Optimización de memoria en Python

1. Procesamiento por partes
    a. Para manejar grandes volúmenes de datos, es fundamental no cargar todo en memoria a mismo tiempo, esto se puede dividir en (**chunks**), lo que ocasiona es que en fragmentos mas pequeños estos consumen menos memoria
    b. Procesar cada chunk por separado, liberando la memoria después de cada procesamiento
2. Reducción del tamaño de los datos
    a. Cambiar el tipo de los datos de columnas o variables a versiones más eficientes en donde por ejemplo se pase a utilizar un float32 en vez de un float64
    b. Uso de numpy en vez de listas, dado que los arrays son mas eficientes para datos numéricos.
3. Gestión de memoria
    a. Liberar recursos cuando ya no se necesiten (del)
    b. Evitar copias innecesarias de datos grandes

Interacción con APIs

1. Autenticación
    - Es necesario autenticar las solicitudes a las APIs mediante claves de acceso. Esto garantiza que las acciones sean realizadas por usuarios autorizados
    - AWS utiliza Access keys y secret keys
    - GCP utiliza service accounts y archivos json de credenciales
    - Ambos sistemas permiten autenticación basada en roles (**IAM**)
2. Operación con APIs
    - Se puede realizar diversas operaciones como crear recursos tipo buckets de almacenamiento, cargar o descargar archivos y consultar información sobre recursos existentes
3. Manejo de Respuestas
    - Las Apis devuelven respuestas estructuradas (JSON u otros formatos) que contienen información relevante, es importante saber interpretar estas respuestas para tomar decisiones o registrar resultados.
4. Prácticas de seguridad
    - Nunca incluir credenciales en el código
5. Gestión de errores
    - Las interacciones con APIs pueden fallar debido a distintos factores, como lo son autenticaciones, problemas de conexión, permisos o errores en parámetros es importante implementar manejo de excepciones dado que esto permite identificar y resolver estos problemas de manera eficiente.
    - Procesar adecuadamente los códigos de estado