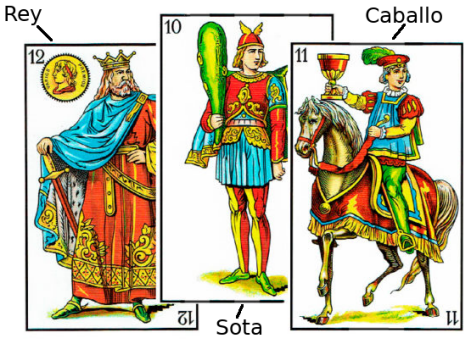
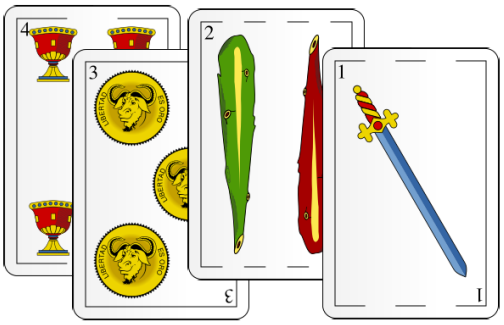
Parcial 1 - Algoritmos I Taller: Tema B

Ejercicio 1

Se van a implementar algunos aspectos del juego *escoba-del-quince* en Haskell.





Para ello

a) Definir el tipo `Palo` que consta de los constructores `Espada`, `Basto`, `Oro`, `Copa`. Los constructores no toman parámetros. **El tipo `Palo` no debe estar en la clase `Eq`**. Luego programa la función usando *pattern matching*:

```
mismo_palo :: Palo -> Palo -> Bool
```

que dados dos valores `p1` y `p2` del tipo Palo debe devolver `True` si y sólo si `p1` y `p2` son del mismo palo (se construyen con el mismo constructor).

b) Como en este juego no se utilizan las cartas 8 y 9 definir el tipo Figura que consta de constructores `Uno`, `Dos`, `Tres`, `Cuatro`, `Cinco`, `Seis`, `Siete`, `Sota`, `Caballo` y `Rey`. El valor de una carta está dado por su número, salvo para la Sota, el Caballo y el Rey donde sus valores se pueden ver en la tabla:

Figura	Valor
`Uno`	1
`Dos`	2
`(:)`	(:)
`Sota`	8
`Caballo`	9
`Rey`	10

Programar la función

```
valor_figura :: Figura -> Int
```

que dada una figura `f` devuelve su valor según la tabla de más arriba. Ahora definir el tipo `Carta` que tiene un único constructor `Naipe` que toma dos parámetros. El primero de tipo `Figura` y el segundo de tipo `Palo`. Finalmente programar la función

```
suma_cartas :: Carta -> Carta -> Maybe Int
```

que dadas dos cartas `c1` y `c2`, si tienen el mismo palo devuelve la suma de los valores de ambas cartas usando el constructor `Just`, y si tienen distinto palo devuelve `Nothing`

Ejercicio 2

En esta versión de la escoba las cartas solo se pueden combinar si son del mismo palo, por lo tanto programa la función:

```
compatibles :: Carta -> [Carta] -> [Figura]
```

que dada una carta `c` y una lista de cartas `cs` devuelve las figuras de las cartas de `cs` que son del mismo palo que `c`, y que al sumarles el valor de `c` no supera 15. Por ejemplo

```
 compatibles (Naipe Seis Oro) [Naipe Rey Oro, Naipe Tres Basto, Naipe Sota Oro]
=
 [Sota]
```

Ejercicio 3

Se va a representar una lista de reproducción en Haskell. Para ello definir los tipos `Duracion` como sinónimo de `Int`, y el tipo `Nombre` como sinónimo de `String`. Luego definir el tipo `Cancion` que tiene un único `Tema` constructor con dos parámetros, el primero de tipo `Nombre` y el segundo del tipo `Duracion`.
Por último definir el tipo `Estado` con constructores `Escuchado` y `NoEscuchado` (sin parámetros ambos) y definir el tipo recursivo `PlayList` que tiene dos constructores:

- `EnLista`: Toma tres parámetros. El primero de tipo `Cancion` (la canción que se agrega a la lista de reproducción), el segundo de tipo `Estado` (indica si la canción ya se escuchó) y el tercero de tipo PlayList (la lista de reproducción a la que se agrega el nuevo tema).
- `Vacia`: La lista de reproducción que no tiene ningún tema dentro.

Programar la función:

```
tiempo_reproduccion :: Playlist -> Int
```

que dada una lista de reproducción `pl` devuelve la suma de las duraciones de los temas que tienen estado `Escuchado`.

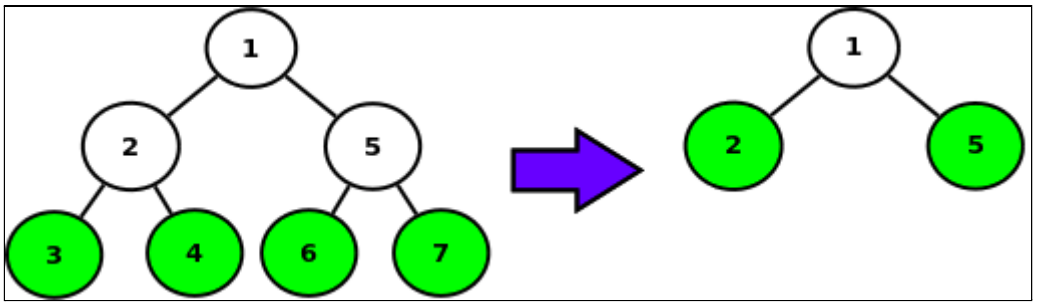
- Inventar un ejemplo concreto con una lista de al menos 3 elementos, ejecutarlo y decirlo como comentario en lo que se suba al parcial.

Ejercicio 4*

Definir la función

```
a_podar :: Arbol a -> Arbol a
```

que dado un árbol `as` no vacío (`a_podar Hoja` no debe estar definido) devuelve el un nuevo árbol cuyas ramas finales se han eliminado. Por ejemplo:



otro ejemplo: