IWI131 Programación (Fase 2) — Ejercicios Semana 2: Listas y Tuplas

1. La empresa *PyCornerShop* dispone de una aplicación que le permite a sus usuarios conseguir productos del supermercado desde la comodidad del hogar usando sus teléfonos móviles. La información de los repartidores y usuarios se encuentra almacenada en dos listas llamadas `repartidores` y `usuarios`, respectivamente.

La lista `usuarios` contiene tuplas de 2 elementos: Código del usuario y ubicación del domicilio (tupla con coordenadas):

```
usuarios = [(1221, (5, 2)), (441, (8, 2)), (587, (10, 1)), ...]
```

La lista `repartidores` contiene tuplas de 4 elementos: El nombre del repartidor, su ubicación actual (tupla con coordenadas), disponibilidad (valor booleano) y lista de visitas a usuarios. Esta lista contiene tuplas con el código del usuario y la cantidad de veces que ha sido visitado por el repartidor:

```
repartidores = [
        ('rayo macuin', (10, 2), True, [(1221, 5), (441, 8), (587, 2)]),
        ('reparti dhor', (9, 3), True, [(1221, 2), (441, 5), (587, 3)]),
        ('eliseo al-azar', (5, 5), False, [(1221, 8), (441, 2)]), ...
]
```

Si un repartidor nunca ha visitado a un usuario, no aparecerá en la lista una tupla con el código de ese usuario.

Para que la *PyCornerShop* pueda funcionar de manera eficiente, le solicita a Ud. que implemente una función llamada `buscar_repartidor(usuarios,repartidores,codigo)` que reciba como parámetros las listas antes mencionadas y un código de usuario. La función debe retornar una lista de repartidores disponibles y cercanos, ordenados de manera descendente, según el número de visitas que hayan realizado al usuario indicado. Un repartidor se considera cercano si está ubicado a menos de 4 km del usuario y para esto considere que las tuplas de ubicación tienen valores enteros medidos en km. Finalmente, si no hay repartidores cercanos, la función debe retornar una lista vacía.

Ejemplos:

```
>>> buscar_repartidor(usuarios,repartidores,587)
['reparti dhor', 'rayo macuin']
>>> buscar_repartidor(usuarios,repartidores,441)
['rayo macuin', 'reparti dhor']
>>> buscar_repartidor(usuarios,repartidores,1221)
[]
```

Nota: Para calcular la distancia d entre un usuario ubicado en (x_1, y_1) y un repartidor ubicado en (x_2, y_2) se debe utilizar la fórmula: $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

IWI131 Programación (Fase 2) — Ejercicios Semana 2: Listas y Tuplas

2. El prestamista Vito Corleone, aburrido de perseguir a sus clientes, ha solicitado la ayuda de algún astro de la programación para resolver sus problemas logísticos.

Vito cuenta con la lista `clientes`, la que tiene tuplas con el nombre de cada cliente, el monto adeudado y la fecha del último pago (también dentro de una tupla).

```
clientes = [
('Don Ramon', 3500, (9, 4, 2014)),
('Miguel', 2785, (30,10, 2014)),
('Cesar', 100, (28, 5, 2015)), ...]
```

Nota: Esto sólo es un ejemplo, considere que la lista puede tener muchos clientes. Sin embargo, asuma que cada nombre de cliente aparece sólo una vez.

1. Implemente la función `deuda_total(clientes)`, que reciba como entrada la lista de tuplas `clientes`, y retorne la suma de las deudas de todos los deudores.

```
>>> deuda_total(clientes)
6385
```

2. Implemente la función `mayor_deudor(clientes, ultimo)`, que retorne el nombre del cliente que tiene la mayor deuda, y que además su último pago haya sido realizado en el año `ultimo`. Si no hay clientes con pagos en el año `ultimo`, retorne un string vacío.

```
>>> mayor_deudor(clientes, 2014)
'Don Ramon'
```

```
>>> mayor_deudor(clientes, 2015)
'Cesar'
```

```
>>> mayor_deudor(clientes, 2016)
''
```

