Funcional 5: integración

Esta práctica incluye ejercicios para hacer una práctica integral de los temas que se vieron sobre paradigma funcional.
Varios son parciales viejos o adaptaciones.

Ventas de PCs

Una empresa de venta de computadoras está desarrollando un sistema para llevar registro de ventas. Para ello cuenta con la siguiente información:

Lista de los vendedores de la empresa

```
vendedores = ["Martin", "Diego", "Claudio", "José"]
```

Lista de ventas de la forma (fecha, nombre vendedor, lista de componentes de la máquina. La fecha es una tupla de la forma (dia, mes, anio) y los componentes son Strings.

```
ventas = [((1,2,2006), "Martin", ["Monitor GPRS 3000", "Motherboard ASUS 1500"]),
          ((1,2,2006), "Diego", ["Monitor ASC 543", "Motherboard Pindorcho"]),
          ((10,2,2006), "Martin", ["Monitor ASC 543", "Motherboard ASUS 1200"]),
          ((12,2,2006), "Diego", ["Monitor GPRS 3000", "Motherboard ASUS 1200"]),
          ((4,3,2006), "Diego", ["Monitor GPRS 3000", "Motherboard ASUS 1500"])]
```

Lista de precios de los componentes, de la forma (nombre componente, precio).

```
precios = [("Monitor GPRS 3000", 200), ("Motherboard ASUS 1500", 120),
           ("Monitor ASC 543", 250), ("Motherboard ASUS 1200", 100),
           ("Motherboard Pindorcho", 30)]
```

Se pide desarrollar las siguientes funciones utilizando al menos una vez cada uno de los siguientes conceptos:

- Listas por comprensión
- Aplicación parcial
- Composición de funciones
- Funciones de orden superior

Indicar en cada caso el lugar en donde ha sido utilizada cada uno de los conceptos solicitados.

1.1. precioMaquina/1, recibe una lista de componentes, devuelve el precio de la máquina que se puede armar con esos componentes, que es la suma de los precios de cada componente incluido.
P.ej.

```
Main> precioMaquina ["Monitor GPRS 3000", "Motherboard ASUS 1500"]
--> 320 ($200 del monitor + $120 del motherboard)
```

1.2. cantVentasComponente/1, recibe un componente y devuelve la cantidad de veces que fue vendido, o sea que formó parte de una máquina que se vendió.
P.ej.

```
Main> cantVentasComponente "Monitor ASC 543"
--> 2
```

La lista de ventas no se pasa por parámetro, se asume que está identificada por ventas.

1.3. vendedorDelMes/1, se le pasa un par que representa (mes,año) y devuelve el nombre del

vendedor que más vendió en plata en el mes. O sea no cantidad de ventas, sino importe total de las ventas. El importe de una venta es el que indica la función precioMaquina. P.ej.

```
Main> vendedorDelMes (2,2006)
--> "Martin" (vendio por $670, una máquina de $320 y otra de $350)
```

1.4. ventasCriterio/1, que devuelva el importe total de las ventas según el criterio que se le pasa como parámetro. Ejemplos de uso:

```
esDeLaFecha fecha (fechaVenta, _, _) = fecha == fechaVenta

Main> ventasCriterio (esDeLaFecha (1, 2, 2006))
--> 600 (una máquina de $320 y otra de $280)
```

Usando esa función definir las funciones que permitan:

1.4.1. Obtener las ventas de un mes, de forma que:

```
Main> ventasMes (2,2006)
--> 1050
```

1.4.2. Obtener las ventas totales realizadas por un vendedor sin límite de fecha, de forma que:

```
Main>  ventasVendedor "Diego"
--> 900
```

1.4.3. huboVentas/1 que indica si hubo ventas en un mes determinado, p.ej.

```
Main> huboVentas (1, 2006)
--> False
```

Viajes en avión

Se cuenta con información viajes que hicieron determinadas personas; de cada persona se tienen los tramos que hizo en avión, con origen, destino y duración en horas de vuelo de cada tramo. P.ej.

```
viajes =
   [ ("pepe"  , [("BsAs","Washington",17),("Washington","Tokyo",22)]),
     ("lucho" , [("BsAs","Boston",10)]),
     ("luisa" , [("Roma","Caracas",15),("Caracas","Lima",7),
                 ("Lima","Tokyo",21)]),
     ("pichu" , [("Bombay","Teheran",8),("Baku","Moscu",10)]),
     ("juana" , [("Madrid","Varsovia",11),("Moscu","Karachi",10),
                 ("Delhi","Bandung",8)]) ]
```

Se pide:

2.1. Definir la función **tiempoEnElAire**/1 que recibe una lista de tramos y devuelve el total de horas voladas. P.ej. la consulta

```
Main>  tiempoEnElAire [("Roma","Caracas",15),("Caracas","Lima",7),("Lima","Tokyo",21)]
43
```

Para probar pueden armar la función **tramosDe**/2 que reciba persona y data de viajes, y devuelva los tramos de esa persona. Entonces p.ej. pueden probar así:

```
Main>  tiempoEnElAire (tramosDe "luisa" viajes)
43
```

Nota: De esta armen dos definiciones, una que use recursividad y la otra que no.

2.2. Definir la función **tramosTerrestres**/1 que recibe una lista de tramos y devuelve los tramos terrestres que la persona tuvo que hacer, que son los tramos entre el destino de cada tramo y el origen del siguiente, a menos que coincidan.

