4A. EJERCICIO: VALIDACIÓN DE DATOS.

Unidad 4: Manejo de errores y depuración.

Datos
Nombre: Keith Alexis Gutiérrez Ibarra
Docente: Adolfo Aldair Duque Borja
Fecha de elaboración: 01/12/2025

Introducción

Una parte fundamenta en el desarrollo de aplicaciones es asegurar de que los datos introducidos por los usuarios sean correctos y seguros, cuando un programa no valida adecuadamente la información es común que aparezcan errores o fallo durante su ejecución afectando el funcionamiento, por esta razón contar con un sistema de validación ayuda a evita problemas y facilita su detección mediante el uso de excepciones y mensajes de depuración.

En esta actividad se construirá un registro de usuarios con los campos de nombre, edad y correo electrónico, para ello se aplicarán técnicas de validación y manejo de errores que permiten identificar y controlar situaciones en las que los datos no cumplen con las condiciones necesarias, además se integrarán mensajes de depuración que ayudarán a encontrar posibles fallas durante el proceso de desarrollo, fortaleciendo la confiabilidad del programa asegurando que cada dato ingresado sea verificado correctamente y que cualquier error sea manejado de forma clara y controlada.

Desarrollo

Ejercicio 1

a) Análisis del problema: Para este ejercicio se nos piden las siguientes condiciones que debe tener el Código.

- Implementar una función llamada validar_datos(nombre, edad, correo) que verifique que *nombre* sea una cadena de texto no este vacía, la *edad* sea un número entero mayor que cero y que el *correo* contenga el símbolo @.
- Usar manejo de excepciones (try, except, finally) para capturar los diferentes tipos de errores como TypeError, ValueError, entre otros que puedan surgir por datos incorrectos, informar si los datos fueron registrados correctamente o si ocurrió alguna falla durante el proceso de validación y asegurarse que el mensaje final se ejecute mediante la cláusula finally sin importar si hubo o no error.
- Incluir al menos 3 excepciones distintas como ValueError, TypeError o ZeroDivisionError.

- Crear una función de prueba llamada probar_validaciones() cuya finalidad es ejecutar validar_datos con distintos datos de entrada de prueba de los diferentes casos posibles.
- Agregar mensajes de depuración usando print() o el módulo logging, con el fin de identificar qué parte del proceso se está ejecutando y en qué punto ocurre un error durante el desarrollo.

b) Diseño del algoritmo

Se define la función principal de validación llamada validar_datos, encargada de recibir nombre, edad y correo como parámetros.

a) Se imprimen los datos que se encuentran en proceso de validación.
b) Se inicia un bloque try donde se realizan todas las validaciones necesarias sobre los datos recibidos.
c) Se verifica que los tipos de datos sean correctos, comprobando que el nombre sea una cadena de texto y que la edad sea un entero, en caso contrario se lanza la excepción TypeError.
d) Se valida que la edad sea un número mayor a cero y si no cumple esta condición se genera la excepción ValueError.
e) Se comprueba que el nombre no esté vacío y que el correo electrónico contenga el símbolo @, en caso contrario se genera una excepción ValueError.
f) Se incluye un ejemplo de manejo de errores con ZeroDivisionError, al realizar una operación de división utilizando la edad como divisor
g) Dentro del bloque except, se capturan específicamente tres tipos de excepciones TypeError, ValueError, ZeroDivisionErrorr y se incluye un except general para cualquier error inesperado.
h) El bloque finally siempre muestre un mensaje final, indicando que la validación ha terminado

Se crea la función probar_validacion, cuya finalidad es ejecutar la función validar_datos con diferentes entradas.

a) Se incluyen varios casos de prueba diseñados para generar errores y mostrar los diferentes tipos de errores.

