFinal a xs =  xs ++ [a]
```

Ejercicio 5

Defina una función abs: Int -> Int que calcula el valor absoluto de un número.

Respuesta

```haskell
valAbs :: Int -> Int
valAbs n
    | n < 0 = -n
    | otherwise = n
```

Ejercicio 6

Define una función **edad** :: (Nat, Nat, Nat) -> (Nat, Nat, Nat) -> Int que dada dos fechas indica los años transcurridos entre ellas. Por ejemplo:

edad (20, 10, 1968) (30,4,1987) = 18

Respuesta

```haskell
--contempla los negativos
edad :: (Int,Int,Int) -> (Int,Int,Int) -> Int
edad (dd,mm,aaaa) (dd1,mm1,aaaa1) | (mm1, dd1)<(mm, dd) =
(absolute aaaa1 - aaaa) - 1
                                  | otherwise = absolute aaaa1 -
aaaa

--no contempla nada jiji
edadClase :: (Int,Int,Int) -> (Int,Int,Int) -> Int
edadClase (x,y,z) (p,q,r)
                          |(q<y) || (q==y && p<x) = r-z-1
                          |otherwise = r-z
```

Ejercicio 7

La disyunción excluyente *xor* de dos fórmulas se verifica si una es verdadera y la otra es falsa. Defina la función *xor* que calcule la disyunción excluyente a partir de la tabla de verdad.

Respuesta

```haskell
xor :: Bool -> Bool -> Bool
xor True True = False
xor False False = False
xor _ _ = True
```

*Ahora dina la función *xor2* que calcule la disyunción excluyente pero sin que considere todos los posibles valores de las entradas. Cuál será la diferencias entre ambas definiciones?

Respuesta

```haskell
xor1 :: Bool -> Bool -> Bool
xor1 x y = if (x==y) then False else True

--manera 2
xor2 :: Bool -> Bool -> Bool
xor2 True y = not y
xor2 False y = y
```

Ejercicio 8

Defina una función que dado un número natural, decida si el mismo es primo o no.

Respuesta

```haskell
nuPrim :: Int -> Bool
nuPrim num = length(dividir num) == 2

dividir :: Int -> [Int] -- funcion auxiliar
dividir n = [x | x <- [1..n], n `mod` x == 0]
```

Ejercicio 9

Defina una función que dado un número natural n, retorne la lista de todos los números naturales primos menores que n

Respuesta

```haskell
listNaturales :: Int -> [Int]
listNaturales 0 = []
listNaturales n = if nuPrim(n-1) then listNaturales(n-1) ++ [n]
else listNaturales(n-1)

--otra manera
listNat :: Int -> [Int]
```

```
listNat n = [x | x <- [1..n-1], nuPrim x]
```

Ejercicio 10

Defina una función que dada una lista, retorne la reversa de la misma.

Respuesta

```
reverList :: [a] -> [a]
reverList [] = []
reverList (x:xs) = reverList(xs) ++ [x]
```

Ejercicio 11

Defina una lista de números , devuelva la lista solo con los números primos.

Respuesta

```
listPrim :: [Int] -> [Int]
listPrim xs = [x| x <- xs, nuPrim x]
```

Ejercicio 12

Defina una función que dada una lista decida si es un palíndromo o no.

Respuesta

Palíndromo = “nolocasescolon”

```
palindromo :: (Eq a) => [a] -> Bool
palindromo xs = xs == reverse xs
```

Ejercicio 13

Defina una función que dados tres números **a,b,c** devuelva la cantidad de raíces reales de la ecuación *ax^2+bx+c*

Respuesta

La cantidad de raíces que tiene una ecuación cuadrática depende de los valores de los coeficientes a,b y c y para ello utilizamos el valor del determinante:

determinante = b^2 - 4ac

donde si:

determinante > 0	tiene 2 raíces
determinante = 0	tiene 1 raíz
determinante < 0	no tiene raíces

```haskell
cantRaices :: Int -> Int -> Int -> Int
cantRaices a b c
                | (b^2 - 4*a*c) > 0 = 2
                | (b^2 - 4*a*c) == 0 = 1
                |otherwise = 0
```