CUNEF
UNIVERSIDAD

Proyecto Final – Gestión de Pedidos Farmacéuticos

Contexto del problema

Nuestro cliente, una distribuidora farmacéutica recibe diariamente pedidos desde farmacias y hospitales. Nosotros, desde el departamento de bussiness intelligence, habremos de diseñar e implementar una solución en Python que cubra las necesidades del cliente, ayudándole a gestionar su almacén, la solicitud de nuevos pedidos a fábrica, y elaborando informes que ayuden a mejorar la gestión del sistema.

Objetivo General

El objetivo de este proyecto es desarrollar una aplicación de línea de comandos que permita gestionar pedidos farmacéuticos de manera interactiva. El proyecto se organiza en tres grandes bloques:

1. Interfaz de usuario por línea de comandos.
2. Gestión de una base de datos no estructurada usando listas, diccionarios y bucles.
3. Análisis de datos sobre una base estructurada usando Pandas.

Los alumnos trabajaréis con datos no estructurados (JSON) para convertirlos en estructuras procesables en Python (listas, diccionarios y pandas), aplicando bucles, condicionales y funciones para analizar y resumir información.

También aprenderéis a integrar la información de datos históricos de los clientes en resúmenes usando Pandas, matplotlib y numpy, y (opcionalmente) podréis entrenar modelos de predicción de parámetros clave sobre dichos datos.

Instrucciones Generales

El proyecto debe estructurarse en distintos archivos .py, que contendrán las funciones necesarias para realizar las distintas tareas. Es conveniente organizar las funciones siguiendo una estructura que responda a la lógica interna del programa. Por ejemplo, se pueden concentrar las funciones relativas a la interfaz de usuario en un mismo archivo, las funciones relativas al manejo del stock y los pedidos en otro, y las funciones de análisis de datos históricos en un tercero.

Recuerda usar la estructura `if __name__ == "__main__"` para la parte ejecutable del código, que debe encontrarse en un archivo llamado `main.py`.

El projecto debe entregarse en una carpeta comprimida en formato .zip, con el siguiente nombre:

```
proyecto_herramientas_programacion.zip
```

La carpeta debe contener todos los archivos .py del proyecto, además de otras carpetas con datos de entrada y salida (según se indica en las instrucciones). Mantén el código en la carpeta principal, y un solo nivel de profundidad en las carpetas con datos.

El proyecto puede desarrollarse en pareja o individualmente (se pueden considerar grupos más grandes con permiso expreso del profesor). Al finalizar el proyecto, todos los grupos deberán exponer su trabajo, y cada alumno debe demostrar individualmente una comprensión clara del proyecto para poder obtener una nota favorable.

Notas técnicas

- Debes utilizar diccionarios para representar la información en memoria.
- Los pedidos cargados desde JSON deben transformarse a estructuras Python.
- Para el análisis y resumen, se recomienda convertir las listas/diccionarios a DataFrames de pandas.
- Usa bucles y condicionales no triviales.
- Maneja posibles errores: archivos no encontrados, campos faltantes, etc. El programa no debe romper su flujo de ejecución por estos motivos.

Elementos a considerar

Almacén

Nuestro cliente dispone de un almacén, el cuál está organizado en varios módulos. Al tratarse de almacenamiento de fármacos, los módulos de almacenaje tienen algunas restricciones especiales, como el disponer de una capacidad de refrigeración a ciertos valores especiales de temperatura (-70ºC, -20ºC, 5ºC, ambiente).

Nuestra base de datos guardará la información sobre los módulos del almacén (en forma de diccionario), guardando los siguientes campos:

- Identificador del módulo (por ejemplo “MOD001”).
- Capacidad máxima (número entero).
- Temperatura (en Celsius, float).
- Stock: Listado de productos disponible en el almacén.
    - Cantidad de cada producto.

Pedidos

Cada pedido, recibido de uno de los clientes, contendrá la siguiente información sobre el cliente:

- Identificador del cliente.
- Tipo de cliente (“farmacia”, “centro médico”, “hospital”, “otro”).
- Fecha del pedido.
- Listado de productos.
    - Cantidad de cada producto.

