Arellano Granados Angel Mariano

218123444

Computación Tolerante a Fallas

D06 2023B

Parte 2

Otras herramientas para el manejar errores

Introducción

Existen muchas metodologías y estrategias para lograr un sistema tolerante a fallos, dentro del lenguaje de programación Python lo más común es usar la estructura de try-except, sin embargo, en el presente documento se abordará una alternativa usando la palabra reservada *raise* para generar un programa tolerante a fallos.

Desarrollo

Se creo un código de ejemplo para entender el funcionamiento de la palabra reservada *raise* la cual no sirve para detectar excepciones de manera manual y precisa.

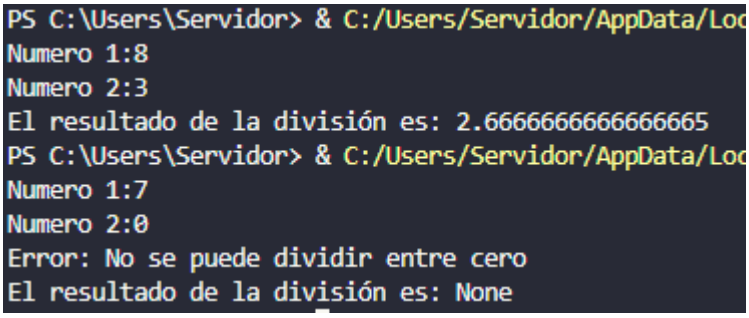
Código:

```
def dividir(a, b):
    if b == 0:
        raise ZeroDivisionError("No se puede dividir entre cero")
    else:
        return a / b

resultado = None

a = float(input("Numero 1:"))
b = float(input("Numero 2:"))

try:
    resultado = dividir(a, b)
except ZeroDivisionError as error:
        print("Error:", error)

print("El resultado de la división es:", resultado)
```

La función dividir toma dos argumentos, a y b. Si b es igual a cero, se lanza manualmente una excepción ZeroDivisionError con un mensaje específico.

Se reciben por consola 2 números y se envían a la función de dividir donde su el número “b” es igual a cero mostrara la excepción que creamos, la llamada a la función se coloca en un bloque de try-excep para que el programa no pare tras mostrar la excepción por fines prácticos del ejemplo.

Ejecución:



En la primera ejecución podemos ver que la división de dos números flotantes funciona como debería, y en la segunda ejecución podemos ver que se intenta dividir entre cero donde entonces salta la excepción que creamos.

En resumen, este código simula una situación en la que se intenta dividir por cero. En lugar de utilizar directamente un bloque try y except, utiliza una verificación manual del divisor y luego lanza y maneja la excepción usando la función raise. Sin embargo, cabe recordar que el uso de bloques try-except es una forma más idiomática y recomendada para manejar excepciones en Python.

Conclusión

En la actualidad los programas de computo son usados por la enorme mayoría de personas en el mundo por ellos nosotros como programadores debemos crear programas tolerantes a fallas para cuando sean sometidos a una gran cantidad de usuarios estos logren cumplir con estándares de calidad y accesibilidad actuales.