Ejercicio Final Integrador: Gestión Jerárquica de Datos (Sistema de Archivos y Recursividad)

Modalidad: Trabajo en equipos de 2 o 3 alumnos.

Objetivo General: Desarrollar una aplicación en Python 3.x que implemente un sistema de persistencia y consulta de datos utilizando una estructura de directorios jerárquica. El proyecto debe aplicar conocimientos de diseño de estructuras de datos, manipulación avanzada de archivos CSV, recursividad para la lectura del sistema de archivos, y funciones de la librería estándar os para gestionar la estructura de carpetas.

Fase 1: Diseño y Documentación del Modelo de Datos (Investigación)

El primer paso es definir la temática y la estructura de datos jerárquica que el equipo utilizará.

1. Definición del Dominio y Estructura:

◦ El equipo debe **escoger un modelo de datos** (por ejemplo, Países, Productos tecnológicos, Universidades, etc.).

◦ Deben definir al menos **tres niveles de jerarquía/filtrado** que se mapearán directamente a una estructura de carpetas.

▪ *Ejemplo:* Países -> Continente / Región / Tipo de Gobierno.

◦ Cada nivel de la jerarquía debe tener un archivo CSV al final donde se almacenarán los ítems individuales con sus atributos.

◦ Los ítems individuales (ej: Argentina, un celular, una universidad) deberán estar representados internamente en Python utilizando **diccionarios**.

2. Documentación del Diseño (Entregables Específicos):

◦ **Video Explicativo (Máximo 8 minutos):** El equipo debe grabar un video explicando el dominio elegido, el **diseño de la estructura de datos investigada** y cómo se mapea esta estructura a la jerarquía de carpetas. El video debe demostrar el entendimiento del problema de persistencia jerárquica.

◦ **README.md:** El repositorio en GitHub debe contener un README.md que incluya un **resumen conciso de la estructura de datos y la lógica de filtrado/almacenamiento** basada en directorios, además de las instrucciones de uso.

Fase 2: Implementación Técnica Centralizada

La aplicación debe ser construida siguiendo buenas prácticas de codificación (ej. usando 4 espacios para indentación, y modularizando mediante funciones).

1. Manipulación de Archivos y Directorios (os y CSV):

◦ La aplicación debe usar la librería os de Python para:

▪ Verificar la existencia de directorios o archivos.

▪ Crear la estructura de carpetas necesaria de forma dinámica al intentar agregar un nuevo ítem (siguiendo los niveles jerárquicos definidos).

▪ Construir rutas de archivos y directorios de manera segura.

◦ La persistencia de datos debe ser gestionada mediante archivos CSV, utilizando el manejo de archivos con la cláusula with para garantizar el cierre automático.

2. Lectura Recursiva del Sistema de Archivos (Obligatorio):

◦ Se debe implementar una función utilizando **recursividad** para recorrer la estructura de carpetas completa.

◦ Esta función recursiva debe:

▪ Recibir la ruta actual del directorio.

▪ Definir un **caso base** (condición de corte) claro (ej: si es un archivo CSV, leer el contenido).

▪ Definir un **paso recursivo** (ej: si es un directorio, llamar a la función para cada subdirectorio/archivo).

◦ El objetivo es que esta función recoja **todos los ítems** almacenados en todos los archivos CSV de la jerarquía en una única estructura de datos (e.g., una lista de diccionarios) para su posterior procesamiento (consultas, estadísticas, ordenamientos).

3. Manejo de Excepciones:

◦ Se debe usar try y except para controlar posibles errores durante la lectura o escritura de archivos (ej: FileNotFoundError, OSError, o errores de formato de datos).

Fase 3: Funcionalidades Mínimas del Sistema (CRUD y Consultas)

El programa debe presentar un menú que permita realizar las siguientes operaciones sobre el conjunto total de ítems recopilados mediante la recursividad. Todas las funciones deben **utilizar el patrón de datos (diccionarios)** establecido en la Fase 1 y deben incluir **validaciones estrictas**.

1. Alta de Nuevo Ítem (Creación Jerárquica)

Esta función corresponde a la operación de Inserción (Create) y debe ser la única que crea nuevas carpetas si es necesario.

• **Entrada de Datos:** Pedir al usuario que ingrese los valores para los **3 niveles de jerarquía** definidos por el equipo (ej: Continente, Región, División). Además, solicitar los atributos del ítem individual (ej: Nombre, Población, Superficie).