Los productos en sí se describen en el siguiente apartado.

Productos

Los productos ofrecidos por nuestro cliente están definidos de antemano en una base de datos (que en nuestro caso representaremos como un diccionario). Los productos tienen los siguientes campos identificativos:

- Código del producto (por ejemplo: “MED001”).
- Nombre del producto (por ejemplo: “ibuprofeno”).
- Precio unitario en euros.
- Categoría (por ejemplo: “antiinflamatorio”).

Cada pedido llega en formato JSON, que contiene información sobre el cliente y los productos solicitados. La empresa necesita una herramienta para procesar estos pedidos, integrarlos en un sistema interno y obtener métricas clave para tomar decisiones.

Parte 1 del Proyecto: Interfaz de línea de comandos

En primer lugar, debes escribir una función de Python, que será la función principal del programa. Cuando se llame al script main.py, este debe llamar la función que gestiona la interfaz.

La interfaz ofrecerá al usuario una serie de opciones básicas, las cuales, al ser seleccionadas, darán lugar a otro menú en el que se le ofrecerán nuevas opciones relacionadas con la inicial.

El menú seguirá en funcionamiento en tanto el usuario no seleccione la opción Salir del programa (5).

Estructura de la interfaz:

```
Main menu

1 - Información general.

2 - Estado del Almacén.
      1 - Mostrar los módulos del almacén.
      2 - Mostrar el estado de un módulo.
          [Input: ¿Qué modulo deseas ver?]
           1 - Display molecular structure
           [Prompt: Which molecule do you want to display?]
      B - Vuelve al menú anterior.
3 - Pedidos.
      1 - Mostrar los pedidos sin procesar.
      2 - Procesar pedido.
          [Input: Por favor, indica la localización del pedido].
      3 - Mostrar los pedidos en marcha.
      B - Vuelve al menú anterior.
4 - Informes históricos.
      [Mostrar "en construcción"].
      B - Vuelve al menú anterior.
5 - Salir del programa.
```

La sección 4 de la interfaz se completará cuando se publique la segunda parte del proyecto, con las opciones que correspondan a este caso.

Parte 2 del Proyecto: Gestor de almacén y pedidos

En esta sección del proyecto, tendremos que diseñar el código interno que nos permitirá realizar dos tareas fundamentales:

- Gestionar los pedidos entrantes
  - Verificaremos si hay producto en stock.
  - Efectuaremos pedidos a fábrica cuando sean necesarios.
- Gestionar el almacén de nuestro cliente
  - Actualizaremos el stock.
  - Consideraremos los pedidos a fábrica con el fin de minimizarlos.

Para atender los pedidos de sus clientes (farmacias y hospitales), nuestro cliente dispone de ciertos fármacos en stock, guardados en los distintos módulos de su almacén. Cuando recibe un nuevo pedido, lo primero que debe hacer es comprobar si el pedido está en el almacén, en cuyo caso se atenderá directamente. Si no lo está, habrá que solicitarlo a fábrica.

Manejo de la base de datos

La base de datos del almacén, que contiene la información sobre los diferentes módulos y su stock, se almacenará en un archivo en formato JSON.

Al inicio del programa, la base de datos debe cargarse en memoria (esto es, leerse del archivo JSON y guardarse en forma de lista de diccionarios), quedando disponible y pasándose como argumento a aquellas funciones que la necesiten.

Al finalizar el programa, la base de datos debe actualizarse, escribiendo cualquier cambio en el estado del almacén de vuelta en el archivo JSON.

NOTA: Esta base de datos no debe funcionar como variable global, sino pasarse como argumento.

Nuevos pedidos

De cara a gestionar los nuevos pedidos, debe ser el usuario (un empleado de nuestro cliente, que usa la interfaz definida en el apartado 1) el que cargue el nuevo pedido. Para ello, debe indicarle al programa el nombre del archivo que contiene el pedido, el cual se guardará en una carpeta `Pedidos` dentro de nuestro proyecto. El archivo en cuestión tendrá formato JSON (ver `pedidos_ejemplo.json`).

