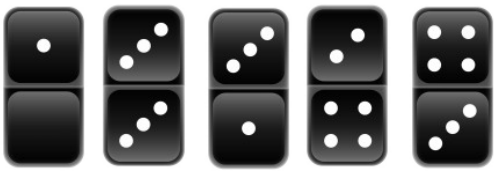
Parcial 1 - Algoritmos I Taller: Tema C

Ejercicio 1

Vamos a definir algunas funciones relacionadas con el juego domino (con numeración hasta 4)



a) Definir el tipo `Numeracion` que tiene constructores `Cero`, `Uno`, `Dos`, `Tres` y `Cuatro`. Los constructores no toman parámetros. El tipo `Numeracion` **no debe estar en la clase Eq**, pero sí en la clase `Show`. Programar la función

```
misma_numeracion :: Numeracion -> Numeracion -> Bool
```

que dados dos valores `n1` y `n2` de tipo `Numeracion` devuelve `True` si y sólo si son iguales (se construyen con el mismo constructor).

b) Definir el tipo `Domino` que consta de un único constructor `Ficha` que toma dos parámetros, ambos de tipo `Numeracion`. El primer numero se refiere al dumero de arriba del dominó y el segundo al de abajo. Programar la función

```
encajar :: Domino -> Domino -> Maybe Numeracion
```

que dados dos fichas `f1` y `f2` de tipo `Domino`, si el número de abajo de `f1` coincide con el número de arriba de `f2`, devuelve el número de abajo de `f2` usando el constructor `Just`. Si el número de abajo de `f1` no coincide con el número de arriba de `f2` se considera que no encajan y devuelve `Nothing`.

Ejemplos:

```
encajar (Ficha Cero Tres) (Ficha Tres Uno)
=
Just Uno

encajar (Ficha Uno Cero) (Ficha Dos Tres)
=
Nothing
```

Ejercicio 2

Programar la función

```
compatibles :: [Domino] -> Numeracion -> [Numeracion]
```

que dada una lista `ds` de dominós y una numeración `n` devuelve la numeración de abajo de las fichas en `ds` que tienen en la numeración de arriba a `n`.

Por ejemplo:

```
compatibles [Ficha Cero Uno, Ficha Tres Dos, Ficha Cero Cuatro] Cero
=
[Uno, Cuatro]
```

Ejercicio 3

Vamos a implementar un calendario usando tipos en Haskell. Para ello definir el tipo `Evento` como un sinónimo de `String`. Además definir el tipo `Categoria` que tiene constructores `Cumple`, `Reunion`, `Otro`. Por último a partir de los tipos anteriores, definir el tipo recursivo `Calendario` cuyos constructores son:

- `Agendar`: Toma tres parámetros. El primero de tipo `Evento` (el nombre del evento que se agrega), el segundo de tipo `Categoria` (el tipo de evento) y el tercero de tipo `Calendario` (el calendario al que se agrega el nuevo evento).
- `SinEventos`: No toma parámetros y representa la agenda vacia.

Finalmente programar la función

```
listar_reuniones :: Calendario -> [Evento]
```

que dado un calendario `ca` devuelve una lista de eventos `es` con los eventos de `ca` que son de la categoría `Reunion`.

- Inventar un ejemplo concreto con un calendario de al menos 3 elementos, ejecutarlo y decirlo como comentario en lo que se suba al parcial.

Ejercicio 4*

Definir la función

```
a_min :: Arbol a -> a
```

que dado un árbol `as` no vacío (`a_min Hoja` no debe estar definido) devuelve el elemento más chico dentro de `as`. Completar el tipado de la función para incluir los *type*

clases necesarios para programarla.