

Ejercicios extra

Laboratorio de AYEDI

Ejercicio 1

Van a representar el sistema de matrículas vehiculares utilizado entre 1995 y 2016 en Argentina. Aunque ahora se utiliza el sistema del *Mercosur* todavía muchos vehículos tienen las patentes con el formato viejo. Un ejemplo de patente es:



Para representar las letras de la matrícula deben definir el tipo `Letras` como un sinónimo de tripla (tres-upla) donde cada componente es un `Char`. De esta manera un valor del tipo `Letras` guarda las tres letras de una patente. Adicionalmente deben definir el tipo `Numeracion` como un sinónimo de `Int`.

Por último definan el tipo `Matricula` que consta de un único constructor `Patente` que toma dos parámetros:

- El primer parámetro es del tipo `Letras`
- El segundo parámetro es del tipo `Numeracion`

Deben incluir al tipo `Matricula` en las clases necesarias para que esté definida la relación `<=`.

Puesto que el tipo `Letras` permite combinaciones inválidas para patentes (por ejemplo la terna `('A', '@', '#')` no es de una patente válida) van a definir la función:

```
letra_valida :: Char -> Bool
```

que dado un carácter `c` devuelve `True` si y sólo si `'A' <= c` y `c <= 'Z'` (*Haskell* nos permite hacer esto con valores del tipo `Char`). Luego definan usando *pattern matching*:

```
letras_validas :: Letras -> Bool
```

que dado una tripla `t` devuelve `True` si y sólo si cada una de las tres componentes cumplen simultáneamente con la condición que verifica la función `letra_valida`

Por último definan usando *pattern matching* la función

```
matricula_valida :: Matricula -> Bool
```

que dada una matrícula `m` verifica que sus letras sean válidas y que la numeración sea un número entre 0 y 999 (inclusive).

Ejercicio 2

Supongamos que buscamos un auto sospechoso del cual sólo tenemos su numeración. Van a definir, usando recursión y *pattern matching*, la función:

```
filtrar_patentes :: [Matricula] -> Numeracion -> [Matricula]
```

que dada una lista de patentes **ps** y una numeración **n** devuelve una lista con las patentes de **ps** que tienen numeración **n**

Ejercicio 3

Se representará un registro donde se guardará cierta información sobre las patentes. Para ello deben definir el tipo **Titular** como sinónimo de **String** (el nombre y apellido del titular de la patente) y el tipo **Estado** que tiene dos constructores **SinDeuda** y **ConDeuda**, ambos sin parámetros. El tipo **Estado** **no debe estar** en la clase **Eq**.

Finalmente deben definir el tipo recursivo **Registro** que tiene dos constructores:

- El constructor **AgregaReg**: Toma cuatro parámetros, el primero del tipo **Matricula**, el segundo del tipo **Estado** (indica si la patente tiene o no deudas), el tercero del tipo **Titular** (el nombre y apellido del dueño) y el cuarto del tipo **Registro** que es a donde se agrega el nuevo registro.
- El constructor **SinRegs**: No toma parámetros. Representa la ausencia de registros (parecido a la lista vacía).

Sobre el registro se pueden hacer varias consultas. Definan usando *pattern matching* la función:

```
consulta :: Registro -> Titular -> Estado -> [Matricula]
```

que dado un registro **rs**, un titular **ti** y un estado **ei** devuelve una lista de las patentes que están en **rs** que son del titular **ti** y presenta un estado **ei**.

IMPORTANTE: Recordar que no se puede utilizar el operador **==** para comparar valores del tipo **Estado**

TIP: Puede serles de utilidad definir una función por *pattern matching*

```
mismo_estado :: Estado -> Estado -> Bool
```