Final de Paradigmas de Programación 18/02/2012 Nombre:

Punto 1

Considerando los siguientes ejemplos de código

Caso 1:

```
#Cliente
        alquileresVencidos
                "alquileres es una colección de alquileres"
                ^ alquileres select: [:alquiler | alquiler estaVencido]
```

Caso 2:

```
#Cliente
        alquileresVencidos(Cliente)
          "alquileres es una colección de alquileres"
          |alqVencidos |
          alqVencidos := Set new.
          alquileres do: [ :alq |
                    alq estaVencido ifTrue:[
                    alqVencidos add: alq
                            ].
                ].
                ^alqVencidos
```

1. ¿Qué solución elegirías y por qué?
2. Desarrolle soluciones similares para los casos 1 y 2 en Haskell.
3. Desarrollar las principales diferencias entre las soluciones declarativo y procedural.

Punto 2

Dada la siguiente función:

```
func [ ] xs  _ = []
func xs [ ]  _ = []
func (x:xs) (y:ys) z = z x y : func xs ys z
```

1. Dado el siguiente dominio e imagen propuesto para la función ¿ Encuentra algún error ? Si es incorrecto corríjalo y justifique.

   ```
   func :: [a] -> [b] -> (a-> b) -> [b]
   ```

2. Considerando este ejemplo.
   a) Además de la recursividad, ¿qué otro concepto se está utilizando?
   b) Dada una lista de flores con sus cantidades con este formato (nombreFlor, cantidad)

   ```
   cantidades = [("orquídea", 20),("jazmín", 30),("rosa", 40)]
   ```

   Y una lista de precios por unidad de las mismas flores en el mismo orden con este formato (nombreFlor, precioUnidad)

   ```
   precioUnidad = [("orquídea", 40),("jazmín", 15),("rosa", 20)]
   ```

b.1) Aplique la función del punto 2.1 para obtener una lista de tuplas con este formato:
`(nombreFlor, cantidad, precioUnidad)`

b.2) Utilice la función del punto 2.1 para obtener una lista de tuplas con este formato:
`(nombreFlor, precioTotal)`

c) ¿Cuáles son las ventajas de usar el concepto del punto a? ¿Qué pasaría en el punto b si no existiera dicho concepto?

d) Propuesta: Reescribir la solucion sin utilizar recursividad.

Punto 3

Dados las siguientes opciones de código:

Opción 1

```
#Cliente
    saldo
      | total |
    total := 0.
      self facturas do: [:fact |
            fact items do: [:item | total := total + item cantidad * item producto precioUnitario]
      ].
       ^total
```

Opción 2

```
    #Cliente
     saldo
            ^self facturas inject: 0 into: [:tot :fact | tot + fact importe]
    #Factura
      importe
            ^self items inject: 0 into: [:tot :item | tot + item importe]
    #Item
      importe
            ^self cantidad * self producto precioUnitario
```

1. ¿Cuál de las dos le parece mejor y con que concepto esta relacionada esta elección?
2. En caso de incorporarse un nuevo tipo de documento que no tiene ítems. ¿Qué hay que modificar/agregar en los modelos anteriores?
3. ¿Cómo se refleja la ventaja de un modelo sobre el otro respecto de este agregado?

Punto 4

1. ¿Cuáles son las diferencias entre el concepto de **functor** en Prolog y el de **tupla** en Haskell respecto al concepto de tipo.
2. ¿Qué ventajas se obtiene cuando se usa un predicado inversible?