INSTITUTO TECNOLÓGICO DE OAXACA
Ingeniería en Sistemas Computacionales

Portafolio de evidencias

Asignatura:

Programación lógica y funcional

Docente:

Ramírez López Sergio Saúl

Alumno:

Chirino Cruz Ángel Omar

Fecha de entrega:

15 de Diciembre del 2024

Cuarta unidad

Determina si un número es divisible por 3, 5 o ambos y devuelve un par (número, "resultado")

```
funbiz x
   | mod x 3 == 0 && mod x 5 == 0 = (x, "BizzBuzz")
   | mod x 3 == 0 = (x, "Bizz")
   | mod x 5 == 0 = (x, "Buzz")
   | otherwise = (x, "")
```

Aplica la función funbiz a cada elemento de una lista

```
funn xs = [funbiz x | x <- xs]
```

Calcula el factorial de un número

```
factorial 0 = 1
factorial 1 = 1
factorial x = x * factorial (x - 1)
```

Genera una lista de pares (número, factorial) para cada elemento de una lista

```
factoria1 xs = [(x, factorial x) | x <- xs]
```

Calcula el número de Fibonacci de un número dado

```
fibonacci 0 = 0
fibonacci 1 = 1
fibonacci x = fibonacci (x - 1) + fibonacci (x - 2)
```

Genera una lista de números de Fibonacci hasta el índice dado

```
fibo x = [fibonacci y | y <- [0 .. x]]
```

Encuentra el máximo elemento de una lista

```
maximum' [] = error "Esta vacía la lista, intenta de nuevo"
maximum' [x] = x
maximum' (x:xs)
   | x > maxTail = x
   | otherwise   = maxTail
   where maxTail = maximum' xs
```

Encuentra el mínimo elemento de una lista

```
minimum' [] = error "Máximo de una lista vacía"
minimum' [x] = x
minimum' (x:xs)
   | x < minTail = x
   | otherwise   = minTail
   where minTail = minimum' xs
```

Genera una lista de números decrecientes hasta 0
```
repeat' 0 = [0, 1]
repeat' x = x : repeat' (x - 1)
```

Determina la suma de los elementos de una lista
```
sumlista [] = 0
sumlista [x] = x
sumlista (x:xs) = x + sumlista xs
```

Determina el producto de los elementos de una lista
```
productolista [] = 0
productolista [x] = x
productolista (x:xs) = x * productolista xs
```

Encuentra la longitud de una lista sin usar length
```
lengthlista [] = 0
lengthlista [x] = 1
lengthlista (x:xs) = 1 + lengthlista xs
```

Determina si un elemento está en una lista
```
buscar x [] = False
buscar x (y:xs)
   | x == y = True
   | otherwise = buscar x xs
```

Reversa una lista
```
reversa [] = []
reversa (x:xs) = reversa xs ++ [x]
```

Determina si una lista es un palíndromo
```
palindromo :: Eq a => [a] -> String
palindromo xs
   | xs == reversa xs = "Es palindromo"
   | otherwise = "No es Palindromo"
```

Duplica cada elemento de una lista
```
listados [] = []
listados (x:xs) = x : x : listados xs
```

Elimina todas las apariciones de un elemento dado en una lista
```
eliminarList _ [] = []
eliminarList y (x:xs)
   | y == x    = eliminarList y xs
```

```
    | otherwise = x : eliminarList y xs

Genera una lista de enteros desde 1 hasta n
listaEnteros 0 = []
listaEnteros n = listaEnteros (n - 1) ++ [n]

Genera una lista alternada entre dos números
listaDos 0 0 = []
listaDos a b = listaDos (a + 1) (b - 1) ++ [a] ++ [b]

Quita las vocales de un texto
quitarvocales [] = []
quitarvocales (x:xs)
    | x `elem` ['a', 'e', 'i', 'o', 'u'] = quitarvocales xs
    | otherwise = x : quitarvocales xs

Dada una lista de listas, suma los elementos de cada lista interna
sumarListas [] = []
sumarListas (xs:xss) = sum xs : sumarListas xss


Combina todas las listas internas en una sola lista
soloLista [] = []
soloLista (xs:xss) = xs ++ soloLista xss

Quita todas las listas vacías de una lista de listas
eliminarVacias [] = []
eliminarVacias (xs:xss)
    | null xs = eliminarVacias xss
    | otherwise = xs : eliminarVacias xss

Cuenta el número total de listas internas
contarlistas [] = 0
contarlistas (_:xss) = 1 + contarlistas xss

Cuenta el número total de elementos en todas las listas internas
contarlistas2 [] = 0
contarlistas2 (xs:xss) = length xs + contarlistas2 xss

Invierte cada lista interna en una lista de listas
invertirCadaLista [] = []
invertirCadaLista (x:xs) = invertir x : invertirCadaLista xs

Inserta un valor al inicio de cada lista interna y las concatena
concatenarConValor [] _ = []
```

```
concatenarConValor (x:xs) val = (val : x) ++ concatenarConValor xs val
```