IWI131 Programación (Fase 2) — Ejercicios Semana 2: Listas y Tuplas

3. Una organización internacional de conservación construyó un domo de cristal para mantener un micro clima en una cierta superficie. Para esto, se instalaron miles de paneles de cristal, contando cada uno de ellos con varios sensores que reportan información diariamente a un computador central. Cada sensor transmite una vez al día una tupla con su medición, indicando la fecha, su identificador único y el valor de la medición. Como ejemplo, una medición sería: `(2016, 3, 27, 'aa:01:10', 0.0021)`

Todas estas mediciones son colocadas en una lista en el computador central, la cual es mantenida **ordenada descendentemente** por el valor medido. Esto facilita la recuperación posterior de mediciones, debido a que por el gran volumen del registro resulta imposible ordenarlo con cualquiera de los algoritmos disponibles en la actualidad. A modo de **ejemplo**, un posible registro sería:

```
registro = [
(2016, 1, 25,'d2:00:10', 0.0112),
(2015, 10, 24,'3e:15:c2', 0.0105),
(2015, 11, 3,'28:0b:5c', 0.0009), ...
]
```

Usted debe:

1. Construir la función `agregar_medicion(registro, A, M, D, id, val)`, la que agrega la información capturada al `registro`. Considere `A` el año, `M` el mes y `D` el día de la medición.

```
>>> agregar_medicion(registro, 2016, 3, 27, 'aa:01:10', 0.0021)
>>> print(registro)
[(2016, 1, 25, 'd2:00:10', 0.0112), (2015, 10, 24, '3e:15:c2', 0.0105),
(2016, 3, 27, 'aa:01:10', 0.0021), (2015, 11, 3, '28:0b:5c', 0.0009)]
```

2. Construir la función `corregir_medicion(registro, A, M, D, id, nvo_valor)` que corrige el valor medido de un sensor. La función recibe como parámetro el `registro`, el año, mes y día de la medición a corregir, el identificador único y el nuevo valor medido. Considere que luego de esta actualización el registro debe permanecer ordenado como se indicó en el enunciado.

```
>>> corregir_medicion(registro, 2016, 3, 27, 'aa:01:10', 0.0221)
>>> print(registro)
[(2016, 3, 27, 'aa:01:10', 0.0221), (2016, 1, 25, 'd2:00:10', 0.0112),
(2015, 10, 24, '3e:15:c2', 0.0105), (2015, 11, 3, '28:0b:5c',0.0009)]
```

3. Construir la función `sensores_sobre_umbral(registro, u)`, la que recibe un valor de medición usado como umbral y también el `registro`. Esta función entrega un listado con todos aquellos sensores, cuyo valor medido se encontró por sobre el umbral `u` entregado. Este listado no debe contener identificadores duplicados.

```
>>> sensores_sobre_umbral(registro, 0.01)
['aa:01:10', 'd2:00:10', '3e:15:c2']
```

IWI131 Programación (Fase 2) — Ejercicios Semana 2: Listas y Tuplas

4. Cineton, una nueva cadena de cines creada por emprendedores de la USM, está ingresando al mercado cinematográfico. Por eso, necesita de su ayuda para implementar ciertas funciones en Python y con ellas manejar la cartelera. Para ello, se cuenta con la información de la cartelera de cine en una lista de tuplas como `cartelera`. El formato de cada tupla en la lista es el siguiente:

```
(mes, país, nombre_película, año_filmación, [sala1, sala2, ...])
```

Un ejemplo se muestra a continuación:

```
cartelera = [
('febrero','FRANCIA','El muelle', 1962, ['sala1','sala3']),
('febrero','FRANCIA','La dama de honor', 2004, ['sala1','sala4']),
('abril','RUSIA','Padre del soldado', 1964, ['sala3','sala2','sala4']),
('enero','CHILE','Gloria', 2013, ['sala1','sala2']),
('mayo','MEXICO','Cumbres', 2013, ['sala3','sala2']),
('julio','FRANCIA','Melo', 1986, ['sala3','sala1']),
('junio','BELGICA','Rondo', 2012, ['sala4','sala2']),
('marzo','ALEMANIA','Tiempo de Canibales', 2014, ['sala1','sala2']),
('marzo','ALEMANIA','Soul Kitchen', 2009, ['sala3','sala4']),
...]
```

Para entender mejor, en `cartelera` la película 'Gloria' (Chilena) creada en 2013, se exhibirá el mes de 'enero' en las 'sala1' y 'sala2'.

