UNCUYO
UNIVERSIDAD
NACIONAL DE CUYO

FACULTAD
DE INGENIERÍA

Paradigmas de Programación

Unidad 4

Problema Paradigma Funcional

¡A trabajar!

Problema de Comidas

Problema de comidas

Queremos definir a una persona en base a indicadores nutricionales como el nivel de colesterol y su peso.

Una persona puede comer distintas comidas:

- Ensalada: de x kilos, aporta la mitad de ese peso x para la persona y no agrega colesterol.
- Hamburguesa: tiene una cantidad de ingredientes, el colesterol aumenta un 50% para la persona y lo hace engordar en (3 * la cantidad de ingredientes) kilos.
- La palta aumenta 2 kilos a quien la consume.

Hora de trabajar

Modele los tipos de datos necesarios

```haskell
type Kilos = Int
type Ingrediente = String

data Persona = Persona {
    colesterol :: Float,
    peso :: Kilos
} deriving (Show, Eq)

data Comida' = Ensalada Kilos | Hamburguesa [Ingrediente] |
 Palta deriving (Eq, Ord, Show)
```

Hora de trabajar

Implemente la función **comer** la cual dada un persona y un alimento, modifique el peso y colesterol de la persona.

Una persona puede comer distintas comidas:

- Ensalada: de x kilos, aporta la mitad de ese peso x para la persona y no agrega colesterol.
- Hamburguesa: tiene una cantidad de ingredientes, el colesterol aumenta un 50% para la persona y lo hace engordar en (3 * la cantidad de ingredientes) kilos.
- La palta aumenta 2 kilos a quien la consume.

```haskell
data Persona = Persona { colesterol :: Float, peso :: Kilos } deriving (Show, Eq)
data Comida' = Ensalada Kilos | Hamburguesa [Ingrediente] | Palta deriving (Eq, Ord, Show)

comer' :: Comida' -> Persona -> Persona
comer' (Ensalada kilos) persona = persona {
  peso = peso persona + div kilos 2
}

comer' (Hamburguesa ingredientes) persona = persona {
  peso = peso persona + (3 * fromIntegral (length ingredientes)),
  colesterol = colesterol persona * 1.5
}

comer' Palta persona = persona {
  peso = peso persona + 2
}
```

Hora de trabajar

Implemente la función **almorzar** la cual dada una persona y lista de comidas, la persona coma todas esas comidas

```haskell
data Persona = Persona { colesterol :: Float, peso :: Kilos } deriving (Show, Eq)
data Comida' = Ensalada Kilos | Hamburguesa [Ingrediente] | Palta deriving (Eq, Ord, Show)

almuerzo' :: [Comida']
almuerzo' = [Ensalada 1, Hamburguesa ["cheddar", "bacon"], Palta, Ensalada 3]

--Version normal
almorzar' :: Persona -> Persona
almorzar' persona = foldr comer' persona almuerzo'

--vesion recursiva
almorzarRecur' :: Persona -> Persona
almorzarRecur' persona = comerAlmuerzo persona almuerzo'
  where
    comerAlmuerzo :: Persona -> [Comida'] -> Persona
    comerAlmuerzo persona [] = persona
    comerAlmuerzo persona (x:xs) = comerAlmuerzo (comer' x persona) xs
```

Hora de trabajar

Implemente la función **contieneComida** que dada una lista de comidas y una comida en particular, determine si se encuentra esa comida.

```haskell
data Persona = Persona { colesterol :: Float, peso :: Kilos } deriving (Show, Eq)
data Comida' = Ensalada Kilos | Hamburguesa [Ingrediente] | Palta deriving (Eq, Ord, Show)

--version recursiva
contieneComidaRec :: Comida' -> [Comida'] -> Bool
contieneComidaRec _ [] = False
contieneComidaRec comida (x:xs) = comida == x || contieneComidaRec comida xs

--version no recursiva
contieneComida' :: Comida' -> [Comida'] -> Bool
contieneComida' comida comidas = elem comida comidas
```

Más trabajo

Ahora se propone definir cuándo una comida es sabrosa según el siguiente criterio:

- Las ensaladas son sabrosas cuando pesan más de un kilo
- Las hamburguesas son sabrosas cuando tienen cheddar
- Las paltas siempre son sabrosas

```haskell
data Persona = Persona { colesterol :: Float, peso :: Kilos } deriving (Show, Eq)
data Comida' = Ensalada Kilos | Hamburguesa [Ingrediente] | Palta deriving (Eq, Ord, Show)

sabrosa' :: Comida' -> Bool
sabrosa' (Ensalada kilos) = kilos > 1

--version no recursiva
sabrosa' (Hamburguesa ingredientes) = "cheddar" `elem` ingredientes
--version recursiva
sabrosa' (Hamburguesa []) = False
sabrosa' (Hamburguesa (ingrediente:ingredientes)) = ingrediente == "cheddar" || sabrosa' (Hamburguesa ingredientes)

sabrosa' Palta = True
```

Finalmente

Dada una lista de personas, cree una función que devuelva como resultado aquellas personas que se encuentran **en forma** y **disfruten de una comida en particular**.

- Una persona se considera en forma si su colesterol es menor a 100 y su peso es menor a 80.
- Una persona disfruta la comida si tiene un peso par y la comida es sabrosa.

```haskell
data Persona = Persona { colesterol :: Float, peso :: Kilos } deriving (Show, Eq)
data Comida' = Ensalada Kilos | Hamburguesa [Ingrediente] | Palta deriving (Eq, Ord, Show)

pesoPar :: Persona -> Bool
pesoPar = even . peso

enForma :: Persona -> Bool
enForma persona = colesterol persona < 100 && peso persona < 80

disfrutar' :: Comida' -> Persona -> Bool
disfrutar' comida alguien = pesoPar alguien || sabrosa' comida

filtroPersonas :: [Persona] -> Comida' -> [Persona]
filtroPersonas personas comida = filter (disfrutar' comida) (filter enForma personas)
```

Repositorio Github

Link de repositorio de github con código ejemplo:

https://github.com/bautifrigole/Paradigmas

WILL THE LEGENDARY FACU AND BAUTI RETURN?