Análisis de Caso

Estructuras de Dato en Python y Sentencias Iterativas👩‍💻

Análisis de Caso

Estructuras de Dato en Python y Sentencias Iterativas

Instrucciones generales Referente:

El análisis de casos tiene como objetivo permitir a los participantes aplicar de manera práctica los conceptos y habilidades aprendidos en la lección a una situación realista o simulada. Se espera que el caso propuesto presente un desafío concreto, donde los participantes deban analizar, tomar decisiones y proponer soluciones efectivas basadas en su conocimiento.

Situación inicial 📍

Una empresa de tecnología llamada **DataSolvers** se especializa en la gestión de grandes volúmenes de datos para diversos clientes, ayudándoles a analizar y organizar su información de manera eficiente. Actualmente, DataSolvers está desarrollando un sistema para analizar y categorizar datos financieros provenientes de diferentes fuentes.

En su fase actual, el equipo de desarrollo necesita implementar estructuras de datos eficientes para almacenar y procesar esta información. Debido a la complejidad del proyecto, se enfrentan a un desafío: optimizar el almacenamiento y acceso a los datos mientras utilizan sentencias iterativas para analizar grandes conjuntos de información. Se espera que el equipo utilice listas, diccionarios y conjuntos en Python para manejar la carga de datos de forma efectiva.

Descripción del Caso 🔎

Eres un desarrollador de **DataSolvers** encargado de liderar la optimización de las estructuras de datos para el sistema de análisis financiero. Tu misión es diseñar una serie de funciones que utilicen sentencias iterativas y estructuras de datos adecuadas para procesar y analizar los datos financieros, garantizando eficiencia y precisión en los resultados. Tendrás que resolver los siguientes desafíos basándote en los conceptos estudiados.

Instrucciones 💡

Analiza el siguiente código base en Python:

```
class AnalizadorFinanciero:

        # Calcula el total de ingresos en una lista de transacciones

    def calcular_total_ingresos(self, transacciones):

        total = 0

        for ingreso in transacciones:

            total += ingreso

        return total


     # Filtra y retorna solo los ingresos mayores a un umbral dado

    def filtrar_ingresos_altos(self, transacciones, umbral):

        ingresos_altos = []

        for ingreso in transacciones:

            if ingreso > umbral:

                ingresos_altos.append(ingreso)

        return ingresos_altos


    # Agrupa ingresos en un diccionario por categorías

            def agrupar_por_categoria(self, transacciones, categorias):

        agrupado = {}

                for categoria, ingreso in zip(categorias, transacciones):

            if categoria in agrupado:
```

```
            agrupado[categoria].append(ingreso)
        else:
            agrupado[categoria] = [ingreso]
    return agrupado
```

A partir del mismo, realiza lo siguiente:

1. **Análisis de la Estructura de Datos**
    - Analiza cómo las estructuras de datos en Python (listas y diccionarios) ayudan a organizar y manipular los datos en el código de ejemplo. Describe las principales ventajas de usar listas para almacenar transacciones y diccionarios para categorizar ingresos.
    - Identifica posibles limitaciones o mejoras al utilizar estas estructuras en el proyecto de DataSolvers.
2. **Optimización de Sentencias Iterativas**
    - Explica cómo se pueden optimizar las sentencias iterativas en las funciones `calcular_total_ingresos()` y `filtrar_ingresos_altos()` para mejorar la eficiencia.
    - Describe cómo podrías usar expresiones generadoras o comprensión de listas para optimizar estos procesos.
3. **Implementación de Pruebas para las Funciones**
    - Crea una serie de pruebas para validar que las funciones `calcular_total_ingresos()`, `filtrar_ingresos_altos()`, y `agrupar_por_categoria()` funcionan correctamente con diferentes conjuntos de datos.
    - Implementa una estructura de prueba que incluya datos de prueba representativos y explica cómo verificar los resultados.
4. **Aplicación de Estructuras de Datos Avanzadas**
    - Investiga cómo podría utilizarse un conjunto (`set`) en el código para optimizar la verificación de categorías únicas en el sistema financiero.
    - Describe los beneficios de utilizar conjuntos en Python para tareas de deduplicación y acceso rápido en comparación con listas.
5. **Refactorización del Código**
    - Utilizando el análisis previo, refactoriza el código base para incorporar las mejoras propuestas. Implementa comprensión de listas y optimiza el uso de estructuras de datos para que el sistema sea más eficiente.

Entregables 📬

Un reporte que incluya:

- Análisis de las ventajas y limitaciones de las estructuras de datos utilizadas en el código.
- Código de las funciones optimizadas, incorporando comprensión de listas y estructuras avanzadas como conjuntos.
- Descripción detallada de las pruebas implementadas para cada función y los resultados obtenidos.
- Ejemplos de casos de prueba, con los datos de entrada y los resultados esperados.
- Reflexión sobre la experiencia de aplicar optimización de estructuras de datos y sentencias iterativas en este caso.

Archivos con el código fuente de las funciones optimizadas y los casos de prueba implementados en Python.

¡Muchas gracias!

Nos vemos en la próxima lección