Parcial 1 - Algoritmos I Taller: Tema B

Ejercicio 1:

a) Si tengo una función con la siguiente declaración

```
f :: (Eq a, Ord a) => a -> a -> a
f x y | x<y = x
      | otherwise = y
```

puedo decir que:

1) a sólo puede tomar valores que representan números (Int, Float, etc.)
2) x sólo puede tomar valores numéricos.
3) a es una variable de tipo.
4) x es una variable de tipo.
5) Ninguna de las anteriores.

b) Si tengo la función definida en el punto a) y el tipo enumerado:

```
data TipoEnum  = Val1 | Val 2 | Val3
```

puedo decir que:

1) `TipoEnum` no es un tipo enumerado.
2) f Val1 Val2 retorna Val1.
3) f Val1 Val2 retorna Val2.
4) No se puede instanciar a con el tipo `TipoEnum`.
5) Ninguna de las anteriores.

c) Si tengo la función definida en el punto a) y el tipo enumerado:

```
data TipoEnum  = Val1 | Val 2 | Val3 deriving (Eq,Ord,Show)
```

puedo decir que:

1) `TipoEnum` no es un tipo enumerado.
2) f Val1 Val2 retorna Val1.
3) f Val1 Val2 retorna Val2.
4) No se puede instanciar a con el tipo `TipoEnum`.
5) Ninguna de las anteriores.

d) Teniendo en cuenta la definición de la función del punto a) se puede decir de la invocación `f (5::Int)` que:

1) No se puede invocar f en 5 puesto que falta un parámetro.
2) `f 5 es` una función de tipo a -> a.
3) `f 5 es` una función de tipo Int -> a.
4) `f 5` es una función de tipo Int -> Int.
5) Ninguna de las anteriores.

Ejercicio 2:

Se va a representar el stock de Artículos en una Librería, usando tipos en Haskell. Los artículos que tenemos en cuenta son: `Libros, Agendas, Cuadernos`. La idea es poder detallar para cada tipo de material, las características más importantes. En tal sentido identificamos las siguientes características de cada uno de los materiales a tener en cuenta:

`Libro`

- `Categoria`, que es un tipo enumerado con las siguientes opciones: `Literatura`, `Infantiles`, `Autoayuda`, `Comics`.
- `Editorial`, que es un tipo enumerado con las siguientes opciones: `Altea`, `Minotauro`, `Panini`.
- `Titulo`, que es un sinónimo de `String` indicando el título del libro.
- `Precio`, que es un sinónimo de `Int` indicando el precio.

`Agenda`

- `Marca`, que es un tipo enumerado con las siguientes opciones: `Monoblock`, `Papikra`.
- AnioAgenda, que es un sinónimo de `Int` indicando el año de la agenda.
- `Precio`, que es un sinónimo de `Int` indicando el precio.

`Cuaderno`

- `Marca`, que es un tipo enumerado con las siguientes opciones: `Monoblock`, `Paprika` .
- `Precio`, que es un sinónimo de `Int` indicando el precio.

Para ello:

a) Definir el tipo `ArtículosLibrería` que consta de los constructores `Libro, Agenda y Cuaderno`, constructores con parámetros descritos arriba. (Se deben definir también los tipos enumerados `Categoria, Editorial , Marca, AnioAgenda)`. **Los tipos** `ArticulosLibreria, Editorial y Marca` **<u>no deben</u> estar en la clase** **`Eq`** , ni en la clase **`Ord`**.

b) Definir la función `librosBaratos` de la siguiente manera:

```
librosBaratos :: [ArticulosLibreria] -> Precio -> [ArtículosLibrería]
```

que dada una lista de `ArticulosLibreria` **`ls`** y un precio **`p`**, devuelve la lista de libros con precio menor o igual a **p**.
NOTA: Dejar como comentario dos ejemplos donde hayas probado la función `librosBaratos` con una lista con al menos 3 `ArticulosLibreria`.

c) Definir igualdad para el tipo de `ArticulosLibreria` : de tal manera que, dos artículos de tipo **`Libro`** son iguales sólo si tienen la misma **`Editorial`** y el mismo **`Precio`**, dos

`Agendas` son iguales sólo si tienen el mismo **AnioAgenda** y el mismo **Precio**, mientras que dos cuadernos son iguales si tienen el mismo **Precio**. Como es de suponer los `Libros, Agendas y Cuadernos` son distintos entre sí.
NOTA: Dejar como comentario en el código dos ejemplos en los que probaste la igualdad.

Ejercicio 3

Queremos hacer un programa, para que les profesores de una academia de Inglés puedan saber si sus alumnos de un nivel pueden pasar al siguiente nivel o no.

a) Definir un tipo recursivo `NotasDeIngles`, que permite guardar las notas que tuvo cada estudiante de un nivel en el período. El tipo `NotasDeIngles`, tendrá dos constructores:

1) `EvolucionDelEstudiante`, que tiene 6 parámetros:

- `String`, para el nombre y apellido del alumno
- `Nivel` (Tipo Enumerado con el Nivel actual que está cursando: Uno, Dos, Tres)
- `Int` (con la nota del primer parcial, entre 1 y 10)
- `Int` (con la nota del segundo parcial, entre 1 y 10)
- `Int` (con la nota del final 1 a 10,)
- `NotasDeIngles,` recursión con el resto de las notas.

2) `NoHayEstudiantes`, que es un constructor sin parámetros, similar al de la lista vacía, para indicar que se terminaron las notas.

La condición para poder obtener el siguiente nivel se describen a continuación según las notas obtenidas en las diferentes instancias:

- Si el estudiante está en Nivel Uno o Dos, debe sacar por lo menos 7 en un parcial, y 8 en otro y haber tenido en el final al menos un 7.
- Si el estudiante está en el Nivel Tres debe tener al menos un 8 de promedio entre los dos parciales, y al menos un 8 en el final.

b) Programar la función `pasaDeNivel`, que toma como primer parámetro `notas` del tipo **`NotasDeIngles,`** y como segundo parámetro el nombre de un estudiante (de la lista) de tipo **`String`** y retorna un valor de tipo **`Bool,`** indicando si el estudiante con ese nombre **pasa de nivel o no**.

```
pasaDeNivel :: NotasDeIngles -> String -> Bool
```

NOTA: Dejar como comentario un ejemplo donde hayas probado `pasaDeNivel` con un parámetro de tipo **`NotasDeIngles`** que tenga al menos 3 estudiantes.

c) Programar la función `devolverNivel` con la siguiente declaración:

```
devolverNivel :: NotasDeIngles -> String -> Maybe Nivel
```

que toma una variable **`notas`** de tipo `NotasDeIngles`, y como segundo argumento un `nombre`, que identifica al estudiante, y en caso que este esté en **`notas`** (en un nivel **`n`**), retorna **Just `n`** y **`Nothing`** en caso contrario.

NOTA: Dejar como comentario un ejemplo donde hayas probado la función