Evaluación Grupal

Integrantes: Mariel Gajardo

1. **¿Qué es una excepción en programación y por qué es importante manejarla correctamente?**

R: Una excepción en programación es un evento que interrumpe el flujo normal de ejecución de un programa cuando ocurre un error o una situación inesperada.

2. **¿Cuáles son los tipos de excepciones más comunes?**

R: Las excepciones más comunes son ZeroDivisionError (cuando el código está haciendo una división y por ingreso manual o por otro error, se hace una división por 0) y TypeError, cuando el código está destinado para hacer una operación y se le entregan tipos de datos que son incompatibles entre sí, o ValueError, que sucede cuando el código está esperando un tipo específico de dato (número, texto o booleano) y recibe uno inadecuado.

3. **¿Cómo funciona la sentencia try/except y cuándo se debe utilizar?**

R: La sentencia try/except funciona poniendo el código que puede fallar dentro del bloque try, y si ocurre una excepción, el programa ejecuta el código del bloque except en lugar de crashear. Se debe utilizar cuando sabemos que una operación puede generar errores, como cuando pedimos input del usuario, abrimos archivos, o hacemos cálculos que pueden dar división por cero.

4. **¿Cómo se pueden capturar múltiples excepciones en un solo bloque de código?**

R: Se pueden capturar de varias formas: con una tupla except (ValueError, TypeError): cuando queremos el mismo manejo para varios tipos, con múltiples bloques except separados cuando cada tipo necesita diferente tratamiento, o con except Exception para capturar cualquier excepción y usar la variable e para obtener detalles del error.

5. **¿Qué es el uso de raise en Python y cómo se utiliza para generar excepciones en validaciones?**

R: raise sirve para lanzar excepciones de forma intencional en nuestro código. En validaciones se usa para detener la ejecución cuando los datos no son válidos, como raise ValueError("El número debe ser positivo") cuando validamos entrada de usuario. También se puede usar raise sin argumentos dentro de un except para volver a lanzar la misma excepción capturada.

6. **¿Cómo se pueden definir excepciones personalizadas y en qué casos sería útil?**

R: Se crean definiendo una clase que herede de Exception: class MiError(Exception): pass. Son útiles cuando necesitamos excepciones específicas para nuestra aplicación, como class UsuarioNoEncontradoError(Exception): en un sistema de usuarios, o class ProductoAgotadoError(Exception): en un e-commerce. Esto hace el código más claro y permite manejar errores específicos del dominio.

7. **¿Cuál es la función de finally en el manejo de excepciones?**

R: El bloque finally se ejecuta SIEMPRE, pase lo que pase en try/except. Su función es garantizar que ciertas operaciones de limpieza se ejecuten sin importar si hubo errores o no. Es como una "red de seguridad" para asegurar que recursos se liberen correctamente, archivos se cierren, o configuraciones se restauren.

8. **¿Cuáles son algunas acciones de limpieza que deben ejecutarse después de un proceso que puede generar errores?**

R: Las más importantes son: cerrar archivos abiertos (con file.close() o mejor usar with), cerrar conexiones de base de datos para no agotar el pool de conexiones, liberar locks o semáforos que se tomaron, limpiar archivos temporales creados durante el proceso, restaurar variables globales o configuraciones que se modificaron, y registrar en logs el estado final del proceso para debugging.