• **Validaciones Estrictas:**

◦ Validar que los datos de entrada no estén vacíos.

◦ Validar el **tipo de dato** de cada atributo (ej: asegurar que la población y la superficie sean numéricos enteros o flotantes).

◦ Validar la **lógica de negocio** (ej: que los valores numéricos importantes sean positivos y mayores a cero).

• **Persistencia:** Utilizar la librería os para **verificar y crear la estructura de directorios** correspondiente a la jerarquía ingresada. Una vez construida la ruta, **agregar** el nuevo ítem (en formato CSV) al archivo correspondiente (usando modo 'a').

2. Mostrar Ítems Totales y Filtrado (Lectura Global)

• **Lectura Centralizada:** Invocar la **función recursiva obligatoria** (definida en la Fase 2) para leer y consolidar **todos los ítems** almacenados en *todos* los archivos CSV de la jerarquía en una única lista de diccionarios.

• **Mostrar:** Presentar una lista clara de todos los ítems registrados, mostrando también su ubicación jerárquica para demostrar el éxito de la lectura recursiva.

• **Filtrado:** Permitir al usuario filtrar esta lista global por al menos uno de los atributos definidos en la Fase 1.

3. Modificación de Ítem (Actualización/Update)

Esta función corresponde a la operación de Actualización (Update).

• **Identificación:** El sistema debe permitir al usuario **identificar un ítem único** para modificar (ej: por ID único o por la combinación completa de sus atributos jerárquicos y su nombre).

• **Proceso:**

1. Buscar el ítem en la lista global obtenida recursivamente.

2. Si se encuentra, solicitar al usuario qué atributo desea modificar y el nuevo valor.

3. Aplicar **validaciones estrictas** al nuevo valor ingresado (tipo y lógica de negocio, similar a la función de Alta).

4. Una vez modificado el ítem en la memoria, el sistema debe **sobrescribir** únicamente el archivo CSV que contiene ese ítem para reflejar el cambio (usando modo 'w').

4. Eliminación de Ítem (Baja/Delete)

Esta función corresponde a la operación de Eliminación (Delete).

• **Identificación:** Permitir al usuario especificar el ítem a eliminar (ej: por ID único o por nombre exacto).

• **Proceso:**

1. Buscar y remover el ítem de la estructura de datos en memoria.

2. Si la eliminación es exitosa, el sistema debe **sobrescribir** el archivo CSV específico (el archivo de "hoja" donde residía el ítem) con la lista actualizada de ítems para ese archivo, garantizando la persistencia del cambio.

◦ **Nota de Robustez:** Es necesario usar manejo de excepciones (try/except) para controlar escenarios donde el ítem a eliminar no es encontrado o hay problemas de escritura.

5. Funcionalidades Adicionales (Estadísticas y Ordenamiento)

• **Ordenamiento Global:** Permitir ordenar la lista completa de ítems (obtenida recursivamente) por al menos dos atributos diferentes.

• **Estadísticas Básicas:** Calcular y mostrar estadísticas basadas en la totalidad de los datos recogidos, incluyendo la cantidad total, un promedio o suma de un atributo numérico clave, y un recuento de ítems por categoría de primer nivel.

Entregables (Obligatorios)

1. Repositorio en GitHub:

◦ Código Python funcional, modularizado y comentado.

◦ Archivo CSV inicial (opcional, si se proveen datos base).

◦ **README.md** detallado, incluyendo el resumen del diseño de la estructura de datos.

2. Video Tutorial/Presentación:

◦ Cumpliendo con las especificaciones de la Fase 1.

Criterios de Evaluación

La calificación estará fuertemente basada en los nuevos requerimientos implementados:

• **Recursividad (30%):** Correcta implementación de una función recursiva con caso base y paso recursivo claro, que recorra la jerarquía de carpetas y consolide los datos.

• **Diseño y Uso del Sistema de Archivos (30%):** La lógica de escribir datos usando la librería os y mapeando los filtros a la creación/navegación de directorios debe ser precisa. La **estructura de datos elegida debe ser coherente**.

• **Funcionalidades (25%):** Correcta implementación de la adición de ítems (escritura jerárquica), ordenamiento y estadísticas globales.

• **Documentación (15%):** Calidad del código (legibilidad, modularización), y cumplimiento de la entrega del vídeo y el README.md.