# Parcial 1 - Algoritmos I Taller: Tema A

## Ejercicio 1

En las siguientes preguntas marque la respuesta correcta.

a. Si tengo una función con la siguiente declaración de tipos de la función `f`
`f :: (Eq b) => a -> (a -> b) -> Bool` puedo decir que:
   1. Es una función polimórfica Paramétrica
   2. Es un función polimórfica Ad hoc
   3. Es una función recursiva
   4. es un constructor
   5. ninguna de las anteriores.

b. Si tengo una función con la siguiente declaración
`f :: a -> b -> (a -> b) -> b`, puedo decir que:
   1) Es una función polimórfica característica
   2) Es un función polimórfica Ad hoc
   3) Es una función recursiva
   4) es un constructor
   5) ninguna de las anteriores.

c. Si tengo la siguiente definición de un tipo y una función, puedo decir que:
```
data Radio = AM | FM | SW deriving Eq
esFM :: Radio -> Bool
esFM (Radio) = (Radio == FM)
```
   1) No puedo usar igualdad, debería usar pattern matching
   2) Es un error conceptual, se deben usar los constructores para definir la función, y no el nombre del tipo.
   3) La función no cubre todos los casos
   4) 1 y 2 son correctas
   5) Ninguna de las anteriores

d. Los constructores son :
   1) Un sinónimos de tipos
   2) Son funciones constantes que sirven para definir los elementos de un tipo cualquiera.
   3) Son funciones que pueden tomar parámetros, y sirven para representar cualquier elemento de un tipo abstracto de datos.
   4) Es un sinónimo de función polimórfica.
   5) ninguna de las anteriores.

# Ejercicio 2

Se va a representar el stock de una casa de electrodomésticos usando tipos en Haskell. Los dispositivos que tenemos en cuenta son: Televisores, Home Theater . La idea es poder detallar para cada tipo de electrodoméstico, las características más importantes. En tal sentido identificamos las siguientes características:

`Televisor`

- `PulgadasTele`, que es un tipo enumerado con las siguientes opciones: `Veinticuatro`, `TreintayDos`, `Cuarenta`, `Setenta`
- `TipoPantalla`, que es un tipo enumerado con las siguientes opciones: `LED`, `QLED` `OLED`
- `Precio`, que es un sinónimo de `Int` indicando el precio

`HomeTheater`

- `Potencia`, que es un sinónimo de `Int` indicando la potencia de salida
- `TipoDeSalida`, que es un tipo enumerado con las siguientes opciones: `Mono`, `Estereo`, `CincoPUno`
- `Precio`, que es un sinónimo de `Int` indicando el precio

Para ello:

**a)** Definir el tipo `Dispositivo` que consta de los constructores `Televisor` y `HomeTheater`, con los constructores con parámetros descritos arriba (Se deben definir también los tipos enumerados `PulgadasTele,TipoPantalla y TipoDeSalida)`.
**Los tipos `Dispositivo y PulgadasTele` no deben estar en la clase `Eq`**, ni en la clase **`Ord`**.

**b)** Definir la función `cuantosTelevisores` de la siguiente manera:

```
cuantosTelevisores :: [Dispositivo] -> PulgadasTele -> Int
```

que dada una lista de `Dispositivos ld` y un valor `p` de `PulgadasTele`, me devuelve un entero indicando la cantidad de televisores que hay en `ld` con las pulgadas `p`.
**NOTA:** Dejar como comentario un ejemplo donde hayas probado la función `cuantosTelevisores` con una lista con al menos 3 `Dispositivo`.

**c)** Definir igualdad para el tipo de `Dispositivo` : de tal manera que, dos dispositivos de tipo `Televisor` son iguales sólo si tienen el mismo **precio** y las mismas **pulgadas**, mientras que dos `HomeTheater` son iguales solo si tienen el mismo **precio**. Como es de suponer los televisores y HomeTheater son distintos.
**NOTA:** Dejar como comentario en el código dos ejemplos en los que probaste la igualdad.

**d)** Definir la función, hay dos dispositivos iguales de manera consecutiva en una lista de dispositivos. La función `hayDosIguales,` tiene la siguiente definición de tipos:

```
hayDosIguales :: [Dispositivo] -> Bool
```

Dada una lista de `Dispositivo ld,` debe devolver `True` en caso que en la lista `ld` existan dos dispositivos que sean iguales de manera consecutiva, y `False` en caso contrario.
**NOTA:** Dejar como comentario en el código dos ejemplos en los que probaste la función.

# Ejercicio 3

Queremos hacer un programa, para que el profe de una materia pueda computar la situación de regularidades y promociones al final del cursado.

a) Definir un tipo recursivo `NotasDelCuatri`, que permite guardar las notas que tuvo cada alumno en el año. El tipo `NotasDelCuatri`, tendrá dos constructores:

   1) `NotasDelAlumno`, que tiene 4 parámetros:

   - `String`, para el nombre y apellido del alumno
   - `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 recuperatorio 1 a 10,)
   - `NotasDelCuatri,` recursión con el resto de las notas.

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

- La condición de regularidad al finalizar el cuatrimestre puede ser, `Regular`, `Promocional` o `Libre`. A continuación se define la condición final en base a las notas obtenidas:
- Para ser considerado **`Regular`**, es necesario sacar 6 o más en cada parcial, o haber aprobado alguno de los dos parciales y haber aprobado el recuperatorio.
- Para ser considerado **`Promocional`**, es necesario haber aprobado ambos parciales y tener un promedio mayor o igual a 8 entre esas notas.
- Los alumnos son considerados con condición **`Libre`**, cuando reprobaron ambos parciales, o aprobaron solo uno y luego reprobaron el recuperatorio.

b) Programar la función `esRegularAlumno`, que toma como primer parámetro `notas` del tipo **`NotasDelCuatri,`** y como segundo parámetro el `nombre del alumno` de tipo **`String`** y retorna un valor de tipo **`Bool,`** indicando si el alumno con `nombre` es **regular** o no.

```
esRegularAlumno :: NotasDelCuatri -> String -> Bool
```

**NOTA:** Dejar como comentario un ejemplo donde hayas probado `esRegularAlumno` con un parámetro de tipo **`NotasDelCuatri`** que tenga al menos 3 alumnos.

c) Programar la función `devolverNotaP1` con la siguiente declaración:

```
devolverNotaP1 :: NotasDelCuatri -> String -> Maybe Int
```

que toma una variable notas de tipo `NotasDelCuatri`, y como segundo argumento un `nombre`, que identifica el alumno, y en caso que el alumno esté en `notas`, retorna la nota del primer parcial y Nothing en caso contrario.

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