Lo que devuelve la función es una lista de pares, cada par es un tramo terrestre.

Veamos en los ejemplos los tramos terrestres de cada viajero:
- para pepe, lucho y luisa: no hay tramos terrestres
- para pichu: uno solo, Teheran-Baku.
- para juana: Varsovia-Kiev y Karachi-Delhi.

Entonces si consulto

```
Main> tramosTerrestres (tramosDe "juana" viajes)
[("Varsovia","Moscu"),("Karachi","Delhi")]
```

2.3. Definir la función **pasaPor**/2 que recibe una ciudad y una lista de tramos (en ese orden) y devuelve True si la persona que hizo esos tramos pasó por la ciudad, o sea, si la ciudad es origen o destino de alguno de esos tramos.

2.4. Definir la función **quienes**/2 que recibe la info de los viajes y una condición, y devuelve los nombres de los viajeros a los que les pasó lo que dice la condición, que es una condición sobre los tramos que voló la persona; dicho de otra forma, para los que la condición evaluada sobre los tramos "da true".

Usarla para saber (las respuestas esperadas entre paréntesis)

- quiénes volaron más de 30 horas (pepe y luisa)
- quiénes hicieron más de dos tramos aéreos (luisa y juana)
- quiénes pasaron por Moscú (pichu y juana)
- quiénes no pasaron por Buenos Aires (luisa, pichu y juana)
- quiénes no hicieron ningún tramo terrestre (pepe, lucho y luisa)

2.5. Ahora quiero usar la función quienes para saber quiénes cumplieron dos o más condiciones, p.ej. quiénes volaron más de 30 horas pero hicieron menos de tres tramos aéreos.
Definir la función **andf**/2 que recibe una lista de funciones y un valor, y devuelve True si todas las funciones evaluadas sobre el valor dan True. Ejemplos

```
andf [odd, (0<), esMultiploDe 3] 9    True
andf [odd, (0<), esMultiploDe 3] 11   False (no es múltiplo de 3)
andf [odd, (0<), esMultiploDe 3] -9   False (no es positivo)
```

Resolver, usando quienes y andf, estas consultas

- quiénes volaron más de 30 horas pero hicieron menos de tres tramos aéreos
- quiénes pasaron por Moscú e hicieron más de un tramo terrestre.

Procesador de Texto

Nos piden un programa que permita hacer operaciones de edición de texto sobre artículos:

criticaHobbit = [“Esta”, “novela”, “de”, “Tolkien”, “narra”, “las”, “aventuras”, “de”, “un”, "nardo"…]
recetaTortaFrita = [“Ingredientes”, “1”, “huevo”, "2", "naranjas", "naturales"…]

articulos = [criticaHobbit, recetaTortaFrita, …]

Recordando que un String es una lista de caracteres, programar las funciones que se solicitan a continuación utilizando por lo menos una vez

- composición de funciones
- aplicación parcial
- funciones de orden superior
- recursividad
- listas definidas por comprensión

3.1. Definir la función **palabrasQueNoEmpiezanConEnCada/2** dados un carácter y una lista de artículos, devuelve una lista con la cantidad de palabras de cada artículo que no empiezan con esa letra. P.ej. (suponiendo que los artículos terminan donde se muestra)

```
Main> palabrasQueNoEmpiezanConEnCada 'n' articulos
[7, 4,…]
```

3.2.1. Definir la función **articulosLargos/1** dada una lista de artículos permite conocer los que tienen más de 500 palabras.

3.2.2. Definir la función **articulosParejosEn/2** dada una lista de artículos y un número, devuelve los artículos para los que todas las palabras tienen esa cantidad de letras.

3.2.3. Definir la función **articulosDificiles/1** dada una lista de artículos permite conocer los que tienen más de 20 palabras difíciles. La lista de palabras difíciles se guarda de la siguiente manera:

palabrasDificiles = ["adláter", "inveterado", "prosapia", "piscolabis", …]

3.3. Definir la función **articulosQueCumplen/2** dada una lista de artículos y una lista de condiciones que tienen que cumplir permite conocer los artículos que cumplen todas las condiciones dadas.

3.3.1. Escribir la consulta que nos permiten obtener los artículos largos, parejos en 6 letras, y difíciles, de una lista de artículos, usando **articulosQueCumplen**.

3.4. Operaciones sobre texto: interesa trabajar sobre el texto de un artículo. En particular,

3.4.1. Eliminar un carácter de un artículo. Debe devolver el texto completo.

```
Main> eliminarCaracter 'n' criticaHobbit
"Esta ovela de Tolkien arra las aveturas de u …"
```

3.4.2. Restringir el texto de un artículo a una determinada cantidad de caracteres

```
Main> restringir 40 criticaHobbit
"Esta novela de Tolkien narra las avent"
```

3.5. Dados los siguientes requerimientos:

a) Invertir el texto de un artículo y luego pasarlo a mayúsculas
b) Eliminar las palabras difíciles o que comiencen con 'X' mayúscula.
c) Eliminar las palabras de más de 10 letras, luego las de menos de 3 letras y por último pasarlo a mayúsculas.
d) Saber si en un artículo al menos una palabra empieza con 'z'.

Mostrar ejemplos de invocación y de respuesta. Si es necesario defina funciones auxiliares o locales.

3.6. Definir la función **hayPlagio/2**, que dados dos artículos dice si hay al menos 50 palabras de más de 3 letras en común.