INSTITUTO TECNOLÓGICO SUPERIOR DEL SUR DE GUANAJUATO

TAREA:

EJERCICIOS EN HASKELL

Materia:

Programación Lógica y Funcional.

Semestre:
6°A

Ing. Sistemas Computacionales

Elaborada por:

Paola Montserrat Ruiz Carmen, S22120146

Profesor:

Gustavo Ivan Vega Olvera.

Uriangato, Gto. a 28 de febrero del 2025

Ejercicio 1: Aplicar Descuento e IVA

```haskell
-- 1 Aplicar descuento e IVA
type Cesta = [(String, Double, Double)] -- (Producto, Precio, Porcentaje)

aplicarDescuento :: Double -> Double -> Double
aplicarDescuento precio descuento = precio * (1 - descuento / 100)

aplicarIVA :: Double -> Double -> Double
aplicarIVA precio iva = precio * (1 + iva / 100)

calcularPrecioFinal :: Cesta -> (Double -> Double -> Double) -> Double
calcularPrecioFinal cesta funcion = sum [funcion precio porcentaje | (_, precio, porcentaje) <- cesta]
```

Valores de entrada:

Cesta de compra:

- Producto1: Precio = 100, Descuento = 10%
- Producto2: Precio = 50, Descuento = 5%

Salida esperada:

- Precio final con descuentos: 137.5
- Precio final con IVA (21% aplicado a cada producto): 162.5

```
ghci> cestaEjemplo = [("Producto1", 100, 10), ("Producto2", 50, 5)]
ghci> calcularPrecioFinal cestaEjemplo aplicarDescuento
137.5
ghci> calcularPrecioFinal cestaEjemplo aplicarIVA
162.5
```

Ejercicio 2: Aplicar una función a una lista

```haskell
-- 2 Aplicar una función a una lista
aplicarALista :: (a -> b) -> [a] -> [b]
aplicarALista f xs = map f xs
```

Valores de entrada:

- Lista de números: [1, 2, 3, 4] Función: Multiplicación por 2

Salida esperada:

- Lista resultante: [2, 4, 6, 8]

```
ghci> aplicarALista (*2) [1,2,3,4]
[2,4,6,8]
ghci>
```

Ejercicio 3: Contar palabras en una frase

```haskell
-- 3 Contar palabras en una frase
contarPalabras :: String -> [(String, Int)]
contarPalabras frase = [(palabra, length palabra) | palabra <- words frase]
```

Valores de entrada:

- Frase: "Hola mundo Haskell"

Salida esperada:

- Lista de palabras con sus longitudes: [("Hola", 4), ("mundo", 5), ("Haskell", 7)]

```
ghci> contarPalabras "Hola mundo Haskell"
[("Hola",4),("mundo",5),("Haskell",7)]
ghci>
```

Ejercicio 4: Convertir notas a calificaciones

```haskell
-- 4 Convertir notas a calificaciones
calificacion :: Double -> String
calificacion nota
    | nota >= 95 = "Excelente"
    | nota >= 85 = "Notable"
    | nota >= 75 = "Bueno"
    | nota >= 70 = "Suficiente"
    | otherwise  = "Desempeño insuficiente"

convertirNotas :: [(String, Double)] -> [(String, String)]
convertirNotas notas = [(map toUpper materia, calificacion nota) | (materia, nota) <- notas]
```

Valores de entrada:

- Notas: [("Matematicas", 90), ("Historia", 72)]

Salida esperada:

- Asignaturas en mayúsculas con calificación: [("MATEMATICAS", "Notable"), ("HISTORIA", "Suficiente")]

```
ghci> convertirNotas [("Matematicas", 90), ("Historia", 72)]
[("MATEMATICAS","Notable"),("HISTORIA","Suficiente")]
ghci>
```

Ejercicio 5: Calcular módulo de un vector

```haskell
-- 5 Calcular módulo de un vector
moduloVector :: [Double] -> Double
moduloVector v = sqrt (sum (map (^2) v))
```

Valores de entrada:

- Vector: [3, 4]

Salida esperada:

- Módulo: 5.0

```
ghci> moduloVector [3,4]
5.0
ghci>
```

Ejercicio 6: Encontrar valores atípicos

```haskell
-- 6 Encontrar valores atípicos

-- Función para calcular la media de una lista de números
media :: [Double] -> Double
media xs = sum xs / fromIntegral (length xs)

-- Función para calcular la desviación típica de una lista de números
desviacionTipica :: [Double] -> Double
desviacionTipica xs = sqrt (sum (map (\x -> (x - m) ** 2) xs) / fromIntegral (length xs))
  where m = media xs

-- Función para encontrar valores atípicos basados en un rango
valoresAtipicos :: [Double] -> [Double]
valoresAtipicos xs
  | null xs = []  -- Si la lista está vacía, no hay valores atípicos
  | otherwise = filter (\x -> x < limiteInferior || x > limiteSuperior) xs
  where
    m = media xs
    d = desviacionTipica xs
    limiteInferior = m - 3 * d  -- Límite inferior del rango
    limiteSuperior = m + 3 * d  -- Límite superior del rango
```

Valores de entrada:

- Muestra de números: [1,2,3,4,5,100]

Salida esperada:

- Valores atípicos: [100.0]

```
ghci> valoresAtipicos[1,2,3,4,5,100]
[100.0]
```