



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 $(3 * \text{la cantidad de ingredientes})$ kilos.
- La palta aumenta 2 kilos a quien la consume.



Hora de trabajar

Modele los tipos de datos necesarios

```
1 type Kilos = Int
2 type Ingrediente = String
3
4 data Persona = Persona {
5     colesterol :: Float,
6     peso :: Kilos
7 } deriving (Show, Eq)
8
9 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 $(3 * \text{la cantidad de ingredientes})$ kilos.
- La palta aumenta 2 kilos a quien la consume.

```
1 data Persona = Persona { colesterol :: Float, peso :: Kilos } deriving (Show, Eq)
2 data Comida' = Ensalada Kilos | Hamburguesa [Ingrediente] | Palta deriving (Eq, Ord, Show)
3
4 comer' :: Comida' -> Persona -> Persona
5 comer' (Ensalada kilos) persona = persona {
6   peso = peso persona + div kilos 2
7 }
8
9 comer' (Hamburguesa ingredientes) persona = persona {
10   peso = peso persona + (3 * fromIntegral (length ingredientes)),
11   colesterol = colesterol persona * 1.5
12 }
13
14 comer' Palta persona = persona {
15   peso = peso persona + 2
16 }
```


Hora de trabajar

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

```
1 data Persona = Persona { colesterol :: Float, peso :: Kilos } deriving (Show, Eq)
2 data Comida' = Ensalada Kilos | Hamburguesa [Ingrediente] | Palta deriving (Eq, Ord, Show)
3
4 almuerzo' :: [Comida']
5 almuerzo' = [Ensalada 1, Hamburguesa ["cheddar", "bacon"], Palta, Ensalada 3]
6
7 --Version normal
8 almorzar' :: Persona -> Persona
9 almorzar' persona = foldr comer' persona almuerzo'
10
11 --vesion recursiva
12 almorzarRecur' :: Persona -> Persona
13 almorzarRecur' persona = comerAlmuerzo persona almuerzo'
14 where
15     comerAlmuerzo :: Persona -> [Comida'] -> Persona
16     comerAlmuerzo persona [] = persona
17     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.

```
1 data Persona = Persona { colesterol :: Float, peso :: Kilos } deriving (Show, Eq)
2 data Comida' = Ensalada Kilos | Hamburguesa [Ingrediente] | Palta deriving (Eq, Ord,
  Show)
3
4 --version recursiva
5 contieneComidaRec :: Comida' -> [Comida'] -> Bool
6 contieneComidaRec _ [] = False
7 contieneComidaRec comida (x:xs) = comida == x || contieneComidaRec comida xs
8
9 --version no recursiva
10 contieneComida' :: Comida' -> [Comida'] -> Bool
11 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 tienen más de un kilo
- Las hamburguesas son sabrosas cuando tienen cheddar
- Las paltas son sabrosas

```
1 data Persona = Persona { colesterol :: Float, peso :: Kilos } deriving (Show, Eq)
2 data Comida' = Ensalada Kilos | Hamburguesa [Ingrediente] | Palta deriving (Eq, Ord, Show)
3
4 sabrosa' :: Comida' -> Bool
5 sabrosa' (Ensalada kilos) = kilos > 1
6
7 --version no recursiva
8 sabrosa' (Hamburguesa ingredientes) = "cheddar" `elem` ingredientes
9 --version recursiva
10 sabrosa' (Hamburguesa []) = False
11 sabrosa' (Hamburguesa (ingrediente:ingredientes)) = ingrediente == "cheddar" || sabrosa'
    (Hamburguesa ingredientes)
12
13 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.

```
1 data Persona = Persona { colesterol :: Float, peso :: Kilos } deriving (Show, Eq)
2 data Comida' = Ensalada Kilos | Hamburguesa [Ingrediente] | Palta deriving (Eq, Ord, Show)
3
4 pesoPar :: Persona -> Bool
5 pesoPar = even . peso
6
7 enForma :: Persona -> Bool
8 enForma persona = colesterol persona < 100 && peso persona < 80
9
10 disfrutar' :: Comida' -> Persona -> Bool
11 disfrutar' comida alguien = pesoPar alguien || sabrosa' comida
12
13 filtroPersonas :: [Persona] -> Comida' -> [Persona]
14 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?**