Práctico de Repaso General: Fundamentos de

Sección I: Fundamentos y Patrones de Funciones

Ejercicios para practicar definiciones básicas, if, guardas, y pattern matching.

1. (Inicial) esMayorDeEdad
Define una función que tome una Int (edad) y devuelva True si es mayor o igual a 18, y False en caso contrario. Usa if-then-else.
2. (Inicial) signoNumero
Define una función que tome un número y devuelva:

- 1 si el número es positivo.
- -1 si el número es negativo.
- 0 si el número es cero.
  Usa guardas.

3. (Inicial) esFinDeSemana
Define una función que tome un String que representa un día de la semana ("lunes", "martes", etc.) y devuelva True si es "sabado" o "domingo", y False en cualquier otro caso. Usa pattern matching.
4. (Intermedio) tipoDeTriangulo
Define una función que tome tres longitudes de lados (Int, Int, Int) y devuelva un String:

- "Equilatero" si los tres lados son iguales.
- "Isosceles" si exactamente dos lados son iguales.
- "Escaleno" si los tres lados son diferentes.
  Usa guardas o if anidados.

5. (Intermedio) invertirTupla3
Define una función que tome una tupla de 3 elementos (a, b, c) y devuelva la tupla con los elementos invertidos (c, b, a). Determina su tipo más general.
6. (Intermedio) operacionCurrificada
Define una función aplicarSi que tome una función (Int -> Int), un predicado (Int -> Bool) y un número Int. Si el predicado es verdadero para el número, aplica la función al número. Si no, devuelve el número sin cambios.

- aplicarSi :: (Int -> Int) -> (Int -> Bool) -> Int -> Int
- Ejemplo: aplicarSi (\x -> x*2) even 4 debería devolver 8.
- Ejemplo: aplicarSi (\x -> x*2) even 5 debería devolver 5.

7. (Avanzado) reescribirConLet
La siguiente función calcula las raíces de una ecuación cuadrática. Reescríbela usando una cláusula let...in para hacerla más legible y eficiente, evitando cálculos repetidos.
raices :: Double -> Double -> Double -> (Double, Double)
raices a b c = ((-b + sqrt(b*b - 4*a*c)) / (2*a), (-b - sqrt(b*b - 4*a*c)) / (2*a))

Sección II: Recursión

Ejercicios para practicar la definición de funciones recursivas sobre listas y números.

1. (Inicial) sumarImpares
Define una función recursiva que tome una lista de Int y devuelva la suma de solo los números impares de la lista.
2. (Inicial) contarElemento
Define una función recursiva que tome un elemento a (que se pueda comparar con Eq) y una lista [a], y devuelva la cantidad de veces que ese elemento aparece en la lista.
3. (Intermedio) invertirLista
Define una función recursiva que invierta el orden de los elementos de una lista.

- (Pista: invertirLista [1, 2, 3] debe ser [3, 2, 1]).
- (Pista: Necesitarás el operador de concatenación ++ que definiste en prácticos anteriores, o puedes usar el de la librería estándar).

4. (Intermedio) eliminarPrimeraOcurrencia
Define una función recursiva que tome un elemento a y una lista [a], y devuelva una lista igual pero sin la primera aparición de ese elemento. Si el elemento no está, devuelve la lista original.
5. (Intermedio) soloIndicesPares
Define una función recursiva que devuelva una lista con los elementos que están en las posiciones pares (0, 2, 4, ...).

- Ejemplo: soloIndicesPares [10, 20, 30, 40, 50] debe devolver [10, 30, 50].

6. (Avanzado) aplanarLista
Define una función recursiva que tome una lista de listas [[a]] y la "aplane", convirtiéndola en una sola lista [a].

- Ejemplo: aplanarLista [[1, 2], [], [3, 4]] debe devolver [1, 2, 3, 4].

7. (Avanzado) esPalindromo
Define una función que determine si una lista es un palíndromo (se lee igual al derecho y al revés).

- (Pista: Puedes usar invertirLista de un ejercicio anterior).

Sección III: Listas por Comprensión

Ejercicios para definir funciones usando la sintaxis de listas por comprensión.

1. (Inicial) duplicarElementos
Define una función que tome una lista [Int] y devuelva una lista con cada elemento multiplicado por 2.

- Ejemplo: duplicarElementos [1, 5, 10] debe devolver [2, 10, 20].

2. (Inicial) obtenerPares
Define una función que tome una lista [Int] y devuelva una lista que contenga solo los números pares.
3. (Intermedio) productoCartesiano
Define una función que tome dos listas, [a] y [b], y devuelva la lista de todos los pares posibles (producto cartesiano).

- Ejemplo: productoCartesiano [1, 2] ['a', 'b'] debe devolver [(1, 'a'), (1, 'b'), (2, 'a'), (2, 'b')].
- (Pista: Necesitarás dos generadores).