Cuando se le indica un nuevo pedido, el programa debe realizar las siguientes acciones:

1. Cargar el archivo (si no existe, avisa del problema sin que el programa acabe).
   a. Comprueba que el pedido no haya sido procesado, y avisa en caso contrario.
2. Leer el pedido, verificando si los productos solicitados están en stock.
3. Planificar el envío, generando un nuevo diccionario con los datos de dicho envío.
   a. El envío puede partirse en dos si fuese necesario obtener algunos productos del almacén.
   b. Se debe indicar la fecha del envío, que será tres días después del pedido si este se encuentra en stock, o diez días después si hay que solicitarlo al almacén.
   c. El envío (o envíos, si son varios) se debe escribir en un archivo JSON, que se guardará en la carpeta Envios, la cual debe estar dentro del proyecto.
   d. Una vez se ha procesado el pedido, se debe anotar su ID en una lista de pedidos procesados, para poder mostrarlo como tal.

4. Actualizar el stock:
    a. Restando los productos que se hayan enviado del almacén.
        i. Muestra el módulo del que se obtiene y el estado del mismo tras el cambio.
    b. Añadiendo los productos que se hayan obtenido de fábrica.
        i. Muestra los módulos que han recibido nuevo stock y su estado tras el cambio.
        ii. Cuidado con el stock máximo del módulo en cuestión.

Indicaciones: Organiza las tareas en funciones pequeñas que sean independientes entre sí (por ejemplo, si se hace un pedido a fábrica, una función puede ocuparse de sumar los nuevos elementos al stock, mientras que otra función se ocuparía de restar los elementos del stock que van en el pedido; comprobar si hay o no hay stock sería una tercera función).

NOTA: Una aplicación de este tipo, en el mundo real, tendría que lidiar con el problema de pedidos asíncronos (esto es, pedidos que llegan al mismo tiempo y que pueden estar "atacando" a la misma parte del stock). En nuestro caso, sin embargo, no nos preocuparemos por este detalle.

Predicción de la demanda

Dado que las solicitudes a fábrica tienen un cierto coste (en forma de transporte), es conveniente que nuestro sistema de pedidos a fábrica aproveche al máximo cada solicitud, aprovechando para rellenar el stock disponible no solo para cubrir la actual demanda, sino para aumentar la disponibilidad de cara a atender futuros pedidos. Sin embargo, debemos mantener un equilibrio a la hora de realizar las solicitudes, ya que la capacidad de almacenaje de que disponemos es limitada, y además los productos caducan, y no queremos que lo hagan en el almacén de nuestro cliente (ya que esto supondría un coste adicional).

Esta cuestión es en realidad un problema abierto, llamado predicción de la demanda, el cual se maneja en muchos contextos empresariales con la ayuda de herramientas de machine learning.

De cara a este proyecto, no obstante, podéis trabajar con alguna asunción más sencilla de cara a rellenar el stock del almacén cuando se recibe un pedido.

Elegid vuestra estrategia para rellenar el almacén cuando el stock es insuficiente. Deberéis explicar esta durante la defensa del proyecto.

(Opcional) Control del almacenaje del fármaco según sus características

Cada fármaco tiene unas necesidades de almacenaje concretas. Para gestionar esto, podemos usar un diccionario que almacene cada fármaco con sus características (usando los códigos como claves), incluyendo la temperatura a la que debe conservarse.

De este modo, cuando se solicita un nuevo fármaco a fábrica, no basta con almacenarlo en cualquier módulo que disponga de stock suficiente, sino que deberemos, además, de asegurarnos de tener espacio disponible en el módulo o módulos que tengan la temperatura de refrigeración adecuada.

Usando el archivo `farmacos.json`, ajusta la funcionalidad de solicitud de nuevos fármacos para tener en cuenta esta restricción. Debes permitir que un mismo fármaco sea repartido entre varios módulos de almacenaje, si uno de ellos no es suficiente para guardarlo.