Modulo 2 Fundamentos de programación Python para ingenieros de datos

Actividad: Análisis de caso Estructuras de Dato en Python y Sentencias Iterativas

Karen Tapia Landa

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.

---

1. Análisis de la Estructura de Datos

Uso de listas:

- Aplicación en el código: Las transacciones se almacenan en listas (transacciones, categorías).
- Ventajas:
    - Acceso por índice.
    - Permiten duplicados.
    - Ideales para operaciones secuenciales (suma, filtrado, etc.).
- Limitaciones:
    - Verificar si un elemento existe requiere búsqueda secuencial.
    - Poca eficiencia para duplicación o búsquedas frecuentes.

Uso de diccionarios:

- Aplicación: En agrupar_por_categoria, los ingresos se agrupan por clave (categoría).
- Ventajas:
    - Acceso rápido a datos por clave.
    - Claridad al organizar información jerárquica (categoría => ingresos).
- Limitaciones:
    - No permite claves duplicadas.

En proyectos como DataSolvers:

- Listas: Buenas para flujos simples de datos.
- Diccionarios: Eficientes para estructurar por categoría.
- Mejorar: Usar sets para verificación de unicidad y evitar operaciones costosas.

---

2. Optimización de Sentencias Iterativas

Función calcular_total_ingresos

Antes:

```
total = 0
for ingreso in transacciones:
   total += ingreso
```

Optimizado con expresión generadora:

```
return sum(transacciones)
```

Función filtrar_ingresos_altos

Antes:

```
ingresos_altos = []

for ingreso in transacciones:
   if ingreso > umbral:
      ingresos_altos.append(ingreso)
```

Optimizado con list comprehension:

```
return [ing for ing in transacciones if ing > umbral]
```

---

3. Implementación de Pruebas

Datos de prueba:

```
transacciones = [1000, 500, 1200, 700, 300]
categorias = ['sueldo', 'freelance', 'sueldo', 'inversiones', 'freelance']
```

Pruebas:

```
af = AnalizadorFinanciero()

assert af.calcular_total_ingresos(transacciones) == 3700
assert af.filtrar_ingresos_altos(transacciones, 800) == [1000, 1200]
```

```
assert af.agrupar_por_categoria(transacciones, categorias) == {
   'sueldo': [1000, 1200],
   'freelance': [500, 300],
   'inversiones': [700]
}
```

- Resultado esperado: Las funciones deben devolver los valores esperados sin error.

Conclusión

- Aplicar comprensión de listas y sets hace el código más conciso y eficiente.
- Se mejora la legibilidad, reduciendo líneas sin perder claridad.
- Las estructuras como dict y set son clave para trabajar con datos clasificados o deduplicados, es decir Identificar y eliminar registros repetidos para conservar solo una versión única
- Las optimizaciones no solo mejoran velocidad, también hacen el código más mantenible.