CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS
Universidad de Guadalajara

Otras herramientas para el manejo de errores II

08 de Febrero del 2022

Computación Tolerante a Fallas

Sección D06

Facilitador: MICHEL EMANUEL LOPEZ FRANCO

Alumna: **MEZA CONSEPCIÓN MARLÉN**

Carrera: INCO
Código de alumno: 219748243

Se comenzó por crear un programa que funcione como una calculadora básica, con las operaciones Suma, Resta, Multiplicación, División, y Residuo. El usuario ingresa un valor que corresponde a una operación posible en el menú, y mediante *if* y *elif* es que se piden los números y se realizan las operaciones; el *while True* permite que aún al ingresarse un valor que no corresponde a las operaciones del menú, el programa vuelva a mostrar el menú para la realización de nuevas operaciones, hasta que el usuario pida salir. Al salir de las operaciones, se debe imprimir en pantalla "Proceso Finalizado, Vuelva pronto."

```
while True:
    print("\n\t    --Calculadora simple--")
    print("1. Suma \t 2. Resta \t 3. Multiplicación")
    print("4. División  5. Residuo  6.Salir")
    opc= (int(input("Operación a realizar: ")))

    if (opc ==1):
        num1=(int(input("Primer número: ")))
        num2=(int(input("Segundo número: ")))
        print("\n",num1,"+",num2,"=", num1+num2)
    elif (opc ==2):
        num1=(int(input("Primer número: ")))
        num2=(int(input("Segundo número: ")))
        print("\n",num1,"-",num2,"=", num1-num2)
    elif (opc ==3):
        num1=(int(input("Primer número: ")))
        num2=(int(input("Segundo número: ")))
        print("\n",num1,"*",num2,"=", num1*num2)
    elif (opc ==4):
        num1=(int(input("Dividendo: ")))
        num2=(int(input("Divisor: ")))
        print("\n",num1,"/",num2,"=", num1/num2)
    elif (opc ==5):
        num1=(int(input("Primer número: ")))
        num2=(int(input("Segundo número: ")))
        print("\n",num1,"%",num2,"=", num1%num2)
    elif (opc ==6):
        break

print("\nProceso Finalizado. Vuelva pronto.")
```

Captura 1. Programa inicial

El programa funcionaba correctamente en cuanto a resolver las operaciones.

```
        --Calculadora simple--
1. Suma      2. Resta    3. Multiplicación
4. División  5. Residuo  6.Salir

Operación a realizar: 1

Primer número: 76

Segundo número: 45

 76 + 45 = 121

        --Calculadora simple--
1. Suma      2. Resta    3. Multiplicación
4. División  5. Residuo  6.Salir

Operación a realizar: 2

Primer número: 76

Segundo número: 123

 76 - 123 = -47

        --Calculadora simple--
1. Suma      2. Resta    3. Multiplicación
4. División  5. Residuo  6.Salir

Operación a realizar: 3

Primer número: 87

Segundo número: 3

 87 * 3 = 261
```

Captura 2. Funcionamiento básico inicial

```
        --Calculadora simple--
1. Suma      2. Resta    3. Multiplicación
4. División  5. Residuo  6.Salir

Operación a realizar: 4

Dividendo: 77

Divisor: 11

 77 / 11 = 7.0

        --Calculadora simple--
1. Suma      2. Resta    3. Multiplicación
4. División  5. Residuo  6.Salir

Operación a realizar: 5

Primer número: 89

Segundo número: 10

 89 % 10 = 9

        --Calculadora simple--
1. Suma      2. Resta    3. Multiplicación
4. División  5. Residuo  6.Salir

Operación a realizar: 6

Proceso Finalizado. Vuelva pronto.
```

Captura 3. Funcionamiento básico inicial

El detalle es que, al ingresar un 0 como el segundo número en la realización de una división o residuo, marca un error y finaliza el programa, sin permitir realizar otra operación o siquiera salir delas operaciones y mostrar el último mensaje en pantalla.

Nótese que al regresar el error, muestra la línea donde se genera el error y el tipo de error, son factores importantes para la captura de excepciones.

```
        --Calculadora simple--
1. Suma      2. Resta    3. Multiplicación
4. División  5. Residuo  6.Salir

Operación a realizar: 4

Dividendo: 60

Divisor: 0
Traceback (most recent call last):

  File "C:\Users\macon\Documents\Sem VI\SO\Manejo Errores\untitled0.py", line 29, in <module>
    print("\n",num1,"/",num2,"=", num1/num2)

ZeroDivisionError: division by zero
```

Captura 4. ZeroDivisionError en División

```
        --Calculadora simple--
1. Suma      2. Resta    3. Multiplicación
4. División  5. Residuo  6.Salir

Operación a realizar: 5

Primer número: 41

Segundo número: 0
Traceback (most recent call last):

  File "C:\Users\macon\Documents\Sem VI\SO\Manejo Errores\untitled0.py", line 33, in <module>
    print("\n",num1,"%",num2,"=", num1%num2)

ZeroDivisionError: integer division or modulo by zero
```

Captura 5. ZeroDivisionError en Residuo

Lo mismo sucedía al ingresar los números en cualquier operación, si el input no era entero, marcaba el error y terminaba el programa.

```
        --Calculadora simple--
1. Suma      2. Resta    3. Multiplicación
4. División  5. Residuo  6.Salir

Operación a realizar: 3

Primer número: efrgrt
Traceback (most recent call last):

  File "C:\Users\macon\Documents\Sem VI\SO\Manejo Errores\untitled0.py", line 23, in <module>
    num1=(int(input("Primer número: ")))

ValueError: invalid literal for int() with base 10: 'efrgrt'
```

Captura 6. ValueError al no ingresarse un número entero en cualquier operación

Captura de excepciones

Entonces, en cada *if* y *elif* de las operaciones posibles, se implementó un *try* para que se intenten asignar los valores mediante inputs, seguido de un *except*, donde se especifica el error del que se trata; esto dentro de un *while*, para que si se ingresa un valor no válido, lo vuelva a pedir y así seguir con el flujo del programa.

