Day 4: Exception Handling & User-Friendly Mini Calculator

Today’s focus was on Python's Exception Handling. We learned how to use try/except blocks, raise custom exceptions, and build a user-friendly mini calculator that gracefully handles errors.

# Key Concepts Learned:

• try / except blocks – to catch errors and prevent crashes.

• ValueError – handles invalid numeric input.

• ZeroDivisionError – prevents dividing by zero.

• Custom Exceptions – raising and handling user-defined errors (NegativeNumberError).

• Loops – to keep the program running until the user chooses to exit.

# Mini Project: User-Friendly Calculator

Below is the final code implemented for Day 4:

def add(a,b):  
 return a + b  
def subtract(a,b):  
 return a - b  
def multiply(a,b):  
 return a \* b  
def divide(a,b):  
 return a / b  
def calculator():  
 print("Mini calculator")  
 print("1. Add")  
 print("2. Subtract")  
 print("3. Multiply")  
 print("4. Divide")  
  
class NegativeNumberError(Exception):  
 pass  
  
while True:  
 calculator()  
 try:  
 choice = input("Please enter your choice (1-4): ")  
  
 a = float(input("Enter first number: "))  
 b = float(input("Enter second number: "))  
  
 if a < 0 or b < 0:  
 raise NegativeNumberError("Negative Numbers are not allowed!")  
 print(f"Your entered numbers are {a} and {b}")  
  
 if choice == '1':  
 print("Add Result:", add(a,b))  
 elif choice == '2':  
 print("Subtract Result:", subtract(a,b))  
 elif choice == '3':  
 print("Multiply Result:", multiply(a,b))  
 elif choice == '4':  
 if b == 0:  
 raise ZeroDivisionError("Error: Can't divide by zero")  
 else:  
 print("Divide Result:", divide(a,b))  
 else:  
 print("Invalid choice")  
  
 choice\_for\_repeat = input("Do you want to try again? (y/n): ")  
 if choice\_for\_repeat.lower() != 'y':  
 break  
  
 except ValueError:  
 print("Your entered values are not numbers")  
 except NegativeNumberError as e:  
 print(e)  
 except ZeroDivisionError as e:  
 print(e)

# How It Works:

1. The program shows a menu of operations at the start of each loop.  
2. It takes user input for the operation and two numbers.  
3. Validates numbers:  
 - Rejects negative numbers using a custom exception.  
 - Checks for division by zero only in division.  
4. Performs the chosen operation.  
5. Asks if the user wants to try again. Exits if the answer is not 'y'.  
6. Handles errors gracefully with clear messages.

# Summary:

✔ Learned how to use try/except blocks.  
✔ Created and raised custom exceptions.  
✔ Built a loop-based calculator with safe input handling.  
✔ Improved user experience by allowing repeated attempts without crashing.