COMPUTACION TOLERANTE A FALLAS

Actividad 01 Herramientas para manejar errores

Universidad de Guadalajara

Nombre: Marco Aurelio Domínguez Amezcua

Código: 216818534

Carrera: Ingeniería en computación.

Herramientas para manejo de errores en programación

El manejo de errores en programación es crucial para evitar errores críticos al momento de usar el software. Usualmente, existen que implican errores que provocan fallos que tal vez no pueden ser tan perjudiciales al momento de utilizar un sistema, pero existen otros que detienen totalmente el programa, esto pude implicar perdida de datos o hasta defectos en la funcionalidad, es por esto que se debe realizar prácticas de programación enfocadas para evitar este tipo de errores o fallos, a continuación, describiré algunas de las más comunes dentro de Python.

Try and Except:

Try nos sirve para ejecutar un bloque de código el cual puede contener errores, mientras tanto, *except* es la parte alternativa del código que se ejecuta si y solo si el bloque de código de *try* cuenta con un error, de esta forma podemos evitar que nuestro programa se detenga de forma repentina y no realice los demás procesos que vienen después.

A continuación, muestro un bloque de codigo que debe realizar la misma tarea, ambos están bien, existe un error

```
denominator = int(input("Insert the denominator: "))
numerator = int(input("Insert the numerator: "))

print("The result is ", denominator/numerator)
```

En este bloque de codigo todo parece que esta normal, sin embargo, si nosotros dividimos cualquier numero por *0* nos provocará un error.

```
Insert the denominator: 24
Insert the numerator: 0
Traceback (most recent call last):
  File "C:\Users\Marco Aurelio\OneDrive\Escritorio\6to semestre\Tolerante a fallas - Michel\PythonCodes\main.py", line 5, in <module>
    print("The result is ", denominator/numerator)
ZeroDivisionError: division by zero
```

Como podemos observar ahí mismo nos dice cuál es el error *ZeroDivisionError,* este lo debemos contrarrestar con varias soluciones, una podría ser utilizando condicionales o nuestra herramienta de *try-except*. La solución sería así:

```
denominator = int(input("Insert the denominator: "))
numerator = int(input("Insert the numerator: "))

if numerator != 0:
    print("The result is ", denominator / numerator)
else:
    print("Zero Division Error")
```

Esta es una posible solución, pero es muy simple poner una simple condicional, ya que pueden presentarse más errores a lo largo del programa, y se vería repetitivo el código, por eso podemos optar por la siguiente alternativa:

```
denominator = int(input("Insert the denominator: "))
numerator = int(input("Insert the numerator: "))

try:
    print("The result is ", denominator / numerator)
except ZeroDivisionError as e:
    print(str(e))
```

Types of Exceptions:

Existen varios tipos de *exceptions,* las más conocidas son las siguientes

- *NameError, se genera cuando la variable no se encuentra de forma local ni global.*
- *AttributeError, se genera cuando falla la asignación de atributos o la referencia.*
- *ValueError, se genera cuando una función obtiene un argumento del tipo correcto, pero con valor inadecuado.*
- *ZeroDivisionError, se genera cuando el numerador de una división es cero.*

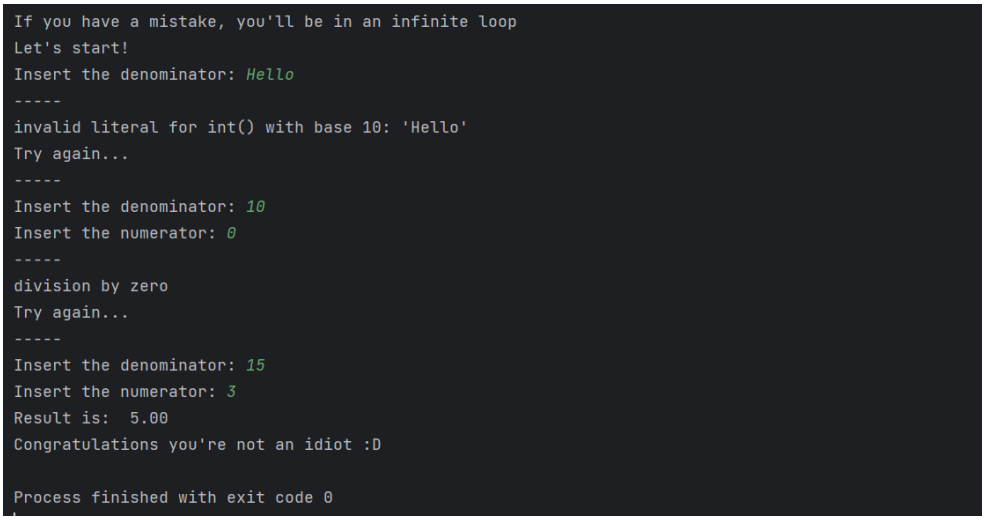
Los anteriores *types of exceptions* los usamos específicamente cuando se van a manejar datos muy básicos, en estos casos no se toman errores de conexión a red o de lectura de archivo.

A continuación, mostraré el ejemplo de cómo se vería un bloque de codigo usando todos estos datos para solamente leer datos correctos:

```
46  print("If you have a mistake, you'll be in an infinite loop\nLet's start!")
47  coin = True
48  while coin:
49      try:
50          denominator = int(input("Insert the denominator: "))
51          numerator = int(input("Insert the numerator: "))
52          print("Result is: ", "{0:.2f}".format(denominator/numerator))
53      except (ZeroDivisionError, ValueError, AttributeError, NameError) as e:
54          print("-----")
55          print(str(e))
56          print("Try again...\n-----")
57      else:
58          print("Congratulations you're not an idiot :D")
59          coin = False
60
```

Con esta herramienta de tolerancia a fallas, podemos ahora cumplir con el objetivo del proyecto sin preocuparnos sin ningún error.

El resultado de la ejecución del programa seria la siguiente:



Conclusión

Es muy importante como evitar errores en nuestros programas, ya que es por esto lo que puede ocasionar un daño ya sea menor o mayor, gracias a diferentes técnicas que existen para contrarrestarlos nosotros podemos hacer uso de ellas para nuestro beneficio, aparte de que nuestro proyecto se vuelve más seguro y profesional.

Bibliografía

- Python exceptions. (n.d.). Programiz.com. Retrieved August 27, 2023, from https://www.programiz.com/python-programming/exceptions
- Pierce, D. (2021, July 23). How to throw exceptions in Python. Rollbar. https://rollbar.com/blog/throwing-exceptions-in-python/
- NeuralNine [@NeuralNine]. (2023, February 9). Advanced Exception Handling in Python. Youtube. https://www.youtube.com/watch?v=ZUqGMDppEDs
- Binaria;, T. [@Tecnobinaria]. (2021, April 21). Controlar ERRORES con TRY - EXCEPT | Curso de Python Intermedio #10. Youtube. https://www.youtube.com/watch?v=BhYSOY6Xbvk