Aunado, en los *elif* de la División y Residuo, se implementó un *try* para que se intente realizar la operación con los datos ingresado, con el *except ZeroDivisonError*, de no ser posible porque se ingresó 0 en el segundo valor, en lugar de terminar el programa, imprimirá que el valor no es válido; e independiente a si se pudo realizar la operación o no, el *finally* se encarga de imprimir que esa operación fue finalizada.

Al implementar lo mencionado anteriormente, nos aseguramos de que el programa continúe independientemente de si se genera algún tipo de error contemplado.

```
    elif (opc ==4):
        while True:
            try:
                num1=(int(input("Primer número: ")))
                num2=(int(input("Segundo número: ")))
                break
            except ValueError:
                print("Valor no valido. Ingrese numeros enteros.")
        try:
            print("\n",num1,"/",num2,"=", num1/num2)
        except ZeroDivisionError:
            print("\nDivision entre 0. Operacion invalida.")
        finally:
            print("Operacion Finalizada.")

    elif (opc ==5):
        while True:
            try:
                num1=(int(input("Primer número: ")))
                num2=(int(input("Segundo número: ")))
                break
            except ValueError:
                print("Valor no valido. Ingrese numeros enteros.")
        try:
            print("\n",num1,"%",num2,"=", num1%num2)
        except ZeroDivisionError:
            print("\nDivision entre 0. Operacion invalida.")
        finally:
            print("Operacion Finalizada.")

    elif (opc ==6):
        break

print("\nProceso Finalizado. Vuelva pronto.")
```

Captura 7. Captura de excepciones en las operaciones División y Residuo

Al capturarse las excepciones, se muestra que es posible ingresar un valor que no sea entero como cualquiera de los dos números en cualquiera de las operaciones, y en lugar de terminar el programa, nos muestra el error en pantalla, pidiendo y permitiendo corregir la falta para continuar con el flujo del programa.

De igual manera, se puede ingresar un 0 como el segundo número en las divisiones y residuos, y el programa permite continuar realizando operaciones, o bien, salir de las operaciones y mostrar el último mensaje en pantalla.

```
        --Calculadora simple--
1. Suma      2. Resta    3. Multiplicación
4. División  5. Residuo  6.Salir

Operación a realizar: 3

Primer número: dfgrt
Valor no valido. Ingrese numeros enteros.

Primer número: yuyumm
Valor no valido. Ingrese numeros enteros.

Primer número: 8

Segundo número: efvvrv
Valor no valido. Ingrese numeros enteros.

Primer número: 7

Segundo número: 3

 7 * 3 = 21

        --Calculadora simple--
```

Captura 8. Manejo de excepción ValueError

```
        --Calculadora simple--
1. Suma      2. Resta    3. Multiplicación
4. División  5. Residuo  6.Salir

Operación a realizar: 4

Primer número: 7

Segundo número: 0

Division entre 0. Operacion invalida.
Operacion Finalizada.

        --Calculadora simple--
1. Suma      2. Resta    3. Multiplicación
4. División  5. Residuo  6.Salir

Operación a realizar: 5

Primer número: 11

Segundo número: 0

Division entre 0. Operacion invalida.
Operacion Finalizada.

        --Calculadora simple--
1. Suma      2. Resta    3. Multiplicación
4. División  5. Residuo  6.Salir

Operación a realizar: 6

Proceso Finalizado. Vuelva pronto.
```

Captura 9. Manejo de excepción ZeroDivisionError

Por ende, es posible que se siga ejecutante el resto del programa independiente a las operaciones.

```
Division entre 0. Operacion invalida.
Operacion Finalizada.

        --Calculadora simple--
1. Suma      2. Resta    3. Multiplicación
4. División  5. Residuo  6.Salir

Operación a realizar: 6

Proceso Finalizado. Vuelva pronto.
```

Captura 10. Muestra de output final

Repositorio en GitHub: https://github.com/MezaConnie?tab=repositories

Conclusión

Como se mencionó con anterioridad, en prácticamente todo programa que hemos desarrollado a lo largo de la carrera, se he dado un énfasis especial a lo importante que son las validaciones, resultando necesarias para su flujo técnico y lógico; aunque bien en la presente práctica se implementaron estas nociones a relativa pequeña escala, permite comprender lo fundamentales/vitales que estas técnicas pueden ser para un sistema.

Si bien anteriormente no conocía sobre estos bloques y palabras reservadas, de manera inevitable, se manejan bastantes errores al programar, ya sea al resolverlos porque se trata de un simple error de sintáxis, o verlos, no saber cómo manejarlos, y optar por cambiar la estructura.

Sólo el realizar esta práctica aunada a la investigación, permite contemplar la utilidad práctica del manejo de errores a mayor escala, como al tratar con archivos, o Bases de Datos, donde un error podría resultar fatal para el sistema o alguno de sus componentes.

Es agradable cuando de vez en cuando una práctica no se siente tanto como una tarea, sino resulta genuinamente interesante, y lo más importante, que realmente se muestra útil, siendo ya tuve la oportunidad de implementar lo aprendido aquí, en un programa de otra materia.