# Tema A

## Ejercicio 1:

Se va a representar el stock de Productos de un Vivero, usando tipos en Haskell. Los productos que tenemos en cuenta son: `Planta, Semilla, Maceta`. La idea es poder detallar para cada tipo de artículo, las características más importantes. En tal sentido identificamos las siguientes características de cada uno de los artículos a tener en cuenta:

`Planta`

- `Categoria`, que es un tipo enumerado con las siguientes opciones: `Bulbosas, Rosales, Frutales, Trepadoras`
- `Floracion`, que es un tipo enumerado con las siguientes opciones: `Primavera, Verano, Invierno`.
- `Hoja`, que es un sinónimo de `String` indicando el tipo de hoja.
- `Precio`, que es un sinónimo de `Int` indicando el precio

`Semilla`

- `TipoSemilla`, que es un tipo enumerado con las siguientes opciones: `Flores , Hortalizas y Aromaticas`
- `Hoja`, que es un sinónimo de `String` indicando el tipo de hoja de la planta
- `Precio`, que es un sinónimo de `Int` indicando el precio

`Maceta`

- `Material`, que es un tipo enumerado con las siguientes opciones: `Plastico, Barro .`
- `Precio`, que es un sinónimo de `Int` indicando el precio

Para ello:

**a)** Definir el tipo `ProductosVivero` que consta de los constructores `Planta, Semilla` y `Maceta`, constructores con parámetros descritos arriba (Se deben definir también los tipos enumerados `Categoria, Floracion, TipoSemilla, Material` y los sinónimos de `Hoja` y de `Peso)`. **No se debe usar** `deriving Eq` **ni** `deriving Ord` **para ninguno de los tipos.** Se puede sin embargo incluir los tipos en la clase `Show`

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

```
cuantasPlantas :: [ProductosVivero] -> Categoria -> Int
```

que dada una lista de `ProductosVivero` **`ls`** y un valor **`c`** de `Categoria de Planta`, me devuelve un entero indicando la cantidad de plantas que hay en **`ls`** con la categoria **c**.
**NOTA:** Dejar como comentario dos ejemplo donde hayas probado la función `cuantasPlantas` con una lista con al menos 3 `ProductosVivero`.

**c)** Definir igualdad para el tipo de **`ProductosVivero`**: de tal manera que, dos productos de tipo **`Planta`** son iguales sólo si tienen la misma **`Categoria`** y la misma **`Hoja`**, dos productos del tipo **`Semilla`** son iguales solo si tienen el mismo **`TipoSemilla`**, la misma **`Hoja`** y el mismo **`precio`**, mientras que dos productos del tipo **`Maceta`** son iguales si tiene el mismo **`Material`** y el mismo **`precio`**. Como es de suponer las Plantas, Semillas y Macetas son distintas entre sí.
**NOTA:** Dejar como comentario en el código dos ejemplos en los que probaste la igualdad.

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

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

Dada una lista de **`ProductosVivero`** **ls**, debe devolver True en caso que en la lista **ls** existan dos productos 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 2

Queremos hacer en Haskell un programa que va a representar una lista de reproducción (parecido a las utilizadas en plataformas como Netflix, Disney, etc) para películas. Para ello

a) Definir un tipo recursivo `MovieList`, que permite guardar las películas que se van reproduciendo por cada perfil creado en la plataforma. El tipo tendrá dos constructores:
   1) `Movie`, que tiene cinco parámetros con los siguientes tipos:
      - `Titulo`, que es el nombre de la película y debe ser sinónimo de `String`
      - `Megusta` pensarlo de la siguiente forma, si me encanta tiene un 2, si me gusta tiene un 1 y si no es para mi, tiene un 0, por lo cual debe ser sinónimo de `Int`
      - `Estado`, que es un tipo enumerado con constructores `Reproducido` y `SinReproducir`
      - `Duracion`, que es la cantidad de minutos que dura la película y debe ser un sinónimo de `Int`
      - `MovieList`, recursión con el resto de las películas.
   2) `SinMovie`, que es un constructor sin parámetros, similar al de la lista vacía, para indicar que no hay películas.

   **NOTA:** Los tipos nuevos definidos no deben estar en la clase `Eq` ni `Ord`

b) Programar la función `minutosReproducidos`

```
minutosReproducidos :: MovieList -> Megusta -> Duracion
```

que dada una Lista de Películas **`mls`** y un valor para Megusta **`mg`** devuelve la suma de los minutos reproducidos de todas las películas de `mls` que han sido reproducidas y que tienen megusta mayor o igual a `mg`.

**NOTA:** Dejar como comentario un ejemplo donde se haya probado `minutosReproducidos` con un parámetro de tipo `MovieList` que tenga al menos 3 películas.

c) Programar la función

```
estadoDePeli :: Titulo -> MovieList -> Maybe Estado
```

que dado un título de película **`t`** y una lista de películas **`mls`** si la película está en **`mls`** con estado **`e`** debe devolver **`Just e`** y si la película no está debe devolver **`Nothing`**

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