Se realiza una llamada a la función probar_validacion, ejecutándose así todas las validaciones definidas y permitiendo observar el manejo de errores en diferentes escenarios.

c) Código en Python

Validación de datos.py

```python
# Funcion para validar datos de entrada
def validar_datos(nombre, edad, correo):
    # Imprimir los datos recibidos
    print(f"\nValidando datos para: {nombre}, {edad}, {correo}")
    # Bloque try-except para manejar excepciones
    try:
        # Validación de tipos de datos
        if not isinstance(nombre, str) or not isinstance(edad, int):
            raise TypeError("Tipo de dato incorrecto.")
        # Validación de edad
        if edad < 0:
            raise ValueError("La edad debe ser mayor a cero.")
        # Validación de nombre vacío
        if len(nombre.strip()) == 0:
            raise ValueError("El nombre no puede estar vacío.")
        # Validación simple del correo electrónico
        if "@" not in correo:
            raise ValueError("El correo electrónico no es válido.")
        # Muestra de error zero division
        división = 100 / edad
        # Mensaje de éxito
        print("Datos válidos")
    # Captura de excepciones de TypeError
    except TypeError as e:
        print(f"Error de tipo: {e}")
    # Captura de excepciones de ValueError
    except ValueError as e:
        print(f"Error de validación: {e}")
    # Captura de excepciones de ZeroDivisionError
    except ZeroDivisionError as e:
        print("Error de división por cero")
```

Validación de datos.py

```python
def validar_datos(nombre, edad, correo):
    # Captura de cualquier otra excepción
    except Exception as e:
        print(f"Error inesperado: {e}")
    # Mensaje final
    finally:
        print("Validación de datos finalizada.")
# Función de datos de prueba
def probar_validaciones():
    # Error de tipo de dato en edad
    validar_datos("Juan Pérez", "a", "juan@xample.com")
    # Error de tipo de dato en nombre
    validar_datos(46478, 30, "46478@example.com")
    # Edad menor a cero
    validar_datos("María Gómez", -5, "maria@example.com")
    # Nombre vacío
    validar_datos("", 22, "nombrevacio@example.com")
    # Correo electrónico inválido
    validar_datos("Luis Martínez", 28, "luismartinezgmail.com")
    #Error de división por cero
    validar_datos("Carlos Sánchez", 0, "carlos@example.com")
    # Caso válido
    validar_datos("Ana López", 25, "ana@example.com")
# Ejecutar las pruebas
probar_validaciones()
```

d) Ejecución y pruebas

Ejemplos del funcionamiento del código:

- Error de tipo de dato en edad:

```
Validando datos para: Juan Pérez, a, juan@xample.com
Error de tipo: Tipo de dato incorrecto.
Validación de datos finalizada.
```

- Error de tipo de dato en nombre:

```
Validando datos para: 46478, 30, 46478@example.com
Error de tipo: Tipo de dato incorrecto.
Validación de datos finalizada.
```

- Edad menor a cero:

```
Validando datos para: María Gómez, -5, maria@example.com
Error de validación: La edad debe ser mayor a cero.
Validación de datos finalizada.
```

- Nombre vacío:

```
Validando datos para: , 22, nombrevacio@example.com
Error de validación: El nombre no puede estar vacío.
Validación de datos finalizada.
```

- Correo electrónico invalido:

```
Validando datos para: Luis Martínez, 28, luismartinezgmail.com
Error de validación: El correo electrónico no es válido.
Validación de datos finalizada.
```

- Error de división por cero:

```
Validando datos para: Carlos Sánchez, 0, carlos@example.com
Error de división por cero
Validación de datos finalizada.
```

- Caso valido:

```
Validando datos para: Ana López, 25, ana@example.com
Datos válidos
Validación de datos finalizada.
```

Conclusión

Implementar un sistema de validación y manejo de errores me permitió comprobar la importancia de controlar adecuadamente los datos ingresados por los usuarios dentro de una aplicación, esto a través de la función validar_datos que logró identificar diferentes tipos de fallas, como errores de tipo, valores inválidos o casos especiales como la división entre cero, demostrando cómo el uso de excepciones mejora significativamente la confiabilidad del programa.

El uso de bloques try, except y finally facilitó la detección y manejo de situaciones inesperadas, evitando que el programa se detuviera abruptamente y mostrara mensajes claros que orientan al usuario o al desarrollador durante el proceso de depuración, también la función de pruebas permitió simular distintos escenarios y verificar que cada validación respondiera de manera adecuada, en general este ejercicio me ayudo a reforzar la importancia de implementar un manejo estructurado de errores para garantizar que la aplicación sea robusta, segura y capaz de gestionar correctamente cualquier situación inesperada.