Examen – IIC1103 2020 TAV

1. [P1a] Supermercado

Enunciado

El gerente de la cadena SomodSud te ha pedido que averigües los precios más bajos de diferentes listas de compras. Esto nace a raíz de que quiere tener los precios más competitivos del mercado. Como el generente no pasó por el Excel-en-te curso de introducción a la programación, no puede generar un código que responda a esta difícil solicitud. Pero gracias a hermosas coincidencias del destino, te tiene a tí como programador experto en python <3.
El generente te ha facilitado una lista llamada "Supermercados" que contiene la información de todos los supermercados de la zona (no te diré cómo obtuvo esta información, sólo la tiene).
Esta lista es de la siguiente forma:

```
Supermercados = [
["Jumbito", [ ["Alfajores", 250, 30], ["Mermeladas",  1500, 5], ["Pan", 1000, 40] ] ],
["Santo es él", [ ["Alfajores", 350, 100], ["Mermeladas",  1000, 10], ["Pan", 1500, 15]] ]
]
```

Esta lista contiene listas que representan a **un** supermercado, en donde la lista **supermercado** contiene en su **primera posición** el **nombre** y en la **segunda posición**, una **lista que contiene todos los artículos que vende** (cada artículo contiene su **nombre**, **precio** de venta y **stock** disponible).

Parte A

Para esta primera parte, deberás crear la función "hay_producto(supermercados, nombre_supermercado, producto)" que recibe como parámetros:
1. La **lista con todos los supermercados**.
2. El **nombre_supermercado** (este nombre siempre estará en la lista supermercados y deberás buscar el **"producto"** dentro de él).
3. Una **lista** llamada "**producto**" de la forma [nombre_producto_buscado, cantidad_buscada].
Esta función debe **retornar un string** de la forma:

```
"El supermercado {nombre_supermercado}, quedará con {stock_supermercado - cantidad_buscada}."
"El supermercado {nombre_supermercado}, no tiene stock suficiene para comprar {nombre_producto_buscado}."
```

El **primer** mensaje se retornará **cuando hay stock suficiente (esto quiere decir que el stock_supermercado - cantidad_buscada >= 0) y el segundo, cuando no haya stock suficiente (stock_supermercado - cantidad_buscada < 0).**
Mucho éxito!

2. [P1b] Supermercado

Parte B

Para esta segunda parte, deberás crear la función "comprar_lista(supermercados, lista_productos)" que recibe como parámetros:
1. La **lista con todos los supermercados**.
2. Una **lista de listas** llamada "**lista_productos**", que contiene elementos (listas) de la forma [nombre_producto_buscado, cantidad_buscada].
Esta función debe **retornar una lista** de todos los supermercados en los que se pueda comprar **TODOS** los productos de esta lista, en el formato:

```
[nombre_supermercado1, nombre_supermercado2, ..., nombre_supermercadoN]
```

Si no hay supermercados en los cuales puedas comprar **TODA** la lista, debes retornar una lista vacía.
Mucho éxito!

3. [P2a] Librería LeoLeo

Enunciado

El dueño de la Librería LeoLeo te ha pedido ayuda para poder gestionar su stock de libros. Esta librería vende y arrienda libros. Deberás leer la información de todos los libros, consolidarla, y luego calcular las ganancias para cada uno de ellos.
Para hacer esto se te avisa que cada libro de la Librería tiene un `Id` único (aunque pueden haber varias copias del mismo libro) y que cada libro puede estar en uno de cuatro estados: Vendido, Arrendado, Disponible y Dañado. Se te entrega el archivo `info_libros.txt`, el cual guarda el id y el estado para cada uno de los libros de la biblioteca. El archivo `info_libros.txt` se ve de la siguiente forma:

```
ID;Estado
1;Disponible
1;Vendido
1;Arrendado
20;Dañado
20;Dañado
3;Disponible
3;Disponible
3;Disponible
20;Dañado
3;Arrendado
```

Para este ejemplo, la bilbioteca tiene 10 libros: tiene tres copias del libro con id 1, una de las cuales está Disponible, otra vendida y otra arrendada; tres copias del libro con id `20`, las cuales todas están dañanas; y por último, cuatro copias del libro con id `3`, tres de las cuales están disponibles y una arrendada.
Nota: Los ID de los libros **no** están necesariamente en orden

Parte A

Tu programa debe leer el archivo `info_libros.txt`, y crear un archivo consolidado con el resumen de cada libro de acuerdo a su identificador. En particular, debe mostrar (para el identificador de cada libro), cuántos libros vendidos hay, cuántos arrendados, cuántos disponibles, y cuántos dañados. Luego, esta información se debe escribir en un archivo `libros_consolidado.txt`, donde cada fila tenga el formato: `ID;Vendidos;Arrendados;Disponibles;Dañados`. Los libros en el archivo consolidado deben estar en orden respecto a su id.
Por ejemplo, si el archivo "info_libros.txt" contiene la siguiente información:

```
ID;Estado
1;Disponible
1;Vendido
1;Arrendado
20;Dañado
20;Dañado
3;Disponible
3;Disponible
3;Disponible
20;Dañado
3;Arrendado
```

El archivo `libros_consolidado.txt` debería verse como:

```
ID;Vendidos;Arrendados;Disponibles;Dañados
1;1;1;1;0
3;0;1;3;0
20;0;0;0;3
```