1. Desarrolle la función `pelicula_por_pais(cartelera, pais)` que recibe la lista de la cartelera y el nombre de un país, y que retorne la lista con las películas realizadas en dicho país. Cada elemento de esta lista resultante es una tupla con el nombre de la película y el año de filmación.

```
>>> pelicula_por_pais(cartelera, 'FRANCIA')
[('El muelle', 1962), ('La dama de honor', 2004), ('Melo', 1986)]
```

2. Desarrolle la función `peliculas_por_sala(cartelera, sala)` que reciba la lista de la cartelera y la sala donde se exhibirán las distintas películas. Esta función debe retornar una lista de tuplas, donde cada tupla tiene la forma `(mes, listapelis)` con `mes` un string de un mes y `listapelis` una lista con todas las películas que se exhibirán en `sala`. Solo deberá incluir tuplas en donde `listapelis` contenga al menos una película.

```
>>> pelicula_por_sala(cartelera, 'sala1')
[('enero', ['Gloria']), ('febrero', ['El muelle', 'La dama de honor']),
('marzo', ['Tiempo de Canibales']), ('julio', ['Melo'])]
```

3. Desarrolle la función `mas_antigua(cartelera)` que retorne el nombre de la película y el país donde fue filmada la película más antigua. Si dos o más películas son las mas antiguas, seleccione cualquiera.

```
>>> mas_antigua(cartelera)
('El muelle', 'FRANCIA')
```

IWI131 Programación (Fase 2) — Ejercicios Semana 2: Listas y Tuplas

5. Inversiones RCaray está siendo investigada por una presunta estafa millonaria. Su presidente generó una asociación ilícita utilizando los dineros de cientos de personas durante los tres últimos años por medio de varias empresas fantasma. Las investigaciones pudieron reunir una información parcial acerca de las personas y las sumas de dineros involucradas. La policía ha recaudado esta información en una lista llamada `estafados`, la que almacena tuplas de la forma `(rut, deuda, empresa, fecha)`, donde rut está en el formato string, deuda en formato entero, empresa en formato string y fecha es una tupla `(dd, mm, aaaa)`. A continuación, se muestra un ejemplo de la estructura `estafados`:

```
estafados = [
('12.234.567-8', 2000000,'pelados_furiosos', (25, 5, 2015)),
('9.111.567-k', 5500000,'multibank', (1, 10, 2014)),
('14.987.007-1', 100000000,'inversiones_seguras', (30, 11, 2016)),
('12.234.567-8', 10000000,'multibank', (2, 7, 2015)),
('12.234.567-8', 2500000,'multibank', (18, 1, 2016)), ...]
```

Una misma persona pudo ser estafada por varias empresas en distintas fechas. De acuerdo a la información anterior, se le solicita implementar las siguientes funciones:

1. `estafado_por(lista, rut)`, función que recibe 2 parámetros. La lista de tuplas `estafados` y un string correspondiente al rut de la persona. Esta debe retornar una lista con las distintas empresas que estafaron a la persona con dicho rut. En caso de que el rut ingresado no se encuentre, entregar el mensaje `'Rut no estafado'`.

```
>>> estafado_por(estafados,'12.234.567-8')
['pelados_furiosos','multibank']
>>> estafado_por(estafados,'19.678.222-2')
Rut no estafado
```

2. `ranking(estafados)`, función que recibe como parámetro la lista de estafados y retorna una lista de tuplas, donde cada tupla `(nombre, dinero)` contiene el nombre de una empresa y la cantidad de dinero que cada empresa logró acumular. La lista debe estar ordenada de mayor a menor por el dinero recaudado.

```
>>> ranking(estafados)
[('inversiones_seguras', 100000000), ('multibank', 18000000),
 ('pelados_furiosos', 2000000), ...]
```

3. `estafados_por_fecha(estafados, inicial, final)`, función que recibe como parámetros la lista de estafados, la tupla fecha inicial y la tupla fecha final en formato tuplas `(dd, mm, aaaa)`. La función retorna una lista de tuplas `(rut, deuda)` ordenadas en forma descendente según la deuda total generada entre las fechas entregadas.

```
>>> estafados_por_fecha(estafados,(03,01,2016),(01,01,2017))
[('14.987.007-1', 100000000), ('12.234.567-8', 2500000), ...]
```