Guía de Ejercicios Manejo de Excepciones

Escribe un programa que solicite al usuario ingresar un número entero. El programa debe intentar convertir la entrada a entero usando `try-except`. Si la conversión falla (por ejemplo, si el usuario escribe "hola"), debe capturar la excepción y mostrar el mensaje: **"Error: Debes ingresar un número entero válido"**. Si la conversión es exitosa, mostrará el número ingresado.

Crea un programa que evalúe expresiones matemáticas simples proporcionadas por el usuario. El programa hará lo siguiente:

1. Solicitará al usuario que ingrese **dos números** (primero y segundo valor) y una **operación** (suma "+", resta "-", multiplicación "*" o división "/").
2. Intentará realizar la operación en un bloque `try`.
3. Manejará **excepciones específicas**:
   - Si ocurre una **división entre cero** (`ZeroDivisionError`), mostrar: `"Error: No se puede dividir entre cero"`.
   - Si el usuario ingresa un **nombre de variable no definido** en la expresión (simulado provocando un `NameError`), mostrar: `"Error: Variable no definida en la operación"`.
   - Si los tipos de datos son incompatibles para la operación (por ejemplo, intentar sumar un número con un texto, provocando un `TypeError`), mostrar: `"Error: Tipos de datos incompatibles"`.
4. Si todo es correcto, mostrará el resultado con el formato:
   `"El resultado de {num1} {operacion} {num2} = {resultado}"`.

Crea un programa que pida al usuario ingresar una **edad** (número entero). Usa `try-except` para manejar errores, pero además implementa lo siguiente:

1. Intenta convertir la entrada a entero. Si falla (por ejemplo, el usuario escribe "veinte"), captura la excepción y muestra: `"Error: Debes ingresar un número entero"`.
2. Si la conversión es exitosa, usa `raise` para lanzar manualmente una excepción si:
   - La edad es menor a 0 → lanzar `ValueError` con el mensaje: `"La edad no puede ser negativa"`.
   - La edad es mayor a 120 → lanzar `ValueError` con el mensaje: `"La edad no puede ser mayor a 120 años"`.
3. Captura estos `ValueError` lanzados con `raise` y muestra el mensaje correspondiente.
4. Si la edad es válida, muestra: `"Edad registrada: {edad} años"`.

Enunciado del ejercicio:

Crea un programa que simule un **sistema básico de registro de estudiantes** para una biblioteca, con las siguientes funcionalidades:

1. **Diccionario inicial**: Tendrás un diccionario predefinido de libros disponibles con su cantidad en stock (ej: `{"Python": 5, "Matemáticas": 2, "Historia": 0}`).

2. **Funciones con manejo de excepciones y raise**:
   - **Prestar libro**: Pide al usuario el nombre de un libro y reduce su stock en 1.
     - Si el libro no existe en el diccionario → lanzar (`raise`) una excepción personalizada `LibroNoEncontradoError` (debes crearla heredando de `Exception`).
     - Si el stock es 0 → lanzar (`raise`) una excepción `StockAgotadoError` (también personalizada).
   - **Devolver libro**: Pide el nombre de un libro y aumenta su stock en 1.
     - Si el libro no existe → lanzar `LibroNoEncontradoError`.
   - **Consultar stock**: Muestra el stock actual de un libro solicitado.
     - Si el libro no existe → lanzar `LibroNoEncontradoError`.

3. **Interfaz principal con `try-except`**:
   - Muestra un menú con opciones: (1) Prestar, (2) Devolver, (3) Consultar stock, (4) Salir.
   - Cada opción debe ejecutarse dentro de un bloque `try-except` que capture las excepciones personalizadas y también `ValueError`/`KeyError` genéricos.
   - Las excepciones personalizadas deben mostrar mensajes específicos; las genéricas un mensaje de error inesperado.

4. **Requisito adicional**: Valida que las opciones del menú sean números (1-4) usando `try-except` para `ValueError`.

El ejercicio integra: manejo de múltiples excepciones, creación de excepciones personalizadas con `raise`, y validación de entrada de usuario en diferentes niveles.