4. [P2b] Librería LeoLeo

Parte B

Para esta parte, deberás calcular las ganancias de LeoLeo. Para esto, deberás leer la información del archivo de texto plano `info_ventas.txt`, que contiene la categoría del libro, el valor de venta y de arriendo por cada libro (según su identificador). Particularmente, y para el mismo ejemplo anterior, este archivo se vería de esta forma:

```
ID;Categoría;Valor_Venta;Valor_Arriendo
1;Novela;2000;500
3;Misterio;8000;2500
20;Novela;7000;3000
```

Para calcular las ganancias por libro (según su identificador), debes sumar los productos entre la cantidad de libros vendidos y su valor venta, la cantidad de libros vendidos y su valor de arriendo, y a eso restarle la cantidad de libros dañados por su valor venta. Debes escribir la ganancia por libro en un archivo `ganancias.txt`. Los libros deben estar ordenados de mayor a menor ganancia, y si hay empate, se debe usar la categoría de cada libro para ordenar. Particularmente hay **solo** 3 categorías, y se deben ordenar primero `Misterio`, luego `Novela` y por último `Manual`. Puedes asumir que nunca habrán dos libros con la misma ganancia y la misma categoria.

En base a los archivos `info_libros.txt` e `info_ventas.txt` anteriores, el archivo `ganancias.txt` debería verse como:

```
ID;Categoría;Ganancia
3;Misterio;2500
1;Novela;2500
20;Novela;-21000
```

Notar que los libros con id `3` y `1` generaron la misma ganancia, pero como el libro `3` es de `Misterio` y el 1 es `Novela`, el `3` debe ir primero en el archivo `ganancias.txt`.

5. [P3a] DCC Parking

Enunciado

Con el boom en la venta de automóviles, el DCC ha decidido crear un sistema de gestión de estacionamientos, llamado *DCC Parking*. Como programador experto, te han pedido ayuda para desarrollarlo.

Parte A

En particular, deberás crear las siguientes clases:
Vehiculo: Esta clase tiene como atributos un string `patente`, un string `tipo`, y un float `peso`. Además, tiene un método __repr__ que retorna un string con el formato:

```
Vehiculo tipo: automóvil peso: 1000 kgs.
```

Se debe crear esta clase de tal manera de poder crear objetos de la siguiente manera:

```
vehiculo1 = Vehiculo("AAAA11","motocicleta",500)
vehiculo2 = Vehiculo("AABB12","automovil",1500)
```

Estacionamiento: Esta clase tiene como atributos un string `nombre`, un int `cantidad_maxima`, una lista `vehiculos` (de objetos de la clase Vehiculo) y un int `peso_max`. El constructor recibe como parámetro la información para los atributos nombre, cantidad_maxima y peso_max. Tiene además los métodos:

-
  - *agregar_vehiculo(vehiculo)*: Recibe como parámetro un objeto de la clase Vehiculo, y lo agrega a vehiculos solo si la cantidad de autos que tiene el estacionamiento es menor a `cantidad_maxima` y el peso del vehículo es menor al atributo `peso_max`. Retorna `True` si el vehiculo pudo ser agregado al estacionamiento y `False` en caso contrario. Puedes asumir que nunca se intentará agregar dos vehiculos con la misma patente.
  - *eliminar_vehiculo(patente)*: Recibe como parámetro un string con la patente de un auto. Si el estacionamiento tiene un vehiculo con esa patente, lo elimina de la lista vehiculos y retorna `True`. En caso contrario, retorna `False`.

6. [P3b] DCC Parking

Enunciado

Con el boom en la venta de automóviles, el DCC ha decidido crear un sistema de gestión de estacionamientos, llamado *DCC Parking*. Como programador experto, te han pedido ayuda para desarrollarlo.

Parte B

Para esta parte deberás utilizar las clases creadas en la parte A. Además deberás crear la clase **Sistema:** Esta clase tiene como atributo una lista `estacionamientos` de objetos de la clase Estacionamiento. Tiene los siguientes métodos:

- *agregar_estacionamiento(estacionamiento)*: método que recibe un objeto de la clase Estacionamiento y lo agrega a la lista estacionamientos.
- *estacionamientos_libres*: Retorna un int con la cantidad de plazas libres **en todos los estacionamientos**.
- *estacionamiento_mas_lleno*: Retorna un string con el nombre del estacionamiento con más vehículos. Si hay dos o más estacionamientos con una cantidad máxima de vehículos, se puede retornar cualquier nombre de esos estacionamientos.
- *cantidad_tipos*: método que revisa todos los objetos de la lista estacionamientos, y retorna una lista de listas con el tipo de vehículo, y la cantidad de vehiculos para ese tipo de vehículo (**en todos los estacionamientos**), ordenada por orden alfabético de los tipos de vehículo. Un ejemplo de retorno de esta función es el siguiente:

```
[
["automovil", 120],
["camion", 5],
["furgoneta", 60],
["motocicleta", 50]
]
```

Este retorno representa que hay 120 automoviles en todos los estacionamientos del sistema, 60 furgonetas, 5 camiones y por último 50 motocicletas.
Puedes asumir que solo habrá 6 tipos de vehículos: automovil, camion, furgoneta, motocicleta, motoneta y triciclo.