```
data Día = Día {año :: Int, mes :: Int, día :: Int} deriving Eq
data Invitado = Invitado {nombre :: String, calendario :: [RestricciónParaJuntarse]}

cantidadDeDíasDelMes unMes unAño = 30 -- Solo escribo esto por si quieren probarlo en GHC, para que no se les rompa
```

-- Punto 1.a

```
type RestricciónParaJuntarse = Día -> Bool
```

-- Punto 1.b

```
permiteIrEl :: Día -> RestricciónParaJuntarse -> Bool
permiteIrEl día restricción = (not . restricción) día

puedeIrEl :: Día -> Invitado -> Bool
puedeIrEl díaParaReunirse = all (permiteIrEl díaParaReunirse) . calendario

enCuálesPuedeJuntarse :: [Día] -> Invitado -> [Día]
enCuálesPuedeJuntarse días invitado = filter (flip puedeIrEl invitado) días
```

-- Punto 1.c

```
{-
Gracias a Lazy Evaluation, sería posible empezar a evaluar esto, pero como queremos ver si "TODAS" las restricciones le permiten ir, seguiría evaluando infinitamente a menos que alguna determine que NO puede ir.

Así que la única forma en que termine la consulta, sería que para cada uno de los días, haya al menos una restricción que NO le permita ir, devolviendo una lista de días vacía.

En cualquier otro caso, queda evaluando "infinitamnete" (hasta que nos quedemos sin memoria).

-}
```

-- Punto 2.a

```
tengoUnaCita :: Día -> RestricciónParaJuntarse
tengoUnaCita díaDeLaCita díaDeLaReunión = díaDeLaCita == díaDeLaReunión
```

-- Punto 2.b

```
esFeriado :: [Día] -> RestricciónParaJuntarse
esFeriado díasFeriados díaDeLaReunión = elem díaDeLaReunión díasFeriados
```

-- Punto 2.c

```
podríaIrIndeseable :: Invitado -> RestricciónParaJuntarse
podríaIrIndeseable personaIndeseable díaDeLaReunión = puedeIrEl díaDeLaReunión personaIndeseable
```

-- Punto 2.d

```
esFinDeMes :: RestricciónParaJuntarse
esFinDeMes díaDeLaReunión = cantidadDeDíasDelMes (mes díaDeLaReunión) (año díaDeLaReunión) - 4 > día díaDeLaReunión
```

-- Punto 2.e

```
uhJustoTengoTurnoConElDentista :: RestricciónParaJuntarse
uhJustoTengoTurnoConElDentista _ = True
```

-- Punto 2.todos

```
pepe = Invitado {
  nombre = "Pepe",
  calendario = [
    tengoUnaCita (Día 2017 07 22),
    esFeriado [Día 2017 05 25],
    podríaIrIndeseable (Invitado "Sofía" []),
    esFinDeMes,
    uhJustoTengoTurnoConElDentista
  ]
}
```

-- Punto 3

```
agendarCitaEl día invitado = invitado {calendario = tengoUnaCita día : calendario invitado}
```

-- Punto 4.a

```
type Reunión = [Día]

mejorDíaParaJuntarse :: Reunión -> [Invitado] -> Día
mejorDíaParaJuntarse reunión invitados = máximoSegún (flip cantidadDePersonasQueIríanEl invitados) reunión
```

```
cantidadDePersonasQueIríanEl :: Día -> [Invitado] -> Int
cantidadDePersonasQueIríanEl díaDeLaReunión = length . quiénesPuedenJuntarseEl díaDeLaReunión

quiénesPuedenJuntarseEl :: Día -> [Invitado] -> [Invitado]
quiénesPuedenJuntarseEl díaDeLaReunión = filter (puedeIrEl díaDeLaReunión)

-- Forma 1 de pensar el "máximoSegún", con find y all
máximoSegún :: (Num n, Ord n) => (a -> n) -> [a] -> a
máximoSegún criterio conjunto = find (esElMejorSegún criterio conjunto) conjunto

esElMejorSegún :: (Num n, Ord n) => (a -> n) -> [a] -> a -> Bool
esElMejorSegún criterio conjunto elMejor = all (\unoPeor -> criterio elMejor >= criterio unoPeor) conjunto
-- Dejo el find únicamente por si el lector no quiere importar la biblioteca de listas
find :: (a -> Bool) -> [a] -> a
find criterio = head . filter criterio

-- Forma 2 de pensar el "máximoSegún", con foldl1 (tiene mejor rendimiento pero es más compleja)
máximoSegún2 :: (Num n, Ord n) => (a -> n) -> [a] -> a
máximoSegún2 criterio = foldl1 (mejorSegún criterio)

mejorSegún :: (Num n, Ord n) => (a -> n) -> a -> a -> a
mejorSegún criterio elemento1 elemento2
  | criterio elemento1 >= criterio elemento2 = elemento1
  | otherwise = elemento2
```

-- Punto 4.b

```
confirmarReunión :: Reunión -> [Invitado] -> [Invitado]
-- Podía interpretarse que había que devolver a toda la lista. o bien, solo a los que pueden ir. El primer caso sería:
confirmarReunión reunión invitados = map (confirmarSiPuedeIrEl (mejorDíaParaJuntarse reunión invitados)) invitados

confirmarSiPuedeIrEl :: Día -> Invitado -> Invitado
confirmarSiPuedeIrEl díaDeLaReunión invitado | puedeIrEl díaDeLaReunión invitado = agendarCitaEl díaDeLaReunión invitado
                                             | otherwise                         = invitado
```

-- Punto 5

```
maratónDeReuniones :: [Reunión] -> [Invitado] -> [Invitado]
maratónDeReuniones reuniones invitados = foldr confirmarReunión invitados reuniones
```