4. (Intermedio) filtrarTuplas
Define una función que tome una lista de tuplas [(Int, Int)] y devuelva una lista de solo aquellas tuplas donde el primer elemento es estrictamente menor que el segundo.
5. (Avanzado) numerosPrimos
Define una función que tome un número n y devuelva la lista de todos los números primos menores o iguales a n.

- (Pista: Puedes usar la función divisores que definimos en los ejercicios anteriores. Un número p es primo si divisores p == [1] (o [1, p] dependiendo de tu definición).

6. (Avanzado) matrizIdentidad
Define una función que tome un entero n y genere una lista de listas que represente una "matriz identidad" de n x n.

- Ejemplo: matrizIdentidad 3 debe devolver [[1, 0, 0], [0, 1, 0], [0, 0, 1]].
- (Pista: Necesitarás generadores dependientes y un if-then-else dentro de la expresión).

7. (Avanzado) sumaDeParesDistintos
Define una función que tome un número n y devuelva la lista de todos los pares (x, y) de números distintos entre 1 y n (inclusive).

- (Pista: x <- [1..n], y <- [1..n], y una guarda para asegurar que sean distintos).

Sección IV: Algoritmos de Ordenamiento

Ejercicios basados en los PDF ClasificacionInterna.pdf y EjCodigo_Sorting.pdf.

1. (Inicial) estaOrdenada
Define una función recursiva que devuelva True si una lista de Int está ordenada de menor a mayor, y False en caso contrario.

- (Pista: estaOrdenada [1, 2, 2, 5] es True. estaOrdenada [1, 3, 2] es False).

2. (Inicial) insertarEnOrden
Define la función insert :: Int -> [Int] -> [Int], que toma un número y una lista ya ordenada, e inserta el número en la posición correcta para que la lista siga ordenada. (Esta es la función clave de Insertion Sort).
3. (Intermedio) minimoYResto
Define una función que tome una lista no vacía y devuelva una tupla (Int, [Int]) donde el primer elemento es el mínimo de la lista, y el segundo es la lista sin ese mínimo. (Esta es la lógica de Selection Sort).

- Ejemplo: minimoYResto [3, 1, 4, 2] debe devolver (1, [3, 4, 2]).

4. (Intermedio) particionar
Define la función particionar :: Int -> [Int] -> ([Int], [Int]). Toma un "pivote" y una lista, y devuelve dos listas: la de los menores o iguales al pivote, y la de los mayores. (Esta es la función clave de QuickSort).

- Ejemplo: particionar 5 [8, 3, 1, 6, 5, 10] debe devolver ([3, 1, 5], [8, 6, 10]).
- (Pista: Puedes usar dos listas por comprensión).

5. (Avanzado) merge
Define la función merge :: [Int] -> [Int] -> [Int]. Toma dos listas ya ordenadas y las "mezcla" en una sola lista ordenada. (Esta es la función clave de MergeSort).

- Ejemplo: merge [1, 4, 8] [2, 5, 9] debe devolver [1, 2, 4, 5, 8, 9].

6. (Avanzado) quickSort
Usando tu función particionar (o listas por comprensión, como en el PDF), implementa la función quickSort completa.
7. (Avanzado) selectionSort
Usando tu función minimoYResto (o las minimo y delete del PDF), implementa la función selectionSort completa.

Sección V: Tipos de Datos (Avanzado)

Ejercicios basados en los últimos temas de Practica1.pdf.

1. (Inicial) data Figura
Define un tipo de dato Figura que pueda ser un Circulo (con un radio Float) o un Rectangulo (con una base Float y una altura Float).
2. (Inicial) funcion area
Usando pattern matching sobre tu tipo Figura, define una función area :: Figura -> Float que calcule el área de la figura.
3. (Intermedio) data ListaPropia
Define un tipo de dato polimórfico ListaPropia a que sea recursivo. Debe tener dos constructores:

- Vacia (para la lista vacía).
- Cons (que toma un valor a y una ListaPropia a).

4. (Intermedio) sumarListaPropia
Define una función recursiva sumarListaPropia :: ListaPropia Int -> Int que sume todos los elementos de tu lista personalizada.
5. (Avanzado) data ArbolBinario
Define un tipo de dato polimórfico ArbolBinario a para un árbol de búsqueda binario. Debe tener dos constructores:

- HojaVacia.
- Nodo (que toma un ArbolBinario a (izquierdo), un valor a, y un ArbolBinario a (derecho)).

6. (Avanzado) insertarEnArbol
Define una función insertarEnArbol :: (Ord a) => a -> ArbolBinario a -> ArbolBinario a que inserte un nuevo elemento en el árbol binario de búsqueda, manteniendo el orden.
7. (Avanzado) aplanarArbol
Define una función aplanarArbol :: ArbolBinario a -> [a] que recorra el árbol "en orden" (in-order traversal) y devuelva una lista ordenada con todos sus elementos.